Benchmark test for cross-correlation simulation

Victor S C Canela

November 12, 2018

1 Overview of the code structure

The structure of the simulation is shown in the following diagram. The colours correspond to the roles the different sections of the code play.

- Blue. Only are visited once, at the beginning of the simulation and outside the main trajectory loop. The sections contain:
  - Definition of parameters and variables types.
  - Allocation of memory to arrays.
  - Initialisation of counters, and variables.
  - Populate arrays.
- Black. Section A, B and E are visited once every loop.
  - Section A. Important values are calculated: mean photon numbers for cavities a and b, probabilities to be in atomic states.
  - Section B. Calculate correlation and jump probabilities.
  - Section E. Write correlation and important values (from Sects. A and B) to output file.
- Grey. Every loop, one of them is chosen randomly (according to the probabilities calculated in Sect. B). The different paths are:
  - Section C, tag 0. A continuous evolution takes place (changing the values by a small amount), whereby the state vector is integrated using a 4th order Runge-Kutta.
  - Section D. A second choice has to be made:
    - Cavity a emission, tag 1.
    - Cavity a reflection, tag 2.
    - Cavity b emission, tag 3.
    - Cavity b reflection, tag 4.

  That is, the path on each loop, according to the sections visited, is one of

  $$A \to B \to \{C_0, D_1, D_2, D_3, D_4\} \to E\,.$$
- Green. The benchmark test section, where an output file is created with values extracted from all possible paths.

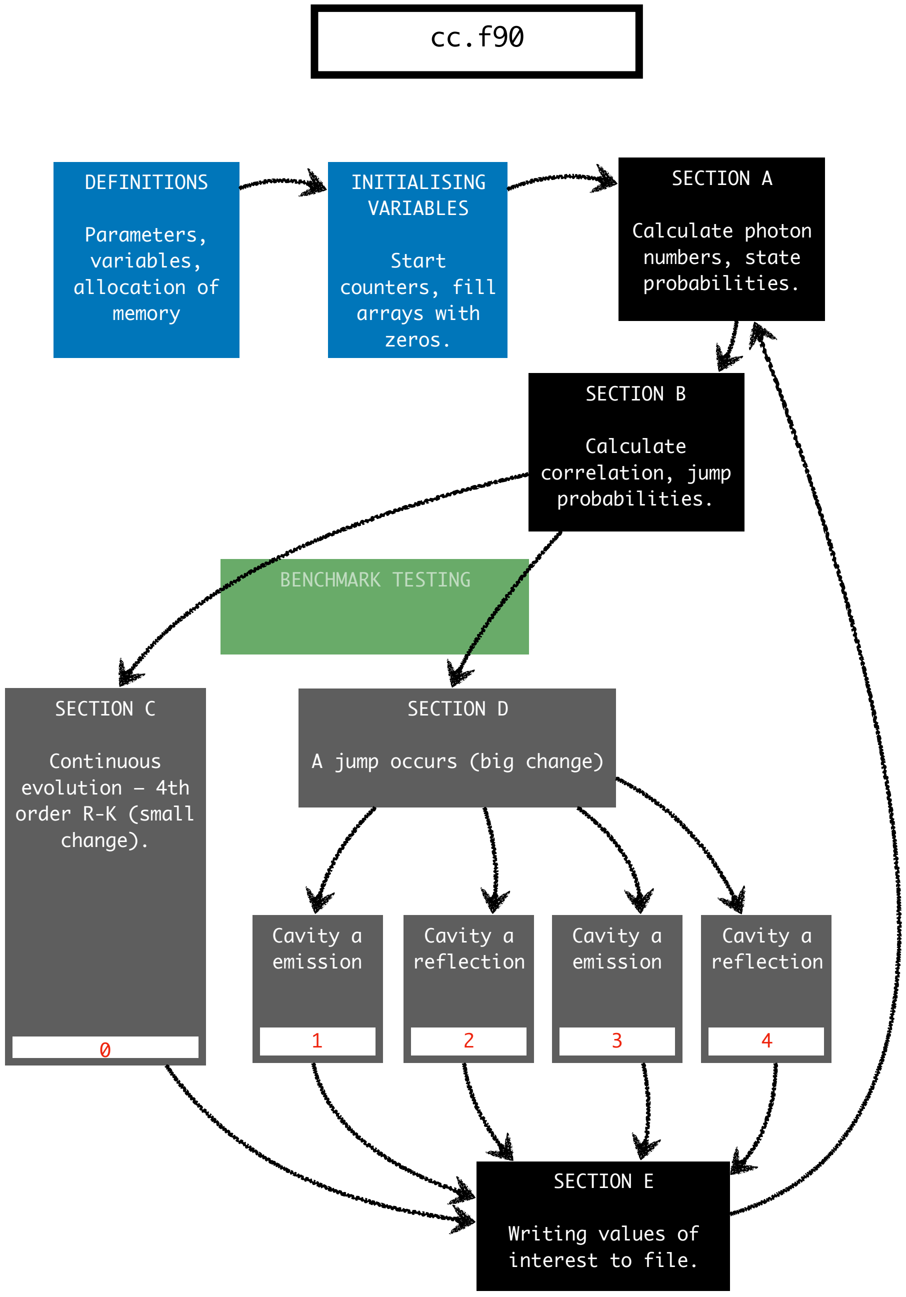
cc.f90
DEFINITIONS
Parameters, variables, allocation of memory
INITIALISING VARIABLES
Start counters, fill arrays with zeros.
SECTION A
Calculate photon numbers, state probabilities.
SECTION B
Calculate correlation, jump probabilities.
BENCHMARK TESTING
SECTION C
Continuous evolution - 4th order R-K (small change).
0
SECTION D
A jump occurs (big change)
Cavity a emission
1
Cavity a reflection
2
Cavity a emission
3
Cavity a reflection
4
SECTION E
Writing values of interest to file.

Figure 1: Diagram of the simulation cc.f90. Each section of the code is colour coded describing its role and loop appearance. Note that on each loop, only one grey box is chosen (the section tags used for the benchmark test are shown in red). After Section E, the program advances in time and starts from Sectino A.

2 The benchmark test

In order to test the validity of the trajectory, the output of the code-to-be-tested—`cc.f90`—is compared to a trusted output (obtained from a stable version). It is important to not change any parameters of the trajectory so that the output can be properly compared.

2.1 The output

The output file consists of columns which contain the change in time of the following variables (from column left to right):

k	last_sect	photon_a	photon_b	p_gg	p_ee	p_ff	prob_atom_decay	prob_t_a	prob_t_a	prob_t_a	prob_t_a
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

The variables are,

- `k`, the time-step.
- `last_sect`, the tag of the last section visited.
- `photon_a`, the photon number value for cavity a.
- `photon_b`, the photon number value for cavity b.
- `p_gg`, the probability of the lower atomic state.
- `p_ee`, the probability of the intermediate atomic state.
- `p_ff`, the probability of the higher atomic state.
- `prob_atom_decay`, probability of atomic emission.
- `prob_t_a`, probability of a transimission from cavity a.
- `prob_r_a`, probability of a reflection from cavity a.
- `prob_t_b`, probability of a transimission from cavity b.
- `prob_r_b`, probability of a reflection from cavity b.

2.2 When is the output recorded?

There are 5 possible paths to take at each time-step. According to the labels used in the figure, they are,

1. $A \to B \to C \to E$
2. $A \to B \to D_1 \to E$
3. $A \to B \to D_2 \to E$
4. $A \to B \to D_3 \to E$

5. $A \rightarrow B \rightarrow D_4 \rightarrow E$

To make sure we check all of them, we cycle through a variable, `last_sect`, and compare it to a counter where we only write to output when the tag section matches the corresponding path. This is even more necessary as the paths are taken with very different frequencies e.g. path 1 will dominate in frequency over all other paths.

If we force to write output after a particular section has been visited, we can check for bugs without having to wait a long time for the program to "naturally" visit the very-rarely-visited section.

The output writing condition operates:

1. Start the counter: `temp_bm` $= 0$.
2. Wait 10^3 timesteps, to skip the initial transient behaviour.
3. Write to file when the last section visited matches the current counter, that is, write iff `last_sect` $=$ `temp_bm`.
4. If the above condition is not met, do nothing. If it is met, write to file and advance counter (mod 5), `temp_bm` $+= 1$.
5. Go back to 3.