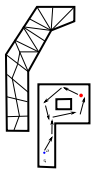

fpakc

fpakc User Manual

Jorge Gonzalez

December 15, 2020
v0.1

Contents

1 Introduction **3**
- 1.1 About fpakc . . . 3
- 1.2 Main Guidelines . . . 3
- 1.3 How to collaborate . . . 4

2 Operation Scheme **5**
- 2.1 The Particle Method . . . 5
- 2.2 Injection of new particles . . . 5
- 2.3 Pushing . . . 5
  - 2.3.1 2D Cylindrical . . . 6
  - 2.3.2 1D Cartesian pusher . . . 6
  - 2.3.3 1D Radial pusher . . . 6
- 2.4 Find new cell . . . 6
- 2.5 Variable Weighting Scheme . . . 6
- 2.6 Reset of particle array . . . 6
- 2.7 Interaction between species . . . 6
- 2.8 Scattering . . . 7
- 2.9 Electromagnetic field . . . 7

3 Installation **8**
- 3.1 Required Packages . . . 8
  - 3.1.1 Gfortran . . . 8
  - 3.1.2 Ifort . . . 8
  - 3.1.3 OpenBlas . . . 8
  - 3.1.4 JSON-Fortran . . . 8
  - 3.1.5 Gmsh . . . 8
- 3.2 Cloning the repository . . . 9
- 3.3 Compiling . . . 9
- 3.4 Running the code . . . 9

4 Input File **10**
- 4.1 Mesh file . . . 10
- 4.2 Case file . . . 10
  - 4.2.1 output . . . 10
  - 4.2.2 geometry . . . 11
  - 4.2.3 species . . . 11
  - 4.2.4 boundary . . . 12
  - 4.2.5 boundaryEM . . . 12

4.2.6 inject 12
4.2.7 reference 13
4.2.8 case 13
4.2.9 parallel 14

5 Example runs **15**
5.1 1D Cathode 15
5.2 ALPHIE Grid system 15
5.3 Flow around cylinder 15

Glossary **16**

Acronyms **17**

Chapter 1

Introduction

1.1 About fpakc

The Finite Element PArticle Code (fpakc) is a simulation tool that models species in plasmas (ions, electrons and neutrals) following the trajectories of macro-particles as they move and interact between them and the boundaries of the domain. The code is currently in very early steps of development and further improvements 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 ease 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 to be ease to use. Input files are required to be in a *human* format, meaning that the different options can be easily understander without constant reference to the user guide. fpakc is aimed to be used in a wide range of applications and by a variety of scientist: from very established ones to newcomers to the field and students.

These are foundation stones of the code and code 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 represent a large amount of real particles. For now own, macro-particles will be referred as just particles by abusing of language. In the evolution of these particles, external forces (as the electromagnetic field), interaction between particles (as collisions) and interaction with the boundaries of the domain are included.

At each time step, particles are first pushed accounting for possible acceleration by external forces. Then, the cell in which the particle ends up is located. If a boundary is encountered, the interaction between the particle and the boundary is calculated. Next, collisions for the particles in each cell are carried on. This may include different collision processes for each particle. Finally, the particles properties are scattered into the mesh nodes. These properties are density, momentum and the stress tensor. Non-dimensional units are used for this, but output files are converted into dimensional units. If requested, the electromagnetic field is computed.

More in depth explanation of the different steps are given in the following sections.

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.

2.3.1 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.

Cross-sections are read from files. These files contains two columns: one for the relative energy (in eV) and another one for the cross-section (in m^{-2}).

2.3.2 1D Cartesian pusher

2.3.3 1D Radial pusher

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 neighbor search, starting from the previous cell containing the particle. In the process of finding the new cell, it is possible that a particle encounters a boundary. When the particle interacts with the boundary, the particle may continue its life in the simulation (reflected) or might be eliminated from it (absorbed). Once that the new cell is found or that the particle life has been terminated, the pushing is complete.

2.5 Variable Weighting Scheme

One of the issues in particle simulations, specially for axisymmetrical 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. This schemes detect when a particle change cells and modify its weight accordingly. To avoid particles having a larger weight than the rest, particle can be split in multiple particles if weight become too large.

2.6 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 parallelisation and is very CPU consuming.

2.7 Interaction between species

For each cell, interaction among the particles in it are carried out. The type of interaction between the different particles is defined by the user. In general, the maximum number of interaction in a cell is computed. For each collision,

a pair of particles is selected. A loop over all possible collisions for the pair of particles is performed. If a random number generated is above the probability of collision for the type divided by the maximum one, the collision take place.

Collisions can change the velocity of the particles involved (elastic), create new particles (ionization-recombination) or change the type of particle (charge-exchange). Right now, only elastic collisions are implemented.

2.8 Scattering

The properties of each particle are deposited in the nodes from the containing cell. This process depend on the cell type, but in general, each node receive a proportional part of the particle properties as a function of the particle position inside the cell. 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.

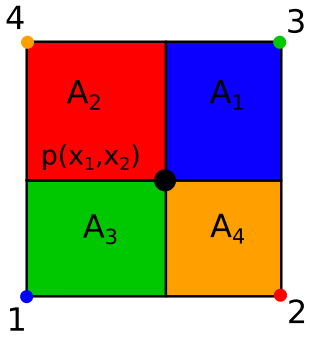

Figure 2.1: Example of how a particle is weighted in a quadrilateral cell.

2.9 Electromagnetic field

WIP.

Chapter 3

Installation

3.1 Required Packages

In order to properly compile fpakc, the following packages are required.

3.1.1 Gfortran

The Open Source free compiler GFortran[7] 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[6] 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 Cloning the repository

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.

3.3 Compiling

To compile the code, just execute make in the fpakc root directory. If everything is correct, an executable named *fpakc* will be generated.

3.4 Running the code

To run a case, simply execute:

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

in a command line. 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 capitalisation. 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.
- **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.

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:

- **type**: Character. Type of geometry. Current accepted vaules are
  - **2DCyl**: Two-dimensional grid (z-r) with symmetry axis at $r = 0$. For Gmsh mesh format, the coordinates x and y correspond to z and r respectively.
  - **1DCart**: One-dimensional grid (x) in Cartesian coordinates For Gmsh mesh format, the coordinates x corresponds to x.
  - **1DRad**: One-dimensional grid (r) in radial coordinates For Gmsh mesh format, the coordinates x corresponds to r.
- **meshType**: Character. Format of mesh file. Currently, only the value **gmsh** is accepted, which makes reference to Gmsh v2.0 output format.
- **meshFile**: Character. Mesh filename. This file is searched in the path **output.path** and must contain the file extension.

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**.

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.
- **type**: Character. Type of boundary. Accepted values are:
  - **reflection**: Elastic reflection of particles.
  - **absorption**: Particle is eliminated from the domain. **Careful**: This boundary will change in the future to absorb particle. A new **transparent** boundary type will be implemented.
  - **axis**: Identifies the symmetry axis for 2D cylindrical simulations. It has no actual function.
- **physicalSurface**: Integer. Identification of the edge in the mesh file.

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 need 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.
  - **sccm**: Standard cubic centimeter.
- **v**: Real. Module of velocity vector, in $^{m}/_{s}$.
- **n**: Real. Array dimension 3. Direction of injection. Norm of vector must be equal 1.

- **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.
  - **Delta**: Dirac's delta distribution function. All particles are injected with velocity **v** times the value of **n** in the specified direction.
- **T**: Real. Array dimension 3. Temperature in each direction.
- **physicalSurface**: Integer. Identification of the edge in the mesh file.

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.
- **mass**: Real. Reference particle mass.
- **temperature**: Real. Reference temperature.
- **radius**: Real. Reference atomic radius.

4.2.8 case

This object determines the simulation time, time step, pushers, weighting scheme and solver for the electromagnetic field. Accepted variables are:

- **tau**: Real. 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.
- **time**: Real. Total simulation time in s.
- **pusher**: Character. Array dimension 'number of species'. Indicates the type of pusher used for each species:
  - **2DCylNeutral**: Pushes particles in a 2D z-r space without any external force.
  - **2DCylCharged**: Pushes particles in a 2D z-r space including the effect of the electrostatic field.
  - **1DCartCharged**: Pushes particles in a 1D Cartesian space accounting the the electrostatic field.
  - **1DRadCharged**: Pushes particles in a 1D cylindrical space (r) accounting the the electrostatic 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.

4.2.9 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

5.1 1D Cathode

5.2 ALPHIE Grid system

5.3 Flow around cylinder

Glossary

GFortran Open-source compiler for fortran source code. 8

Git Git is a distributed version-control system for tracking changes in a set of files. 16

GitLab GitLab is a web-based lifecycle tool that provides a Git-repository manager. 9

Gmsh A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities.. 8, 10, 11

ifort Intel® Fortran compiler. 8

Open Source Source code that is made freely available for possible modification and redistribution. 3, 8

OpenBLAS Open-source implementation of BLAS and LAPACK APIs. 8

OpenMP Shared-memory parallelization. 14

Acronyms

CPU Central Processing Unit. 10, 11, 14

fpakc Finite Element PArticle Code. 3–6, 8–10, 13, 14

I/O input/output. 8

JSON JavaScript Object Notation. 8, 10

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-c

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

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

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

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