Dusty FARGO-ADSG: a quick user manual

Clément Baruteau

What is Dusty FARGO-ADSG ?

Dusty FARGO-ADSG is an extension of the original 2D grid-based hydrodynamical code FARGO. It models the evolution of the gas and dust of a protoplanetary disc with embedded planets, optional energy equation (AD, originally standing for 'adiabatic') and optional gas self-gravity (SG). Dust can be treated either as Lagrangian particles with a power-law size distribution and/or as a zero/low-pressure fluid with fixed size. We first describe below the new features of Dusty FARGO-ADSG regarding the evolution of the disc gas. We then detail the Lagrangian particles and low-pressure fluid implementations of the disc dust. This quick tour ends up with a short section dedicated to installation, compilation and execution of the code.

New features about the disc gas

Compared to the previous public [1] version of FARGO-ADSG, which dates back from 2007, Dusty FARGO-ADSG features a number of improvements for the gas part, in particular by the inclusion of several possible source terms in the energy equation (β-cooling, thermal cooling with the Bell & Lin opacities, thermal, entropy or radiative diffusions, simplified stellar irradiation...). A bug in the expression of the viscous heating has also been corrected. Disc photo-evaporation due to X-rays has been implemented (like in Rosotti et al. 2013, MNRAS, 430 ; ADS link). The simple disc turbulence model of Baruteau & Lin (2010, ApJ, 709 ; ADS), with the stochastic excitation of wave-like modes, is available. As for boundary conditions, it is now possible to use the so-called wave-killing zones near the grid's inner and outer edges, for which the mass surface density and radial velocity of the gas are damped towards the radial profiles obtained by calculating the pure viscous evolution of the disc on a 1D grid simultaneously to the gas equations solved on the polar grid (the azimuthal velocity of the gas is damped towards its axisymmetric instantaneous profile). It is a very simplified way to model the viscous evolution of a global disc in the absence of planets, yet it can be useful for long-term simulations. Several parameter files can be found in the "in" sub-directory which show how to active or use the aforementioned options.

More details on how the disc gas is modelled in the code can be found in :

- the first Fargo papers – Masset 2000 (A&A Supp., 141 ; ADS) and Masset 2002 (A&A, 387 ; ADS) – for the general equations and the FARGO algorithm,
- Baruteau & Masset 2008a (ApJ, 672 ; ADS) for the gas energy equation, Baruteau & Masset 2008b (ApJ, 678 ; ADS) for the gas self-gravity solver, or in my PhD thesis.

Dust modelled as Lagrangian particles

Solver and general options – When dust is modelled as Lagrangian particles, we treat them as test particles, which means that they feel the gravity of the star (direct and indirect terms), of the planets (if any), of the gas disc (if gas self-gravity is included) and the drag acceleration from the gas (both the Epstein and the Stokes drag laws are implemented with a simple interpolation between both regimes as in Paardekooper 2007 (A&A, 462 ; ADS). However, dust particles do not feel each other (no dust self-gravity, no collisions, no growth nor fragmentation). Dust drag onto the gas (dust feedback) is actually implemented in the present version of the code, but only in a beta version which is valid only for same-sized particles (it will not work for a size distribution). The effects of gas turbulence are modelled by applying stochastic kicks to the orbital radius and azimuth of the particles at each hydrodynamical timestep. Kicks

1. The website hosting the original FARGO code and its various branches, including FARGO-ADSG, is no longer available since Summer 2023.

follow a Gaussian distribution with mean and standard deviation as in Charnoz et al. (2011, ApJ, 737 ; ADS – see their equation 21, where we discard the terms with spatial derivatives of the dust's turbulent diffusion, which are generally small). The coefficient of dust's turbulent diffusion, D, is taken to be $D = \nu(1+4\mathrm{St}^2)/(1+\mathrm{St}^2)^2$, with ν the local kinematic viscosity of the gas, and St the local Stokes number of the dust (Youdin & Lithwick 2007, Icarus, 192 ; ADS). Note that it is possible to discard any of the above forces in the code's parameter file (parameters `DustFeelPlanets`, `DustFeelSG`, `DustFeelDisk`, `DustFeedback` and `DustFeelTurb`). The particles number is set by `NbPart` in the input parameter file.

The equations of motion for the particles are solved with a semi-implicit first-order Euler integrator which is similar to, but different from, that implemented in the code Athena (Zhu et al. 2014, ApJ, 785 ; ADS). Differences between both integrators are found to be marginal, except when the particles friction time is much shorter than the hydrodynamical timestep. When the friction time is much shorter than the hydro timestep, the so-called short-friction time approximation may be used instead of the particles integrator (e.g., Johansen & Klahr 2005, ApJ, 634 ; ADS – see `SFTApprox` in the input parameter file, which can be deactivated, for instance for testing purposes). We have checked that the particles integrator behaves well for dust particles that have a friction time much shorter than the hydro timestep, and gives very similar results to the short-friction time approximation. Note also that the use of a leapfrog integrator (second-order in time) has a minor impact on the results for typical applications of the code (gas/dust simulations over a few thousand orbits at most).

In the calculation of the gas drag acceleration, the gas quantities need to be interpolated at the particles location. By default, a Triangular-Shaped Cloud interpolation scheme is used, but other interpolation schemes are also available : Cloud-In-Cell and Nearest-Grid Point – usually, Cloud-In-Cell provides good enough results. The same interpolation scheme is used when calculating the feedback acceleration of the dust particles on the gas. The interpolation scheme can be changed via the `Interpolation` parameter in the input parameter file. The functions that take care of the particles update are `SemiUpdateDustPositions` and `UpdateDustVelocities` in `src/DustUpdate.c`. The interpolation scheme is implemented in function `interpolation` in `src/Dsys.c`. Note that by default, particles that enter the planets Hill radius are removed from there and put back in the disc at a different location. This avoids a large accumulation of particles immediately near the planets. If you are interested in the particles' behavior near the planets, you can deactivate this feature by setting `RemoveDustFromPlanetsHillRadius` to No in the input parameter file.

Initial conditions – The initial orbital radius of the dust particles is sorted out according to a power-law probability distribution, whose power-law exponent is set by `DustSlope` in the input parameter file (`DustSlope` should be set to `SigmaSlope`-1 if you would like a uniform dust-to-gas surface density ratio initially). The minimum and maximum orbital radii where particles are inserted in the grid are also defined in the input parameter file (`RMinDust` and `RMaxDust`). The initial azimuth of the particles is sorted out randomly between 0 and 2π. Their initial radial velocity is zero[2], and their initial azimuthal velocity accounts for the star's gravity and the disc's self-gravity (if included). This ensures that the only difference in initial azimuthal velocities between gas and dust is due to the gas pressure gradient. The function that takes care of the particles initialization is `InitDustSystem` in the `src/Dsys.c` source file.

Size distribution – The particles have a size distribution such that the number of particles in the size interval $[s, s+ds]$, which we denote by $n(s)ds$, is a power-law function of s. The negative value of the power-law exponent is set by `SizePartSlope` in the input parameter file (it therefore takes positive values). The minimum and maximum sizes of the particles are set in physical units (meters) in the input parameter file (`SizeMinPart` and `SizeMaxPart`). Given that `NbPart` is usually up to about 10^6 to maintain a tractable computational cost, it may not be possible to use a realistic, ISM-like value for `SizePartSlope` of 3.5. This is not a problem for observational predictions of the dust's thermal emission (see Baruteau et al. 2019, MNRAS, 486 ; ADS). Taking `SizePartSlope` = 1 ensures that there is roughly the same particles number per decade of size.

Internal density and code units – The internal density of the particles is set in physical units by the `Rhopart` parameter in the input parameter file (in g cm^{-3}). This implies that the code's units of length and mass need to be specified in the input parameter file. By default, the code's unit of length is 1 Astronomical Unit (AU) and the code's unit of mass is 1 Solar mass (the units of time and of temperature follow by the convention that the gravitational constant and the reduced ideal gas constant are equal to unity in code units ; see function `ComputeCodeUnits` in `src/LowTasks.c`).

2. This could be improved in the future by using the initial gas radial velocity interpolated at the particles position.

The code's units of length and mass can be changed via the parameters `FactorUnitLength` and `FactorUnitMass` in the input parameter file. For example, in `in/template.par`, `FactorUnitLength` is set to 10 so that the code's unit of length is 10 AU (and the grid thus extends from 4 to 22 AU).

Outputs – There are two kinds of outputs the code writes for the dust particles. On the one hand, the code writes `dustsystatX.dat` ascii files, which contain 6 columns : the particles orbital radius, azimuth, radial velocity, azimuthal velocity, Stokes number and physical size in meters (this requires the keyword `WriteDustSystem` set to `Yes` in the input parameter file). The first four columns are in code units. On the other hand, the code may write `dustX.dat` binary files, which contain 2D arrays of the total dust surface density (requires the keyword `WriteDustDensity` set to `Yes` in the input parameter file).

Parallelization – The particles evolution is parallelized with MPI using FARGO's domain decomposition into rings, which means that the particles' position and velocity are communicated from one CPU to another when particles cross the radial boundary between two CPUs. You should be aware that, *on some very rare occasions*, this may cause the code's execution to suddenly 'freeze' if at some point there are too many particles that cross the boundary between two CPUs : the MPI messages to be sent from one CPU to another are too big and cause `MPI_Wait` instructions to fail, with the consequence that the code's execution is put on hold indefinitely. Should this happen, please restart the code by changing the number of CPUs (and check that the `RestartWithNewDust` parameter is set to `No` in the input parameter file). I am well aware of this problem but still have not found a smart way to solve it.

Dust modelled as a low-pressure fluid

Solver and general options – Dust can also be modelled in the code as an additional low-pressure fluid which is coupled to the gas via gas drag. This requires setting `DustFluid` to Yes in the input parameter file. Contrary to the Lagrangian approach, a low-pressure fluid can only simulate dust of a given size (parameter `Sizepart`). Unless the short-friction time approximation is used (by setting `SFTApprox` to Yes), the same momentum equation as for the gas is solved for the dust fluid on the polar grid, with the addition of the gas drag term. In particular, if `DustFeelSG` is set to Yes in the parameter file, the self-gravitating acceleration of the gas is included in the dust's momentum equation (although, just like for the Lagrangian approach, the dust's self-gravity is currently ignored). The viscous term can be included for the dust fluid upon specification of the dust's alpha turbulent viscosity (parameter `DAlphaViscosity`). Alternatively, and probably preferentially, dust turbulence can be modelled as a diffusion term in the dust's continuity equation (see Zhu et al. 2012, ApJ, 755 ; ADS – their equation 8), where, just like for Lagrangian particles, the dust's turbulent diffusivity, D, is taken to be $D = \nu(1+4\mathrm{St}^2)/(1+\mathrm{St}^2)^2$, with ν the local kinematic viscosity of the gas, and St the Stokes number of the dust. This requires the keyword `DustDiffusion` set to `Yes` in the input parameter file.

Internal density and code units – Just like when dust is modelled as Lagrangian particles, the dust's internal density needs to be specified in g cm^{-3}, which implies that the code's units of length and mass also need to be specified in the input parameter file (see the *Internal density and code units* paragraph in the previous subsection).

Initial conditions – Like for the gas, the initial conditions for the dust fluid are related to its surface density (which is set by the parameter `DustToGasDensityRatio`) and its aspect ratio (parameters `DAspectRatio` and `DFlaringIndex`). The latter points to the main caveat of modelling dust as a low-pressure fluid, which is to determine how low the dust's pressure (or sound speed) can be, which is problem- and resolution-dependent (see tests in Zhu et al. 2012, ApJ, 755 ; ADS). As a first guess inspired by many test cases, setting the dust's aspect ratio to $1/10^{\mathrm{th}}$ that of the gas gives good agreement with the Lagrangian approach.

Boundary conditions – Wave-killing zones with the same radial extent as for the gas can be used for the dust fluid, where the density and velocity components are damped towards their instantaneous, axisymmetric profiles. However, we suggest to use instead an open boundary condition for the dust fluid by setting `InnerBoundaryDust` to O in the parameter file.

Outputs – The code will write `dustX.dat` binary files with 2D arrays of the dust surface density and velocity field, just like for the gas.

Installation, compilation and first run

The code's github repository contains 3 sub-directories : `src` (where the source files are), `in` (where the input parameter files are), and `lib` with an archive file to install the FFTW 2.1.5 library required for simulations with gas self-gravity. This library needs not be installed if you don't plan on using gas self-gravity. If you do, enter the `lib` sub-directory and install the library as follows, which assumes that MPI is already installed on your environment :

```
tar zxvf fftw-2.1.5.tar.gz
cd fftw-2.1.5
./configure --enable-mpi --prefix='/home/cbaruteau/dustyfargoadsg/lib/fftw2_1_5'
make
make install
```

where /home/cbaruteau/dustyfargoadsg/lib should be changed to the full path of the dustyfargoadsg/lib directory on your machine. Also, if you use a bash environment, you will need to add the following line to your .bashrc file in your home directory :

export FFTW_PREFIX=/home/cbaruteau/dustyfargoadsg/lib/fftw2_1_5

where again the full path to the dustyfargoadsg/lib/fftw2_1_5 directory should be indicated accordingly. Type `bash` in the command line of your terminal to account for the above changes.

To compile the code's source files, go to the `src` directory. There are three compilation options depending on whether MPI and/or FFTW are installed on your machine :

```
make BUILD=parallelfftw  # with MPI and FFTW
make BUILD=parallel      # with MPI but without FFTW
make BUILD=sequential    # without MPI nor FFTW
```

These compilation options (`BUILD=...`) need only be specified the first time source files are compiled. Any time after you may simply type `make`. Other compilation options can be set by editing the makefile in the `src` directory (options set by default are `-O3`). In particular, in the makefile you can have different compilation options depending on your local environment (laptop, super-computing clusters...). Simply define a global environment variable named FARGO_ARCH (much like FFTW_PREFIX above) and edit the top part of the makefile. If you set for instance FARGO_ARCH to CALMIP, you will see in the makefile that the Intel compilers icc and mpiicc will be used.

Next, go back to the main directory where the executable `fargo` has been written. Before running the code, you need to create the output directory where the simulations results are going to be written (contrary to FARGO3D, the present code does not automatically create the directory upon execution !). The output directory is set by `Outputdir` in the parameter file, so if you're using one of the provided template input files in the `in` directory, simply type `mkdir out1`.

Now, to run your first simulation with Dusty Fargo-ADSG, simply type `./fargo -v in/template.par`. If you have compiled with MPI, you may type instead `mpirun -np 4 ./fargo -vm in/template.par` or any equivalent MPI execution command specific to your MPI environment. Over 4 cpus, it takes about 10 minutes to complete the run with the in/template.par parameter file. Outputs may be visualized with the public python program `fargo2python` located here.

⋆⋆⋆