Computational Physics Education with Python

A. Bäcker

Institut für Theoretische Physik, TU Dresden, D–01062 Dresden, Germany

In recent years, the usual courses on theoretical and experimental physics have been supplemented by courses on computational physics at various universities. In this text a short overview on our experience in establishing such a course within the context of theoretical physics using Python as a programming language is given. The main aim of the course is to enable the students to solve problems in physics with the help of numerical computations. In particular by making use of graphical and interactive exploration a more profound understanding of the underlying physical principles and the problem at hand should be gained.

After setting up a list of possible topics for the course, one of the first choices to be made concerned the programming language. The decision for Python was driven by the following main points

- freely available (Linux, Mac and Windows) so that the students could be provided with a CD to take home and install.
- full programming language, easy to learn (even for beginners)
- very readable and compact code allows for a short development time for the programms and therefore to concentrate on the physics problem
- interactive plotting capabilities (e.g. `matplotlib`)
- Object orientated programming (OOP) is possible, but not required so that the emphasis can be on the solution of physics problems using the computer and not on learning the deeper aspects of Python. However, for students with advanced programming knowledge this makes python also attractive.

The computational physics course was started in 2002 and since then run every summer term with student numbers increasing from about 20 to 70 covering now more than 50% of each years physics students. The lectures (2 hours) covering both physical and numerical aspects are accompanied by tutorials (2 hours) and weekly exercise sheets. The students hand in their solutions by e-mail and the programms are then printed, tested, corrected, marked and returned in the next tutorial to provide individual feedback. In addition, extensively commented sample solutions are posted on the web-page of the course. The topics range from elementary numerical methods (differentiation, integration, zero finding), differential equations, random numbers, stochastic processes, Fourier transformation, nonlinear dynamics and quantum mechanics.

The students knowledge of programming languages turned out to be rather diverse, ranging from no experience at all to detailed expertise in C++. With a few exceptions, no previous knowledge of Python was present. In order to provide the necessary basics a detailed introduction is provided, which makes extensive use of the interactive capabilites of Python (using IPython). This covers the use of variables, simple arithmetic, loops, conditional execution, small programms and subroutines. Of particular importance for numerical computations is the use of arrays as provided by `numpy` which allows to write efficient code without explicit

loops. Again this makes the code compact and also enhances the readability and speed of programming. For further numerical routines, e.g. solving ordinary differential equations or computation of special functions, `scipy` is used.

Finally a short introduction to plotting using matplotlib is given, e.g. after starting ipython with support for interactive plotting via `ipython -pylab` one can simply do

```
x = linspace(0.0, 2.0*pi, 100)  # Array of x values
plot(x,sin(x))                  # graph of sin(x) vs. x
plot(x,cos(2*x))                # add another graph
```

and then for example zoom into the resulting plot using the mouse.

A first non-trivial example is the dynamics of a driven pendulum which can be described by the coupled differential equation

$$\ddot{\varphi} = \sin\varphi + 1/4\cos t \qquad \Longleftrightarrow \qquad \begin{aligned} \dot{\varphi} &= v \\ \dot{v} &= \sin\varphi + 1/4\cos t \end{aligned} \tag{1}$$

for which this simple programm computes the time evolution

```
from pylab import *                       # plotting routines
from scipy.integrate import odeint        # routine for ODE integration

def derivative(y, t):
    """Right hand side of the differential equation.

       Here y = [phi, v].
    """
    return array([y[1], sin(y[0]) + 0.25* cos(t)])   # (\dot{\phi}, \dot{v})


def compute_trajectory(y0):
    """Integrate the ODE for the initial point y0 = [phi_0, v_0]"""
    t = arange(0.0, 100.0, 0.1)           # array of times
    y_t = odeint(derivative, y0, t)       # integration of the equation
    return y_t[:, 0], y_t[:, 1]           # return arrays for phi and v


# compute and plot for two different initial conditions:
phi_a, v_a = compute_trajectory([1.0, 0.9])
phi_b, v_b = compute_trajectory([0.9, 0.9])
plot(phi_a, v_a)
plot(phi_b, v_b, "r--")
xlabel(r"$\varphi$")
ylabel(r"$v$")
show()
```

The resulting plot is shown as a screenshot in fig. 1 and shows that in the considered system a moderate change in the initial condition leads to quite different behaviour already after short times.

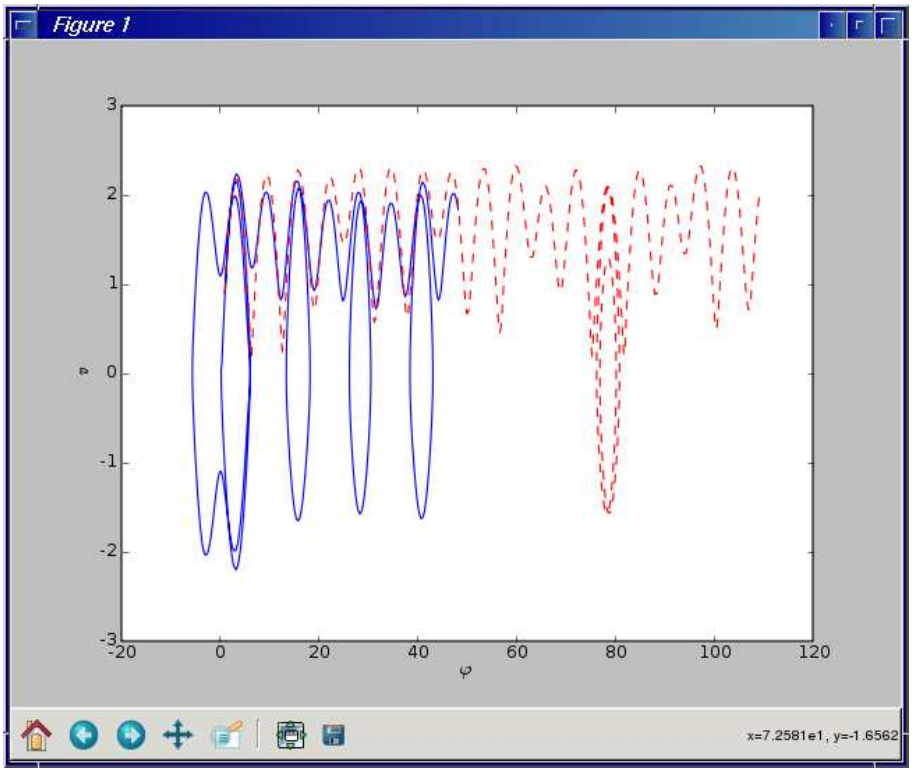

Figure 1: Dynamics of the driven pendulum described by (1) for two different initial conditions, visualized using `matplotlib`.

A more advanced example is the visualization of the quantum probability densities for wave functions of the hydrogen atom, which in spherical coordinates reads

$$\Psi_{n,l,m}(r,\vartheta,\varphi) = \mathrm{Y}_{lm}(\vartheta,\varphi)\cdot\sqrt{\frac{(n-l-1)!(2/n)^3}{2n[(n+l)]!}}(2r/n)^l e^{-r/n}\; L^{2l+1}_{n-l-1}(2r/n) \quad . \tag{2}$$

Here n, l, m are the quantum numbers characterizing the wave function, Y is the spherical harmonics, numerically determined using `scipy.special.sph_harm` and L is the associated Laguerre polynomial, determined by `scipy.special.assoc_laguerre`.

As $\psi(x, y, z)$ is a scalar function defined on $\mathbb{R}^3$ one can either use a density plot or plot equi-energy surfaces. Instead of writing the corresponding (non-trival) visualization routines from scratch, we use the extremely powerful visualization toolkit (VTK, `www.vtk.org`) whose routines are also accessible from Python. So the first step is to generate a data file suitable for VTK choosing a rasterization in Cartesian coordinates (x, y, z), on $[-40, 40]^3$ with 100 points in each direction,

```
# vtk DataFile Version 2.0
h_data.vtk Data for hydrogen wave functions
ASCII

DATASET STRUCTURED_POINTS
DIMENSIONS   100 100 100
ORIGIN    -40.0 -40.0 -40.0
```

```
SPACING    0.8 0.8 0.8
POINT_DATA   1000000
SCALARS scalars float
LOOKUP_TABLE default
... 1000000 real numbers corresponding to |\psi_{5,2,1}(x,y,z)|^2 ...
```

Then this data set can be visualized using `MayaVi` by starting

```
mayavi -d h_data.vtk -m IsoSurface -m Axes
```

This reads the datafile `h_data.vtk`, loads the IsoSurface module and adds axes to the plot. By modifying the value for the isosurface one obtains a plot as shown in fig. 2a) for $n, l, m = 5, 2, 1$. Adding a scalar-cut-plane, a different look-up table and varying the view leads to fig. 2b). This short example shows that with only a moderate effort, highly instructive and also appealing visualizations of data are possible.

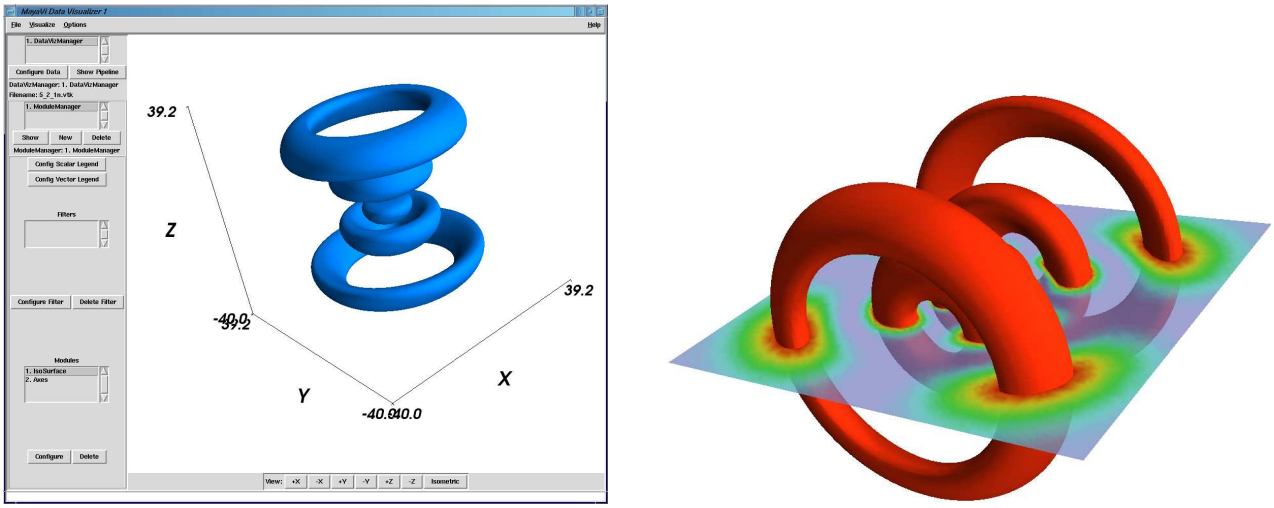

Figure 2: (a) Screenshot of a MayaVi visualization showing a surface of constant value of $|\psi(x, y, z)|^2$ for the hydrogen atom. (b) Same type of plot, together with a scalar cut-plane and a different lookup-table with a transparency gradient.

At the end of the course the students give a short presentation on a small project of their own choice. In particular here the creativity of the students was not limited by python, which enabled them to create highly illustrative dynamical visualizations, for example also in 3D using `VPython` (`www.vpython.org`).

To summarize one can say that the initial experiment of using Python for teaching computational physics has proven to be highly successful. It even turned out that several students continued to use python for various tasks, like data analysis in experimental physics courses or during their diploma thesis, not just in our group.

Acknowledgements

I would like to thank Prof. R. Ketzmerick with whom the concept of the computational physics course described here was developed jointly. Moreover, I would like to thank all the many people involved in setting up and running the course. Finally, all authors and contributors to the mentioned software deserve a big thanks!