Maggs-Rossetto simulations

This is some code I wrote to simulate the Maggs-Rossetto algorithm in 2D on the square lattice and 3D on the simple cubic lattice. Most of the development has gone into the 2D version as a test case before moving to 3D.

OpenMP and some form of MPI (I use OpenMPI and haven't tested any other implementation, but it doesn't do anything particularly sophisticated so might well work with a different one) are required. Make simply with

```
$ make
```

To recompile I use

```
$ make cleaner && make
```

to ensure `gcc` rebuilds all the module files, etc. correctly. Run with

```
$ ./exec_name inputfile
```

for the serial versions or

```
mpirun -np numslots exec_name inputfile
```

for the parallel ones; optionally add `-v` before the input file for more verbose output. My default input file is located at `ROOT_DIR/in/start.in`, but there's also a file `scripts/run.pl` which will take various arguments and generate appropriate input files, time stamp everything and store the output, and plot graphs of various things using gnuplot. I'm currently (Sept. 2017) extending it to generate job scripts for the SGE on UCL clusters.

common.f03

This one's a standard fortran common file: contains definitions of various quantities needed everywhere else. The system size, charge array, fields, etc. etc. are all in here. I have tried to keep the number of globals down as much as possible, but have not been successful. Some of the quantities in there could be added to a "parameters" derived type, for example, but I haven't got around to doing this. Also contains simple arrays containing positive and negative neighbours of each site in each direction. These are very, very simple and would need to be sorted out for any more complicated lattice. Finally contains the random number generator I use, which is a lagged Fibonacci generator. The code for this is taken from (unpublished, so far as I know) code by Michael Faulkner, used in his thesis and also in papers *INCLUDE PAPER REFS.*

input.f03

A module to read input from a file, the path to which should be given as the final command-line argument. Basically the subroutine loops over lines and checks the first word against a long list of cases. If any line doesn't match, a short warning is printed, but the program continues; if the verbose flag is set, the program prints out what it's just read in. After reading in the entire input file, some basic manipulations are done to sort out what's been read in. Filenames are trimmed and basic quantities like β calculated.

`setup.f03`

`linear_solver.f03`

`update.f03`

`output.f03`

`main.f03`