Radio Frequency Montecarlo Library for orbit following codes

Yuri Sánchez Alves
Barcelona Supercomputing Center
(Dated: August 2024)

I. INTRODUCTION

The aim of this work is to provide an insight on the content and functioning of the RFOF library [1]. The purpose of the library is to handle the interaction between charged particles and RF-wave fields in guiding centre orbit following Monte Carlo codes. As such, this library does not provide any output file, since it is not a code on its own, but rather it provides the key electromagnetic parameters of the charged particles to an external EM wave solver code, such as AORSA.

The work on this library has comprised a few weeks, and an effort to be able to run the code without any kinds of issues has been made. However, no relevant results have been obtained from this library, and no standalone working version has been produced. Despite this, several improvements have been made to the code, and a better understanding has been achieved.

In order to get the library to work and couple it to a EM solver code, more efforts should be dedicated to it. This brief article aims to serve as a guide for whomever might want to work with the RFOF library in the future, to provide an improved version and an explanation on the modifications.

II. FILES

All the necessary files for compilation and execution in the library can be found in the github link provided [2].

The executable file is given by the *rftest.F90* file, which is simply a call to the run_dumorb subroutine in the *dummy_orbit.F90* file. The run_dumorb traces a dummy orbit, given by the parallel velocity. Between each orbit-step the RFOF operator is called to check if the resonance has been reached.

The run_dumorb subroutine calls the subroutine init_dumorb, which is the responsible of reading the electromagnetic parameter values from the input and create the electromagnetic field and initialise the markers. The local magnetic field is set in this subroutine as well as the estimate for the time of a poloidal orbit tauBounce.

Once the dummy orbit has been initialised, it is time to find its resonances. Now, for each time step, the step_orbit_dumorb is called. This subroutine calculates the new magnetic field and updates the markers.

The markers are set in the file *RFOF_markers*. The data in the markers are stored as pointers to the internal markers. These markers should be accessed by the EM solver code (AORSA for instance), so that both the EM solver and the RFOF library access the same memory and can exchange data. However, as it will be discussed later, the inadequate use of RFOF_markers seems to be responsible of memory access issues.
There is an extensive set of files that are used in this process.

- *RFOF_constants.F90* This file contains the definition of the physical constants such as masses or conversion factors.
- *RFOF_diagnostics.F90* This file comprises a set of routines that are responsible for the reading of data from the plasma such as the energy or the momentum.
- *RFOF_Efield_update.F90* This file recalculates the electric field in each time step performed.
- *RFOF_Kick.F90* This file is called in the *RFOF_main.F90* file, when called proceeds to do a Montecarlo kick to the charged particles. After the kick, the markers and the diagnostics are updated.
- *RFOF_local_magnetic_field.F90* This file reads the condition on whether the particles should be pushed through both real space and velocity space. If both of them are wanted, the subroutine local_magnetic_field_interface_to_RFOF shall be used. This file is responsible of storing in the magnetic field

parameters, the corresponding markers. This file is where the issue with the markers memory accessing is noticed. More on this problem is discussed later.

- *RFOF_master* This file is the responsible of adressing the resonance condition. If the particle is in the resonance condition, the Montecarlo kick is given to it.
- *RFOF_numerics* This file has the goal of performing polynomial evaluations, calculate Bessel functions and numerical derivatives.
- *RFOF_parameters* This file reads variable values from the input files and assigns them to the corresponding RF parameters.

There are several more remaining files that are used but are of lesser importance to the understanding of the library.

III. ISSUES ENCOUNTERED

A series of problems came up when trying to compile together the files, and others arose during the execution of the rftest. These issues are listed below in the order they appeared.

- The file *euitm_schemas.F90* was not existent. This file is where the type waves is supposed to be defined. Also, this file needed the *coherentwave.F90*, which in turn needed the *Fullwave.F90* and *GlobalParam.F90*. The former required the *Local.F90*.

  The file *GlobalParam.F90*, needed to contain the definition of the following parameters of type global:

  - `real, allocatable :: ntor(:)`
  - `real, allocatable :: p_frac_ntor(:)`
  - `real, allocatable :: pow_i(:)`
  - `real, allocatable :: pow_ntor_i(:,:)`
  - `real, allocatable :: pow_ntor_e(:)`
  - `real, allocatable :: cur_tor_ntor(:)`
  - `real :: power_tot`
  - `real :: pow_e`
  - `real :: cur_tor`
  - `real :: frequency`
  - `character :: name`
  - `character :: type`

  The file *Local.F90*, needed to contain the definition of the following parameters of type local:

  - `real, allocatable :: e_plus(:,:,:)`
  - `real, allocatable :: e_minus(:,:,:)`
  - `real, allocatable :: e_para(:,:,:)`
  - `real, allocatable :: e_plus_ph(:,:,:)`
  - `real, allocatable :: e_minus_ph(:,:,:)`
  - `real, allocatable :: e_para_ph(:,:,:)`

  These parameters are needed by RFOF_waves, but are not defined in the source repository, so the files were created and the variables defined.

- A non-existent input file "input.rfof" was expected to be used in the following files:
  - dummy_orbit.F90
  - euitm_waves_interface.F90
  - RFOF_diagnostics.F90
  - RFOF_parameters.F90
  - RFOF_waves.F90

  The input file consists of the following sections, each providing different input parameters. When the file input.rfof is opened in any of the files, only some of these input sections are read by that file.
  - input_control
  - input_magnetifield
  - input_marker
  - input_wavefields
  - input_resonance_memory
  - IO_control
  - simplify_rfof
  - rz_boundingbox

  After observing which variables were requested by the library files to be read from a provided input file, the file *input.rfof* was created. It is provided in this document as an annex.

- Memory issues appeared in the file *euitm_waves_interface.F90*, as the allocation of memory for the types was not properly done.

  As an example,

```
allocate( waves%coherentwave(jfreq)%fullwave%local%e_plus(nnphi,max_npsi,max_ntheta))
```

  was written without being preceded by previous allocations.

```
allocate(waves%coherentwave(jfreq)%fullwave)
allocate(waves%coherentwave(jfreq)%fullwave%local)
```

  When these were introduced, the issue was fixed.

- Now, the config.h file was not provided either. The headers that were supposed to exist were created and afterwards there were no issues in compilation. However depending on the definition of one *USE*, the program would present issues during execution. If **USE_MAGNETIC_FIELD_INTERFACE** was defined in the config.h file, the programme would run. However, if it was not defined the programme would encounter memory issues. The use of this header is explained by the author of the code [1]. " In case you wish to have an RF-operator that pushes particles in real space as well as in velocity space, then you also need to write a subroutine to provide RFOF with your magnetic field data. The subroutine should be called local_magnetic_field_interface_to_RFOF. "

  In order to understand where this issue is caused, one should look at the *RFOF_local_magnetic_field.F90*. If the **USE_MAGNETIC_FIELD_INTERFACE** is defined, the subroutine local_magnetic_field_interface_to_RFOF will be accessed. If that header is not defined, the parameters of the local magnetic field, will be read from the input (R, phi and z) or from the markers. These condition on the header is given by the #ifdef statements found in *RFOF_local_magnetic_field.F90*.

- The memory issues mentioned occured in the *dummy_orbit.F90* file, in the subroutine run_dumorb. When the init_dumorb subroutine is called, the *RFOF_local_magnetic_field.F90* is accessed and the issue is encountered when Blocal%psi = marker%psi, because marker%psi is not initialized. However, if instead, we either assign an arbitrary value directly to Blocal%psi or define the psi in the input file and generate the corresponding marker , the problem is removed from the init_dumorb and another one appears later when the subroutine step_orbit_dumorb is called. This occurs when trying to read the marker%mass, which is properly initialized when accessed by the init_dumorb, but is not initialized when accessed by the step_orbit_dumorb. This is what Is being currently worked on understanding.

  The init_dumorb uses the markers as well but after defining psi does not create any kind of trouble. This routine calls the make_markers and set_marker_pointers, whereas the subroutine step_orbit_dumorb calls the update_marker routine. This means that some memory is not properly allocated in this step in the step_orbit_dumorb, but it is not clear how.

IV. FINAL COMMENTS

In order to summarise this document, notable changes have been to the library, mainly creating files that were not available in the original repository. The current situation is that the *RFOF_markers.F90* should be understood better, since the segmentation faults occur due to uninitialised markers as pointers.

With regards to the coupling to the EM solver code AORSA, the file aorsa3din_mod.f will have its data modified by the RFOF library, such as the electron density, central electron temperature, magnetic field at axis. The coupling has to be carefully performed using the markers in RFOF_markers.

V. REFERENCES

[1] https://portal.eufus.eu/documentation/ITM/doxygen/imp5/rfof/docs/index.html
[2] https://github.com/yuri-BSC/RFOF_LIBRARY

Appendix A: Appendix

```
&input_control
    dt = 1e-4,
    NtimeSteps = 10
/

&input_magnetifield
    R0 = 5.5
    aminor = 0.53
    B0 = 2.6
    q = 1
/

! Inventat
&input_marker
    weight = 1
    R = 5.5
    phi = 43.4
    z = 0.0
    mass = 1   ! the mass is given en amu, then in RFOF_markers it transforms to kg
    charge = 1.0
    E = 5.0e3
    xi = 0.0
/

&input_wavefields
  nfreq = 1,
  nnphi = 1,
  RFpower = 1.0,
  EfieldNormalisation = 0.5,
  freq = 30e6,
  nphi = 12,
  kperp = 0.1
/


&input_resonance_memory
    nStoreTimes = 10
/

&IO_control
    start_time_event_output = 0e-4
    output__2D_RZ_out = .FALSE.
    output__Orbit = .FALSE.
    MAX_number_of_points_stored_in_the_Orbit = 10000
    output__rf_kicks = .TRUE.
    MAX_number_of_points_stored_in_the_rf_kick = 10000
    output__resonace_predictions = .FALSE.
    MAX_number_of_points_stored_in_the_resonance_memory = 10000
/

&simplify_rfof
    simplify__static_resonance_position_during_RF_kick = .TRUE.
    simplify__drift_velocity_does_not_affect_resonance_condition = .TRUE.
    simplify__parallel_velocity_does_not_affect_resonance_condition = .TRUE.
    simplify__assume_zero_larmor_radius_in_KPERPxRHO = .FALSE.
    simplify__kpar_is_nphi_over_R = .TRUE.
```

```
    width_of_rf_resonance_layer = 1.0d-2
/


&rz_boundingbox
    Rmin = 0.1d0
    Rmax = 20.0d0
    Zmin = -20.0d0
    Zmax = 20.0d0
/
```