Exercise 6 FYSA120 C++ numerical programming Winter 2014

Email the *commented* solution code (*.cpp) to :
address: `FYSY160(at)gmail.com` Subject-line: `demo6`
If you run into trouble, please send questions also to that address.

1. The sample code `gauss_seidel_serial.hpp` uses the iterative Gauss–Seidel algorithm for solving a linear set of equations `Ax=b`. The algorithm is from Wikipedia (web link). The code `gauss_seidel_main.cpp` is for testing. The task is to test the performance of the algorithm and whether it works well in parallel.

   - Parallelize the innermost `j`–loop in `gauss_seidel_serial.hpp` using OpenMP. The sample code `numerics/openmp_reduction.cpp` shows how this is done.
   - Add a call to this function to `gauss_seidel_main.cpp`, along with the timing code.
   - Compare the timing of the serial vs. parallel code. Didn't get any speedup? Try a larger matrix.

   The number of OpenMP threads is set programmatically to `4` using

   ```
   omp_set_num_threads(4); // overrides OMP_NUM_THREADS
   ```

   or using an environment variable, in the linux shell

   ```
   export OMP_NUM_THREADS=4  (bash)
   setenv OMP_NUM_THREADS 4  (csh and tcsh) .
   ```

   Extra:
   Before we get carried away by home–made implementations of numerical routines, Armadillo `solve()` outperforms my Gauss–Seidel code by a factor of 10. The `solve()` is "just" a wrapper to library calls to BLAS and LAPACK. This is actually great, because the BLAS/LAPACK interface is hideous, while `solve()` is nice and simple! If you are interested, the Armadillo code in calc.phys.jyu.fi is
   `/usr/local/include/armadillo_bits/glue_solve_meat.hpp`,
   which calls functions in
   `/usr/local/include/armadillo_bits/auxlib_meat.hpp`.
   This is what I mean when I emphasize the importance of user–friendly interfaces and encourage to hide boring details to headers. Armadillo does it well, that's why people use it.