RMT Code: Computational considerations

Overview

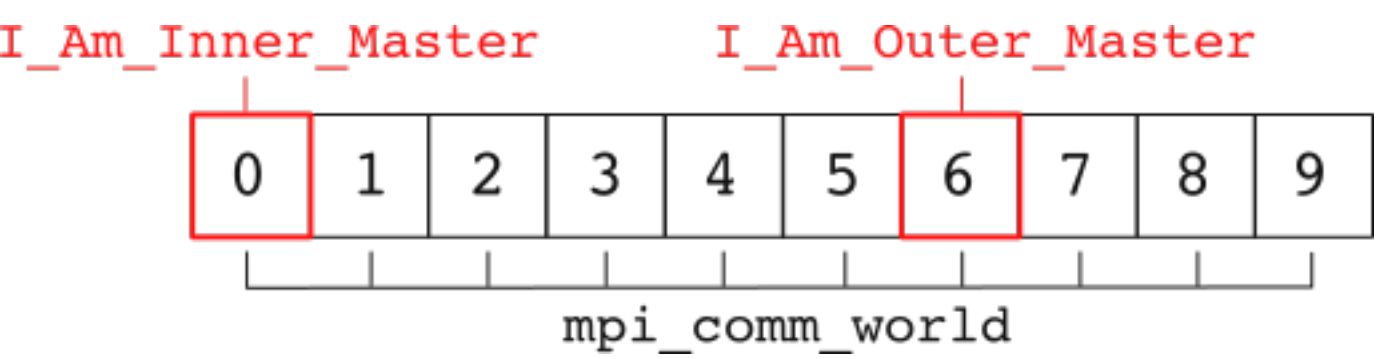
A note on precision

The code is designed to work with a flexible floating point precision. You can set the number of decimal places in the module precisn.f90 by adjusting the parameter decimal_precision_long. The code has been tested for decimal_precision_long=15.

Parallelisation Strategy

The code employs the standard R-matrix paradigm of dividing configuration space into two regions. Within each region a different parallelisation strategy is employed. In any given calculation, the bottleneck will exist in one of these two regions. For calculations comprising a high degree of atomic structure, the inner region tends to dominate. For those comprising many channel functions (large angular momentum expansion) the outer region can dominate. The skill in optimising the calculation is to balance the workload in each region by a judicious allocation of cores.

Communication between the two regions is handled by the ‘region-master’ cores (essentially the first core in each region). Rather than having a separate communicator for this, every core has a logical flag set at the start of the calculation: `I_Am_Inner_Master` is set on all inner region cores and is true only for the inner-region master. Similarly, `I_Am_Outer_Master` is set for the outer region cores. Calls to subroutines named `First_PEs_share_<something>` or `First_PE_receives_<something>` and so on then are called from all cores in a given region, and the logical flags are used to determine which cores are involved in the communication using `MPI_Comm_World`.

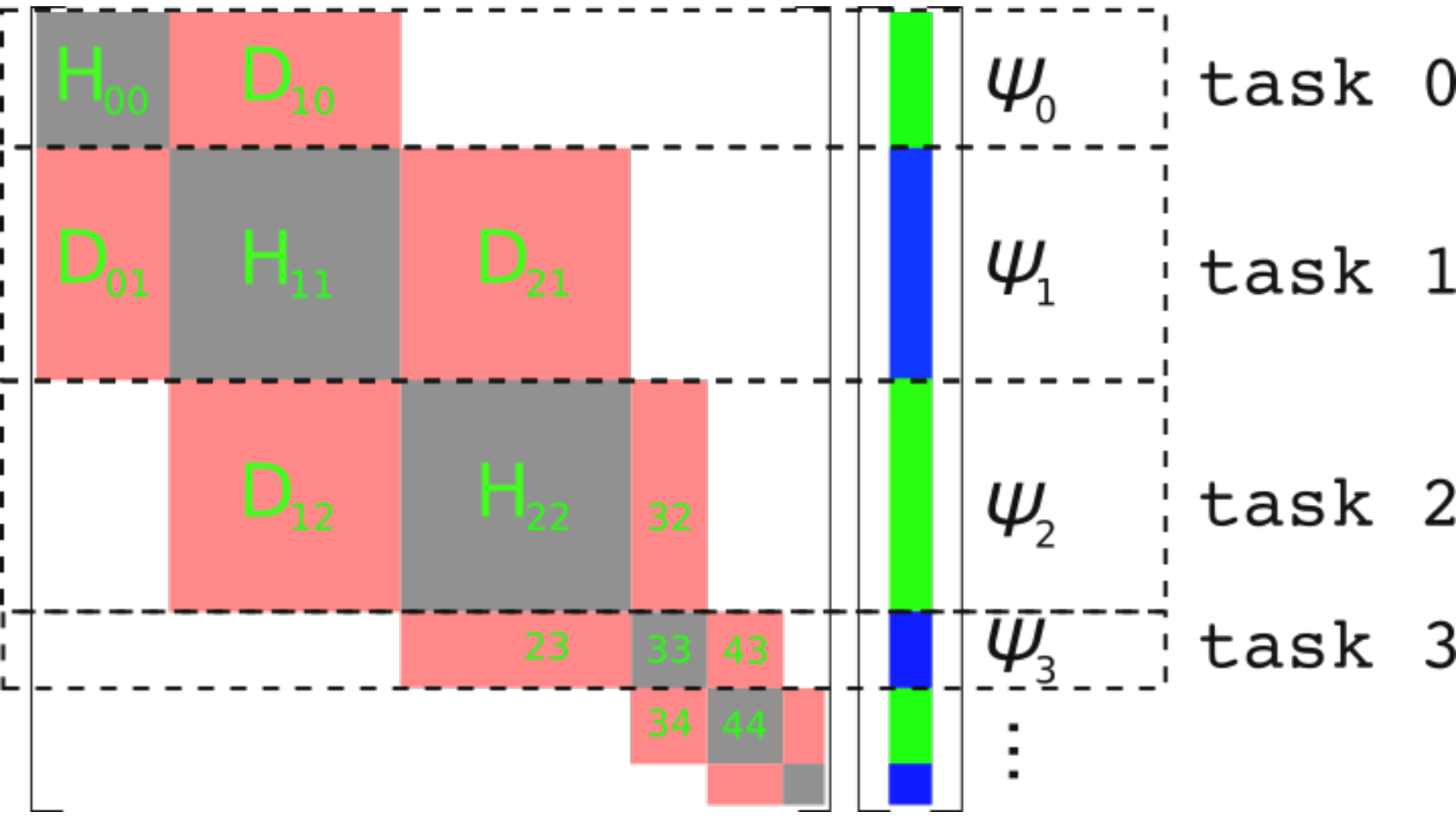


Inner region parallelisation

The calculation in the inner region involves repeated application of matrix vector multiplications. This calculation is parallelised in three layers using both distributed (MPI) and shared (OpenMP) paradigms.

Layer 1

Both the Hamiltonian matrix, and the wavefunction vector are divided into symmetry blocks containing states of a given angular momentum, as shown in the diagram below. The first layer of parallelism is to assign each symmetry block to an MPI task. In the simplest arrangement, one MPI task is assigned to each block, so (refering to the diagram below) \(H_{00}\), \(D_{10}\) and \(\psi_{0}\) are local to MPI task 0, \(H_{11}\), \(D_{01}\), \(D_{12}\) and \(\psi_{1}\) on local to MPI task 1 etc. Here, \(H_{ii}\) refers to the energies of the states which have total angular momentum \(L=i\), while \(D_{ij}\) refers to the dipole matrix elements coupling a state with \(L=i\) to a state with \(L=j\). (Obviously there are multiple states within each block, but for layer 1, this level of abstraction is appropriate.)



The code expects at least one core per symmetry block, so for a calculation with \(L_{max}=N\) you need to allocate at least \(N+1\) cores to the inner region.

The communication for this layer is handled between nearest neighbours. Each MPI task needs only the wavefunction data from the tasks above and below. At the beginning of each iteration, the data is transferred between the tasks using the communicator `Lb_m_comm` (Lb for L block, m for master, see below). This occurs in the subroutine `Parallel_Matrix_Vector_Multiply` in the module `live_communications`.

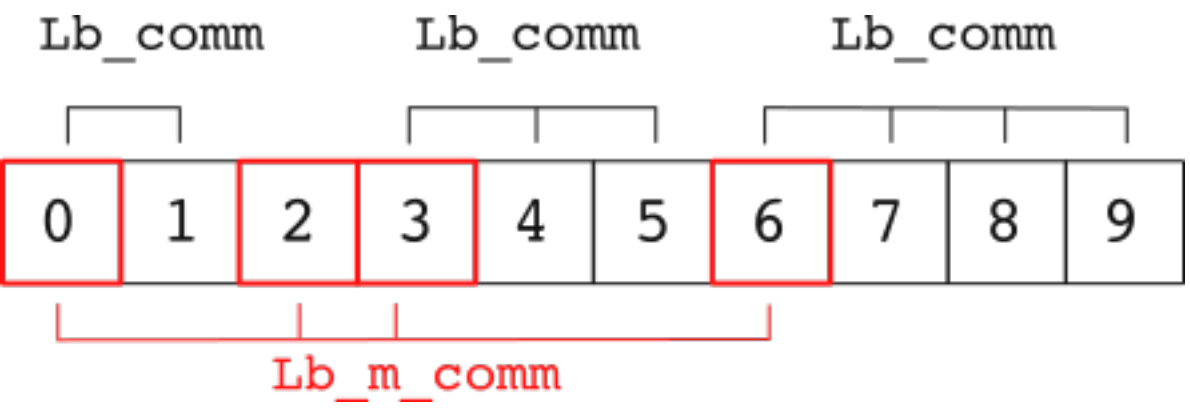
We note that there are other possible couplings when the magnetic quantum number `ml` is non-zero, but the parallel scheme remains the same, albeit with additional dipole blocks for each symmetry.

Layer 2

Evidently, there are systems where certain symmetries will contain substantially more states than others. As suggested by the diagram above, in this case the dipole blocks are substantially larger, and the number of multiplications required on each iteration grows with the square of the size of the block. Hence the code has the flexibility to assign multiple MPI tasks to each block. The tasks are allocated by a routine which first assigns one task per block, and then calculates which block has the most work per task, assigning an additional task, until all tasks are allocated. The table below shows a typical allocation of 144 tasks to the 10 blocks used in a calculation for singly ionised neon.

Block	States	Tasks
0	429	7
1	970	29
2	1126	37
3	979	29
4	726	17
5	500	9
6	369	5
7	305	4
8	284	4
9	276	3

Each block is now controlled by the so-called ‘block master’ which communicates with the neighbouring blocks and distributes data to the tasks within the block. Each task within a block handles an equal number of rows of the matrix vector multiplication. The relevant portions of the dipole blocks are distributed to each task at the start of the calculation by the block master and the wavefunction on each iteration. Communication within each block is handled on the communicator `Lb_comm`.



Layer 3

The final layer of parallelism in the inner region is shared memory parallelisation on each MPI task. From layers 1 and 2, each MPI task has a chunk of the Hamiltonian matrix and wavefunction vector with which it performs matrix-vector multiplications. The shared memory parallelism is implemented in two ways.

First, several do loop structures with independent loops are farmed out to the shared memory threads with a simple `!$OMP PARALLEL DO`. Secondly, all library routines for linear algebra which support shared-memory processing will execute in parallel on all available shared memory threads.

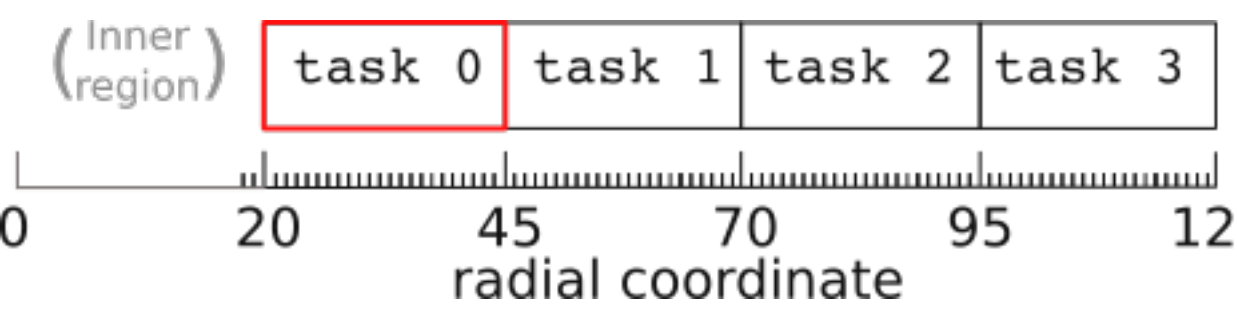
In practice, we have found only marginal performance gain from using more than one OpenMP thread in the inner region.

Outer region parallelisation

The outer region parallelisation is somewhat easier to envisage than the inner region, as it is actually a division of physical space. Two layers of parallelism are employed in the outer region, one using MPI and one using OpenMP.

Layer 1

The major division in the outer region is a division of the physical space. Thus, an outer region of 100 a.u. might be divided into smaller sectors of 25 a.u with each sector handled by an MPI task. Because the outer region uses an explicit grid based representation of the wavefunction, this corresponds to each MPI task handling a set number of grid points.



In practice, the outer region master (first MPI-task in the outer region, highlighted red above) is allocated a smaller number of grid points than the rest of the outer region, to allow additional resource for the extra communication responsibilities with the inner region.

Performance is enhanced by reducing the number of grid points per sector (and increasing the number of outer-region MPI tasks to maintain the size of the outer region). However, the finite-difference rule implementation requires a minimum of 24 grid points per sector. If this limit is reached, further performance can be extracted from the second layer of parallelism.

Layer 2

As in the inner region, additional performance can be extracted with the use of OpenMP parallelism of each MPI task. Thus a number of shared memory threads can be allocated per outer region task (this is controlled separately from the number of OpenMP threads in the inner region). In the outer region, most of the calculation takes place for each electron-emission channel independently of the other emission channels, so all do loop structures which loop over the variable `channel_ID` are parallelised with the `!$OMP PARALLEL DO` structure.

As with the inner region, we have found that in practice the greatest performance gains are obtained with providing additional MPI tasks rather than OpenMP threads. However, because of the hard limit on

how small the outer region sectors can be made with additional MPI parallelisation, the OpenMP layer becomes more useful in the outer region, especially in systems with many electron-emission channels.

Input data

Data files (from prmatrix?)

- H
- d
- d00
- Splinewaves
- Splinedata

Input parameters

RMT reads input from the file ‘input.conf’ which controls the setup of the laser pulse(s), the parallel structure of the calculation and various other calculation specific parameters. Of these parameters, some are required, and others can be left to default values. The list below contains all the input parameters which can be set, along with their default values.

Lines in the input file may be commented with the hash symbol #. All inputs begin with the option name (All options are case sensitive) then an equals sign, followed by the value of the option. for example:

```
Frequency  = 2.0
```

true/false options can be written with or without the fortran dots, for example

```
USE_2Colour_Field          = false
```

```
Keep_CheckPoints           =.false.
```

are both valid.

all spaces are stripped from the file, but tab characters will break the input.

There is no need to include quotation marks for strings e.g:

```
Version_root               = He_w390_I101014_05
```

Required variables

- `No_Of_PEs_to_Use_Inner`
  - The number of MPI-tasks to use in the inner region
  - Must be at least equal to the number of symmetry blocks which is given by \(L_{\mathrm{max}}+1\)

- `No_Of_PEs_to_Use_Outer`
  - The number of MPI-tasks to use in the outer region
- `X_Last_Master`
  - The number of grid points allocated to the outer-region master
  - Must be \(\geq\) 24
  - Must be a multiple of 4
  - Is usually smaller than `X_Last_Others`
- `X_Last_Others`
  - The number of grid points allocated to the outer-region master
  - Must be \(\geq\) 24
  - Must be a multiple of 4
- `DeltaR`
  - the grid spacing in the outer region
  - numerical stability requires a small DeltaR
  - (For a strong field calculation we usually we go for 0.08 a.u)
  - larger grid spacings can be used when the energy of the outgoing electron will be smaller
- `Steps_Per_Run_Approx`
  - The approximate number of time-steps in the calculation
  - The actual number of time-steps will depend on the checkpointing conditions (you have to complete a round number of time-steps essentially)
  - The time-step is calculated using `Final_T` and the `Steps_Per_Run_Approx` (see next)
- `Final_T`
  - The end-time of the calculation in atomic units
  - Used with `Steps_Per_Run_Approx` to calculate the time-step
  - The stability will depend on many parameters, but usually, we aim for a time-step of 0.01 a.u, so `Steps_Per_Run_Approx = 100 Final_T`

Optional Parameters (default value)

Electric Field Parameters

- `USE_2Colour_Field` (false)
  - Use two independently controllable pulses
  - For historical reasons, the primary pulse is referred to as the IR, the secondary as the XUV
- `USE_Leakage_Pulse` (false)
  - For certain applications (particularly attosecond transient absorption spectroscopy) it is useful to include a low intensity replica of the IR pulse locked in time with the XUV pulse
  - Hence this is not an independently controllable pulse- the only parameter you can set is the intensity
- `Frequency` (0.0)
  - Frequency of the (primary) IR pulse in a.u. (800 nm = 0.0569 a.u)
- `Periods_Of_Ramp_On` (0.0)
- `Periods_Of_Pulse` (0.0)
  - The pulse profile is a \(\sin^2\) envelope

    - `Periods_Of_Ramp_On` sets both the ramp on and off
    - `Periods_Of_Pulse` is the total number of field cycles including the ramp on and off
    - Thus, `Periods_Of_Pulse` \(\ge\) `2*Periods_Of_Ramp_On`
    - The code supports non-integer numbers of cycles
- `ceo_phase_deg` (0.0)
    - The carrier envelope phase measured in degrees
- `Intensity` (0.0)
    - The peak intensity of the (primary) IR pulse in units of \(10^{14} \mathrm{Wcm} ^{-2}\)
- `Intensity2IR` (0.0)
    - The intensity of the leakage (secondary) IR pulse in units of \(10^{14} \mathrm{Wcm} ^{-2}\)
- `Frequency_XUV` (0.0)
- `Periods_Of_Ramp_On_XUV` (0.0)
- `Periods_Of_Pulse_XUV` (0.0)
- `ceo_phase_deg_xuv` (0.0)
- `Intensity_XUV` (0.0)
    - All parameters for XUV pulse set as for IR pulse
- `time_between_peaks_in_fs` (0.0)
    - time between the peak of the (primary) IR pulse and XUV pulse measured in fs
    - negative time delay = XUV pulse arrives first

Atomic Structure Parameters (Default values)

- `ml` (0)
    - The total magnetic quantum number of the system
- `lplusp` (0)
    - The sum, modulo 2 of the angular momentum and parity of the ground state
    - e.g. G.S of Ne: 1Se gives \(L=0\) \(\pi=0\): -> `lplusp=0`
    - e.g. G.S of C: 3Pe gives \(L=1\) \(\pi=0\) -> `lplusp=1`
- `GS_finast` (1)
    - which of the included symmetries is the G.S
    - e.g. Ne, including 1Se, 1Po, 1De etc. G.S is 1Se -> `GS_finast=1`
    - e.g. Ne+, including 2Se, 2Po, 2De etc. G.S is 2Po -> `GS_finast=2`
- `Adjust_GS_Energy` (false)
    - Can choose to adjust the ground state energy to a given value
- `GS_Energy_Desired` (0.0)
    - Ground state energy in a.u. (measured relative to ionisation threshold)
    - e.g. for He: `GS_Energy_Desired=-0.9035`
- `Remove_Eigvals_Threshold` (true)
- `Surf_Amp_Threshold_Value` (1e0)
- `neigsrem_1st_sym` (14)
- `neigsrem_higher_syms` (20)
    - For numerical stability the code will remove the highest energy eigenvalues of the inner region Hamiltonian matrix

    - These energies are usually non-physical, so their removal is advised
    - The cutoff for these energies is set by `Surf_Amp_Threshold_Value`
    - The number of eigenvalues removed is set by `neigsrem_1st_sym` for the first symmetry block and `neigsrem_higher_syms` for the others
    - For most purposes the default values are advised

Calculation parameters

- `Dipole_Output_Desired` (false)
- `Dipole_Velocity_Output` (false)
    - Output the expectation value of the dipole (length and velocity) operator (for High Harmonic Generation for instance)
- `No_Of_OMP_Threads_Inner` (1)
- `No_Of_OMP_Threads_Outer` (1)
    - Number of OpenMP threads to employ in the inner and outer regions
- `Taylors_Order` (8)
    - Order of the propagtor to employ
- `Version_root` (”)
    - version identifier to suffix to all output files
    - final suffix also contains intensity identifier
- `Timesteps_Per_Output` (20)
    - the interval at which output data is written to file
    - note that EField, and expec files are written *every* timestep regardless of `Timesteps_Per_Output`
- `Checkpts_Per_Run` (1)
    - write output at intermediate checkpoints for restarting calculations

Additional parameters

- disk_path
    - by default, the executable will look in the run-time directory for the input files, `input.conf`, `H`, `d00` etc.
    - should you want to place the input files in a different location, for instance for a batch job, the path to the input files should be set as the environment variable `disk_path`
    - e.g. batch job script for queuing system:

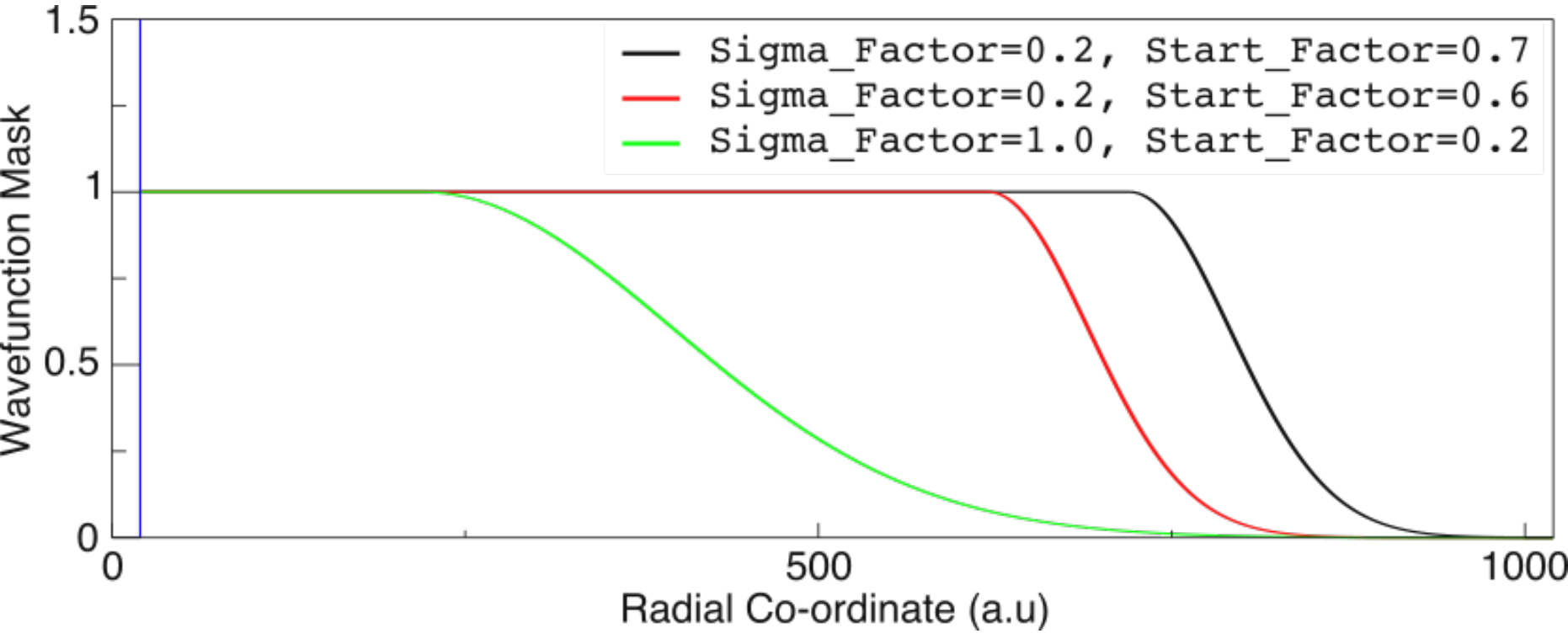
```
>> ls
submit.pbs run_dir_1 run_dir_2
>> ls run_dir*
run_dir_1:
d d00 data ground H input.conf rmt.x Splinedata Splinewaves state
run_dir_2:
d d00 data ground H input.conf rmt.x Splinedata Splinewaves state
>> tail submit.pbs
```

```
# variable PBS_ARRAY_INDEX loops over values 1,2
export disk_path=$PWD/run_dir_$PBS_ARRAY_INDEX
mpirun -n 32 $disk_path/rmt.x > $disk_path/rmt.log
```

- Absorbing boundary
  - By default, no absorbing boundary is enabled in the outer region. This means if your outer region is not sufficiently large the outgoing wavefunction will reflect from the boundary and create spurious oscillations in the results.
  - Enabling the absorbing boundary is not possible at run-time, as it is hard-coded in the module `calculation_parmaters`
  - There are four relevant variables therein
    - `Absorb_Desired` (Logical- set true to enable)
    - `Sigma_Factor`
    - `Start_Factor`
    - `Absorption_Interval`
  - Essentially, the absorbing boundary is a Gaussian mask which multiplies the outer region wavefunction after every `Absorption_Interval` iterations.
  - The un-masked proportion of the outer region is set by `Start_Factor`. `Start_Factor=0.7` would start the absorbing boundary 70% in to the outer region
  - `Sigma_Factor` sets the severity of the absorption. the larger this is, the gentler the absorption
  - The equation for the mask function is given by $$M(x)=\begin{cases} 1 &\quad x< x_{\mathrm{start}} \\\\ e^{-\left(\frac{x-x_{\mathrm{start}}}{\sigma}\right)^2} &\quad x_{\mathrm{start}} \le x \le x_{\mathrm{last}}\end{cases}$$

where x_{last} is the outermost point in the outer region, $x_{\mathrm{start}} =$ `start_factor`$\;\times\; x_{\mathrm{last}}$ and $\sigma=\frac{x_{\mathrm{last}}}{2}\;\times\;$`sigma_factor`

The mask function for an outer region of 1000 a.u. is shown for a variety of parameters below. Note that the outermost point is 1020 a.u. as the inner region is 20 a.u.



Execution

In order to execute the RMT code, the following are required in the directory

- The input data files: H, d, d00, Splinedata, Splinewaves
- The input file: input.conf
- The executable: rmt.x
- The directories: data, ground, state

The total number of MPI tasks is given by

```
No_Of_PEs_to_Use_Inner + No_Of_PEs_to_Use_Outer
```

And in the case that `No_Of_OMP_Threads_Inner = No_Of_OMP_Threads_Outer = 1` the basic execution command (for 32 MPI tasks, for instance) is

```
>> mpiexec -n 32 ./rmt.x > ./rmt.log
```

If using more than one OpenMP thread in either region, you should refer to your machine documentation on how to execute hybrid MPI/OpenMP jobs.