Time integration and differential equations: Worksheet #1

Dr. rer. nat. Martin Schreiber 2020-04-27

This first worksheet will make you familiar with the basic steps for implementing time integrators for linear ODEs. This will be the fundament used over and over again throughout the next worksheets. Make sure that you'll stick to the suggested interfaces which will make further development as well as debugging much easier.

1 Linear ODEs

We use identifiers given in capitalized letters in the following section headings.

1.1 ODE_LIN_DEC: Declining or increasing behavior

The ODE is given by

$$\frac{d}{dt}u(t) = \lambda u(t)$$

with $u \in \mathbb{R}$ and $\lambda \in \mathbb{R}^+$ for an exponential growth behavior or $\lambda \in -\mathbb{R}^+$ for a declining behavior.

The initial condition is given by

$$u(0) = 1/2.$$

1.2 ODE_LIN_OSC: Linear oscillator

The ODE is given by

$$\begin{aligned}\frac{d}{dt}u_1(t) &= -\lambda_1 u_2(t) \\ \frac{d}{dt}u_2(t) &= \lambda_2 u_1(t)\end{aligned}$$

with $u \in \mathbb{R}^2$ and $\lambda \in \mathbb{R}^+$.

The initial conditions are given by

$$\begin{aligned}u_1(0) &= 1/2 \\ u_2(0) &= -1/3.\end{aligned}$$

1.3 ODE_LIN_GEN: General complex-valued linear ODE

The previously discussed ODEs can be compactly written using complex valued and $u, \lambda \in \mathbb{C}$, leading to

$$\frac{d}{dt}u(t) = \lambda u(t).$$

The initial conditions should be given by

$$\begin{aligned}Re(u(0)) &= 1/2 \\ Im(u(0)) &= -1/3.\end{aligned}$$

2 Realization

The first assignment is on the development of a simple expclicit time integrator. The interfaces which you'll design here will be reused also in the next worksheets, so please pay attention to them. It's not required to exactly follow these interfaces, but it's rather a suggestion how they can look like to be future-proof. Please also care about a certain amount of commenting.

If you're using Python, typing of variables, e.g.

```
def fun(i: float):
   ...
```

is highly recommend.

2.1 Time integrator

2.1.1 ODE class

[ASSIGNMENT] Develop a class for each ODE including a `setup(...)` routine which takes the ODE parameter(s) λ as input.

2.1.2 ODE tendencies

[ASSIGNMENT] For each ODE class, add a function `du_dt(...)` which takes the following input:

1. Current state variables $u_i = u(t_i)$ as a real or complex valued vector (e.g. numpy array). In case of a scalar, you just have a single-valued array field.
2. The time t_i as an input.

Return the time tendencies (from the particular ODE formulation).

These functions must have the same signature (number of parameters and types of parameters) to be used in a transparent way by the following methods.

2.1.3 Time step integrator

[ASSIGNMENT] The integrator should be a general interface to time integrate based on a given method and a given ODE.

Implement a function `int_u_dt(...)` which takes the following data as input:

1. String which requests a particular time integration method (e.g. "ERK1" for first order forward Euler)
2. ODE function itself which should be time integrates
3. Current state variable u_i
4. Timestep size Δt
5. (Current timestamp t_i)

Implement explicit forward Euler.

Return the solution at the next time step.

2.1.4 Full time integration for simulation

[ASSIGNMENT] Provide a function which takes the following parameters:

- the ODE itself
- the time integration method string
- time step size
- number of time steps

[ASSIGNMENT] Store all approximated solutions for all time steps and return this.

2.2 Plotting functionality

[ASSIGNMENT] Provide a plotting function which plots the approximated solution(s) over the simulated timeframe.

- Write out the solution to a PDF or PNG file so that you can easily use it later on.
- Use proper names for axes, title, etc.
- Most important: Make it nice looking!
  Check out the following repository for some helper routines: https://github.com/schreiberx/plot_config

2.3 Studying the ODE behavior

[ASSIGNMENT] First, determine parameters for ODE_LIN_GEN which show the same behavior as the other linear ODEs (e.g. if you plot the numerical solution, both should match). In particular, make yourself familiar to the complex-valued ODE formulation!

Then, test different lambda parameters of ODE_LIN_GEN, e.g.

- larger/smaller values,
- negative/positive,
- imaginary/real.

Plot the result for different values using a sufficiently small time step size so that results wouldn't change visually for smaller time step sizes.