Scientific Programming
(Wissenschaftliches Programmieren)

Exercise 10

Preparation

- The exercises 1A and 1B should be preferably solved by two developers, each of them making one part of the exercise in a **separate** repository. Once each developer finished the work, the repositories should be synchronized / merged as shown in the lecture. Please note, that the two repositories must have a common ancestor (e.g. before starting the exercise, one developer should clone the repository of the other and work in that).
- In case, the exercises 1A and 1B are both solved by the same developer, the developments should happen in two different branches, which should be then merged to main accordingly.

1A. Optimized solver algorithm (Developer 1, branch “optimization”)

- Write a small standalone “profiler” script `profile_solvers`, which creates a random linear system of equations (coefficient matrix and right-hand-side vector) of a given size using the `numpy.random.random()` routine and and then calls the `solve()` routine for that matrix.
- Run the profiler script with different matrix sizes, up to a size, which is large enough that the calculation takes ca. 5-10 seconds. How do the execution times scale with the dimension of the equations (linear, quadratic, cubic, etc.)? You can measure the execution time of a program by prepending the `time` command when starting the scipt (e.g. `time python profile_solvers.py` or `time python3 profile_solvers.py`).
- Replace the hand-coded solver algorithm within the `solve()` routine by a call to the `numpy.linalg.solve()` routine. How does it affect the execution speed for the same matrix sizes?
- Make sure, all tests still pass.
- Commit your changes (add also the profile script to the repository).

1B. Linear solver as standalone program (Developer 2, branch “fileio”)

- Create a standalone Python program/script `linsolver.py`, which reads a linear system of equations from a file `linsolver.in`, solves the linear system of equations and writes the result into a file `linsolver.out`. If the system of equations is linearly dependent, it should print an error message and exit with a non-zero exit code (generated with `sys.exit(`*`EXIT_CODE`*`)`) to signalize to the operating system that the execution failed.
- The script should import the `solvers` module (from `solvers.py`) and use the routine therein to solve the linear system of equation.
- The script should do its file I/O using the following two routines from a new module `linsolverio.py`:
  - `read_input()`: reads the input from a given file and returns the matrix *A* and the right hand side vector *b*.
  - `write_result()`: writes the result vector *x* into a given file

- The input file should have following format:

$$\begin{array}{cccc} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{nn} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \\ b_1 & b_2 & \dots & b_n \end{array}$$

  Example:

```
2.0  4.0  4.0
1.0  2.0 -1.0
5.0  4.0  2.0
1.0  2.0  4.0
```

- The output file should contain the solution vector in one row, provided the linear system of equation could be solved. Otherwise, the output file should not be created.

  Example output when the linear system of equation has a solution:

```
0.666666666666667 0.416666666666667 -0.5
```

- Make sure, the `linsolver` script exists with a well formatted error message (and non-zero exit code) gracefully (and not with an unhandled exception), if the input file is not found.
- Test your script with all 3 different test input data files from the test set as `linsolve.in`. Make sure, the exit codes are correct for each case (zero, if the linear system of equations could be solved, non-zero otherwise).
- Make sure, that the documentation of the routines in the `linsolverio` module appear in the Sphinx-generated API documentation.
- Commit your changes to the repository.

Note: You migh find the `numpy.loadtxt()` and the `numpy.savetxt()` routines helpful when implementing the functions for reading the input and writing the result.

Note: When you run a program from the shell, you can print out the exit code of the program by issuing

```
echo $?
```

immediately as next command after the program has finished.

2. Synchronization of concurrent developments

- Merge the two concurrent developments in one repository.
- Check, whether the unit tests and the standalone script with file I/O work as expected. If any problems arise, fix them and commit the changes to the repository.
- Synchronize the two repositories, so that the main branch of both repositories contain the same project status (a working `linsolve` program reading its input from a file, writing its results to a file and using the optimized Numpy routine for solving the linear system of equations).