Exercise C3a: Unit tests

Software

In this task you will learn to write and run unit tests for you code. To make the experience smoother we will use a testing framwork called check. To install the library on the stuDAT computers run:

```
bash ./install_check.sh
```

This works on other linux computers as well, however, some package managers have it in their repositories as well (which is the preferred way of installing it).

Introduction

In this exercise you are supposed to get familiar with testing your code. The idea is to write your code with named segments (functions). The function names should reflect the task that the function has. Writing your code using named segments has two very valuable effects. First of, your code will be more readable, helping us, the teaching assistants, to help you with debugging your code. Secondly, you can easily verify that your small named segments is doing the correct thing, making it more likely that you get the correct result in the end.

Tasks

Task 1

We have written a Newton II solver that, given initial conditions for the position and velocity, computes the trajectory of a projectile that is subject to gravity and air resistance (drag). In every time step is updates the positions and velocities according to the acceleration from gravity and drag

$$\begin{aligned}\boldsymbol{a} &= -\alpha\boldsymbol{v} - \beta\hat{\boldsymbol{e}} \\ \boldsymbol{x} &\leftarrow \boldsymbol{x} + \boldsymbol{v}\Delta t \\ \boldsymbol{v} &\leftarrow \boldsymbol{v} + \boldsymbol{a}\Delta t,\end{aligned}$$

where α is a constant proportional to the drag coefficient and β to the mass of the particle.

Our solver is, however, broken. It is your task to **find and fix** the bug! Use the unit tests to your advantage. The program should give reasonable output and all tests should pass when you are done. See the section *The makefile and project structure* below for instructions for how to run the program and tests.

Note: If the program is broken in a way that it never exits, press CTRL+C in the terminal to abort.

Task 2

Your task is now to add a function (in src/vector.c) that calculates something related to vectors, this could, e.g., be the L2 norm. You should then verify that the function is doing this correctly by writing a test (in unit-test/main.c).

The makefile and project structure

The project is structured as follows:

- `src/`

  This folder holds all the C files that are related to the "real" program.

  - `main.c` the main file for the simulation
  - `vector.c` contains help functions called by `main.c`

- `include/`

  This folder contains all the header files that holds all of your data types (e.g., all your structs) and function prototypes. Specifically

  - `vector.h`

- `unit-test/`

  - `include/` contains header files needed for tests
    - `test_main.h`
  - `src/` contains the test itself
    - `test_main.c`
  - `test.mk` holds extra information for the makefile which is needed to compile the test

Note that you may need to add -lm and -lsubunit to LIB if you installed the check through apt on Linux. On some occations we have also noted that you need to include -lpthread and -lrt as well for the compilations to work

Valid targets for the makefile are:

- make program
- make test
- make clean

The first target creates your program (this could be your MD program for example). The second target creates your unit test that verifies that your functions is doing the correct thing. The third target removes all files that are created during the compilation, this includes the binary files. The makefile is hopefully structured in an understandable way, at least for the things you need

to modify. We will very briefly try to explain how you add more files to the program if needed. Note, always add your source files under src/ and header files under include/ that are used for the program. This requirement is related to some magic in the makefile. If you put them somewhere else, make won't find your files and it will very likely give you a super cryptic error message. Name the header file that is related to the source file with the same name, e.g., the header file that is related to the vector.c file should be named vector.h. To add more files to the compilation of the program add them in the makefile under the tag OBJ like this (note the ending of .o):

```
OBJ = \
    obj/vector.o \
    obj/one_more_source_file.o
```

To add more files to the test compilation add them in the file unit-test/test.mk under the tag TEST, like so:

```
TEST = \
    obj/main.o \
    obj/one_more_test_file.o
```

To change or add more CFLAGS to the test, change the tag CFLAGS in the file unit-test/test.mk. To change or add more CFLAGS to your program you do that in the CFLAGS placed in makefile (these flags will also propagate to the test).

Run the main program as

```
./program
```

and run the tests as

```
./run-test
```

You should plot the output to convince us that your solution is correct. To plot the output of the Newton II simulator you can redirect the. output to a file and plot it with your favorite tool (Python)

```
./program > trajectory.dat
python
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> data = np.loadtxt('trajectory.dat')
>>> plt.plot(data[:, 1], data[:, 2])  # Plot x versus y
>>> plt.xlabel('x')
>>> plt.xlabel('y')
```