RMT- output

List of output files

The following files are updated during the calculation every `Timesteps_Per_Output` iterations

- `CurrentPosition`
  - Summary of the current status of the calculation (key parameters, and values of variables)
- `rmt.log`
  - Output printed by executing code
  - Debug information during setup
  - Summary of the current status of the calculation
- `pop_all.<version_number>`
- `pop_inn.<version_number>`
- `pop_out.<version_number>`
  - Contain the total/inner/outer population
- `timing_inner.<version_number>`
  - Timing information recorded on the inner region master
- `timing_outer0.<version_number>`
  - Timing information recorded on the outer region master
- `timing_outer1.<version_number>`
  - Timing information recorded on the first (non-master) outer core
  - The two outer timing files allow the balancing of work on the outer region master

Additionally:

- `hstat.<version_number>`
  - Checkpoint status file which is read by the RMT code for restart
  - Is written only after the latest Checkpoint
- `EField.<version_number>`
  - The electric field strength (in a.u) is written at every time-step

Optional outputs

- `expec_z_all.<version_number>` `(Dipole_Output_Desired=true)`
  - The expectation value of the dipole length operator at every time-step
- `expec_v_all.<version_number>` `(Dipole_Velocity_Output=true)` and

(`Dipole_Output_Desired=true`)
  - The expectation value of the dipole velocity operator at every time-step

Channel populations

The `data` directory contains files describing the population in

- The ground state
- Each of the electron emission channels

after every `Timesteps_Per_Output` iterations

The channel numbering follows the standard R-matrix protocol, in ascending order sorted by

1. The total angular momentum of the final state
2. The parity of the final state (even then odd) (only relevant for calculations with non-zero m_l)
3. The energy of residual ion state to which the emitted electron is coupled
4. The angular momentum of the emitted electron

E.G. consider a calculation for argon comprising the $^2P^o$ and $^2S^e$ ionisation thresholds, with $L_{\mathrm{max}}=3$. The possible final states are $^1S^e$, $^1P^o$, $^1D^e$ and $^1F^o$ and the 11 channels are ordered as:

State	Residual ion	Continuum electron	Channel ID
$^1S^e$			
	$^2P^o$ +	ϵp	1
	$^2S^e$ +	ϵs	2
$^1P^o$			
	$^2P^o$ +	ϵs	3
	$^2P^o$ +	ϵd	4
	$^2S^e$ +	ϵp	5
$^1D^e$			
	$^2P^o$ +	ϵp	6
	$^2P^o$ +	ϵf	7
	$^2S^e$ +	ϵd	8
$^1F^o$			
	$^2P^o$ +	ϵd	9
	$^2P^o$ +	ϵg	10
	$^2S^e$ +	ϵf	11

Wavefunction data

The initial and final wavefunction is recorded in the `ground` and `state` directories respectively. The wavefunction is written out in parallel by each MPI task, and thus the wavefunction must be reconstructed from the unformatted files in postprocessing. A utility code (reform) is provided to accomplish this, but the basic procedure is outlined here.

The utility code is compiled against the same common code base as RMT, so it has access to the input files input.conf, H, Splinedata and Splinewaves. The code expects to be run from either the 'ground' or 'state' directories and thus looks in the parent directory (whence the rmt.x executable is run) for these files. This is important if you are performing post-processing on a different machine, and soft links to input files are not preserved.

Inner region wavefunction

Each block master in the inner region writes out the spline-coefficients for each channel wavefunction in a file `psi_inner<block_ID>.<version_number>`. To reconstruct the wavefunction in each channel, the utility code reads the spline information from the input files `Splinedata` and `Splinewaves`, and the atomic structure information from the input file `H`.

The output of the utility script are the files `InnerWave<Channel_ID>` where the channel ID is as described above, which contain the value of the wavefunction at a set of grid points in the inner region.

Outer region wavefunction

Each outer region MPI task writes out its wavefunction data per channel into a file named `psi_outer<outer_ID>.<version_number>` where the `<outer_id>` is essentially the rank of the MPI task in the outer region labelled from `0` to `No_Of_PEs_to_Use_Outer-1`. The utility script reads in the data from every `<outer_ID>`, calculates what the corresponding grid point should be, and writes out the wavefunction for each channel in files `OuterWave<Channel_ID>` where the channel ID is as described above.

Execution

Presuming that the `reform` code has been succesfully compiled, it can be copied or linked into the `ground` or `state` directory as required and then is run simply as

```
>> ./reform
```

The code does have OpenMP loop directives enabled, so multicore machines can be exploited simply by exporting the variable `$OMP_NUM_THREADS` prior to execution.