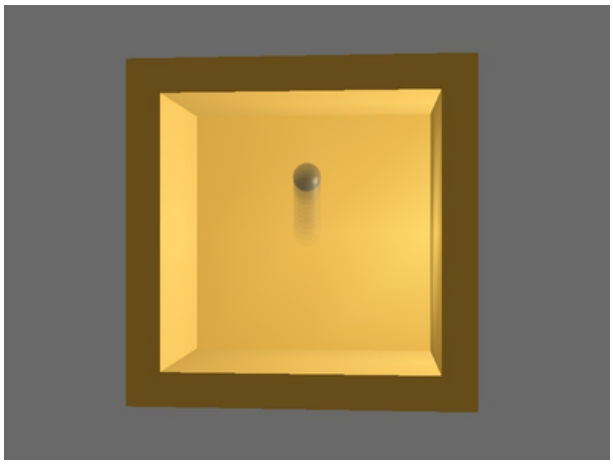
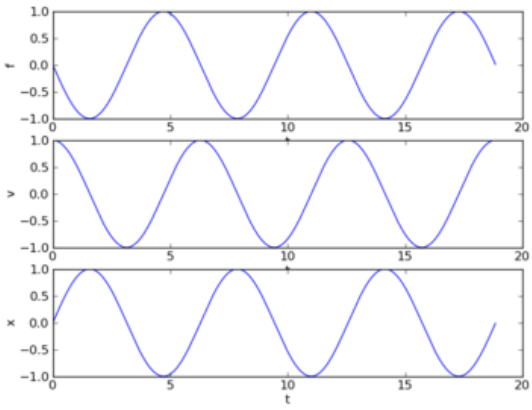
ODE: ORDINARY DIFFERENTIAL EQUATIONS

JOHAN JANSSON MATTIAS SANDBERG





1. To read

- Newton's laws of motion (ch. 17)
- Particle-Spring System (ch. 18)
- Generalized Fundamental Theorem (ch. 77)
- Time Stepping Error Analysis (ch. 80)

2. Goals

2.1. **Understanding.**

- Time-stepping for ODE (initial value problems)
- The generalized fundamental theorem.
- Trapezoidal method using simple fixed-point iteration.
- Energy conservation.
- Residual versus error, growth of local errors.
- Understand how to formulate a vector-valued/array ODE system.

2.2. **Skills.**

- To explain the steps in the proof of the generalized fundamental theorem.
- Modify the Python function `timestep()` from module 1 to handle ODE.
- To experimentally show order of convergence

Simulation Technology Module.

- Show conservation of energy by numerical experiments
- To experimentally show how the error $e(T)$ grows with regard to end time T

3. Software Specifications

Follow the interface specifications in the template Python file in the "python" subdirectory. The file **ode.py** includes the functions (some of them to be finished) needed for this module, continue adding your own code to the functions. When you're finished or want to check how well you're doing, submit your "ode.py" (must be named exactly that) to the Web-CAT system at http://webcat.csc.kth.se.

You will also find a file **main.py** where you can implement the different test cases specified in the questions below.

4. Examination

The examination of this module consists of the questions below. Each question gives 1 point (a question with two sub-questions gives $0.5 + 0.5 = 1$ point). To pass the module 3.5/5 points are necessary.

5. Questions

Consider the harmonic oscillator equation

$$\begin{aligned} \dot{x}(t) &= v(t), \\ \dot{v}(t) &= -x(t). \end{aligned} \tag{1}$$

5.1. **Game/Interactive simulation.** To take one time step with an implicit method (the trapezoid method for example), one needs to solve a non-linear algebraic equation. A simple way to do this is by repeating the time-step formula several times, 5 times in this case (fixed-point iteration, we will later see in module 5 why it has this name), here we demonstrate for implicit Euler (note that u is on both the left hand and right hand side):

```
for i in range(0, 5):
  u = u0 + k * f(t, u)
```

We will show in module 5 that performing this iteration a number of times makes `u` converge to the solution of the algebraic equation, and derive what conditions need to be satisfied.

Enter the "game" directory and run: `python particle_simulator.py`. Play around with the time-stepping or the right hand side $f(t, u)$. What does the plot represent? What does the equation model? Can you modify the frequency? Here u and $f(t, u)$ are vector-valued (array) functions:

$$u = \begin{pmatrix} x \\ v \end{pmatrix} \quad f(t, u) = \begin{pmatrix} f_1(t, u) \\ f_2(t, u) \end{pmatrix} \quad \dot{u} = f(t, u)$$

The y-component in `f_simple()` in the game from M1 can be written:

$$\dot{x} = v$$
$$\dot{v} = -cos(t)$$

How would you write `f_simple()` in terms of $f(t, u)$?

5.2. **Time-stepping.**

- Modify the function `timestep()` (that you implemented and used in module 1) to be able to handle the case where f also depends on u, that is, $f = f(t, u)$. In the forward Euler-case the changes are trivial. However for the Trapezoidal method, you need to need to perform a number of fixed-point iterations (as described in the game) every time step.
- Implement a function `solve()` that solves

$$\dot{u} = f(u), \quad u(t_0) = u_0$$

  on a given interval $I = [t_0, T]$ using the methods implemented in `timestep()`. The function `solve()` should be able to handle both a scalar (only one equation) ODE and system of ODEs.
- Test your function `solve()` by solving $\dot{u} = -u$ on $[0, 2]$ with $u(0) = 1$ and timestep $k = 0.1$. This should generate a decaying exponential function, see main.py.
- Verify that your function `solve()` also works for systems of ODEs. Test by computing a solution for the harmonic oscillator equation (1).

5.3. **Conservation of energy.**

The total energy (potential + kinetic) for the harmonic oscillator equation, eq. (1), is given by

$$E(t) = \frac{1}{2}(x(t)^2 + v(t)^2).$$

- Solve the harmonic oscillator equation, eq. (1), on a time interval and study the total energy. Verify that your implementation of the Trapezoidal method conserves the energy. Also try with the forward Euler method, what is the difference?

5.4. **Error estimates I.**

- Show symbolically (using pen and paper) step 1 and 2 of the proof of the generalized fundamental theorem as given in section 77.2.
- The forward Euler method is convergent of order one and the Trapezoidal method is convergent of order two. Verify this by solving the harmonic oscillator, eq. (1), on a given time interval, using different time steps.

5.5. **Error estimates II (Hard).** Here we will examine a general "a posteriori" error estimate (based on the computed solution, i.e. "after" the computation) for the Trapezoidal method, see chapter 80 for the estimate and its derivation. The estimate is based on a "residual" which is defined simply by inserting the computed (piecewise linear) solution into the ODE: $R(u_h) = f(u_h) - \dot{u_h}$, and a "stability factor" which describes the growth of the residual in time.

- Implement `maximum_residual()` and compute the maximum residual and error at the end time $e(T)$ (by using an exact solution) of the Trapezoidal method when used to solve the decaying exponential $\dot{u} = -u$ and the harmonic oscillator, eq. (1). Compute the maximum residual and the error for three different end times $T = 2, 4, 6$ and using the time steps $k = 0.08, 0.04, 0.02, 0.01$.
- Plot the errors and residuals with regards to end time and timestep. Explain the growth/decrease of the error and residual by inspecting the error analysis in chapter 80.