Scientific Programming
(Wissenschaftliches Programmieren)

Exercise 8

1. Unit tests

- Create a `test/` subfolder and move `test_solvers.py` with git into this folder. Additionally create an empty `__init__.py` file in this folder.
- Rewrite the tests in `test/test_solvers.py` so that pytest can automatically check, whether the expected and the obtained results are close enough.
- Run the tests from the shell command line and from within your IDE. Make sure all tests pass in both cases.
- Analyze the test coverage, you should be able to reach 100%.
- Add a short section to the README about how to test the code.
- Commit your changes.

2. Code analysis

- Check the quality of your Python source files (`solvers.py`, `test/test_solvers.py`) with pylint and mypy.
- Change the source files to avoid any complaints from mypy and to obtain a pylint score of 10.0.
- Apply the formatter black to obtain further stylistic changes.
- Commit your changes.

3. Parametrized tests

- Rewrite the tests in `test/test_solvers.py`, to parametrized tests, which read their test data from files:
  - For each test case, create an input file containing A and b and (if applicable) an output file containing the expected solution x in the `test/data/` subfolder within your project:
    `test/data/elimination_3.in`, `test/data/elimination_3.out`,
    `test/data/elimination_4.in`, `test/data/elimination_4.out`,
    `test/data/pivot_3.in`, `test/data/pivot_3.out`,
    `test/data/lindep_3.in`
  - Create a parameterized test routine for verifying successful Gaussian eliminations. This routine should read a given input file with A and b, and a given output file with the expected result x, call the Gaussian elimination and compare the obtained result with the expected one.
  - Create a parameterized test routine to check for linearly dependent systems. This routine should read an input file with A and b, call the Gaussian elimination and verify, that it returned None (signalizing linear dependency).
  - Parametrize the two test routines, so that all four tests are executed.

- Commit your changes (including the new test data in the `test/data/` folder!).

4. *LU-decomposition

- Implement the function `lu_decompose()` for LU decomposition with partial pivoting. The function should produce the LU-factorized matrix (as one matrix, see the form below) along with the permutation vector representing the row-permutations. If the matrix is linearly dependent, the routine should return `None`.
- Implement the function `forward_substitute()` for forward substitution. It should take the LU-decomposed matrix LU (as returned by the LU-decomposition routine) and a vector b as arguments, solve the equation $Ly = b$ and return the solution vector y.
- Implement the function `backward_substitute()` for backward substitution. It should take the LU-decomposed matrix LU (as returned by the LU-decomposition routine) and a vector y as arguments, solve the equation $Ux = y$ and return the solution vector x.
- Add unit tests for all three newly created functions. Make sure to test `lu_decompose()` also for a linearly dependent system.
- Commit your changes to the repository.
- Revise the `gaussian_eliminate()` function so that it solves the linear system of equation by calling these new functions using appropriate arguments. Rename the routine to `solve()` (as it does not use the Gaussian elimination any more). Make sure, it passes all previously stored unit tests.
- Commit your changes to the repository.
- **Note:** Given the lower triangle matrix L and the upper triangle matrix U, the LU-decomposed matrix LU has the form as shown below:
- L = $\begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix}$ U = $\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$ LU = $\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ l_{21} & u_{22} & u_{23} \\ l_{31} & l_{32} & u_{33} \end{pmatrix}$