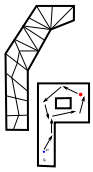


fpakc

fpakc User Manual

Jorge Gonzalez

July 11, 2024
v0.1

Contents

Chapter 1

Introduction

1.1 About fpakc

The Finite element PArticle Code (fpakc) is a simulation tool that models species in plasma (ions, electrons and neutrals) following the trajectories of macro-particles as they move and interact between them and the boundaries of the domain. Particles properties are scattered into a finite element mesh in 1, 2 or three dimensions, with the possibility to choose different geometries. The official repository can be found at: `https://gitlab.com/JorgeGonz/fpakc.git`. The code is currently in the very early steps of development and further refinements are expected very soon.

1.2 Main Guidelines

The fpakc environment aims to be a fully functional tool for the simulation of plasmas from a kinetic point of view. The main guidelines in the creation of the code are:

1. The code is Open Source and freely distributed. This means the code can be used and modified by anyone. Nevertheless, the official version distributed in the official repository will be managed by the developed team, which members have to be given direct permission by the lead developer. fpakc is distributed with a GNU General Public License v3.0 that covers all files in the repository. Moreover, there should always be an open-source free alternative for external tools required by the program (post-processing, mesh generation, input file generation ...).

2. fpakc is coded in a *understandable* way. This means that the code is required to be written in a clear way that is easy to understand and maintain. Variables and procedure names need to be self-understanding. This eases the process of fixing bugs and improving the codes by a large team of developers. For more information, please refer to the fpakc Coding Style document.

3. fpakc requires being ease to use. Input files are required to be in a *human* format, meaning that the different options can be easily understood without constant reference to the user guide. fpakc is aimed to be used in a wide range of applications and by various scientists: from well-established ones to newcomers to the field and also students.

These are foundation stones of fpakc and its development and should always be followed, at least for the releases in the official repository.

1.3 How to collaborate

Right now, development of fpakc is closed to third parties until a stable version with the basic functionality is released. However, if you have a huge interest in the project please contact jorge.gonzalez@upm.com.

Chapter 2

Operation Scheme

2.1 The Particle Method

Fpakc uses macro-particles to simulate the dynamics of different plasma species (mainly ions, electrons and neutrals). These macro-particles could represent a large amount of real particles. For now own, macro-particles will be referred as just particles by abuse of language. During the initiation phase, the input and mesh file(s) are reading. If an initial distribution for a species is specified in the input file, particles to match that distribution are loaded into the cells.

The general steps performed in each iteration are:

1. Firstly, new particles are introduced into the domain as specified in the input file.

2. Particles are then pushed, accounting for possible acceleration by external forces. During this process, if a particle changes cell, it is found using the connectivity between elements. If a particle encounters a boundary instead a new cell, the interaction between the boundary and the wall are computed. A particle may abandon the computational domain and is no longer accounted for.

3. Next, collisions for the particles inside each cell are carried out. This may include different collision processes for each particle. Monte-Carlo collisions (elastic, ionization, charge-exchange...) can be carried out in a specific mesh, to better adjust to the cell size required, similar to the mean-free path. Although not yet implemented, Coulomb scattering will always be performed in the mesh used for scattering, which cell size should be in the order of the Debye length.

4. A new array containing all particles inside the numerical domain is obtained.

5. Finally, particle properties are scattered among the mesh nodes. These properties are density, momentum and the stress tensor.

6. If requested, the electromagnetic field is computed.

7. If the number of iteration requires writing output files, it is done after all steps for the particles are completed.

Fpakc has the capability to configure all the behaviour of the simulation via the input file. Parameters as injection, the kind of pusher used for each species, boundary conditions or collisions are user-input parameters and will be described in Chap. 4.

2.2 Injection of new particles

fpakc has the capability of injecting particles at different velocities, temperatures, distributions and mass flows in boundary sections defined by the user. Particles are distributed uniformly along the edge. Their velocities are computed from the distribution function selected by the user.

The injection of particles is controlled in the moduleInject module.

2.3 Pushing

Particles are pushed in the selected domain. Velocity and position are updated according to the old particle values and the external forces. All the push routines for the different geometries can be found in moduleSolver. The pushers included in Fpakc are:

- 3D Cartesian pusher. Moving particles in a simple 3D Cartesian space.
- 2D Cylindrical. When a 2D cylindrical geometry is used (z, r), a Boris solver[1] is used to move particles accounting for the effect of the symmetry axis. This pusher removes the issue with particles going to infinite velocity when $r \to 0$ by pushing the particles in Cartesian space and then converting it to $r - z$ geometry. Velocity in the θ direction is updated for collision processes, although the dynamic in the angular direction is assumed as symmetric.
- 2D Cartesian pusher. Moving particles in a simple 2D Cartesian space.
- 1D Radial pusher. Same implementation as 2D cylindrical pusher but direction z is ignored.
- 1D Cartesian pusher. Moving particles in a simple 1D Cartesian space. Same implementation as in 2D Cartesian but y direction is ignored.

2.4 Find new cell

Once the position and velocity of the particle are updated, the new cell that contains the particle is searched. This is done by a neighbour search, starting from the previous cell containing the particle. In the process of finding the new cell, a particle might encounter a boundary. When the particle interacts with the boundary, the particle may continue its life in the simulation or might be eliminated from it. Once that the new cell is found or that the particle life has been terminated, the pushing is complete. If a secondary mesh is used for the Monte-Carlo Collision method, the new cell in that mesh in which the particle reside is also found by the same method, although no interaction with the boundaries is accounted for this step.

2.5 Variable Weighting Scheme

One of the issues in particle simulations, specially for axial-symmetrical cases, is that due to the disparate volume of cells, specially close to the axis, the statistics in some cells is usually poor. To try to fix that, the possibility to include a Variable Weighting Scheme in the simulations is available in Fpakc. These schemes detect when a particle changes cells and split it if necessary to improve statistics. The use of a Variable Weighting Scheme is defined by the user in the input file.

Beware that this can increase the number of particles in the simulation and increase computational time.

2.6 Interaction between species

For each cell, interaction among the particles in it are carried out. Fpakc distinguish between two types of interactions: Monte-Carlo Collisions (MCC) and Coulomb Scattering (CS). MCC refers to the process in which two particles interact in short range. These processes include, but are not limited to: elastic collisions, ionization/recombination, charge-exchange, excitation/de-excitation... A secondary mesh, with cell sizes in the range of the mean-free path, can be used for this type of collision. CS refers to the large range interaction that a charged species suffer do to the charge of other particles. The interactions between the different species is defined by the user.

2.6.1 Monte-Carlo Collisions

For each cell, the maximum number of collisions between particle is computed. For each collision, a random pair of particles is chosen. A loop over all possible collisions for the pair of particles chosen is performed. If a random number is above the probability of collision for that specific type, the collision takes place. If not, the next type for the particle pair is checked.

Below are described the type of collision process implemented in fpakc:

- Elastic. In this type of collision, particles exchange energy due to hard-sphere model. Total energy is conserved. Resulting velocity directions are chosen from Maxwellian distribution functions. This interaction is useful for short-range collisions as neutral-neutral and charged-neutral elastic collisions.
- Charge Exchange. When an ion interacts with a neutral particle, an electron is exchanged between the two particles with no exchange of energy. This is called a resonant charge-exchange.
- Electron Impact Ionization. When the relative energy between a neutral and an electron is above the ionization threshold, there is a probability that the neutral particle will become ionized. This ionization emits and additional electron
- Recombination. When an electron and an ion interact, there is a possibility for them to be recombined into a neutral particle. The photons emitted by this process are not modelled yet.

2.6.2 Coulomb Scattering

A simple linearization of the Coulomb operator based on Ref. [9] is implemented. This method assumes that the species involved in the scattering process have a Maxwellian distribution. The method is made to conserve momentum and kinetic energy based on the approach in Ref. [6] for self (same species) and intra (different species) collisions.

The user must specify the charged species that will interact together. The Coulomb logarithm involved in these processes is currently set to a fix value of 10.

This method is not valid for situations in which the distribution functions are far from Maxwellian.

2.7 Reset of particle array

Once that the pushing is complete, the array of particles that remain inside the domain is copied to a new array. The new array containing only the particles inside the domain will be the one used in the next steps. In this section, particles are assigned to the list of particles inside each individual cell. Unfortunately, this is done right now without parallelization and is very CPU consuming.

2.8 Probing

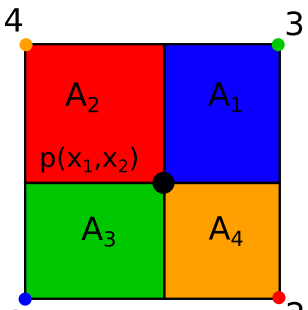
As default, fpakc outputs information of macroscopic quantities (density, velocity, temperature...) in the finite element mesh. However, a lot of information can be extracted from the particle distribution function. Thus, a probing method is provided to extract the distribution function in a specific position.

The particles inside a cell in which the input position is located are distributed into a 3D velocity grid. The user can decide the grid width and the number of points in each direction. The distribution function will be calculated and wrote with a time step decided by the user.

If a particle velocity resides outside the velocity grid (in any direction), it will not be added to the tally of the distribution function. Due to the limitation of only considering particles in the cell, and not neighbour particles, two probes for the same species at different positions but in the same cell will output the same results. A more advance method considering distance between the particles and the probe position as well as particles in neighbour cells could be implemented to improve the statistics of the distribution function.

2.9 Scattering

The properties of each particle are deposited in the nodes from the containing cell. This process depends on the cell type, but in general, each node receives a proportional part of the particle properties as a function of the particle position inside the cell. The figure 2.1 shows how a particle at a generic position $p(x_1, x_2)$ inside the cell is scattered to the four nodes. Each node receives a proportional part of the area formed by dividing the cell in for rectangles, using as an additional vertex the particle position. These properties are dimensionless, but they are converted to the correct units once the output is printed.



2.10 Electromagnetic field

WIP.

2.11 Average scheme

Particle-in-cell codes have an intrinsic statistical noise associated with them. Although this can be reduced by increasing the number of particles, this also increases the CPU requirements of the case.

It is quite common that most cases reach a quasi-steady state after a number of iterations and time-average results can be obtained after to improve analysis, plotting and restarting the case using these time-average results as new species backgrounds. Although this is possible to do once the simulation is finished with post-processing tools, this is limited to the number of iterations printed.

Fpakc implements a simple average scheme that, after a start time provided by the user, scores a mean and standard deviation of all the main species properties, and the electromagnetic field. This scheme is based on the Welford's online algorithm [10]. The averaged data is written in the same format as the input mesh at the end of the simulation.

Chapter 3

Installation

3.1 Required Packages

To properly compile fpakc, the following packages are required.

3.1.1 Gfortran

The Open Source free compiler GFortran[8] from GCC is the basic way to compile fpakc. It is distributed with all GNU/Linux distributions.

3.1.2 Ifort

The Ifort[3] compiler is a proprietary Fortran compiler developed by Intel®. The makefile distributed with Fpakc is prepared to be used with ifort with minor modifications. However, GFortran is recommended for compiling fpakc.

3.1.3 OpenBlas

OpenBLAS[7] is used by fpakc to solve the electromagnetic field in the finite element mesh.

3.1.4 JSON-Fortran

To read JSON[5] input files in Fortran, the API JSON-Fortran[4] is used. This needs to be compiled and placed in a folder accessible from the root directory of Fpakc. The same compiler as the one used to compile Fpakc needs to be used to generate a compatible library.

3.1.5 Gmsh

Although Gmsh[2] is not required to compile and run Fpakc, it is the default tool to generate finite element meshes and post-processing. Right now, the only I/O format available in Fpakc is the v2.0 .msh format.

3.2 Installation steps

Once you have been added to the Fpakc GitLab repository, and have added an ssh key to your account, you will be able to clone the repository using:

```
git clone git@gitlab.com:<YourGitLabUsername>/fpakc.git
```

in which you have to substitute <YourGitLabUsername> for your GitLab username.

Go into the new downloaded directory with

```
cd fpakc
```

Clone the json-fortran repository with:

```
git clone https://github.com/jacobwilliams/json-fortran.git
```

This will create the json-fortran-8.2.0 directory. Go into this new folder and create a new build-gfortran directory, enter it and execute:

```
cmake ../
```

Once it finish, just compile the json-fortran library with

```
make
```

This creates the include and lib folders that contain the required files for fpakc compilation.

Go back to the fpakc root folder and execute

```
make
```

to compile the code. If everything is correct, an executable named *fpakc* will be generated.

3.3 Running the code

To run a case, simply execute:

```
./fpakc path/to/input-file.json
```

in a command line from the root fpakc folder. Substitute path/to/input-file.json with the path to the input file of the case you want to run. The examples in the run directory are presented in Chapter 5.

Chapter 4

Input File

The input files for Fpakc is divided between to files: a mesh file and a case file. The mesh file contains the descriptions for the different elements composing a mesh (nodes, edges and volumes). The case file contains the required information to define a simulation case: reference parameters, initial state, injection of particles, boundary conditions, species, number of iterations...

4.1 Mesh file

Fpakc accepts right now the version 2.0 of Gmsh mesh format .msh in ASCII format. This file contains information about the nodes, edges and volumes that define the finite element mesh used by Fpakc to scatter particle properties and compute the self-consistent electromagnetic field.

4.2 Case file

The required format for the case file is JavaScript Object Notation (JSON). JSON is a case-sensitive format, so input must be written with the correct capitalization. The basic structure and options available for the case file are explained below. The order of the objects and variables is irrelevant, but the structure needs to be maintained.

4.2.1 output

The object **output** in the JSON case file determines the different options to define how and when the simulation produces files: The available options are:

- **path**: Character. Path for the output files. This path is also used to locate the mesh input file.
- **folder**: Character. Base name of the folder in which output files are placed. The date and time is appended to this name. If none is provided, only the date and time is written as the folder name.
- **triggerOutput**: Integer. Determines the number of iterations between writing output files for macroscopic quantities.
- **cpuTime**: Logical. Determines if the Central Processing Unit (CPU) time per iteration is written into a file. Each row in the file determine the iteration, number of particles in the simulation and the multiple times spend per action. Each action in the iteration (pushing, collisions, weighting...) has its own column.
- **triggerCPUTime**: Integer. Determines the number of iterations between writing CPU time for macroscopic quantities. This option is irrelevant if **cpuTime** is set to *false*.
- **numColl**: Logical. Determines if the number of collisions per cell are outputted in a separated file. Trigger between writings is the same as in **triggerOutput**.
- **EMField**: Logical. Determines if the electromagnetic field is printed.
- **probes**: Array of objects. Defines the probes employed for obtaining the distribution function at specific positions. See Sec. 2.8 for more information. The object is structured as follows:

  - **species**: Character. Species name as defined in **species** array.
  - **position**: Real. Array of dimension 3. Units in m. Indicates the position of the probing.
  - **timeStep**: Real Units in s. Optional. Time step for output of the distribution function. If none is provided, the minimum time step of the case is used.
  - **velocity_1, velocity_2, velocity_3**: Real. Array of dimension 2. Velocity range (minimum-maximum) in which the distribution function will be interpolated. The subscripts 1, 2, 3 indicate the three directions of the case.
  - **points**: Integer. Array of dimension 3. Number of points in each direction.

4.2.2 geometry

The object **geometry** contains information about the type of geometry, the mesh file format and the mesh filename. The accepted parameters are:

- **dimension**: Integer. Number of spatial dimensions of the geometry. Current values are: 0, 1, 2 or 3. Zero dimension is a fictitious volume. Geometry used mostly to test collisional effects. No boundary or EM field is solved. No injection can be implemented. Initial state must be read from file. No mesh file is required. The optional argument **geometry.volume** can be used to set a value for the fictitious volume. Otherwise, the volume is set to 1 in non-dimensional units.
- **type**: Character. Type of geometry. Current accepted vaules are
  - **Cart**: Cartesian coordinates. Available for **geometry.dimension** 1, 2 and 3. The coordinates x, y and z correspond to x, y and z respectively.
  - **Cyl**: Cylindrical coordinates $(z - r)$ with symmetry axis at $r = 0$. Only available for **geometry.dimension** 2. The coordinates x and y correspond to z and r respectively.
  - **Rad**: One-dimensional radial space (r). Only available for **geometry.dimension** 1. The coordinates x corresponds to r.
- **meshType**: Character. Format of mesh file. The output will be written in the same format as specified here. Accepted formats are:
  - **gmsh2**: Gmsh file format in version 2.0. This has to be in ASCII format.
  - **vtu**: Visualization Toolkit for unstructured grids (VTU) file format. This has to be in ASCII format.
- **meshFile**: Character. Mesh filename. This file is searched in the path **output.path** and must contain the file extension.
- **volume**: Real. Units of m^{-3}. Used to set a fictitious volume for the 0 dimension. Ignored in the other cases.

4.2.3 species

The array object **species** contains the data relevant to identify the different species in the simulation. Multiple species can be defined as elements in the array. For each species, the following parameters need to be defined:

- **name**: Character. Name of the species.
- **type**: Character. Defines the type of species. Current values are:
  - **neutral**: Neutral species.
  - **charged**: Charged species. The parameter **charge** is required for this type of species.
- **mass**: Real. Particle mass in kg.
- **charge**: Real. Particle charge in elementary charge units. This parameter is only relevant if **type** is **charged**.
- **ion**: Character. Name of species resulting of ionizing the current one.
- **neutral**: Character. Name of species resulting of neutralizing the current one.

4.2.4 boundary

The array object **boundary** determines the interaction between surfaces and particles. These boundaries need to be linked to a specific edge in the mesh. The accepted variables are:

- **name**: Character. Name of the boundary.
- **physicalSurface**: Integer. Identification of the surface in the mesh file.
- **bType**: Array of objects of dimension 'number of species'. Per each species defined in the case, a boundary **type** needs to be provided. Accepted values for **type** are:
  - **reflection**: Elastic reflection of particles.
  - **absorption**: Particle is eliminated from the domain. The particle is first moved into the edge and its properties are scattered among the edge nodes.
  - **transparent**: Particle abandon the numerical domain.
  - **wallTemperature**: Reflective wall with constant temperature that exchange heat with particles. Required parameters are:
    - **temperature**: Real. Units of K. Temperature wall.
    - **specificHeat**: Real. Units of $\mathrm{Jkg^{-1}K^{-1}}$. Specific heat capacity of the material.
  - **ionization**: Per each particle crossing the surface with this type of boundary, a number of ionization events are calculated. A pair of ion-electron is generated for each ionization event, taking as a reference a neutral background. The secondary electron is taken as the same type as the incident particle. The available input is:
    - **neutral**: Object. Information about the neutral background. Required parameters are:
      - **ion**: Character. Species name of the ion generated as defined in object **species**. Required parameter.
      - **mass**: Real. Units in kg. Mass of neutral species. If missing, the mass of the ion is used
      - **density**: Real. Units in $\mathrm{m^{-3}}$. Density of neutral background. Required parameter.
      - **velocity**: Real. Units in $\mathrm{ms^{-1}}$. Array of dimension 3. Mean velocity of neutral background. Required parameter.
      - **temperature**: Real. Units in K. Temperature of neutral background. Required parameter.
    - **effectiveTime**: Real. Units in s. As the particle is no longer simulated once it crossed the boundary, this time represents the effective time in which the particle produces ionization processes in the neutral background. Required parameter.
    - **energyThreashold**: Real. Units in eV. Ionization energy threshold for the simulated process. Required parameter.
    - **crossSection**: Character. Complete path to the cross-section data for the ionization process.
  - **axis**: Identifies the symmetry axis for 2D cylindrical simulations. If , for some reason, a particle interacts with this axis, it is reflected.

4.2.5 boundaryEM

The array object **boundaryEM** determines the boundary conditions for the electromagnetic field. As with the **boundary** definition, these must be linked to an edge identified in the mesh file. The variables for each array element are:

- **name**: Character. Name of the boundary.
- **type**: Character. Type of boundary. Accepted values are:
  - **dirichlet**: Elastic reflection of particles.
- **potential**: Real. Fixed potential for Dirichlet boundary condition.
- **physicalSurface**: Integer. Identification of the edge in the mesh file.

4.2.6 inject

The array **inject** specifies the injection of particles from different surfaces. The injection of particles needs to be associated to a physicalSurface in the mesh file. Multiple injections can be associated to the same surface.

- **name**: Character. Name of the injection.
- **species**: Character. Name of the species that is being injected.
- **flow**: Real. Flow of particles going through the surface.
- **units**: Character. Units for the **flow** parameter. Available values are:
  - **A**: Ampere.
  - **Am2**: Ampere per square meter. This value will be multiplied by the surface of injection.
  - **sccm**: Standard cubic centimetre.
  - **part/s**: Particles (real) per second.
- **v**: Real. Units of ms^{-1}. Module of velocity vector.
- **n**: Real. Array dimension 3. Direction of injection. If no value is provided, the normal of the edge is used as the direction of injection. This vector is normalized to 1 by the code.
- **velDist**: Character. Array dimension 3. Type of distribution function used to obtain injected particle velocity:
  - **Maxwellian**: Maxwellian distribution of temperature **T** and mean **v** times the value of **n** in the specified direction.
  - **Half-Maxwellian**: Half-Maxwellian distribution of temperature **T** and mean **v** times the value of **n** in the specified direction. Only considers the positive part of the half-Maxwellian.
  - **Delta**: Dirac's delta distribution function. All particles are injected with velocity **v** times the value of **n** in the specified direction.
- **T**: Real. Units of K Array dimension 3. Temperature in each direction.
- **physicalSurface**: Integer. Identification of the edge in the mesh file.
- **particlesPerEdge**: Integer. Optional. Number of particles to be injected by each edge in the numerical domain. The weight of the particles for each edge will modified by the surface of the edge to ensure the right flux is injected. If no value is provided, the number of particles to inject per edge will be calculated with the species weight and the surface of the edge respect to the total one.

4.2.7 reference

This object indicates the reference values used by Fpakc to scale the problem variables. The required parameters are:

- **density**: Real. Reference density in $m^{-}3$.
- **mass**: Real. Reference particle mass in kg.
- **temperature**: Real. Reference temperature in K.
- **radius**: Real. Reference atomic radius in m.
- **crossSection**: Real. Reference cross-section in m^2. If this value is present, radius is ignored.

4.2.8 solver

This object determines the input parameters for the solvers used in the case, both for particle pushers and electromagnetic field. Accepted variables are:

- **tau**: Real. Units of s. Array dimension 'number of species'. Defines the different time steps for each species. Even if all time steps are equal, they need to be defined as an array.
- **finalTime**: Real. Units of s. Final simulation time.
- **initialTime**: Real. Units of s. Initial simulation time. If no value is provided, the initial time is set to 0.0 s.
- **pusher**: Character. Array dimension 'number of species'. Indicates the type of pusher used for each species:
  - **Neutral**: Pushes a particle without any external force.
  - **Electrostatic**: Pushes a particle, including the effect of the electrostatic field.
  - **Electromagnetic**: Pushes a particle, accounting for the electromagnetic field.
- **WeightingScheme**: Character. Indicates the variable weighting scheme to be used in the simulation. Check Section 2.5 for more information. If this variable is not present, no scheme is used. The current available schemes are:
  - **Volume**: Modifies particle weight as a function of cell volume.
- **EMSolver**: Character. Determines the solver for the electromagnetic field. If no value is supplied, no field is solved.
  - **Electrostatic**: Solves the Poison equation to obtain the self-consistent electrostatic potential.
  - **ConstantB**: Assumes a constant magnetic field in all the domain. It solves the Poisson equation as in the **solver.EMSolver** option.
- **B**: Real. Units of T. Array of dimension 3. Provides the value of constant magnetic field for the option **solver.EMSolver ConstantB**.
- **initial**: Array of objects. Determines initial values for the species. Required values are:
  - **species**: Character. Name of species as defined in the object **species**.
  - **file**: Character. Output file from previous run used as an initial state for the species. The file format must be the same as in **geometry.meshType** Initial particles are assumed to have a Maxwellian distribution. File must be located at **output.path**.
  - **particlesPerCell**: Integer. Optional. Initial number of particles per cell. If not, the number of particles per cell will be assigned based on the species weight and the cell volume.

4.2.9 average

This object determines the use of an average scheme. If this object exists in the input file, average will be written at the end of the simulation. Acceptable values are:

- **startTime**: Real. Units in s. Simulation physical time in which average scheme will start to compute the mean and standard validation. If no value is provided, the initial time is set to 0.0 s.

4.2.10 interactions

This object determines the different interactions among species. Acceptable values are:

- **folderCollisions**: Character. Indicates the path to in which the cross-section tables are allocated.
- **meshCollisions**: Character. Determines a specific mesh for MCC processes. The file needs to be located in the folder **output.folder**. If this value is not present, the mesh defined in **geometry.meshFile** is used for MCC. The format of this mesh needs to be the same as the one defined in **geometry.meshType**.

- **timeStep**: Real. Units of s. Time step to calculate MCC. If no time is provided, the minimum time step is used.
- **collisions**: Array of objects. Contains the different short range interactions (MCC). Multiple collision types can be defined for each pair of species. Each object in the array is defined by:
  - **species_i**, **species_j**: Character. Define the two species involved in the collision processes. Order is indiferent.
  - **cTypes**: Array of objects. Defines all the collisions between **species_i** and **species_j**. Required values are:
    - **type**: Character. Collision type. Accepted values are **elastic**, **chargeExchange**, **ionization** and **recombination**. Please refer to Sec. 2.6 for a description of the different collision types.
    - **crossSection**: Character. File in **interactions.folderCollisions** that contains the cross-section data as a 1D table of relative energy (in eV) and cross-section (in m^-2).
    - **energyThreshold**: Real. Energy threshold of the collisional process in eV. Only valid for **ionization** and **recombination** processes.
    - **electron**: Character. Name of species designed as electrons. Only valid for **ionization** and **recombination** processes.
    - **electronSecondary**: Character. Optional. Name of species designed as secondary electrons. If none provided, **electron** is used. Only valid for **ionization**.
- **Coulomb**: Array of objects. Contains the information about which species must use the Coulomb linear scattering. This method assumes a Maxwellian distribution for all species involved. Each object in the array is defined by:
  - **species_i**, **species_j**: Character. Define the two species involved in the collision processes. Order is indifferent.

4.2.11 parallel

This object contains the information related to parallelization of Fpakc. Current values accepted are:

- **OpenMP**: Object. Defines the parameters for the OpenMP shared memory parallelization.
  - **nThreads**: Integer. Number of threads to be used by OpenMP. Try to match this to the number of threads of the CPU used.

Chapter 5

Example runs

This chapter presents a description of the different example files distributed with fpakc. All examples in the repository have a README.txt file and a reference output. Plotting of the output is done with Gnuplot or Gmsh.

5.1 1D Emissive Cathode (1D_Cathode)

Emission from a 1D cathode in both, Cartesian and radial coordinates. Both cases insert the same number of electrons from the minimum coordinate and have the same boundary conditions for particles and the electrostatic field. This case is useful to illustrate how fpakc can deal with different geometries by just modifying some parameters in the input file. The same mesh file (mesh.msh) is used for both cases.

5.2 0D Ar-Ar^+ Elastic Collision (0D_Argon)

Test case to check the 0D geometry that includes the elastic collision between Ar and Ar^+. Initial states are read from the Argon_Initial.dat and Argon+_Initial.dat.

5.3 ALPHIE Grid system (ALPHIE_Grid)

Two-dimensional axial-symmetry case to study the counterflow of electrons and Argon ions going through the ALPHIE grid system. A mesh.geo file is provided to easily modify the parameters of the grid system and generate a new mesh with Gmsh.

5.4 Flow around cylinder (cylFlow)

Simple case of neutral Argon flow around a cylinder in a 2D axial-symmetry geometry. Elastic collisions between argon particles are included. Two cases are presented here: one in which the same mesh (meshSingle.msh) for scattering and collisions is used (input.json) and a second one (inputDualMesh.json) in which a mesh is used for scattering (mesh.msh) and a second one is used only for collisions (meshColl.msh).

Glossary

GFortran Open-source compiler for fortran source code. 11

Git Git is a distributed version-control system for tracking changes in a set of files. 21

GitLab GitLab is a web-based lifecycle tool that provides a Git-repository manager. 11

Gmsh A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities.. 11, 13, 14, 19

Gnuplot A portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms.. 19

ifort Intel® Fortran compiler. 11

Open Source Source code that is made freely available for possible modification and redistribution. 5, 11

OpenBLAS Open-source implementation of BLAS and LAPACK APIs. 11

OpenMP Shared-memory parallelization. 18

Acronyms

CPU Central Processing Unit. 13, 18

CS Coulomb Scattering. 3, 8, 9

fpakc Finite element PArticle Code. 5, 7–13, 16, 18, 19

I/O input/output. 11

JSON JavaScript Object Notation. 11, 13

MCC Monte-Carlo Collisions. 3, 8, 17, 18

VTU Visualization Toolkit for unstructured grids. 14

Bibliography

[1] Jay P Boris. "Relativistic plasma simulation-optimization of a hybrid code". In: *Proc. Fourth Conf. Num. Sim. Plasmas*. 1970, pp. 3–67.

[2] Christophe Geuzaine and Jean-FranÃ§ois Remacle. *Gmsh*. https://gmsh.info/.

[3] Intel®. *Intel® Fortran Compiler*. https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/fortran-compiler.html.

[4] *JSON-Fortran*. https://github.com/jacobwilliams/json-fortran.

[5] *JSON, JavaScript Object Notation*. https://www.json.org/json-en.html.

[6] Don S Lemons et al. "Small-angle Coulomb collision model for particle-in-cell simulations". In: *Journal of Computational Physics* 228.5 (2009), pp. 1391–1403.

[7] *OpenBLAS, an optimized BLAS library*. https://www.openblas.net/.

[8] GNU Project. *gfortran - the GNU Fortran compiler*. https://gcc.gnu.org/wiki/GFortran.

[9] Mark Sherlock. "A Monte-Carlo method for Coulomb collisions in hybrid plasma models". In: *Journal of Computational Physics* 227.4 (2008), pp. 2286–2292.

[10] BP Welford. "Note on a method for calculating corrected sums of squares and products". In: *Technometrics* 4.3 (1962), pp. 419–420.