Georg-August-Universität Göttingen
Institut f”ur Theoretische Physik
Prof. Dr. F. Heidrich-Meisner
Dr. E. Bothmann
S. L. Villani
SoSe 2021

Übungen zur Vorlesung
Computergestütztes Wissenschaftliches Rechnen
Blatt 6

Lernziele dieses Übungsblattes

- Verlet-Algorithmus zur Lösung von Newtonschen Bewegungsgleichungen
- Shooting method zur Lösung von Randwertproblemen

Aufgabenmodus

Die Aufgaben dieser Woche verfügen über keine gesonderten Tutorials. Sie sollten alle Aufgaben mit Ihren bisherigen Kenntnissen bearbeiten können. Dafür fallen die Aufgabenstellungen stellenweise etwas feinschrittiger aus.

Diese Woche sind die Aufgaben auf Englisch.

Übungsaufgaben

Aufgabe 16 *Verlet integrator*

On the previous exercise sheet you have seen that using RK2/4 it is possible to improve our solution accuracy compared to the simple Euler method. Now we are going to study another integration algorithm, *velocity verlet*, which has a global discretization error of $\mathcal{O}(\Delta t^2)$ and has the advantage of preserving conserved physical quantities such as the energy, with the numerical solution oscillating around the constant exact energy of the system. The method steps are:

i) $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_i \cdot \Delta t + \mathbf{a}_i \frac{\Delta t^2}{2}$,

ii) calculate $\mathbf{a}_{i+1}$ using $\mathbf{x}_{i+1}$,

iii) $\mathbf{v}_{i+1} = \mathbf{v}_i + \frac{\Delta t}{2}(\mathbf{a}_{i+1} + \mathbf{a}_i)$.

Before studying the stability of the method in problem 17, let’s test it out for a simple problem like the harmonic oscillators of problem 11.

1. Add the following function to your numerics library:

```
void verlet_step(double t, double delta_t, double y[],
                 ode_func func, int dimension,
                 void *params)
{
    ...
```

}

`t` is the current time, `delta_t` is the size of the time step to be performed. `y[]` is the current state vector of the system and has the size `dimension`. The function `func` is the function that calculates the derivatives of the system quantities. Hence,

```
func(t, y1, y2, params);
```

evaluates the function $\mathbf{F}(t, \mathbf{y}_1)$ and stores the result in the array `y2[]`.[1]

2. In `verlet_step`, declare two `double` arrays `a1[]` und `a2[]` with their lengths given by `dimension`. These will be used to store the acceleration for the times $i\Delta t$ and $(i+1)\Delta t$. Use the heap, such that `verlet_step` can also work for very large system dimensions.
3. Fill `a1[]` using `func`. Remember to multiply by the time step Δt. Use it to calculate x_{i+1}, i.e. the first half of `y[]`.
4. Now calculate `a2[]` and use it to calculate v_{i+1}, i.e. the second half of `y[]`.
5. Free the memory.
6. In your solution for problem 11, replace `euler_step` with `verlet_step`. Rerun the simulation and plot the positions of the harmonic oscillators and the total energy of the system to check that the algorithm is working.

Aufgabe 17 *Convergence behaviour of Verlet*

As we did in problem 14 for the RK2 algorithm, let's study the consistency and convergence of our Verlet algorithm by applying it to our reference system, a harmonic oscillator (here: a mass on a spring).

1. Copy your harmonic oscillator simulation from problem 14. Replace the RK2 step in the simulation loop with your new Verlet step.
2. As in problem 14, simulate the oscillator 100 times with a logarithmically distributed Δt in the range $10^{-4} \ldots 10^{0}$. Write the time step and the residuals between the analytical and the numerical end positions for $t = T/4$ to a file.
3. Plot the residuals in a double logarithmic plot against Δt. Determine the convergence order of the algorithm from the slope. If you don't get the expected result, your program has a bug. Fix it and try again.
4. As mentioned before, a major difference between this method and others like RK and Euler is energy conservation. Compare the energy of the system over time obtained from RK2 and RK4 (and possible Euler, if you did the optional subtask 7 of problem 14).

[1] Note that on the previous sheet we used $\mathbf{a}$ and $\mathbf{b}$ to denote the state vectors before and after the time step. Now we use $\mathbf{y}_{1,2}$ instead, to avoid any confusion with the acceleration $\mathbf{a}$ that appears explicitly in the Verlet step.

Aufgabe 18 *Shooting method (Coulomb potential)*

In this problem we will use the shooting method to solve the equation of motion of a positron with charge e in an electric field generated by a point-like nucleus of positive charge e. With the nucleus being much heavier than the positron, we can set the origin of our reference frame equal to the centre of mass of the nucleus. The positron's initial position is $r(t_0)$ and its final position is $r(t_f)$. To simplify the problem even further and deal with only one space dimension, we demand that the positron is moving directly towards (or away from) the origin.

The Coulomb force acting on the positron is given by

$$F(r) = \frac{e^2}{4\pi\epsilon_0}\frac{1}{r^2}.$$

1. Define the following constants: the vacuum permittivity $\epsilon_0 = 8.854\,19 \times 10^{-12}\,\mathrm{F/m}$, the positron charge $e = 1.602 \times 10^{-19}\,\mathrm{C}$ and its mass $m = 9.109 \times 10^{-31}\,\mathrm{kg}$.

2. For the boundaries, use $r(t_0) = 10\,\mathrm{m}$ and $r(t_f) = 10\,\mathrm{m}$, where $t_0 = 0\,\mathrm{s}$ and $t_f = 2\,\mathrm{s}$. We define the state vector $\mathbf{y} = (y_1, y_2) = (r, v)$, so we need to solve

$$\dot{\mathbf{y}} = \begin{bmatrix} v \\ \frac{F(r)}{m} \end{bmatrix}.$$

3. The core of the shooting method is finding the roots of

$$G(v_0) = y_1(t_f) - r(t_f),$$

where $y_1(t_f)$ is the first component of the solution to the initial value problem at time t_f with initial velocity v_0. Implement the function $G(v_0)$ with the signature `double G(double v0)`. Inside of it, solve the initial state problem with initial conditions $y_1 = r(t_0)$ and $y_2 = v_0$, using your implementation of `verlet_step` to get $y_1(t_f)$, and return the difference with the required end position $r(t_f) = 10\,\mathrm{m}$.

4. To find the root of $G(v_0)$, you can use the *Newton-Raphson* method you implemented in the function `find_root` for exercise 7, sheet 3. Start with an initial guess for the velocity of $-5\,\mathrm{m/s}$ and solve the initial state problem. Use a suitable time step (as studied in problem 17).

5. To check whether the Newton-Raphson algorithm has found the right solution, plot $G(v)$ with $v \in [-12, -2]$ and compare its zeros with the found optimal velocity.

6. To appreciate how the trajectories change varying v, plot three trajectories. One with the initial guess, one with the optimal velocity and one with some intermediate velocity, e.g. the mean of the other two.

Selbsttest

- What are the advantages and disadvantages of using verlet over RK2? Articulate your answer.
- What are the advantages and disadvantages of using verlet over RK4? Articulate your answer.
- Are initial and final position boundaries enough to have a unique solution to an ODE?