**FYSS5120 Efficient Numerical Programming - Demo 6**

**Drop solutions before the demo session to the Nextcloud box (link)**
**Please indicate clearly your name in the file name,**
**so that I can tell who solved what.**

1. **Parallelization in Distributed Memory Machines**
   The sample code `mpi4py_eigvals.py` (link to directory) uses MPI parallelism from module `mpi4py`. `mpi4py` requires a system-level MPI installation, such as OpenMPI, Microsoft MPI or Intel MPI, which provide `mpiexec` or `mpirun`. Roughly speaking, `mpi4py` gives you means to code your problem in a parallel fashion, and the system-level MPI knows how to execute that code in your hardware. Run the code `mpi4py_eigvals.py`.

2. **Parallelization in Shared memory Machines**
   Rewrite `mpi4py_eigvals.py` to use the module `multiprocessing.Pool` for shared memory environment. It's best to write the `multiprocessing.Pool` version in a separate file. Run a speed comparison to the MPI version. An example of `multiprocessing.Pool` is `multi_pool.py` (link to directory). Check also that the results are independent of parallelization.

3. Voluntary:
   MPIRE is a multiprocessing library for Python (see link to github page or the the blog post by Sybren Jansen (link to blog @towardsdatascience.com)) MPIRE claims to be faster than `multiprocessing`. The syntax of `WorkerPool` in MPIRE is almost the same as `multiprocessing.Pool`. Installation: Either `git clone` the github package or use `pip`,

   ```
   $ python -m pip install mpire
   ```

   Modify your code to use either `multiprocessing.Pool` or `WorkerPool` in MPIRE, and compare their speeds. Add a progress bar to the MPIRE version.

   MPIRE `map()` collects NumPy arrays to chunks, see link. As a result, `task()` may get arrays shaped (25,300,00), while expecting a single (300,300) array input. To prevent this, you can, for example, convert NumPy arrays to lists before using MPIRE `map()`. Another possibility is to use MPIRE's `make_single_arguments`.