A primer to numerical simulations: The perihelion motion of Mercury

I. Hammer, C. Hanhart, C. Körber and C. Müller

Forschungszentrum Jülich

E-mail: `c.hanhart@fz-juelich.de`

Abstract. Numerical simulations are playing an increasingly important role in modern science. In this work it is suggested to use a numerical study of the famous perihelion motion of the planet Mercury (one of the prime observables supporting Einsteins General Relativity) as a test case to teach numerical simulations to high school students. The paper includes details about the development of the code as well as a discussion of the visualization of the results. In addition a method is discussed how to estimate the size of the effect a priori.

1. Introduction

Numerical simulations play a key role in modern physics for they allow one to tackle theoretical problems not accessible otherwise, e.g., since there are too many particle participating in the system (as in simulations for weather predictions) or the interactions are too complicated to allow for a systematic, perturbative approach (as in theoretical descriptions of nuclear particles at the fundamental level). The goal of this paper is it to introduce a project that could be used to make numerical simulations themselves the topic of the class. On the example of the perihelion motion of the planet Mercury the students are supposed to get in touch with

- the importance of differential equations in theoretical physics;
- the numerical implementation of Newtonian dynamics;
- systematic tests and optimization of computer codes;
- effective tools to estimate the result a priori as an important cross check;
- the visualization of numerical results using `VPython`.

The course as well as this paper is structured as follows: after an introduction to Newtonian dynamics and the concepts of differential equations their discretization is discussed based on Newtons law of gravitation and possible extensions thereof. Afterwards the visualization of the resulting trajectories using `VPython` is introduced and applied to the problem at hand. In particular tools are developed to extract the relevant quantity from the result of the simulation. Finally the principle of dimensional analysis is presented as a tool to cross check, if the results of the simulation are sensible.

We are convinced that in order to excite the students for numerical simulations it is compulsory to demonstrate their power on an example that they catches their interest. This purpose is served perfectly by the case chosen here, since Einsteins equations of General Relativity are fascinating to a very broad public. While their detailed study needs deeper knowledge in Differential Geometry, for the study outlined here very little math is necessary, such that high school students from about 10th grade up should benefit from it. This was already demonstrated in the 'Schülderakademie Teilchenphysik', where an early version of this course was tested successfully on a goup of 10 German high school students from 10th to 13th grade.

2. Trajectories, velocities, accelerations and Newtons second law

We can say that we understand a physical system, if we can demonstrate that the assumed force leads to the trajectories observed. In other words, we need to show that we can calculate the location in space of the object of interest at any point in time, once the initial conditions are fixed properly. If we can neglect the finite size of this object (and in particular its orientation in space) the location is parametrized by a single three vector $\vec{x}(t)$. As will become clear below, in order to describe and control the dynamics of a physical object in addition we need to be able to calculate its velocity, $\vec{v}(t)$, related to the location via

I see here a potential conflict with the implementation because we use $x(t-\Delta t)$ in order to get v and so on

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \ldots \;, \tag{1}$$

for some infinitesimally small Δt and the acceleration, $\vec{a}(t)$ that describes the change of the velocity

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \vec{a}(t)\Delta t + \ldots \;. \tag{2}$$

The dots in the above expressions indicate that there are in general additional terms that may be expressed with higher powers in Δt, however, for sufficiently small Δt those can be safely neglected. Thus we may define the time derivative via

$$\vec{v}(t) = \lim_{\Delta t \to 0} \frac{\Delta \vec{x}(t)}{\Delta t} =: \frac{d\vec{x}(t)}{dt} = \dot{\vec{x}}(t) \;, \tag{3}$$

where $\Delta \vec{x}(t) = \vec{x}(t+\Delta t) - \vec{x}(t)$ and we introduced with the last expression a common short hand notation for time derivatives. Analogously we get

$$\begin{aligned} \vec{a}(t) &= \lim_{\Delta t \to 0} \frac{\Delta \vec{v}(t)}{\Delta t} =: \frac{d\vec{v}(t)}{dt} = \dot{\vec{v}}(t) \;, & (4) \\ &= \frac{d^2\vec{x}(t)}{dt^2} = \ddot{\vec{x}}(t) \;, & (5) \end{aligned}$$

where we introduced in the second line the second derivative.

It was Newton who observed that if a body is at rest it will remain at rest, and if it is in motion it will remain in motion in a constant velocity in a straight line, unless it is acted upon by some force — this is known as Newtons first law. Said differently: a force $\vec{F}$ shows up by changing the motion of some object. This is quantified in Newton's second law

$$\vec{F}(\vec{x},t) = \frac{d}{dt}(m\vec{v}) \ . \tag{6}$$

If the mass does not change ‡ with time this, reduces to the well known

$$\vec{F}(\vec{x}) = m\vec{a}(t) = m\dot{\vec{v}}(t) = m\ddot{\vec{x}}(t) \ . \tag{7}$$

Note that in general the force could depend also on the time or the velocity. We here restrict ourselves to the case relevant for our example where the force depends on the location only. Therefore, as soon as the force $\vec{F}(\vec{x})$ is known for all $\vec{x}$ one can in principle calculate the trajectory, e.g. solving eq. (7) for $\vec{x}(t)$. Sometimes this requires some advanced knowledge in math, sometimes no closed form solution exists. However, alternatively one can calculate the whole trajectory of some test body that experiences this force by a successive application of the rules given in Eqs. (1) and (2):

See previous comment.

(i) For a given time t, where $\vec{x}(t)$ and $\vec{v}(t)$ are known, use Eq. (7) to calculate $\vec{a}(t)$.

(ii) Use Eq. (1) to calculate $\vec{x}(t+\Delta t)$ and

(iii) then use Eq. (2) to calculate $\vec{v}(t+\Delta t)$.

(iv) Go back to (i) with $t \to t+\Delta t$.

Clearly to initiate the procedure at some time t_0 both $\vec{x}(t_0)$ as well as $\vec{v}(t_0)$ must be known — the trajectories depend on these initial conditions. §

Clearly for this procedure to work Δt must be sufficiently small. What this means depends on the process studied. One way to estimate, if Δt is small enough, is to verify, if the relation

$$|\vec{v}(t)| \gg \frac{1}{2}|\vec{a}(t)|\Delta t = \frac{1}{2m}|\vec{F}(\vec{x}(t))|\Delta t \ . \tag{8}$$

holds, since the next term neglected in Eq. (1) reads $(1/2)a(t)(\Delta t)^2$. Eq. (8) also shows that small (large) time steps are necessary (sufficient), if the force is strong (weak), since the time steps need to be small enough that all changes induced by the force get resolved. Clearly, a relation as Eq. (8) can only provide guidance and can not replace a careful numerical check of the solutions: A valid result has to be insensitive the concrete value Δt chosen — in particular replacing Δt by $\Delta t/2$ should not change the result significantly.

3. Example: General Relativity and the perihelion motion of Mercury

For this concrete example the starting point for the force is Newtons law of gravitation

$$\vec{F}_N(\vec{x}) = \frac{G_N m M_\odot}{r^2}\frac{\vec{x}}{r} \ , \tag{9}$$

‡ A well known example where m does change with time is a rocket, whose mass decreases as the rocket rises.

§ In general a differential equation of $n^{\rm th}$ degree (where the highest derivative is n) needs n initial conditions specified. For $n = 2$ those are often chosen as location and velocity at some defined time, but also to pick two location at different times is possible.

where $G_N = 6.67 \times 10^{-11}$ m^{3}kg^{-1}s-2 is the Newtonian constant of gravitation, m is the mass of Mercury and $M_\odot = 2 \times 10^{30}$ kg is the mass of the sun. In addition $r = |\vec{x}(t)|$ denotes the distance between sun and Mercury, when we assume that the sun is infinitely more heavy than the planet and located at the center of the coordinate system. Although this is not exact, since $m/M_\odot \sim 10^{-8}$ this is a pretty good approximation. For later convenience we introduce the Schwarzschildradius of the sun

This possibly can become a problem for our accuracy because it is at the 10% level of our expected result, right? Same for c.

$$r_S = \frac{2G_N M_\odot}{c^2} = 3 \text{ km} , \tag{10}$$

where $c = 3 \times 10^8$ m/s denotes the speed of light. Note the r_S is the characteristic length scale of the gravitational field of the sun for — up to the prefactor — one can not form another quantity with dimensions of a length from G_N, $M_\odot$ and c. This is what we need r_S for later in the discussion. That the Schwarzschild radius is also an important quantity to characterize black holes is not relevant for the discussion at hand. With this Newtons second law reads

$$\ddot{\vec{x}} = \frac{c^2}{2}\left(\frac{r_S}{r^2}\right) . \tag{11}$$

In general an attractive force that vanishes at large distances leads, depending on the initial conditions, either to closed orbits or open orbits — in the latter case the planet simply disappears from the sun. Depending on time available for the course it might be helpful to explain to the students the origin of this, however, since the most transparent reasoning needs an integration as well as the concept of angular momentum not necessarily familiar to the whole class (details are given in the appendix), it appears in general more appropriate to work out together with the students that a moving body can not be kept by a force if its momentum is too large. Then they may try to find closed orbits in the simulation, e.g., by varying the start velocity while keeping the start location fixed.

I am confused. I thought only $1/r$ and the harmonic oscillator potential lead to closed orbits (Bertrand's theorem).

Include this appendix part...

The closed orbits that emerge from a potential that scales as $1/r$ are elliptic and fixed in space. In particular the point of closest approach of the planet to the sun, the perihelion, does not move. However, as soon as the potential deviates from $1/r$, the perihelion moves. Because of this, the behavior of the perihelion is a very sensitive probe of the gravitational potential.

The observed perihelion motion of Mercury is nowadays determined as $(574.10 \pm 0.65)''$ per 100 earth years ||, but it should be noted that the bulk of this number can be understood by the presence of other planets within the Newtonian theory, since their gravitational force also acts on Mercury. However, a residual motion of $\delta\phi_M = (43)''$ in 100 earth years remained unexplained, until Einstein quantified the predictions of General Relativity to this particular observable.

Here should be a brief description of the relation between Newtons dynamics and General Relativity

We allow for a movement of the perihelion by modifying the potential of Eq. (11) with the following ansatz

$$\ddot{\vec{x}} = \frac{c^2}{2}\left(\frac{r_S}{r^2}\right)\left(1 + \alpha\frac{r_S}{r}\right) , \tag{12}$$

|| The symbol $''$ denotes 'arc seconds': $1'' = (1/3600)^o$.

where α is some parameter related to the effects by general relativity. The idea is that the correction must be dimension-less and should scale with some inverse power of r, for we still need to demand that the potential vanishes at large distances. Since the only parameter of the system with dimension of a length is r_S, the correction should scale as (r_S/r). When replacing r by a typical distance of Mercury to the sun, $(r_S/r) \sim 10^{-7}$, it should be sufficient to include this kind of term to first order only. For comparison: the corresponding number for the Venus, the neighbor of Mercury, reads $r \sim 5 \cdot 10^{-8}$. Therefore the expected ratio of residual motion, favoring in the orbit period T, would result in

$$\delta\phi_V \approx \frac{1}{T_V} \delta\phi_M T_M \frac{r_M}{r_V} \approx 9'' \,, \tag{13}$$

which precisely describes the experimental value of $9''$.

4. Numerical Implementation

4.1. Describing the Motion with Python

In this section we present the numerical implementation as well as the visualization of planetary trajectories and in particular the perihelion motion of Mercury. We select `Python` []as programming language, because `Python` is easy to learn, intuitive to understand and an open source language¶. No prior knowledge of `Python` or any other programming language is required because we explain all the steps necessary to create the simulation. We also provide a working example [] that can be used as template.

Citation

One can additional files to the arXiv upload

To start the simulation one needs the "initial" distances and velocities, which are needed as input to their equation of motions (Eq. (1) and Eq. (2)). As described above, here one can simplify the problem by placing one planet (the "infinitely" heavy sun) in the center of the coordinate system and describe the motion of the other planet (mercury) in a two-dimensional plane.

Because the problem does not depend on the initial position of mercury on its orbit around the sun, we extract the parameters at the perihelion with $|\vec{r}_{MS}(0)| = 46 \cdot 10^6$km and $|\vec{v}_M(0)| = 59$km/s $^+$. Since the computer does not understand physical units, one has to express each variable in an appropriate unit. Both, for the numerical treatment and the intuitive understanding, it is useful to select parameters in a "natural range". A particular useful choice for this problem is given by expressing time intervals in days $T_0 = 1$d, where d refers to earth days, and distances in $R_0 = 10^{10}$ m. With this choice, the initial distance of mercury to the sun, the size of the initial velocity of mercury and the acceleration prefactor become

$$r_{MS}(0) = 4.6 R_0 \,, \quad v_M(0) = 0.51 \frac{R_0}{T_0} \,, \quad a_M = 0.99 \frac{R_0}{T_0^2} \frac{1}{(r_{MS}/R_0)^2} \,. \tag{14}$$

In `Python` , this reads

¶ See also appendix Appendix A for an instruction how to set up `Python` on different operating systems.
$^+$ https://nssdc.gsfc.nasa.gov/planetary/factsheet/mercuryfact.html

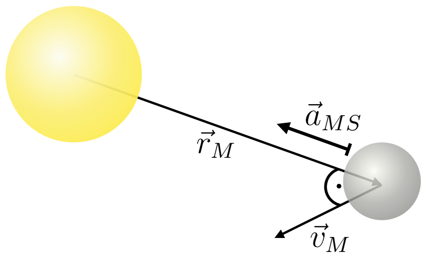

Figure 1. Sun mercury system with relevant vectors. Because the mercury is at its perihelion, its velocity is perpendicular to its direct connection vector with the sun.

```
# Definition of parameters
rM0 = 4.6    # in units of R0
vM0 = 0.51   # in units of R0/T0
aM  = 0.99   # in units of R0/T0**2
T   = 88.    # in units of T0
rS  = 3.e-7  # in units of R0
```

So far we only fixed the length of the vectors. Next, we set up the initial directions, which will describe the motion in the two-dimensional plane. We will build on the existing `Python` module `VPython`, which provides an implementation for treating vectors as well as their visualization. The first object of interest is a `vector`, which takes three-dimensional coordinates as its input. Because of our choice of initial conditions (we picked the initial vectors in the perihelion), the velocity of mercury is perpendicular to the vector which connects mercury and the sun (see figure 1):

```
# Import the class vector from vpythons module visual
from visual import vector
# Initialize distance and velocity vectors
vec_rM0 = vector(0, rM0, 0)
vec_vM0 = vector(vM0, 0, 0)
```

According to Eq. (11), the force which acts on mercury changes the velocity of mercury which eventually changes the position. However, we first need to fix the time step Δt and use the estimate of Eq. (8) as a guidance

Careful: Check units of dt

```
# Definition of the time step
dt = 2 * vM0 / aM / 1000
```

Here the factor 1/1000 makes sure that Δt is indeed consistent with Eq. (8). Now we are in the position to calculate location and velocity of the planet at $t_0 + \Delta t$ using the following comands

I need 1/1000 to get relatively closed orbits – we have to crosscheck codes!

```
# Compute the strength of the acceleration
aMS = aM * ( 1 + alpha * rS / vec_rM_old.mag  ) / vec_rM_old.mag**2
# Multiply by the direction
```

```
vec_aMS = - aMS * ( vec_rM_old / vec_rM_old.mag )
# Update velocity vector
vec_vM_new = vec_vM_old + vec_aMS * dt
# Update position vector
vec_rM_new = vec_rM_old + vec_vM_new * dt
```

Note the beauty of working with the predefined `vector` class: the basic vector operations are already implemented. The difference and sum of two vectors, or the scalar vector multiplication return vectors themselves. Also the magnitude of a vector – `vector.mag` – is a property of the vector and can be easily extracted.

It is handy to use `Pythons functions` to embed repeating structures (”DRY” – Don’t Repeat Yourself)

```
# Define the function
def evolve_mercury(vec_rM_old, vec_vM_old, alpha):
␣␣␣␣#<...Code...>
␣␣␣␣return vec_rM_new, vec_vM_new

# Call the function
vec_rM_new, vec_vM_new = do_time_step(vec_rM_old, vec_vM_old)
```

`Python`′s syntax enforces a clean programming style: it is necessary that the body of the function is indented relative to the definition statement of the function. This is visualized by the long underscore in the previous block. Furthermore, `Python` is an Interpreter language. Each line of the code is execute once the Interpreter passes them. For this reason, all the variables defined before the function are global and thus known to the function. The variables within the function (e.g. `vec_rM_old`, ...) are local and do not live beyond the scope of the function.

Finally, we can express the evolution by a `while`-loop

```
t     = 0
alpha = 0
# Execute the loop as long as t < T
while t < T:
␣␣␣␣vec_rM, vec_vM = evolve_mercury(vec_rM, vec_vM, alpha)
␣␣␣␣t = t + dt
```

where for the start we set the parameter α to zero in order first study the properties of the pure $1/r^2$ force. Note the required indent of the loop structure similar to the indent of a function. In each iteration of the `while`-loop, the previous distance and velocity are used to compute the new values – which directly overwrite the previous values and thus enter the next iteration. The total runtime `T` is the amount of ”virtual” days the simulations should run. To describe at least one full orbital period, T needs to be larger than $T > 88\text{d}$. With the previous choice for `dt`, this corresponds to roughly $N_T = 10^4$ evolution steps.

4.2. Visualizing the Motion with VPython

To start the visualization one has to include further objects of the `VPython` module `visual`. Additional to the vector class, one has to include

```
from visual import vector, sphere, color, curve, rate, display
```

The class sphere will represent mercury and the sun in the simulation

```
# define the initial coordinates; M = mercury, S = sun
M = sphere( pos=vec_rM0,        radius=0.01, color=color.red    )
S = sphere( pos=vector(0,0,0), radius=0.1,  color=color.yellow)
# and the initial velocities
M.velocity = vec_vM0
S.velocity = vector(0,0,0)
```

We have placed the sun in the origin of our coordinate system and choose non-realistic radii sizes for visualization purposes.

To further visualize the trajectory of mercury we add this property to our `mercury` object

```
# add a visible trajectory to mercury
M.trajectory = curve(color=color.black)
```

Next we tell python that the objects shall be drawn from this point on

```
# display the objects from now on
display(background=color.white)
```

and last but not least, instead of updating the object-unrelated vectors in the while loop, we update the object-related vectors which will eventually be drawn

```
t     = 0
alpha = 0
# update the objects
while t < T:
    # set the frame rate: shows four earth days at once
    rate(4*1000)
    # update the drawn trajectory with the current position
    M.trajectory.append(pos=M.pos)
    # update the velocity and position
    M.pos , M.velocity = evolve_mercury(M.pos , M.velocity , alpha)
    # update tim
    t = t + dt
```

If the starting values are chosen as advised in the previous section, then the students should end up with a trajectory as depicted in figure 2. To get the perihelion motion, the term from equation 12 has to be be added. This is a good opportunity to let the students play with the size of α and get a feeling for the impact on the trajectories. However it might be advisable to not start out with the correct mercury values but e.g. a slightly higher initial distance between mercury and sun, since the perihelion motion is better visible then. This is illustrated in figure 2, where we used a value of $\alpha = 10^5$ and $dt = 0.1T_0$.

This is really to large for me to get these orbits. Also I need to run with $\alpha = 10^6$ to get such shapes...

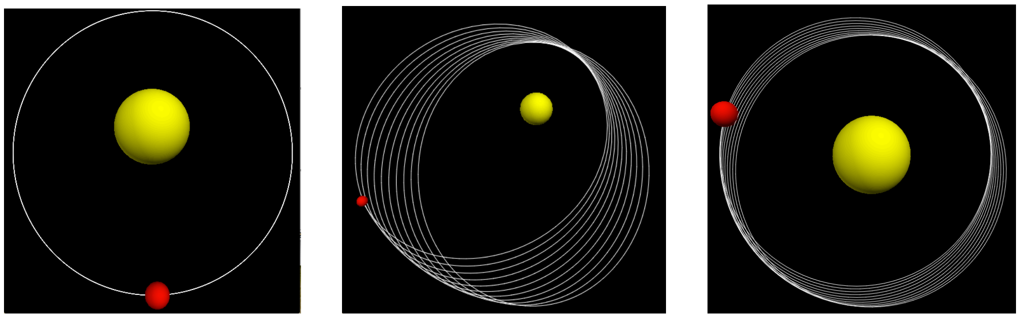

Figure 2. Output for $\alpha = 0$ (left), $\alpha = 10^5$ and bigger starting distance $r_{MS}(0) = 6R_0$ (middle) and $\alpha = 10^5$ with original starting values.

4.3. Running the Program

The complete program needs to be saved as a python file, e.g. `mercury_orbit.py`. Depending on the editor of choice, there might already be a execute functionality e.g. **bla...** for the recommended windows installation (see Appendix A). In case one uses a linux like system and wants to execute the code from a console/terminal, one has to to execute the command

Look up the windows example...

```
python 'address/to/python/file.py'
```

If one already works in the file directory and has used the suggested naming scheme, this reads

```
python mercury_orbit.py
```

Here, the specified `python` interpreter needs to agree with the installed `VPython` version. For more information see Appendix A.

4.4. Extracting the Perihelion Motion

What exactly do we want the students / advise the teachers to do? I think a nice approach would be to let the students implement the code to extract the perihelion motion and make a list of some values of alpha and the corresponding angles. Then discuss, that $\alpha = 1$ is the natural size for α and that to extract the value of angle they need to exploit the linear dependence. Finally they can think on whether the value they get is reasonable and the teacher can explain the rest of section? What do you think?

To calculate the expected size of the perihelion motion for a modified gravitational force, the students need to be abel to extract the relative angle between the current mercury position and the starting position. There are several ways to do this, but the easiest is to extract the point where the distance to the sun gets minimal – the definition of the perihelion– and compare the new position of the perihelion after n turns to the original one. This can be done by looking for a minimal distance of mercury to the sun within a *while*-loop.

why aphelion and not perihelion? I have changed the following code to the perihelion (because it is shorter). We can change it back if desired.

Suppose we want to save the number of turns in the variable `turns`, the maximal number of turns in `max_turns` (here e.g. 10), the initial perihelion is already stored in the vector `vec_rM0` and the goal is to find the final perihelion. In addition to find out when the perihelion is reached, the last two positions `vec_r_last` and `vec_r_before_last` have to be known. Thus, if the last vector is smaller than the previous and the current vector, one has reached the perihelion again. The implementation can be realized as follows

```
# set up vectors
vec_r_last = vec_rM0
turns      = 0
max_turns  = 10
# find perihelion after ten turns
while turns < max_turns:
␣␣vec_r_before_last = vec_r_last
␣␣vec_r_last        = M.pos
␣␣#<...update mercury position...>
␣␣# check if at perihelion
␣␣if vec_r_before_last.mag > vec_r_last.mag < M.pos.mag:
␣␣␣␣turns = turns+1
```

When the loop stops, the last perihelion vector is stored in `vec_r_last`. To compute the angle between two vectors one uses the formula

$$\sphericalangle(\vec{a}_0, \vec{a}) = \cos^{-1}\left(\frac{\vec{a}_0 \cdot \vec{a}}{|\vec{a}_0|\,|\vec{a}|}\right) \tag{15}$$

This is readily implemented in `VPython` via

```
# import function to resolve angle
from visual import acos, pi, dot
angle = acos(
␣␣dot(vec_rM0, vec_r_last.mag) / (vec_rM0.mag * vec_r_last.mag)
) / maxcounter * 380 / 2 / pi
```

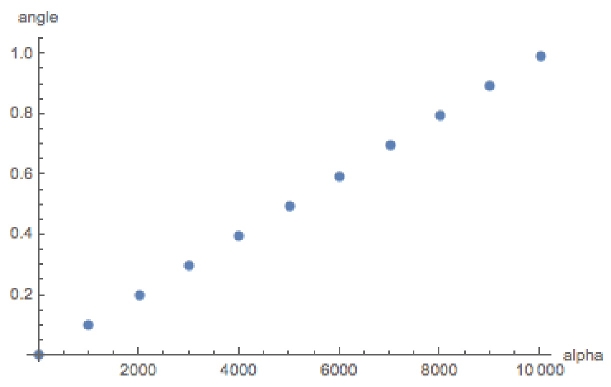

Figure 3. Linear relation between α and the perihelion motion.

As explained above the natural value for α would be 1. However if the student us this value for α, they will find that there is close to no visual change in the trajectories and the distance between the first an the last perihelions is of the order of magnitude of one step in the calculation $v_{aphelion} \cdot dt$. Therefore, to estimate the size of the perihelion motion it is more advisable to use the fact, that there is an approximately linear dependence between α and the perihelion motion, if the angle is not to large. The students can convince themselves of this fact, by plotting some angles over α. This can be done either by hand or using another program like e.g. Excel. One such plot can be found in figure 3.

To get to the angle at $\alpha = 1$ one would ideally employ a linear regression (or ruler fit), but to get the order of magnitude right it is enough to consider only two points. Choosing $\alpha_1 = 1000$ and $\alpha_2 = 2000$ here, this would yield

$$\Theta(\alpha = 1) \approx \frac{0.201 - 0.100}{2000 - 1000} = 0.0001 = 0.36'' \qquad (16)$$

so $3.63'' \cdot 415 \approx 150''$ per 100 earth years. To find out, whether this is a reasonable result, the students could first follow the dimensional analysis and then compare to the actual value for the mercury as outlined above.

5. Tests of stability

6. Dimensional analysis

Dimensional analysis is not only a tool that allows one to cross check if the results of some simulation is of the right order of magnitude, it is also very helpful to identify unusual dynamics in some system. Especially the latter aspect should become clear from the discussion in this section.

The idea of dimensional analysis is that in a system that can be controlled by expanding the relevant quantities (like the force) in some small parameter(s), the coefficients in the expansion should turn out to be of order unity (that means anything between about 0.1 and 10 is fine - but 0.01 or 100 is irritating) — parameters in line with this are called 'natural'. Applied to the problem at hand given by Eq. (11) this statement implies that from naturalness one would expect that the parameter α is of order 1 when one uses for r the average distance Mercury-sun, namely $\bar{r} = 6 \times 10^7$ km. Form this one estimates for the expected angular shift per orbit

$$\delta\phi \simeq 2\pi \left(\frac{r_S}{\bar{r}}\right) = (\pi \times 10^{-7}) \text{ rad} = (2 \times 10^{-5})^o = (7 \times 10^{-2})'' \ , \qquad (17)$$

which leads to a shift of about $30''$ in 100 earth years to be compared to the empirical value of $43''$. Thus indeed the amount of perihelion motion of Mercury is in line with expectations, *if* — and this is an important 'if' — the Newtonian dynamics is simply the leading term of some more general underlying theory. In particular, no new scales enter in the correction terms.

Thanks to Einstein we indeed know that the conclusion formulated above is correct —- the more general underlying theory is General Relativity and indeed Einstein was able to quantitatively explain the perihelion motion of Mercury from his equations.

On the other hand had we found a dramatic deviation of α from unity one would have concluded that there is probably some other dynamics going on that drives this difference.

Indeed, we now several of those hierarchy problems in modern physics: e.g. the so called QCD θ term, expected to be of natural size, is at present known to be at most 10^{-10}. This smallness, called the strong CP problem, is so irritating that physicists like S. Weinberg even proposed that there must exist an additional particle, the axion, whose interactions are in charge of pushing θ even to zero and there are now intense searches for this axion going on various labs.

7. Tasks

Ideas for questions:

- Mass and motion of sun
- Estimate `dt` and also `T` in order to cover roughly two "mercury years".
- What does happen when you change those units?
- Compare values to the values in []. Where does the difference come from?
- Start with $\alpha = 0$. What do you observe?

8. Possible extensions

- **Explore Problem autonomously**
  In chapter 4 we suggested a certain way to present the material. By using a template as well as a step by step instruction, the students are guided rather strictly through the problem solution. We presented this approach with young students and limited time in mind. However if the students are more experienced and enough time is available, it is certainly advisable to give the students space for exploring the problem independently. This could be achieved by the following changes or additions to the concept presented in chapter 4
  - **Build code from scratch**
    Instead of providing the template to the students, they could build the program from scratch. Of course, this requires some basic knowledge in `VPython`, as they could acquire for example by working through a ball in the box example or the like. They could even look up the needed parameters on their own.
  - **Why can we work in one plane?**
    In the code the third coordinate is never used. On the first glance, this could seem like a simplification. Let the students work out on their own, why this does not imply any loss of generality.
  - **What is the impact of the different parameters?**
    Especially if the needed parameters are not specified beforehand, the students might have to experiment a bit, before getting the correct trajectories. But

even if they are given, it might be beneficial to encourage the students to play with a few parameters, like mass or starting velocity, and track their impact on the trajectories. This way the students get a much better feeling for the physics involved.

- **Optimizing performance**
  Simulations always involve a balance between the time needed for the calculations and the accuracy of the results achieved. Even though this is a rather simple example, it contains some opportunities to make this concept accessible to the students.
  - **Consider error due to finite time steps**
    To make the students realise, that the finite time steps lead to inaccuracies in the trajectories encourage them to increase the time steps. This is best done in a program with unmodified gravitational force, as in this case the trajectories are closed ellipses and a deviation is most prominent. If the students choose the time steps big enough, they should witness big deviations. This exercise stresses the point, that by choosing increasingly smaller time steps, deviations from the physical trajectories can be reduced.
  - **Measure calculation time**
    However in practice there is a limit to decreasing the time steps, because the time needed for the calculation grows simultaneously. By including import time

```
start_time = time.time()
main()
print("--- %s seconds ---" % (time.time() - start_time))
```

    the students can measure the time needed by their program. By varying dt they can validate, that there is indeed approximately an anti-proportional dependence. (Note: This only works if the time in the loop is increased by dt, so t=t+dt.)
  - **Verlet integration**
    Of course by optimising the code, an improvement in accuracy can be achieved without increasing the calculation time. The simplest way to demonstrate this might be implementing Verlet integration instead of using the simple Euler method.
- **Extended Problems**
  - **Non-stationary sun**
    It might be interesting to abandon the simplification of a stationary sun, as it nicely illustrates Newton's third law. Here it might also be advisable to reduce the mass of the sun to have a visible result.
  - **Three-body problem**
    Ambitious students could even include another planet and see how the two planets interact. This is especially interesting, when discussing the perihelion

motion of mercury, as it is mainly due to the influence of the other planets. Only a smaller part is due to general relativity.

Acknowledgments

Here they come

Appendix A. Instructions to install VPython

Appendix A.1. Mac

Download VPython for Mac

Appendix B. The code

Here we should put the code

[1] Giovanni Organtini. A ball pool model to illustrate higgs physics to the public. Physics Education, 52(2):023001, 2017.