Physics Education

PAPER

A primer to numerical simulations: the perihelion motion of Mercury

To cite this article: C Körber *et al* 2018 *Phys. Educ.* **53** 055007

PAPER

Phys. Educ. 53 (2018) 055007 (19pp) iopscience.org/ped

A primer to numerical simulations: the perihelion motion of Mercury

C Körber[1], I Hammer[1], J-L Wynen[1], J Heuer[2,4], C Müller[3] and C Hanhart[1]

[1] Institut für Kernphysik (IKP-3) and Institute for Advanced Simulation (IAS-4), Forschungszentrum Jülich, D-52425 Jülich, Germany
[2] Hochschule Hamm-Lippstadt, Marker Allee 76-78, 59063 Hamm, Germany
[3] Schülerlabor JuLab, Forschungszentrum Jülich, D-52425 Jülich, Germany

E-mail: c.koerber@fz-juelich.de and 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 that allows one to estimate the size of the effect as well as the uncertainty of the approach *a priori*. At the same time this enables the students to double check the results found numerically. The course is structured into a basic block and two further refinements which aim at more advanced students.

1. Introduction

Numerical simulations play a key role in modern physics because they allow one to tackle some theoretical problems not accessible otherwise. This might be the case because there are too many particles 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). This paper introduces a project that allows one to demonstrate the power of numerical simulations. On the example of the perihelion motion of the planet Mercury the students are supposed to learn about

- 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`.

We are convinced that in order to excite students for numerical simulations it is compulsory to demonstrate their power on an example that catches their interest. This purpose is served

[4] Present Address: Institut für Neurowissenschaften und Medizin (INM-4), Forschungszentrum Jülich, D-52425 Jülich, Germany

perfectly by the case chosen here, since Einsteins equations of general relativity are fascinating to a very broad public. Their detailed study needs a deep understanding of Differential Geometry. For the course presented here, however, very little math is necessary, such that high school students vaguely familiar to vector calculus and derivatives will benefit from it. This was already demonstrated in the 'Schülerakademie Teilchenphysik', where this course was tested successfully on two groups consisting in total of 24 German high school students from 10th to 13th grade in 2015 and 2017. As an example on how the course can be implemented in practice, we describe our own experiences with it in the second to last section.

The course as well as this paper is structured as follows: after an introduction to Newtonian dynamics and the concept of differential equations, their discretization is discussed based on Newton's 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 addition, 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 estimate the size of the effect studied as well as its expected accuracy. Furthermore the method allows one to cross check the results of the simulation. As is pointed out in the chapters below the course can be finished at some canonical points—in particular we think that for most students a detailed study of the uncertainties of the numerical simulations might be too technical. However, we include this discussion as well both for completeness and as an additional challenge to the more advanced students.

2. Trajectories, velocities, accelerations and Newton's second law

A classical physics system is said to be understood, if the assumed forces acting on it lead to the observed trajectories. In other words, we need to show that we can calculate the location in space of the object of interest at any future 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 $\vec{r}(t)$. Additionally, we need to be able to calculate the objects velocity $\vec{v}(t)$ and acceleration $\vec{a}(t)$ in order to describe and control its dynamics; this will become clear below. The velocity describes a change of location over time which, for some infinitesimally small Δt, can be written as

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \dots \, . \qquad (1)$$

Similarly, the acceleration describes a change in velocity, i.e.

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \vec{a}(t)\Delta t + \dots \, . \qquad (2)$$

The dots in these expressions indicate additional terms that may be expressed in terms of higher powers in Δt. While they are needed in general, for sufficiently small Δt they can be safely neglected. Thus we may define the time derivative via

$$\vec{v}(t) = \lim_{\Delta t \to 0} \frac{\Delta \vec{r}(t)}{\Delta t} =: \frac{\mathrm{d}\vec{r}(t)}{\mathrm{d}t} = \dot{\vec{r}}(t), \qquad (3)$$

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

$$\vec{a}(t) = \lim_{\Delta t \to 0} \frac{\Delta \vec{v}(t)}{\Delta t} =: \frac{\mathrm{d}\vec{v}(t)}{\mathrm{d}t} = \dot{\vec{v}}(t) = \ddot{\vec{r}}(t), \qquad (4)$$

where we introduced the second derivative in the last step.

The dynamics is controlled by Newton's second law

$$\vec{F}(\vec{r}, t) = \frac{\mathrm{d}}{\mathrm{d}t}(m\vec{v}) \, . \qquad (5)$$

If the mass does not change[5] with time, this reduces to the well known

$$\vec{F}(\vec{r}) = m\vec{a}(t) = m\dot{\vec{v}}(t) = m\ddot{\vec{r}}(t) \, . \qquad (6)$$

Note that in general the force could depend also on the time or the velocity. Here, we restrict ourselves to the case relevant for our example, where the force depends on the location only. Therefore, as soon as the force $\vec{F}(\vec{r})$ is known for all $\vec{r}$ one can in principle calculate the trajectory by solving equation (6) for $\vec{r}(t)$. Sometimes this requires some advanced knowledge of mathematics and sometimes no closed form solution exists. However, alternatively one can calculate the whole trajectory of some test body that experiences this force

[5] A well known example where m does change with time is a rocket, whose mass decreases as fuel is burned.

by a successive application of the rules given in equations (1) and (2):

(i) For a given time t, where $\vec{r}(t)$ and $\vec{v}(t)$ are known, use the force to calculate $\vec{a}(t) = \vec{F}(\vec{r}(t))/m$, see equation (6).
(ii) Use equation (1) to calculate $\vec{r}(t + \Delta t)$.
(iii) Use equation (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 $\vec{r}(t_0)$ as well as $\vec{v}(t_0)$ must be fixed—the trajectories depend on these initial conditions[6].

This procedure can only work if Δt is sufficiently small. One way to estimate whether Δt is small enough is to verify whether the relation

$$|\vec{v}(t)| \gg \frac{1}{2}|\vec{a}(t)|\Delta t = \frac{1}{2m}|\vec{F}(\vec{r}(t))|\Delta t \qquad (7)$$

holds. This follows from equation (1) where the first term that was neglected reads $(1/2)a(t)(\Delta t)^2$. The impact of the size of Δt on the simulation is illustrated in figure 1 where only Newtonian gravity was used. While for the calculations for the left panel Δt is chosen significantly below $\Delta t_0 = 2m|\vec{v}(0)|/|\vec{F}(\vec{r}(0))|$ (see equation (7)), for the right panel it was chosen twice as large. Accordingly, only the trajectory in the left panel shows the characteristic feature of a $1/r^2$—force of an ellipse fixed in space. The trajectory in the right panel does not reproduce itself in successive revolutions.

Equation (7) 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 equation (7) can only provide guidance and can not replace a careful numerical check of the solutions: a valid result has to be insensitive to the concrete value chosen for Δt—in particular replacing Δt by $\Delta t/2$ should not change the result significantly. This issue will be discussed in more detail below.

3. Example: general relativity and the perihelion motion of Mercury

For this concrete example the starting point for the force is Newton's law of gravitation

$$\vec{F}_{\mathrm{N}}(\vec{r}) = -\frac{G_{\mathrm{N}} m M_\odot}{r^2}\frac{\vec{r}}{r}, \qquad (8)$$

where $G_{\mathrm{N}} = 6.67 \times 10^{-11}$ m^3 kg^{-1} s^{-2} is the Newtonian constant of gravitation, m is the mass of Mercury and $M_\odot = 1.99 \times 10^{30}$ kg is the mass of the Sun. In addition $r = |\vec{r}(t)|$ denotes the distance between Sun and Mercury, when we assume that the Sun is infinitely heavier than Mercury and located at the center of the coordinate system. Although this is not exact, it is a good approximation because $m/M_\odot \sim 10^{-8}$. For later convenience we introduce the Schwarzschild radius of the Sun

$$r_{\mathrm{S}} = \frac{2G_{\mathrm{N}}M_\odot}{c^2} = 2.95\,\mathrm{km}, \qquad (9)$$

where $c = 3.00 \times 10^8$ m s^{-1} denotes the speed of light. Note that 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. The Schwarzschild radius is also an important quantity to characterize black holes; this is however not relevant for the discussion at hand. With this, Newton's second law reads

$$\ddot{\vec{r}} = -\frac{c^2}{2}\left(\frac{r_{\mathrm{S}}}{r^2}\right)\frac{\vec{r}}{r}\,. \qquad (10)$$

An attractive force that vanishes at large distances leads in general, depending on the initial conditions, either to bounded or to open orbits. In the latter case the planet simply disappears from the Sun, since a given force can capture only bodies with small enough momenta. The students may study those scenarios within their simulations by varying the start velocity while keeping the start location fixed.

The bounded orbits that emerge from a potential that scales as $1/r$ (which corresponds to a force that scales as $1/r^2$) 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, when a potential that vanishes for $r \to \infty$ deviates from $1/r$, the perihelion does move. This makes the behavior of the perihelion a very sensitive probe of the gravitational potential.

[6] In general, a differential equation of nth degree (where the highest derivative is order n) needs n initial conditions specified. For $n = 2$ those are often chosen as location and velocity at some starting time, but one may as well pick two locations at different times.

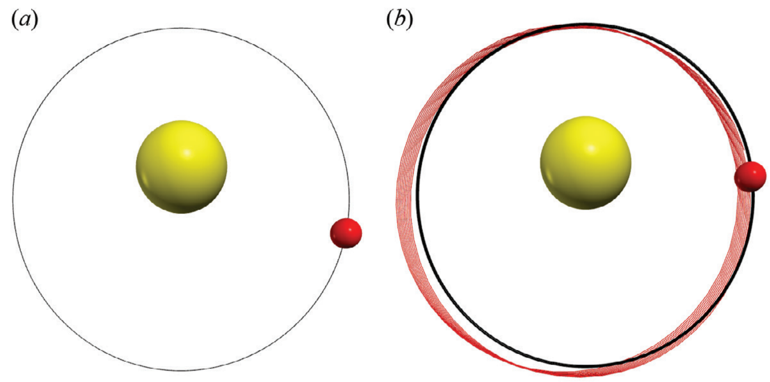

Figure 1. Different Mercury orbits for the purely Newtonian gravity (8) and $\Delta t = \Delta t_0/20$ for (a) and $\Delta t = \Delta t_0 \times 2$ for (b), where the black line represents the orbit of (a) and the red line is the orbit for the larger time steps. The reference time step Δt_0 is defined via inequality (7) as explained in the text. The images are screenshots of the simulation that is described below (with modified colors).

The observed perihelion motion of Mercury is nowadays determined as

$$(574.10 \pm 0.65)'' \text{ per 100 earth years [1]}$$

where the symbol $''$ denotes 'arc seconds': $1'' = (1/3600)^\circ$. 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

$$\delta\Theta_{\mathrm{M}} = (42.56 \pm 0.94)'' \text{ per 100 earth years [1]} \tag{11}$$

remained unexplained, until Einstein quantified the predictions of general relativity to this particular observable [2]

$$\delta\Theta_{\mathrm{GR}} = (43.03 \pm 0.03)'' \text{ per 100 earth years [1]}\,. \tag{12}$$

To allow for a movement of the perihelion we need to modify the force that led to equation (10). This modification should be such that it depends on r and it vanishes for $r \to \infty$. Based on the discussion presented above we may multiply the right hand side of (10) with a factor $(1 + \alpha \frac{r_{\mathrm{S}}}{r})$, where α denotes some dimensionless parameter. This seems like a natural way to parametrize the additional potential because it adds the smallest possible deviation from a $1/r$ potential expressed relative to the characteristic length r_{S}. However, besides r_{S}, which characterizes the gravitational potential of the Sun, there is also a characteristic parameter for the dynamics of Mercury:

$$r_{\mathrm{L}}^2 := \frac{\vec{L}^{\,2}}{m^2 c^2} = \frac{(\vec{r} \times \dot{\vec{r}})^2}{c^2}, \tag{13}$$

where $\vec{L}$ denotes the angular momentum, which is a constant of motion for central potentials. It is easily verified that r_{L}^2 carries dimensions of length squared. We therefore have to add one more term to the potential; in analogy to the one discussed above, we use $\beta \frac{r_{\mathrm{L}}^2}{r^2}$. Here one might wonder why this additional term is chosen proportional to $(r_{\mathrm{L}}/r)^2$ and not to (r_{L}/r). The reason for this choice is indeed not obvious: in general the correction terms added must be scalar quantities—accordingly vectors can enter only as scalar products. Moreover, one is not allowed to use square-roots of those scalar products for these impose wrong mathematical properties to the equations of motion (for the case at hand, e.g. the derivative with respect to the velocity would not be defined at $\dot{\vec{r}} = 0$)—the only vector that is allowed to appear as its length in linear order is the radius, since this length is related to geometrical properties of the system. Combining everything, we use the following ansatz for the modified equation of motion:

$$\ddot{\vec{r}} = -\frac{c^2}{2}\frac{r_{\mathrm{S}}}{r^2}\left(1 + \alpha\frac{r_{\mathrm{S}}}{r} + \beta\frac{r_{\mathrm{L}}^2}{r^2}\right)\frac{\vec{r}}{r}\,. \tag{14}$$

For the system at hand, r_{L}^2 may be estimated from the parameters of Mercury at its perihelion: the corresponding velocity is $\dot{r}(t=0) = |\vec{v}_{\mathrm{M}}(0)| = 59.0$ km s^{-1} (here we

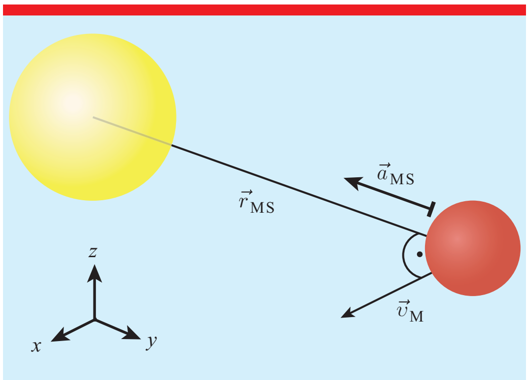

Figure 2. Sun–Mercury system with relevant vectors. Mercury is at its perihelion, therefore its velocity is perpendicular to its direct connection vector with the Sun.

already indicate that we will start the simulation at the perihelion) and the closest distance between Sun and Mercury is $r_{\rm MS} = |\vec{r}_{\rm MS}(0)| = 46.0 \cdot 10^6$ km [3]. Note that at the perihelion $\vec{r}$ and $\dot{\vec{r}}$ are perpendicular to each other (see figure 2). We thus find that $(r_{\rm L}^2/r_{\rm MS}^2) \sim 4 \cdot 10^{-8}$. Furthermore $(r_{\rm S}/r_{\rm MS}) \sim 6 \cdot 10^{-8}$. Therefore one may expect from both correction terms an effect of similar size—for a more detailed discussion about the underlying logic we refer to section 6. Furthermore, any term of higher order in either $r_{\rm L}^2$ or $r_{\rm S}$ should be suppressed by additional seven orders of magnitude and thus can be neglected safely. The parameters α and β can be extracted from a fit to data. However, since we have only a single number to study, the perihelion shift (see equation (11)), we can only fix either α or β (or a linear combination thereof). The parameters can also be calculated from the underlying theory one uses. The actual values depend on the specific theory; general relativity gives [2]

$$\alpha = 0\,, \quad \beta = 3\,. \tag{15}$$

Thus we may also use the simulation explained below to calculate the perihelion motion of Mercury from the input values given in equation (15). Modifications to the Newtonian equation of gravity introduced to account for the perihelion motion of Mercury are discussed in great detail in a very pedagogical way also in [4].

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 choose `Python` [5] 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, as we explain all necessary steps to create the simulation. In the following, we present code examples which tested for `Python` version `3.6.4` and `VPython` version `7.3.2`. We provide online instructions for setting up `Python` and `VPython` on different operating systems [6]. In appendix A, we also provide a working example along with a flowchart in figure A1 showing the logic of the full program as well as possible extensions and template files online.

To start the simulation one needs the 'initial' distance and velocity (equations (1) and (2)). As described above, here one can simplify the problem by placing one object (the 'infinitely' heavy Sun) in the center of the coordinate system and keep it fixed.

Since the trajectories do not depend on where on the orbit of Mercury the simulation is started, we use the values at the perihelion with $|\vec{r}_{\rm MS}(0)| = 46.0 \cdot 10^6$ km and $|\vec{v}_{\rm M}(0)| = 59.0$ km s^{-1} [3] as initial (for $t = 0$) parameters. The computer does not understand physical units, hence 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', e.g. by expressing distances in $R_0 = 10^{10}$ m and time intervals in days, $T_0 = 1$ d, where d refers to earth days. One Mercury year is given by $T_{\rm M} = 88.0\, T_0$. With this choice, the initial distance of Mercury to the Sun, the size of the initial velocity of Mercury and the acceleration prefactor (see equation (14)) become

$$r_{\rm MS}(0) = 4.60 R_0\,, \quad v_{\rm M}(0) = 0.510 \frac{R_0}{T_0}\,, \tag{16}$$

$$a_{\rm MS}(r_{\rm MS}) = \frac{c^2}{2} \frac{r_s}{r_{\rm MS}^2} = 0.990 \frac{R_0}{T_0^2} \frac{1}{(r_{\rm MS}/R_0)^2}\,. \tag{17}$$

In Python, this reads

```
# Definition of parameters
rM0 = 4.60       #  Initial radius of Mercury orbit, in units of R0
vM0 = 5.10e-1    #  Initial orbital speed of Mercury, in units of R0/T0
c_a = 9.90e-1    #  Base acceleration of Mercury, in units of R0**3/T0**2
TM  = 8.80e+1    #  Orbit period of Mercury
rS  = 2.95e-7    #  Schwarzschild radius of Sun,in units of R0
rL2 = 8.19e-7    #  Specific angular momentum, in units of R0**2
```

where `c_a` $= a_{\mathrm{MS}} r_{\mathrm{MS}}^2$, the acceleration without the radius factor. The last two quantities refer to the Schwarzschild radius of the Sun and the parameter r_L^2 defined in equations (9) and (13), respectively.

So far we only fixed the length of the vectors. Next, we set up the initial directions, which will describe the motion in space. 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. With 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 Sun (see figure 2):

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

[7] We use the estimate of equation (7) as a guidance to fix the time step Δt as

```
# Definition of the time step
dt = 2 * vM0 / c_a / 20
```

Here the factor 1/20 makes sure that Δt is indeed consistent with equation (7). Now we are in the position to calculate location and velocity of the planet at $t_0 + \Delta t$ using the following expressions[8]

```
# Compute the strength of the acceleration
temp = 1 + alpha * rS / vec_rM_old.mag + beta * rL2 / vec_rM_old.mag**2
aMS  = c_a * temp / 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
```

[7] Note that the code is designed for VPython versions 7 and later. For VPython versions prior 7, e.g. the statement from `vpython import vector` must be replaced by `from visual import vector`.

[8] Note, that there are different notions for numerically integrating differential equations with different accuracies. Thus, the ordering and exact expressions for updating a_{MS}, v_{M} and r_{M}S are not unique. However, all correct prescriptions lead to the same result for sufficiently small Δt. For this particular problem, the definitions we show are in good balance between being numerically stable and inexpensive to compute.

Note basic vector operations are already implemented in the predefined `vector` class. The difference and sum of two vectors, or the scalar vector multiplication return vectors themselves. Also the magnitude of a vector—`vector.mag`—is an attribute of the vector and can be easily extracted.

It is handy to use `Python`s functions to structure the program and hold repeating code ('DRY'—Don't Repeat Yourself):

```
# Define the coordinate and velocity update function
def evolve_mercury(vec_rM_old, vec_vM_old, alpha, beta):
    # Compute the strength of the acceleration
    temp = 1 + alpha * rS / vec_rM_old.mag + beta * rL2 / vec_rM_old.mag**2
    aMS  = c_a * temp / 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
    return vec_rM_new, vec_vM_new

# Call the function
vec_rM_new, vec_vM_new = evolve_Mercury(vec_rM_old, vec_vM_old, 0.0, 0.0)
```

`Pythons` syntax enforces a clean programming style: it is necessary that the body of the function is indented (by an arbitrary but consistent amount of `spaces`[9]) relative to the definition statement of the function. Furthermore, `Python` is an Interpreter language. Each line of the code is executed when the Interpreter passes it. For this reason, if we define the parameters before the function, they can be used inside of the function. Variables defined within the function (e.g. `aMS`, `vec_aMS`, …) are local and do not exist beyond the scope of the function; they can not be used after the `return` statement.

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

```
t      = 0.0
alpha  = 0.0
beta   = 0.0
# Set position and velocity to their starting points
vec_rM = vec_rM0
vec_vM = vec_vM0
# Execute the loop as long as t < 2*TM
while t < 2*TM:
    # Update position and velocity
    vec_rM, vec_vM = evolve_mercury(vec_rM, vec_vM, alpha, beta)
    # Advance time by one step
    t = t + dt
```

where for the start we set the parameters $\alpha = 0 = \beta$ in order to 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, directly overwriting the previous values. The total runtime `2*TM` is the amount of 'virtual' days the

[9] While `Python` allows using `tabs` as well, this can cause programming errors, because their displayed width depends on the text editor.

simulations should run. To describe at least one full orbital period, this time needs to be larger than T_{M}. With the previous choice for `dt`, this corresponds to roughly $N_T \approx 2 \cdot 10^3$ evolution steps. Note that the exact time it takes for Mercury to complete a full revolution depends on the initial coordinates and velocities, the accuracy of the computation (controlled by the value of Δt) as well as the computing power employed. One should encourage the students to analyze this in the beginning.

4.2. Visualizing the motion with VPython

To start the visualization one has to `import` further objects from the `VPython` module. We change the import statement from before to

```
from vpython import vector, sphere, color, curve, rate
```

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=0.5,  color=color.red   )
S = sphere(pos=vector(0,0,0), radius=1.5,  color=color.yellow)
# And the initial velocities
M.velocity = vec_vM0
S.velocity = vector(0,0,0)
# Add a visible trajectory to Mercury
M.trajectory = curve(color=color.white)
```

We place the Sun in the origin of our coordinate system and choose non-realistic radii for visualization purposes. Last but not least, one should use in the code the vectors directly related to the visualization. Thus the `while`-loop becomes

```
t     = 0.0
alpha = 0.0
beta  = 0.0
# Execute the loop as long as t < 2*TM
while t < 2*TM:
    # Set the frame rate (you can choose a higher rate to accelerate the program)
    rate(100)
    # Update the drawn trajectory with the current position
    M.trajectory.append(pos=M.pos)
    # Update velocity and position
    M.pos, M.velocity = evolve_mercury(M.pos , M.velocity , alpha, beta)
    # Advance time by one step
    t = t + dt
```

If the starting values are chosen as advised in the previous section, the students should end up with a trajectory as depicted in figure 1(a). At this point one might ask students to vary Δt and observe the effect of this on the orbit (see also figure 1(b)). To get the perihelion motion, the additional force term described in equation (14) has to be 'turned on', e.g. by choosing either $\alpha \neq 0$ or $\beta \neq 0$ (figure 3(b)). This is a good opportunity to let the students play with the size of α or β and get a feeling for its impact

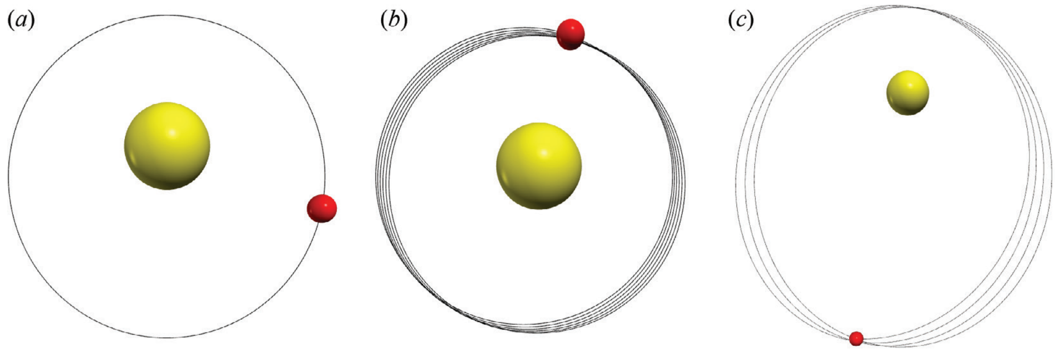

Figure 3. Different Mercury orbits for $\beta = 0$ and (a) $\alpha = 0$ and $\Delta t = \Delta t_0/20$; (b) $\alpha = 10^6$ and $\Delta t = \Delta t_0/20$; (c) $\alpha = 10^6$ and $\Delta t = \Delta t_0/20$ but $r_{\mathrm{MS}}(0) = 6R_0$. Here, the time steps are defined by $\Delta t_0 \equiv 2v_{\mathrm{M}}(0)/a_{\mathrm{M}}(0)$ and the images are screenshots of the simulation (with modified colors). Note that for different starting values of $r_{\mathrm{MS}}(0)$, also the numbers of days for a 'Mercury year' changes.

on the trajectories. In particular they should discover that values of α or β larger than 10^5 are necessary to get a visible effect. For an even more enhanced perihelion motion it is advisable to not use the correct Mercury values but a more excentric trajectory by choosing, e.g. $r_{\mathrm{MS}}(0) = 6R_0$ (figure 3(c)).

This finalizes the basic course—for the not so ambitious students this might be a good point to stop. Already up to here the students should have learned a lot about numerical physics by successfully simulating the perihelion motion. The chapters to come are increasingly technical and thus address more advanced students.

4.3. *Extracting the Perihelion Motion*

Both α and β have similar effects on the perihelion motion. We recommend to set one of them to zero and to vary the other, when analyzing the perihelion motion. In order to calculate the size of the perihelion motion for a modified gravitational force, the students need to

- Extract multiple positions of the perihelion $\vec{r}_{\mathrm{MS}}(t_{\mathrm{ph}}^{(n)})$ for a fixed value of α and β. The simulation time needs to cover several revolutions.
- Calculate the angle between the perihelions for each pair of successive turns and compute the average angle $\delta\Theta(\alpha)$ over all individually computed angles.
- Repeat the steps given above for different values of α (or β). The students should find a linear dependence of $\delta\Theta$ on α or β.
- Interpolate $\delta\Theta(\alpha, \beta)$ to small values of α (or β)—including $\delta\Theta(0, 0) = 0$.

There are several ways to find the position of the perihelion. The easiest one is to look for the point of minimal distance to the Sun—the definition of the perihelion. This can be done within the *while*-loop. To identify a minimal distance to the Sun, one has to know the last two positions `vec_rM_past` and `vec_rM_before_last` in addition to the current position. If the length of the `last` vector is smaller than that of the `before_last` vector and smaller than the current vector, one has passed the perihelion and its position is given by `vec_r_last`. It is useful to save the position of the perihelion in a list `list_perih` for a certain number of turns (here e.g. `max_turns = 10`). The implementation can be realized as follows[10]

[10] Note that one stores `vec_r_last` by making a copy of `M.pos`: `vec_r_last = vector(M.pos)`. This is essential because otherwise, if one changes `M.pos`, one would automatically change `vec_r_last` as well.

```
# Set up vectors
vec_rM_last = vec_rM0
turns       = 0
max_turns   = 10
list_perih  = list()
# Find perihelion for each turn and print it out
while turns < max_turns:
    vec_rM_before_last = vec_rM_last
    # Store position of Mercury in a new vector (since we will change M.pos)
    vec_rM_last = vector(M.pos)
    # <...update Mercury position...>
    # Check if at perihelion
    if (vec_rM_last.mag < M.pos.mag) and (vec_rM_last.mag < vec_rM_before_last.mag):
        list_perih.append(vec_rM_last)
        turns = turns + 1
```

The angle between two vectors can be computed using

$$\sphericalangle(\vec{v}_1, \vec{v}_2) = \arccos\left(\frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1|\,|\vec{v}_2|}\right) . \tag{18}$$

This is readily implemented in VPython via

```
# Import functions to compute angle
from vpython import acos, pi, dot
# Define function for angle extraction
def angle_between(v1, v2):
    return acos( dot(v1, v2) / (v1.mag * v2.mag) ) * 180. / pi
```

where the factor $180/\pi$ in the last line converts the unit of the angle from radians into degrees. To account for the statistical errors (e.g. numerical rounding errors) one can average over a few turns. Depending on the programming proficiency of the students and time constraints, this can be done either by hand or, e.g. by implementing the following code

```
sum_angle=0.
for n in range(1, max_turns):
    # Calculate angle
    sum_angle = sum_angle + angle_between(list_perih[n-1],list_perih[n])
# Display the average
print(sum_angle/(max_turns-1))
```

Note that the perihelion motion is computed from the locations stored in `list_perih` and not based on the initial position. This is important because depending on the initial conditions and numerical uncertainties, the simulation does not necessarily start in the exact perihelion.

As explained above (and further discussed in section 6) the natural value for α (and β) would be of the order of 1. However, if the students use this value for α, they will find that the change in the trajectories is close to invisible and the numerical uncertainty is much larger than the result. It is therefore more advisable to use the fact that there is a linear dependence between α and $\delta\Theta$ to estimate the size of the perihelion motion. On the other hand as soon as the values of α or β get too large the effective

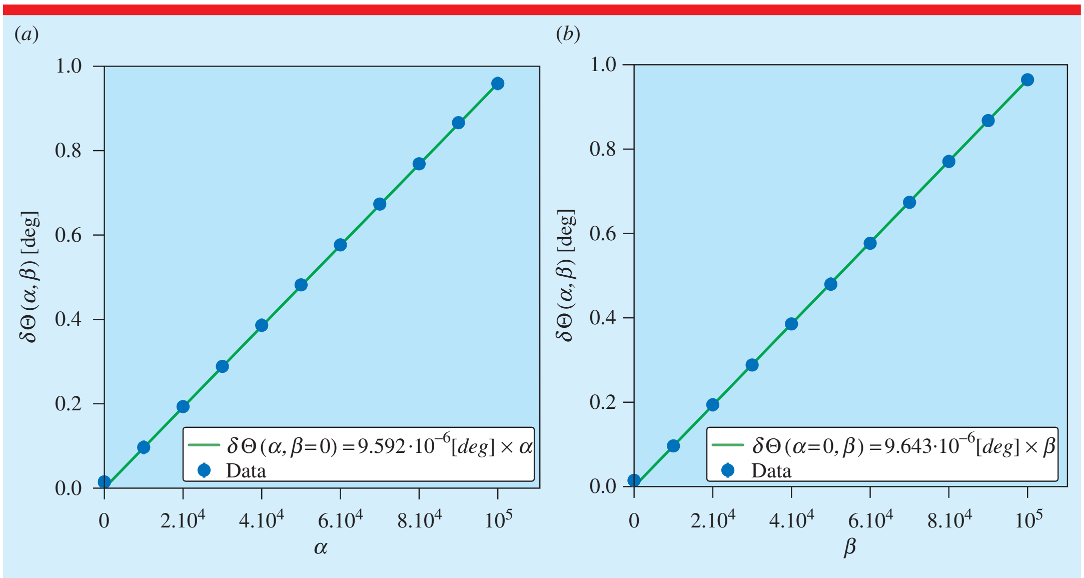

Figure 4. Linear relation between α (a), β (b) and the perihelion motion $\delta\Theta$ for $\Delta t = 2v_{\mathrm{M}}(0)/a_{\mathrm{M}}(0)/200$ and the other parameter set to zero.

ansatz to use only the leading terms in the (r_s/r) and $(r_L/r)^2$ expansion is no longer justified and the method of extraction might get unreliable. At this point we therefore recommend to choose the parameters α and β at most of the order of 10^5—this value will be better justified in section 6.

The students should convince themselves that in this parameter range there is a linear relation between the angles $\delta\Theta$ and the parameters α or β:

$$\delta\Theta(\alpha,\beta) = m_\alpha \cdot \alpha + m_\beta \cdot \beta\,. \tag{19}$$

This can be done either by hand, `Python` (e.g. with `matplotlib` [7]), or using another program like, e.g. Excel. For instance figure 4 demonstrates such plots.

As an example we extract the perihelion motion using the parameters provided by general relativity, namely $\alpha = 0$ and $\beta = 3$ evaluated with $\Delta t = \Delta t_0/200$. Considering one result from the numeric simulation, e.g. $\delta\Theta(\alpha = 0, \beta = 10^5) = 0.964°$, the sought-after angle can be calculated using equation (19)

$$\delta\Theta(\alpha = 0, \beta = 3) = \frac{0.964°}{10^5} \cdot 3 = 0.104''\,. \tag{20}$$

Depending on the knowledge of the students, multiple points and also estimated numerical uncertainties could be used to extract this value by a linear regression. Here, one has to find a proper range for the interpolation. Small values for α and β usually come with relatively larger numerical errors while for larger values one cannot guarantee that the additional terms are sufficiently small. For this reason, we suggest to interpolate between zero and $\alpha_{\max}, \beta_{\max} \sim 10^5$. This will be further explained in the next section. Note that, when comparing the results of the simulation with the experimental result, one Earth year corresponds to $T_{\mathrm{E}}/T_{\mathrm{M}} \approx 365\ \mathrm{d}/88\ \mathrm{d} \approx 4.15$ Mercury years. Therefore, the perihelion motion per 100 earth years is $0.104'' \cdot 415 \approx 43.2''$. This is consistent with the observation $\delta\Theta_{\mathrm{M}} = (42.56 \pm 0.94)''$ (see equation (11)).

This agreement is the second important achievement after which finishing the course appears natural: the students have understood quantitatively the perihelion motion of Mercury. The next section that finalizes the numerical investigation is even more technical and should be worked on by the most advanced students only.

5. Tests of stability and error analysis

Measurements as well as numerical simulations in physics should always be accompanied by estimates of the corresponding uncertainties. It is important for the students to recognize this fact

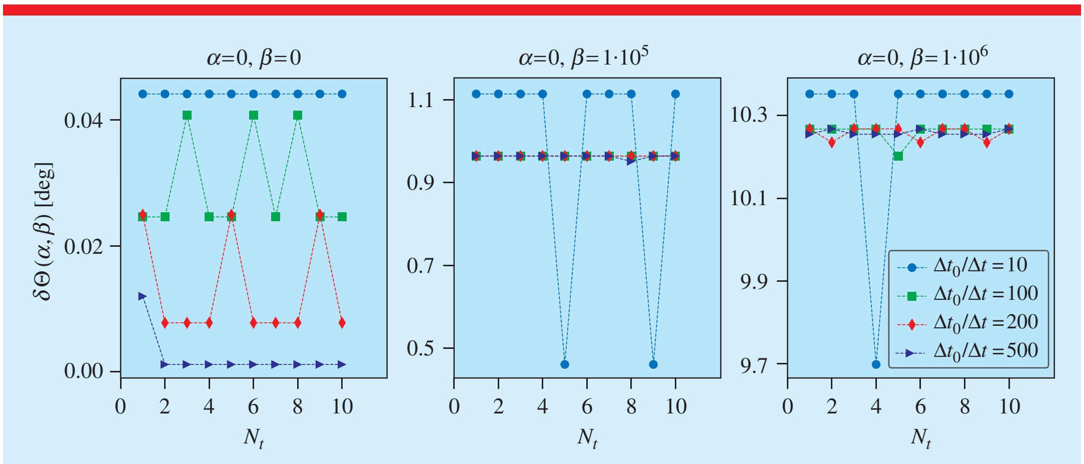

Figure 5. The motion of the perihelion $\delta\Theta$ in degrees depending on the number of turns N_t for different time steps Δt (color) and different values for β (columns). Here, $\Delta t_0 \equiv 2v_{\mathrm{M}}(0)/a_{\mathrm{M}}(0)$. Offsets of $\delta\Theta(\alpha=0,\beta=0)$ from zero can be used as an estimate of the magnitude of the error. Oscillations in $\delta\Theta$ indicate too coarse time steps. Note the change in scale for the different plots.

and to understand what the sources of uncertainties are. The students should therefore explore sources and sizes of inaccuracies arising in this simulation at least qualitatively.

There are many potential sources of uncertainties in a simulation and discussing all of them is be beyond the scope of this work. Instead, we focus on the most accessible source: numerical errors due to finite time steps Δt. Additional sources of errors include the omission of terms in equations (1) and (2) and the infinite mass approximation of the Sun (i.e. keeping the Sun's position fixed).

Consider figure 5, where the angle of the perihelion motion $\delta\Theta$ is shown for the first 10 turns (x-axis) for different choices of Δt (colors) and β (columns). In all cases $\alpha = 0$. As expected, $\delta\Theta$ is approximately constant for sufficiently small time-steps Δt and its value depends solely on β. However, contrary to the correct result $\delta\Theta$ deviates from zero for $\alpha = 0 = \beta$ for all Δt. Using the data points in figure 4 and extrapolating to $\alpha = 0 = \beta$ without enforcing $\delta\Theta(0,0) = 0$ leads to a similar offset. This offset is a numerical error and can be taken as an estimate for the error of the simulation.

Furthermore, one can observe that for large time-steps the values of the perihelion motion oscillate between discrete values at different turns. This can be explained as follows: the sample code we present to find the perihelion vectors only finds the position closest to the Sun amongst the discrete set of vectors evaluated at the discretized time steps and labels it as perihelion vector. However, this is only an approximation: sometimes the program finds a point before and sometimes after the perihelion causing the oscillations observed. The precision of this approximation improves as the time-step shrinks. Hence, strongly visible oscillations in $\delta\Theta$ for a different number of turns indicate that the time-step used is too large.

This problem can be pointed out to the students by asking them to vary the size of the time-steps and observe the effect on the trajectories. This should also be done for a wide range of values to show both good and problematic regimes. It is advantageous to set both α and β to zero for this study, since in this case the trajectories should be closed and deviations from the correct case can be spotted easily. Figure 1(b) shows the Mercury trajectory for $\Delta t = 2v_{\mathrm{M}}(0)/a_{\mathrm{M}}(0) \times 2$. One can clearly see the failure of the simulation to reproduce the physical trajectory shown as the black, solid line. From examples like this it should be apparent that the time steps influence the accuracy of the simulation. The simulation reproduces the actual trajectory only in the limit $\Delta t \to 0$. Thus, in any numerical simulation one always has to identify a proper compromise between numerical accuracy and time spent for the simulation (clearly for $\Delta t \to 0$ the computing time goes to infinity).

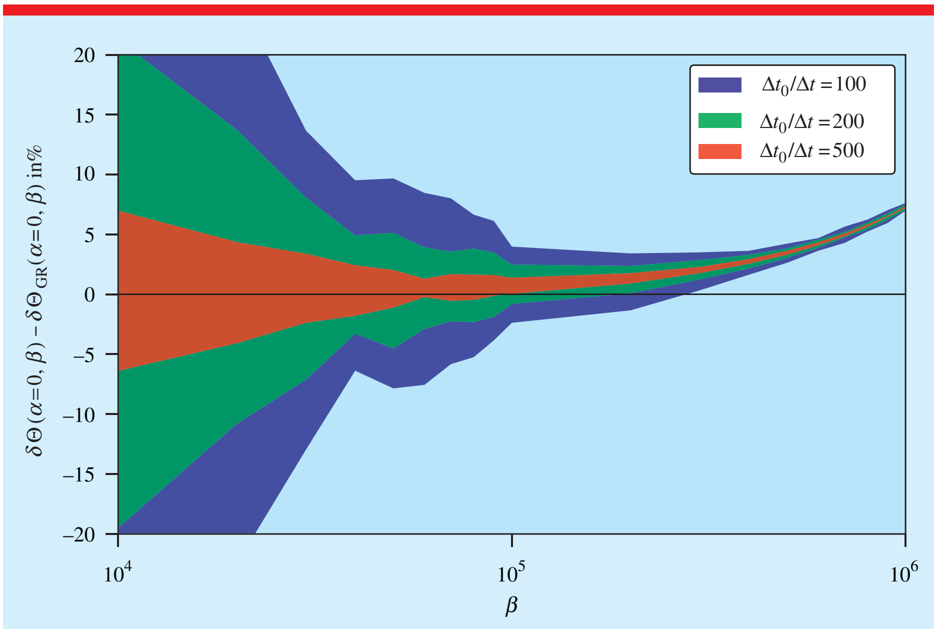

Figure 6. Difference of numerically extracted perihelion motion relative to the value computed from general relativity (equation (25)) for different time steps. The error bands are computed according to equation (21).

As a first estimate, one can approximate the numerical uncertainty of the perihelion motion, $\Delta\delta\Theta(\alpha,\beta)$ at non-zero α and β, by its offset at $\alpha = 0 = \beta$, the amplitude of its oscillation at $\alpha = 0 = \beta$ as well as the amplitude of its oscillation at the non-zero values of α and β used for the actual calculation.

$$\Delta\delta\Theta(\alpha,\beta) = \sqrt{\delta\Theta^2_{\text{mean}}(0,0) + \delta\Theta^2_{\text{std}}(0,0) + \delta\Theta^2_{\text{std}}(\alpha,\beta)}\,, \tag{21}$$

$$\delta\Theta_{\text{mean}}(\alpha,\beta) = \frac{1}{N_t}\sum_{n=1}^{N_t}\delta\Theta_n(\alpha,\beta)\,, \tag{22}$$

$$\delta\Theta^2_{\text{std}}(\alpha,\beta) = \frac{1}{N_t-1}\sum_{n=1}^{N_t}[\delta\Theta_{\text{mean}}(\alpha,\beta) - \delta\Theta_n(\alpha,\beta)]^2\,, \tag{23}$$

where the standard deviations and mean values are obtained from the data sample of N_t turns (orbits of Mercury). Thus, the numerical value extracted is given by

$$\delta\Theta(\alpha,\beta) = \delta\Theta_{\text{mean}}(\alpha,\beta) \pm \Delta\delta\Theta(\alpha,\beta)\,. \tag{24}$$

Note that this has to be repeated for each different time step Δt.

The absolute value of the numerical error mostly depends on the time step Δt and is roughly independent of the values of α and β. Therefore, it is more desirable to pick large values for α and β, because this increases the absolute size of the perihelion motion and thus decreases the relative numerical error. However, there is a competing effect which places upper bounds on the values of α and β. For instance, the prediction for the perihelion motion coming from general relativity allowing for varying values of β is given by

$$\delta\Theta_{\text{GR}}(\beta) = 2\pi\left[\frac{\beta}{4}\frac{r_s^2}{r_L^2} + \mathcal{O}\left(\beta^2\frac{r_s^4}{r_L^4}\right)\right]\,. \tag{25}$$

Thus, for large values of β, quadratic corrections in β become relevant and it is not possible to extract the perihelion growth of Mercury by performing a linear interpolation. We display these competing effects in figure 6. The relative difference between numerically extracted values and the general relativity prediction for $\delta\Theta$ are plotted against β for zero α. The value of β we recommend for the extraction is of the order of 10^5 as will be further motivated in the next section. As can be seen in the figure: for smaller values of β, the relative numerical uncertainty grows, while for larger values, $\beta \gtrsim 10^5$, the numerical values deviate from the assumed linear dependence on

β. Identifying a parameter space for reliable and precise computations is a general challenge for numerical simulations.

6. Dimensional analysis

Dimensional analysis is not only a tool that allows one to cross check, if the results of simulations are of the right order of magnitude, it is also very helpful to identify unusual dynamics in some systems. Especially the latter aspect should become clear from the discussion in this section. The modifications to the Newtonian equation of gravity introduced to account for the perihelion motion of Mercury are discussed also in [4]. To get a deeper understanding of the concepts introduced in this section reading this article is highly recommended.

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 one (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 equation (10) this statement implies that from naturalness one would predict the parameters α and β to be of order one. One expects that unnatural parameters point at dynamics not accounted for explicitly. Employing for the problem at hand the average distance Mercury–Sun $\bar{r}_{\mathrm{MS}} = 6 \times 10^7\,\mathrm{km}$, the concept on naturalness allows us to estimate the expected angular shift per orbit for, e.g. $\alpha = 1$ and $\beta = 0$

$$\begin{aligned}\delta\Theta &\simeq 2\pi \left(\frac{r_{\mathrm{S}}}{\bar{r}_{\mathrm{MS}}}\right) = (\pi \times 10^{-7})\,\mathrm{rad} \\ &= (2 \cdot 10^{-5})^{\circ} = (7 \cdot 10^{-2})^{\prime\prime}, \end{aligned} \tag{26}$$

or for $\alpha = 0$ and $\beta = 1$

$$\begin{aligned}\delta\Theta &\simeq 2\pi \left(\frac{r_{\mathrm{L}}^2}{\bar{r}_{\mathrm{MS}}^2}\right) = (\pi \times 5 \cdot 10^{-8})\,\mathrm{rad} \\ &= (1 \cdot 10^{-5})^{\circ} = (4 \cdot 10^{-2})^{\prime\prime}, \end{aligned} \tag{27}$$

which leads to a shift of about $30^{\prime\prime}$ and $15^{\prime\prime}$, respectively, in 100 earth years. These are to be compared to the empirical value of $43^{\prime\prime}$. Thus the amount of perihelion motion of Mercury is indeed 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 in addition to r_{S} and r_{L}^2. This is by itself already an interesting observation. In case of general relativity $\beta = 3$ is a natural value while α vanishes. This pattern is therefore a non-trivial prediction of general relativity and one should expect that alternative theories of gravitation generate non-vanishing values of α.

Since the dimensional analysis allows one to estimate with little effort a certain effect to be studied in numerical simulation one may also use it as a check of the numerical results: if the simulation had produced a result orders of magnitude different from the expectations of the dimensional analysis, there must be a dynamical reason in the physical system that deserves to be identified—or the code underlying the simulation has an error.

It is even possible to push the idea of naturalness further to estimate the intrinsic uncertainty of a given study. For the problem at hand we identified the expansion parameters (r_{s}/r) and $(r_{\mathrm{L}}/r)^2$ both being of order 10^{-7}. Therefore, as long as natural parameters are employed in the simulation one expects corrections to be suppressed by seven orders of magnitude since those need to scale as $(r_{\mathrm{s}}/r)^2$, $(r_{\mathrm{L}}/r)^4$, or $(r_{\mathrm{s}} r_{\mathrm{L}}^2/r^3)$. In the simulations discussed above we observed that for $\beta = 10^6$ the deviation of the result from the expectation of the underlying theory is 8% (see figure 6). Even this is in line the estimate just discussed, since for such large values of β the effective expansion parameter is $\beta(r_{\mathrm{L}}/r)^2 \sim 10$ %. This is the justification for limiting β for a reliable extraction of the perihelion motion of Mercury to values below 10^5.

Please note, however, dimensional analysis only provides an order of magnitude estimate of a given effect. While it can be used to cross check some explicit detailed evaluation it can by no means replace it, if one aims at precise results.

We do not want to leave unmentioned that in modern physics the concept of naturalness plays a very important role. It is regarded as a serious problem, when parameters deviate significantly from their natural values. Nowadays there are 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. Its interactions would even push Θ to zero. At present various intense experimental searches for this axion are going on at various labs.

7. Possible extensions

- **Explore problem autonomously**

  In section 4 we suggested to present the material by using a template as well as a step by step instruction to guide the students through the problem solution [6]. These instructions can be cut down or left out depending on available time and numerical/computational versatility of the students. This could be achieved by the following changes or additions to the concept presented in section 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`, which they could acquire for example by working through introductory materials (see [8]). Also, they could independently research the parameters relevant for the system.
  - **Why can we work in a plane?**
    In the code the third coordinate of all vectors is set to zero and never used. On the first glance, this might seem like a simplification. This choice is however possible without loss of generality because we are dealing with a central potential. The students could work out the reason behind this choice on their own.
  - **What is the impact of the different parameters?**
    Especially if the 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 the masses or the starting velocities, and observe their impact on the trajectories. This way the students get a better feeling for the physics involved.

- **Optimizing performance**
  Simulations always involve a balance between time needed for the calculations and the demanded accuracy for the results. Even though this is a rather simple example, it contains some opportunities to make these concepts accessible to the students.
  - **Measure calculation time**
    In practice there is a limit to decreasing the time steps, because the time needed for the calculation grows simultaneously. The following snippet shows how the run time of function `main` can be measured:

```
import time
start = time.time()
main()
end = time.time()
print("--- {} seconds ---"\
     .format(end - start)
```

    By varying Δt the students can validate that there is indeed approximately an anti-proportional dependence. (Note: This only works if the time in the loop is increased by Δt, so $t = t + \Delta t$.)
  - **Verlet algorithm**
    Obviously, an improvement in accuracy can be achieved by using a better algorithm without changing Δt. The simplest way to demonstrate this might be given by the implementation of the Verlet algorithm (see e.g. [9] and references therein) instead of using the simple Euler method employed here for solving the differential equation.

- **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 more visible result.
  - **Three-body problem**
    Ambitious students could even include further planets and see how the different 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.

8. Own experiences with the implementation of the course

Work on the subject began with a project work of one of the authors (JH). The project was presented at the Mädchen-Gymnasium Jülich (a German high school) in 11th grade in 2014. Based on this study, the course was developed as outlined in section 4 and used in the Schülerakademie Teilchenphysik in 2015, which aims at high school students from 10th to 13th grade. This academy is financed by the German Research Society (DFG) as the outreach branch of a research grant focussing on basic research in particle and nuclear physics. One of the authors (CH) is acting as a principal investigator for the outreach activities within this scheme. The Schülerakademie Teilchenphysik runs biennially at the Science Center Overbach in Jülich for four days and has about 25 participants coming from various parts of Germany. The academy comprises lectures, a tour to the particle accelerator COSY and the high performance computing facility at the Research Center Jülich as well as one full day of practical work. Given the setting, the participating students are highly motivated and eager to understand.

The course presented in this paper is offered as one out of three hands-on project options. A lecture to the whole group explaining the basics of numerical simulations laid down some foundations in addition to what is presented in this paper. Given the positive to enthusiastic feedback we have received from the participants on the course after its first installation in 2015, we decided to make the course a permanent part of the academy. In 2017 we received similarly positive feedback. About one half of the participants of each year chose to work on the numerical project—out of those about one half had previous programming experience. Most of the participants had already learned about derivatives and vector calculus in school.

The numerical course was allocated for one work day (seven hours) and the students worked in pairs. Each group worked on a laptop provided by us, where the necessary software was pre-installed. Four of the authors (IH, CK, CM, JLW) acted as supervisors for the students although not all at the same time. Each instance of the course was supervised by one to three people. The course was initiated with a brief summary of previously presented methods, motivations and objectives of the simulation. In the first two hours the students were introduced to `Python` and `VPython`. In the following part the students worked with an initial code template (see [6]) and were motivated to come up with own solutions to fulfill the objectives. A large part of the material covered during this day is presented in section 4. With the exception of some younger students, the participants where able to simulate and visualize the motion of Mercury including forces from GR. Only a few participants started to implement the quantitative extraction of the perihelion motion, uncertainty estimates were not a subject as the time frame was too short. Once the basic problem was tackled some students deviated from the suggested path and for instance studied the problem with one additional planet or different starting parameters. This is a very interesting problem by itself for there are parameter ranges where the classical three-body system develops chaotic features. We encouraged such explorations as well. We include the additional material in the paper because we want to provide teachers with sufficient material to fill a working group outside the general curriculum running for a longer period than one day.

9. Summary

In this work a course is presented that should enable advanced high school students to understand quantitatively the perihelion motion of Mercury by using a numerical simulation. At the same time the active participation in the course teaches the central role of differential equations in theoretical physics, the basic concepts of how to use numerical simulations to find their solutions as well as the need to estimate the uncertainties of a given study. In addition the concepts of dimensional analysis and naturalness were introduced which not only allow for an estimate of a given effect *a priori* (to cross check the numerical results) but also to estimate the intrinsic uncertainty of the result.

The course is structured such that students at different levels can stop after different achievements. The basic course contains the set-up of the numerical simulation and its visualization: the students will have observed the impact of

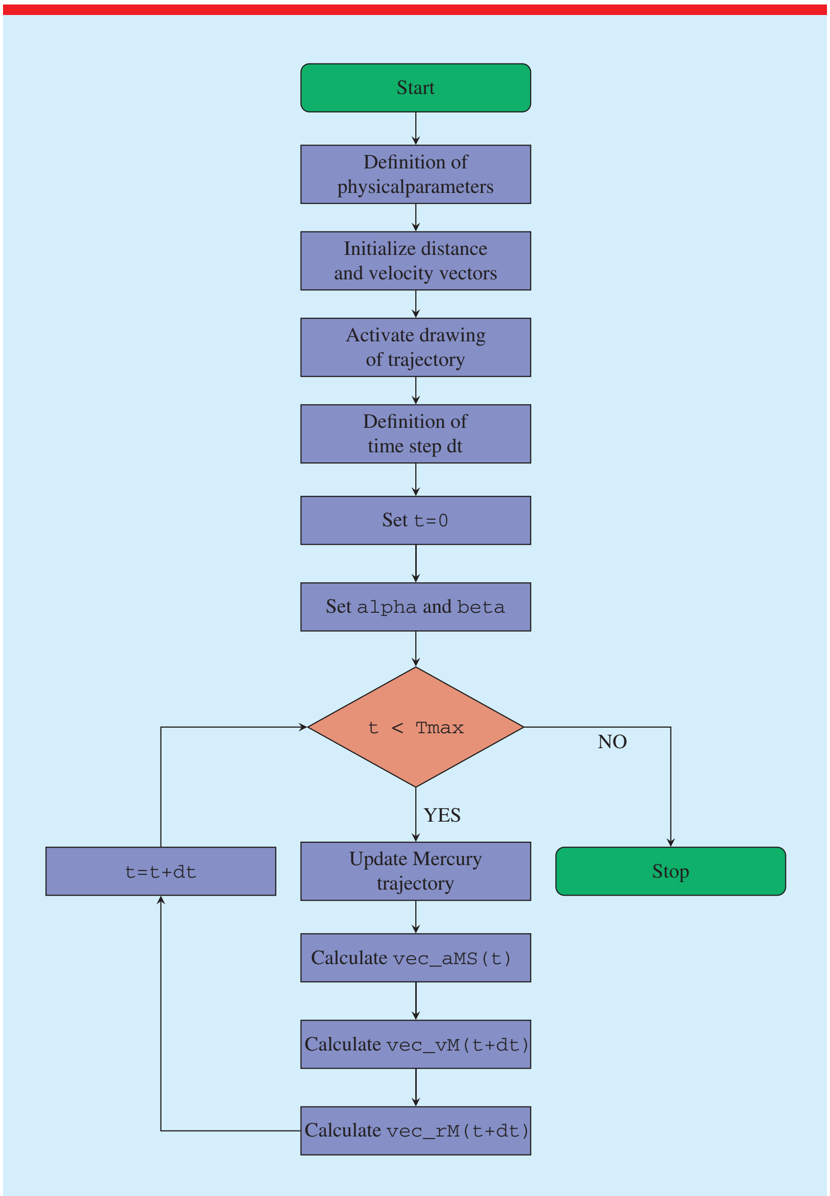

Figure A1. Flowchart demonstrating the logical ordering of the example code.

different forces on the trajectories and some basic features of numerical studies. The more advanced students may proceed to extract the perihelion motion quantitatively from the parameters provided by general relativity. And finally the most advanced students may even follow the discussion of the uncertainties of the simulation.

We are convinced that the course presented in this paper is very well suited to teach high school students not only the power of numerical simulations but also the beauty of theoretical physics.

Acknowledgments

We would like to thank the staff of the Science College Overbach and in particular its director Rusbeh Nawab for providing us with the ideal environment for the Schülerakademie

```python
# Import all objects from the vpython module
from vpython import *

# Definition of physical parameters
rM0 = 4.60    # Initial radius of Mercury orbit,in units of R0
vM0 = 5.10e-1# Initial orbital speed of Mercury, in units of R0/T0
c_a = 9.90e-1# Base acceleration of Mercury, in units of R0**3/T0**2
TM  = 8.80e+1# Orbit period of Mercury
rS  = 2.95e-7# Schwarzschild radius of Sun, in units of R0
rL2 = 8.19e-7# Specific angular momentum, in units of R0**2

# Define the initial coordinates; M = mercury, S = Sun
M = sphere(pos=vector(0, rM0, 0), radius=0.5,  color=color.red)
S = sphere(pos=vector(0, 0, 0),   radius=1.5,  color=color.yellow)
# And the initial velocities
M.velocity = vector(vM0, 0, 0)
S.velocity = vector(0, 0, 0)

# Add a visible trajectory to mercury
M.trajectory = curve(color=color.white)

# Definition of the time step
dt = 2 * vM0 / c_a / 20

# Define the coordinate and velocity update
def evolve_mercury(vec_rM_old, vec_vM_old, alpha, beta):
    # Compute the strength of the acceleration
    temp = 1 + alpha * rS / vec_rM_old.mag + beta * rL2 / vec_rM_old.mag**2
    aMS  = c_a * temp / 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
    return vec_rM_new, vec_vM_new

t     = 0.0
alpha = 0.0
beta  = 0.0
# Execute the loop as long as t < 2*TM
while t < 2*TM:
    # Set the frame rate (you can choose a higher rate to accelerate the program)
    rate(100)
    # Update the drawn trajectory with the current position
    M.trajectory.append(pos=M.pos)
    # Update velocity and position
    M.pos, M.velocity = evolve_mercury(M.pos, M.velocity, alpha, beta)
    # Advance time by one step
    t = t + dt
```

Teilchenphysik. This work is supported in part by NSFC and DFG through funds provided to the Sino-German CRC110 'Symmetries and the Emergence of Structure in QCD'.

Appendix. The code covered by the basic course (visualization of the trajectories)

Below we provide the complete code as well as a flowchart for part 1 as developed above.

Further examples and template files can be found online [6].

ORCID iDs

C Körber https://orcid.org/0000-0002-9271-8022
J-L Wynen https://orcid.org/0000-0002-3761-3201
C Hanhart https://orcid.org/0000-0002-3509-2473

Received 16 March 2018, in final form 24 April 2018
Accepted for publication 14 May 2018
https://doi.org/10.1088/1361-6552/aac487

References

[1] Clemence G M 1947 The relativity effect in planetary motions *Rev. Mod. Phys.* **19** 361–4
[2] Einstein A 1915 Erklärung der Perihelbewegung der Merkur aus der allgemeinen Relativitätstheorie *Sitzungsber. preuss. Akad. Wiss.* **47** 831–9
[3] Williams D R 2017 Mercury fact sheet https://nssdc.gsfc.nasa.gov/planetary/factsheet/mercuryfact.html (Accessed: 24 May 2018)
[4] Wells J D 2012 When effective theories predict: the inevitability of Mercury's anomalous perihelion precession. Masses
[5] Python Software Foundation 2017 Python https://www.python.org/ (Accessed: 24 May 2018)
[6] Körber C and Wynen J-L 2018 ckoerber/perihelion-mercury: Physics Education Release (Version v1.0) *Zenodo* http://doi.org/10.5281/zenodo.1252198
[7] Hunter J D 2007 Matplotlib: a 2D graphics environment *Comput. Sci. Eng.* **9** 90–5
[8] Scherer D *et al* 2017 Vpython—3d programming for ordinary mortals http://vpython.org (Accessed: 24 May 2018)
[9] Hairer E, Lubich C and Wanner G 2003 Geometric numerical integration illustrated by the störmer/verlet method *Acta Numer.* **12** 399–450

Christopher Körber is a member of the Institute of Advanced Simulation (IAS) at Forschungszentrum Jülich (FZJ), Germany. He has finished his PhD in nuclear physics at the University of Bonn, Germany, in 2018. His scientific interests range from numerical simulations over data analysis and visualization to the presentation of scientific content.

Inka Hammer is a PhD student at IAS where she is working in the field of hardonic physics, in particular on the interplay of hadronic molecules and heavy quarkonia. She received her master degree in physics from the University of Bonn in 2016.

Jan-Lukas Wynen is also a PhD student at IAS and the University of Bonn. He simulates and analyzes strongly correlated quantum systems. His scientific interests are focused around numerical simulations including development of algorithms and scientific software.

Joseline Heuer graduated from high school in 2015 and since then she studies Biomedical Engineering at the University of Applied Sciences Hamm-Lippstadt in Hamm, Germany. She participated in the first 'Schülerakademie Teilchenphysik' and her school project on numerical simulations for the mercury trajectory was the seed for the work presented here.

Christian Müller worked as a laboratory technician at the Central Institute for Engineering, Electronics and Analytics at FZJ. A major part of his work was the analysis of various samples using mass spectrometry. Since 2004 he was involved in the formation of and now works at FZJ's student laboratory JuLab.

Prof. Dr. C. Hanhart studied Physics at the University of Bonn, and the University of New South Wales, Sydney, Australia. He did his PhD on a subject in theoretical particle physics also in Bonn. After two post-docs, since 2001, he is a permanent staff member at IAS at FZJ doing research in particle physics. He also teaches at Bonn University.