The M3D-C^1 User's Guide

– VERSION 0.1 –

PPPL M3DC1 Team
RPI SCOREC

April 3, 2024

*************** LICENSE ***************

Copyright (c) 2022 PPPL All rights reserved.

SS: PLEASE FILL OUT THIS PAGE

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Questions to jardin@pppl.gov
Derived from the OSI 3-clause BSD

Contents

1 Obtaining M3D-C1

1.1 License Agreement

M3D-C1 is a code developed with funding from the US Department of Energy, and is intended for open scientific research. If you intend to run M3D-C1, please complete the following steps:

1. Sign the license agreement `https://m3dc1.pppl.gov/M3D-C1_License.pdf` and return to nferraro@pppl.gov.
2. Request access to the code using the form `https://pppl.tiny.us/code-release-form`. This step will involve review by PPPL to clear any potential export control issues.

Please be aware that permission to run M3D-C1 does not carry an implicit agreement to provide technical support for compiling, running, modifying, or interpreting output of, M3D-C1.

1.2 Preinstalled M3D-C1 Executables

For the general user, we recommend using precompiled executables and associated modules for release versions, where available. Installations are presently available on a number of systems, including:

GA Iris:
```
module use /fusion/projects/codes/m3dc1/modules
module load m3dc1/1.14
```
NERSC Perlmutter:
```
module use /global/cfs/projectdirs/mp288/C1/modules/perlmutter
module load m3dc1/1.14-cpu
```
PPPL Cluster:
```
module use /p/m3dc1/modules
module load m3dc1/1.14
```
PU Stellar:
```
module use /projects/M3DC1/modules
module load m3dc1/1.14
```

These modules will modify the user's enviroment variables appropriately to access the M3D-C1 executables, python libraries, and IDL routines.

1.3 Accessing the M3D-C1 Source Code

If you choose to build the code yourself, either to use an unreleased version or to do code development yourself, you will need access to the M3D-C1 code repository. The M3D-C1 source code is located in the Github repository: `PrincetonUniversity/M3DC1`. To get access to this repository, please complete the license form and software access request form as described in section 1.1.

1.4 Makefiles and Dependencies

Some of the build scripts depend on the following environment variables to be set, to specify the location of the M3D-C1 source code and the directory in which to install any compiled executables:

`M3DC1_DIR` should be set to the directory containing the M3D-C1 source code. For example:

```
setenv M3DC1_DIR $HOME/src/M3DC1
```

`M3DC1_INSTALL_DIR` should be set to the directory in which M3D-C1 will be installed. For example:

```
setenv M3DC1_INSTALL_DIR $HOME/M3DC1
```

It is recommended to set these values in your login script.

Makefiles for a number of systems are included in the repository, with filenames `$M3DC1_DIR/unstructured/*.mk`. For most of these systems, environment modules are also included in the repository. These modules will load the appropriate software modules for building on that particular system, and can be loaded using:

NERSC Perlmutter:

```
module use $M3DC1_DIR/modules/perlmutter
module load m3dc1/devel
```

PPPL Cluster:

```
module use $M3DC1_DIR/modules/pppl
module load m3dc1/devel-centos7
```

PU Stellar:

```
module use $M3DC1_DIR/modules/stellar
module load m3dc1/devel
```

It is recommended to place the appropriate `module use` statement in your login script. The makefile should have the name `${M3DC1_ARCH}.mk`. If $M3DC1_ARCH is not defined, it will default to `$HOST`, stripped of any trailing numbers identifying a node index on multinode systems (*e.g.* if `$HOST==''sunfire06''` then `$M3DC1_ARCH` will default to "sunfire").

If you are building M3D-C1 on a system for which no makefile or development module is provided, please refer to the existing makefiles and modules as examples. In general, M3D-C1 requires the following dependencies:

- Compilers for C, C++, and Fortran;
- MPI

- HDF5 compiled with support for Fortran and MPI
- netcdf
- GSL
- FFTW
- PETSc compiled with support for Fortran, complex-valued functions, MUMPS and/or SuperLU_dist
- PUMI meshing libraries

1.5 Building

Once the appropriate makefile has been defined, the M3D-C1 executables can be built by entering `$M3DC1_DIR/unstructured` and running

```
make all
```

This will run the following commands:

`make OPT=1`	Builds the 2D version
`make OPT=1 COM=1`	Builds the complex version
`make OPT=1 3D=1 MAX_PTS=60`	Builds the 3D version
`make OPT=1 3D=1 MAX_PTS=60 ST=1`	Builds the stellarator version
`make a2cc`	Builds a utility for extracting coil currents from a-eqdsk files
`make bin`	Puts executables into the _`$M3DC1_ARCH` subdirectory

1.6 Regression Tests

The M3D-C1 source code includes a suite of regression tests that should be run before commiting any new code to the repository. To run these tests:

```
cd $M3DC1_DIR/unstructured/regtest ./run <arch> <test>
```

where `<arch>` and `<test>` are optional arguments specifying the specific batch script to run, and the specific regression test to run, respectively. By default, `<arch>=$M3DC1_ARCH`.

If `<test>` is not specified, then all the regression tests will be run, using the batch scripts `$M3DC1_DIR/unstructured/regtest/*/base/batchjob.<arch>`.

The `run` script will create new directories in which to run these regression tests, named `$M3DC1_DIR/unstructured/regtest/*/${M3DC1_VERSION}_<arch>/`.

If `<test>` is specified, then only `$M3DC1_ARCH/unstructured/regtest/<test>/base/batchjob.<arch>` will be run (again, in a new directory named as described above). If `<test>` is specified, then `<arch>` must also be specified.

To check the results of the regression tests

```
./check <arch>
```

2 GITHUB

Retrieve the current version of M3D-C1 from the GIT repository. For the first time, to check out the sources, do:

Initial access is with the *clone* command. This copies the source code from the master file into a working directory on your machine. You only do this once on each computer you work on.

module load git

git clone https://github.com/PrincetonUniversity/M3DC1

Subsequent GIT commands used to commit:

- *add/commit/push*: You *add* files to a list of files to update, *commit* the chanes to your branch, and then *push* the changes to the master branch.
- *git commit -m "message describing changes"* (adding *-a* commits all changes)
- *diff* lists the changes you made from the last commit, even if you haven'tpushed your commits to github. To see how your files differ from what's on github, you can do:

  git fetch origin master

  git diff origin/master
- *status* compares your branch with the master branch
- *pull origin master* updates your local branch to the current master branch
- *stash* takes uncommitted changes, saves them for later use, and reverts files in working directory

  stash list   *stash drop*   *stash apply*   *stash pop* (apply and drop)
- *stash pop* removes changes from your stash and reapplies them to working copy
- *stash apply* keeps changes in stash, but reapplies them to working copy
- *reset -hard* discards any changes to local branch since last commit
- *branch* tells you what branch you are in
- *log* (–oneline) (-after 2023-01-31) lists all the commits for the checked-out branch after that date
- *checkout hashtag* replaces your version with the version that has hastag "hashtag" (need only first 7 characters)

2.1 Branches in GIT

To make a new branch called fp-phase2:

- *git checkout master* switch to the master branch
- *git pull* make sure the master branch is up to date
- *git checkout -b fp_phase 2* The =b creates a new branch. This will be identical tomaster to start.
- *git push –set-upstream origin fp_phase2* push this new branch to the remote so others can access it

2.2 Committing changes

For example, to commit changes to newpar.f90

- *git pull* Always do thi before you start committing
- *git add newpar.f90* This stages the current changes in newpar.f90 for commit. You could then make more change before committing, but you would have to add again to get those into the commit. Or, you could add -a to the following to commit all changes.
- *git commit -m "Changed newpar.f90"* Commit changes to your local branch.
- *git push* Push commits to the remote repo. –set-upstream only needs to be done the first time.

2.3 Merging branches

To merge changes on master into fp_phase2

- *git checkout master* switch to the master branch
- *git pull* get the latest commits on the master branch
- *git checkout fp_phase2* switch back to fp_phase2
- *git pull* to get the latest commits to fp_phase2
- *git merge master* merge any new commits into fp_phase2. This makes a commit. You may need to resolve conflicts.
- git push Push the merged commit to remote repo

Inverting fb_phase2 and master here would merge the development branch into master locally, then the push wouuld send the merge tothe remote master

3 Mesh Generation and Management

Given a mesh, all the mesh support needed to run M3D-C^1 is provided by PUMI (Parallel Unstructured Mesh Infrastructure) developed at RPI Scientific Computation Research Center (SCOREC). The PUMI is used to manage the mesh information as it is processed within the M3D-C^1 code. PUMI is a freely available open source software. For more information on how to get and install, visit http://www.scorec.rpi.edu/pumi.

The generation of M3D-C^1 meshes for Tokamak geometries using the M3D-C^1 mesh generation program involves both PUMI and Simmetrix mesh generation software. Simmetrix is commercial software which provides a set of tools and libraries for engineering simulation. As Simmetrix runs only with a valid license, any M3D-C^1 user who wants to install his/her own version of the M3D-C^1 mesh generation program should contact Simmetrix to purchase its license. For more information on Simmetrix, visit http://simmetrix.com. M3D-1^1 users with accounts on `PPPL Portal` or `Princeton Stellar` are allowed to run M3D-C^1 mesh mesh generation programs on those systems.

Note that Simmetrix software is not required to run a M3D-C^1 simulation. Running a M3D-C^1 simulation using a mesh generated by a different mesh generator requires the development of a tool that creates PUMI readable model and mesh files with M3D-C^1 related control information required as the M3D-C^1 input. For more information, contact RPI SCOREC (shephard@rpi.edu).

With respect to the mesh support, there are a number of files involved with housing the geometry and mesh information. First, the model file extensions are the following

- `.smd`: Simmetrix-readable binary format model file
  The model generated with Simmetrix is saved in this format.
- `.dmg`: PUMI-readable binary format model file
  The model generated from PUMI mesh
- `.txt`: M3D-C^1-readable ascii format model file
  The model is generated from mesh generation tool (See Section 3.1)

Second, the mesh file extensions are the following.

- `.sms`: Simmetrix-readable binary format mesh file
  - The mesh generated with Simmetrix is saved in this format.
  - If a mesh is serial (1-part), the mesh file doesn't have a number before the extension
  - If a mesh is distributed (`P`-part, P>1), the mesh file has a number before the extension to represent the global part ID.
- `.smb`: PUMI-readable binary format mesh file
  - This format is used in M3D-C^1 to import/export a mesh
  - No matter if a mesh is serial (1-part) or distributed (`P`-part, P>1), the mesh file has a number before the extension to represent the global part ID.

- `.vtu/pvtu`: binary format mesh file for visualization with Paraview. For more information, visit http://paraview.org.

An overview of the Model/Mesh requirements for the M3D-C^1 mesh generation process are as follows:

- The model and mesh shall be generated as described in Section 3.1.
- The mesh file must be PUMI-readable `.smb` file. Note that a mesh file name contains a number before the extension (.smb) to denote a global part ID.
- The model and mesh file must be present in the work directory
- The name of model and mesh file must be specified in `C1input` file in the work directory
  - mesh_model = model_file
  - mesh_filename = mesh_file.smb (NOTE: do not specify a number before the file extension)
- In a 2D run with `P` processes, there should be `P` mesh files with part ID from `0` to `P-1`
- In a 3D run with `P`×`N` processes where 2D mesh is distributed to `P` parts,
  - there should be `P` mesh files with part ID from `0` to `P-1`
  - in `C1input` file, specify `nplanes` to `N` (e.g. nplanes=8), where `nplanes` describes how many 2D mesh copies to be loaded
  - the M3D-C^1 code should be compiled with options `"3D=1, MAX_PTS=60"`.

The rest of this section is organized as follows: Section 3.1 describes a mesh generation program `m3dc1_meshgen`. Section 3.2 describes a mesh generation program `m3dc1_mfmgen`. Section 3.3 describes a mesh generation program `polar_meshgen`. Section 3.5 presents a mesh partitioning program `"split_smb"` and `"collapse"` which changes the number of parts of the mesh. For how to visualize a mesh with `Paraview`, see Appendix B.

3.1 m3dc1_meshgen

`m3dc1_meshgen` requires an ascii input file of arbitrary name that contains the following parameters.

- modelType: 0, 1, 2, 3, or 4
  - Type 0: a parameterized vacuum region defined by five doubles for analytic expression. For five doubles X_0 X_1 X_2 Z_0 Z_1, vacuum boundary is defined by

$$X = X_0 + X_1 cos(\theta + X_2 * sin(\theta)) \tag{1}$$

$$Z = Z_0 + Z_1 sin(\theta) \tag{2}$$

  - Type 1: a vacuum region defined by piece-wise linear points
  - Type 2: a vacuum region defined by piece-wise polynomials
  - Type 3: spline-fitted 3-region model (plasma, wall and vacuum)
  - Type 4: spline-fitted 3-region model (plasma, wall, and vacuum) with inner & outer boundary points to set resistive wall

- reorder: if 1, reorder PUMI mesh based on adjacency (default: 0) and generate vtk folders for mesh visualization. The mesh before and after reodering is saved in `original-mesh.vtk` and `reordered-mesh.vtk`, respectively. Note that the element order of Simmetrix mesh is not affected.

- inFile: (modelType 0) not required (modelType 1 and 2) geometry file describing the vacuum (modelType 3 and 4) geometry file describing the inner plasma wall

- bdryFile: (modelType 0-3) not required (modelType 4) geometry file describing the outer plasma wall

- outFile: output file name to save model and mesh

- meshSize: relative mesh size for each region (default 0.05)
  for modelType 3, set three doubles for plasma, resistive, vacuum, respectively

- useVacuumParams: for modelType 0 or 3, if 1, use parameterized vacuum wall (default 0)

- vacuumParams: five doubles to describe parameterized vacuum wall. Required if useVacuumParams=1.

- adjustVacuumParams: for modelType 0 or 3, if 1, multiply coordinates and parametric values of nodes on vacuum wall by vacuumFactor. Valid only if useVacuumParams=1 (default 0)

- vacuumFactor: for modelType 0 or 3, an optional double value used to multiply coordinates and parametric values of nodes on vacuum wall when adjustVacuumParams=1. Valid only if adjustVacuumParams=1 (default 2×PI)

- numVacuumPts: optional # interpolation points on parameterized vacuum wall. Valid only if useVacuumParams=1 (default 20)

- meshGradationRate: for modelType 3 or 4, optional mesh gradation rate (default: 0.3). This value should be greater than or equal to 0.3. Otherwise the mesh will be fine everywhere.

- resistive-width: for modelType 3, the width of resistive wall. If resistive-width=0, only plasma region is created (default 0.02)

- plasma-offsetX: for modelType 3, the offset in x direction to the left (default 0.0)

- plasma-offsetY: for modelType 3, the offset in y direction to the bottom (default 0.0)

- vacuum-width: for modelType 3 or 4, the width of vacuum region (default 2.5)

- vacuum-height: for modelType 3 or 4, the height of vacuum region (default 4.0)

Locate input parameter file and all files listed as bdryFile (if applicable) in the work folder and do `m3dc1_meshgen input_param_file`, then the following output will be generated.

- The output model in three formats
  - $M3D - C^1$-readable `.txt`
  - Simmetrix-readable file `.smd` and
  - PUMI-readable `.dmg`
    - ▷ For modelType 0-2, the model is saved in `outFile.*`
    - ▷ For modelType 3 with resistive width `R`, vacuum-width `W` and vacuum-height `H`, the model is saved in `outFile-R-W-H.*`.
    - ▷ For modelType 4 with vacuum-width `W` and vacuum-height `H`, the model is saved in `outFile-W-H.*`.
- The output mesh in three formats
  - Simmetrix-readable`.sms`
  - $M3D - C^1$/PUMI readable `.smb`
  - Paraview
    - ▷ For modelType 0-2 with # mesh faces `F`,
      - if `F` > 1000, the mesh is saved in `outFile-(F/1000).*`
      - if `F` < 1000, the mesh is saved in `outFile-F.*`
    - ▷ For modelType 3 with # mesh faces `F`, resistive width `R`, vacuum-width `W`, vacuum-height `H`,
      - if `F` > 1000, the mesh is saved in `outFile-R-W-H-(F/1000).*`
      - if `F` < 1000, the mesh is saved in `outFile-R-W-H-F.*`
    - ▷ For modelType 4 with # mesh faces `F`, vacuum-width `W` and vacuum-height `H`,
      - if `F` > 1000, the mesh is saved in `outFile-W-H-(F/1000).*`
      - if `F` < 1000, the mesh is saved in `outFile-W-H-F.*`

3.1.1 Type 0 (parameterized vacuum)

The figure 1 illustrates a mesh generated by the following input file.

```
modelType 0
outFile analytic
meshSize 0.04

useVacuumParams 1
vacuumParams 1.65908 0.46 0.2 -0.02504 0.8
numVacuumPts 20

adjustVacuumParams 0
vacuumFactor  6.28319
```

The figure 2 illustrates a mesh generated with the same vacuum region parameters and a higher mesh size value.

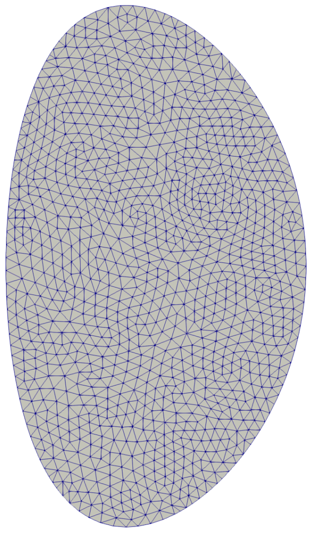


Figure 1: A mesh with vacuum region defined by five parameters for analytic expression

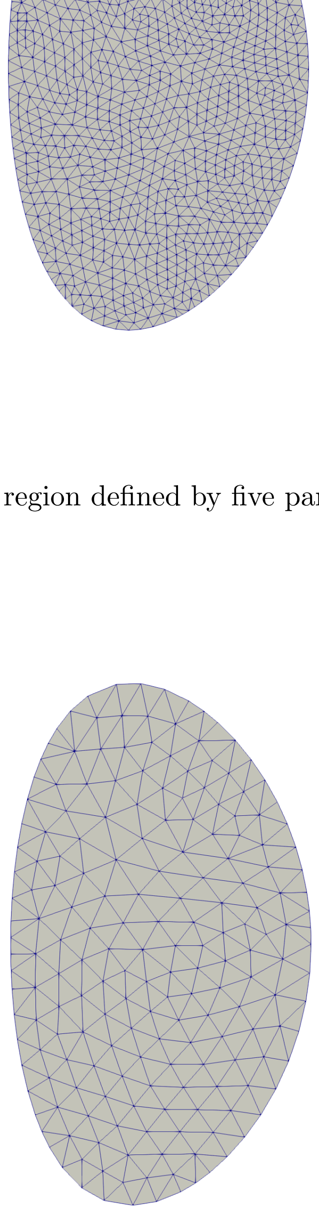


Figure 2: A mesh generated with vacuum region parameters and mesh size 0.1

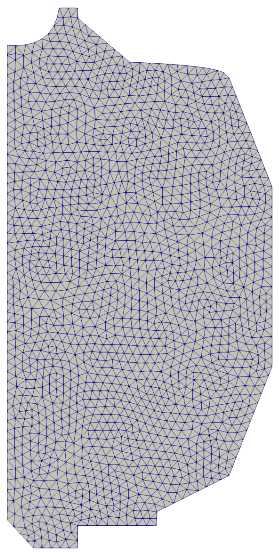


Figure 3: A mesh with vacuum region defined by piece-wise polynomials

3.1.2 Type 2 (piece-wise polynomial vacuum)

The figure 3 illustrates a mesh generated by the following input file. The vacuum region's geometry information is defined by piece-wise polynomials and stored in the file `in-poly`.

```
modelType 2
inFile in-poly
outFile poly
```

The vacuum region's geometry information is defined by piece-wise polynomials and stored in the file `in-poly` and the example file can be found in

`/p/tsc/m3dc1/lib/SCORECLib/rhel7/intel2019u3-openmpi4.0.3/16.0-220226/bin`.

3.1.3 Type 3 (three-regions with inner wall points)

With the model type 3, a geometric model consists of three model faces where each represents plasma region, resistive region and vacuum region, respectively. An ascii file name which describes inner plasma wall boundary has to be provied with the parameter `inFile`.

The figure 4 illustrates a mesh generated by the following input file. In the figure, geometric model faces are different colored.

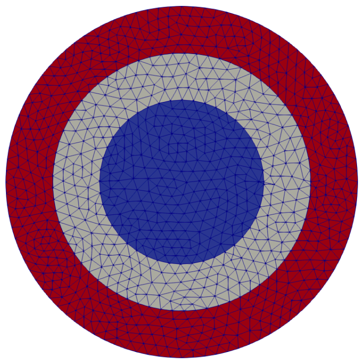


Figure 4: A mesh with spline-fitted 3-region model

```
modelType 3
inFile in-circle
outFile circle
meshSize 0.1 0.5 0.1
useVacuumParams 1
adjustVacuumParams 1
vacuumParams 5.0 1.5 0.0 0.0 1.5
numVacuumPts 20
meshGradationRate 0.4
resistive-width 0.4
```

The example files `circle-input` and `in-circle` can be found in

`/p/tsc/m3dc1/lib/SCORECLib/rhel7/intel2019u3-openmpi4.0.3/16.0-220226/bin`.

3.1.4 Type 4 (three-regions with inner & outer wall points)

With the model type 4, a geometric model consists of three model faces where each represents plasma region, resistive region and vacuum region, respectively. The parameter `inFile` denotes a file name that contains inner plasma wall boundary. The parameter `bdryFile` denotes a file name that contains resistive wall boundary.

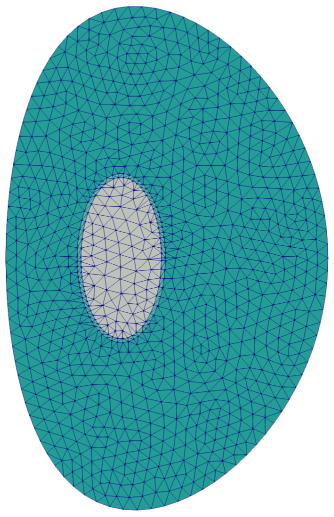


Figure 5: A mesh with inner & outer plama wall boundary points

The figure 5 illustrates a mesh generated by the following input file. In the figure, geometric model faces are different colored.

```
modelType 4
inFile inner_bdry.pts
bdryFile outer_bdry.pts
outFile iter
meshSize 0.7 0.5 0.7
useVacuumParams 1
adjustVacuumParams 1
vacuumParams 8.25 8.0 0.2 0.0 12.5
meshGradationRate 1
```

The example files `bdry-input`, `inner_bdry.pts` and `outer_bdry.pts` can be found in

`/p/tsc/m3dc1/lib/SCORECLib/rhel7/intel2019u3-openmpi4.0.3/16.0-220226/bin`.

3.2 m3dc1_mfmgen

`m3dc1_mfmgen` requires an ascii input file of arbitrary name that contains the following parameters. The parameters can be in any order.

- numBdry: the number of boundary files defined by peice-wise linear points given for the construction of the loops (default: 0)
  - Each boundary file corresponds to a loop in PUMI
  - For `numBdry`$=N$, N lines of `bdryFile` should be provided, $N \geq 0$
- bdryFile: For each boundary file, the user has to provide its file name followed by the unique loop ID and desired mesh size on the loop.
  - Each boundary file corresponds to a loop in PUMI
  - The unique ID can be an arbitrary integer defined by the user
  - The unique ID is used with input parameter "faceBdry" to specify the boundaries (loops) of model face
  - For more than one boundary files (numBdry>1), the boundary files can be in any order
- useVacuum: A parameter to control the vacuum boundary
  - The first number sets the mode of vacuum boundary and can be 0, 1 or 2. If 0, no vacuum boundary will be created. If 1, vacuum boundary will be created without user defined parameters. If 2, a parameterized vacuum boundary will be created by using the parameters defined in the parameter "vacuumParams".
  - The second number is the desired unique loop ID for the vacuum loop
  - The third number defines the mesh size on the vacuum boundary
- vacuumParams: if "useVacuum = 2", the user has to provide five doubles to define parameterized vacuum wall
- numVacuumPts: if "useVacuum = 2", # interpolation points on parameterized vacuum wall (default=20)
- thickWall: three integers and one double to control finite thickness wall
  - The first number can either be 0 or 1. If it is set to 0, no finite thickness wall will be created. If it is set to 1, a finite thickness wall will be created
  - The second number is the loop ID that will be offset for given thickness
  - The third number is the desired unique loop ID for the new loop created from offsetting for the finite thickness wall
  - The last number is the desired wall thickness
- layeredMesh: two integers to create an extruded layeded mesh on the finite thickness wall
  - The first integer is 0, no layered mesh will be created. If 1 (default), an extruded mesh with desired number of mesh layers will be created
  - The second integer defines the number of mesh layers
- numFace: the number of geometric model faces in PUMI (default 1)
  - Each geometric face corresponds to regions (e.g. plasma, resistive, vacuum) in M3DC1
  - For `numFace`$=N$, N lines of `faceBdry` should be provided, $N > 0$

- faceBdry: For each model face, the user has to provide the number of loops, loop ID(s), and desired mesh size
  - The first number gives the total number of loops bounding the face
  - the first number is followed by the loops IDs of the bounding loops. If number of loops $= n$, there should be n loop ID
  - The last number is the desired mesh size on the geometric face
- meshGradationRate: Global mesh gradation rate for the meshing. This parameter is optional and if not specified a default mesh gradation rate = 0.3 is used. This value should be greater than or equal to 0.3. Otherwise the mesh will be fine everywhere.
- outFile: output file name to save model and mesh

Locate input parameter file and all files listed as bdryFile (if applicable) in the work folder and do `m3dc1_mfmgen input_param_file`. The output files are the same as those of `m3dc1_meshgen`.

3.2.1 Mesh with parameterized vacuum wall

This section presents a mesh created with a parameterized vacuum wall. This is equivalent to `Type 0` mesh of `m3dc1_meshgen`.

```
numBdry 0
useVacuum 1 1 0.1
numFace 1
faceBdry 1 1 0.09
outFile analytic-0.09
```

The figure 6 presents the mesh generated by the input file above with two different mesh sizes.

3.2.2 Mesh with single boundary file

```
numBdry 1
bdryFile loop1.dat 3 0.1
numFace 1
faceBdry 1 3 0.2
outFile input1
```

The figure 7 presents the mesh generated by the input file above.

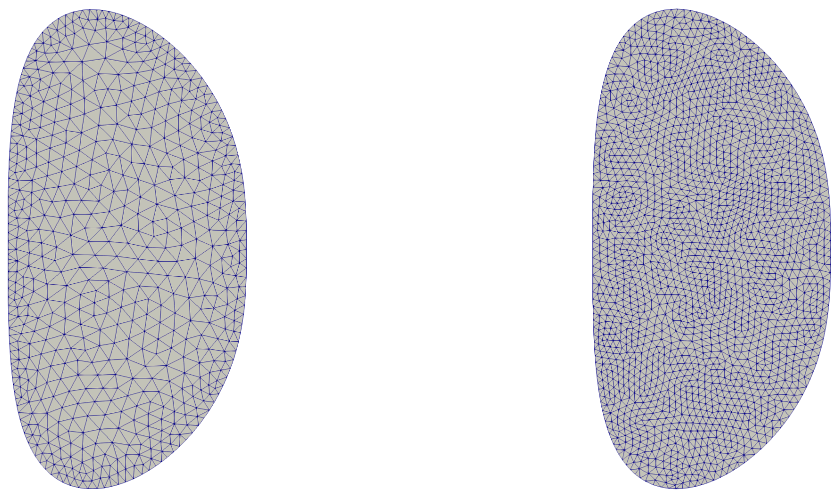


Figure 6: Mesh with a parameterized vacuum region and different mesh size for model face (left) 0.2 (right) 0.09

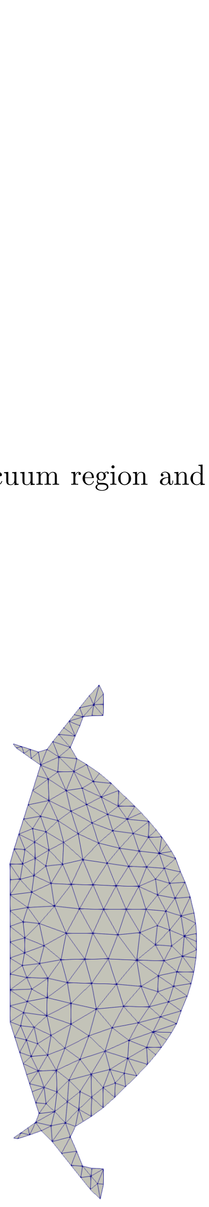


Figure 7: Mesh with single boundary file and no vacuum wall

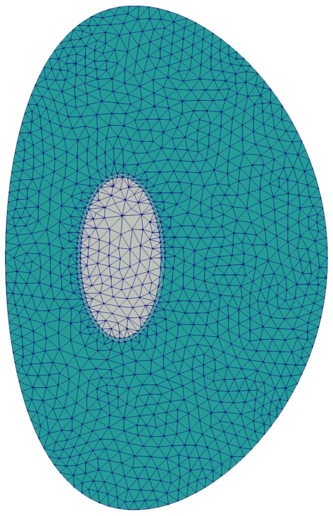


Figure 8: Mesh with two boundary files and a parameterized vacuum region

3.2.3 Mesh with two boundary files and a parameterized vacuum wall

This section presents a mesh created with two boundary files and a parameterized vacuum wall. This is equivalent to `Type 4` mesh of `m3dc1_meshgen`.

```
numBdry 2
bddyFile loop1.pts 1 0.5
bdryFile loop2.pts 2 0.5
useVacuum 2 3 0.5
vacuumParams 8.25 8.0 0.2 0.0 12.5
numFace 3
faceBdry 1 1 0.7
faceBdry 2 1 2 0.5
faceBdry 2 2 3 0.7
meshGradationRate 1
outFile iter
```

The figure 8 presents the mesh generated by the input file above. As you can see, the mesh in Figure 8 and 5 are almost identical.

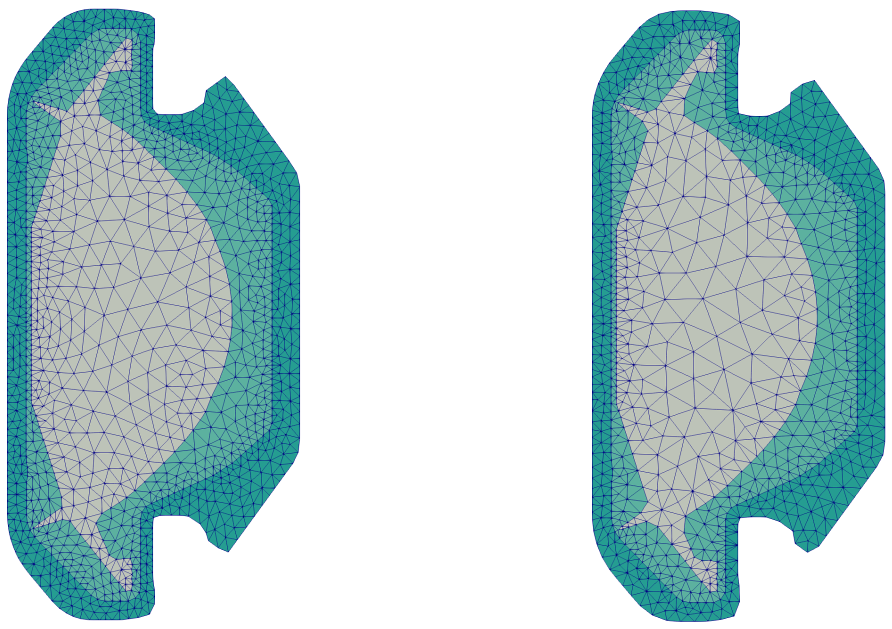


Figure 9: Mesh with three boundary files and different meshGradationRate (left) 0.3 (right) 0.9

3.2.4 Mesh with three boundary files

```
numBdry 3
bdryFile loop1.dat 3 0.1
bdryFile loop2.dat 10 0.05
bdryFile loop3.dat 11 0.09

numFace 3
faceBdry  1 3 0.2
faceBdry  2 3 10 0.1
faceBdry  2 10 11 0.09

outFile input3
```

The figure 9 presents the mesh generated by the input file above.

3.2.5 Mesh with seven boundary files and a vacuum wall

```
numBdry 7
bdryFile loop1.dat 3 0.2
bdryFile loop2.dat 10 0.3
bdryFile loop3.dat 11 0.4
```

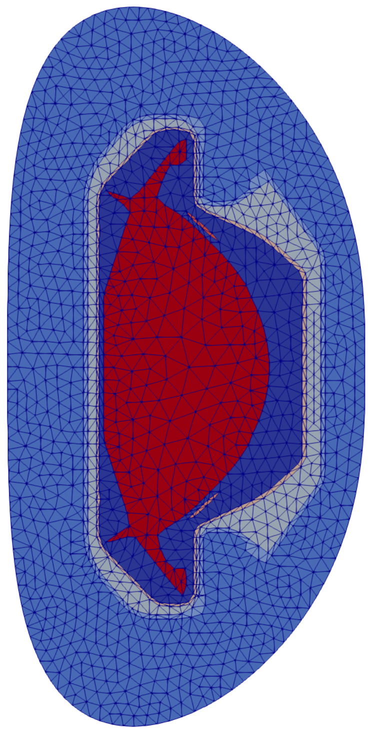


Figure 10: Mesh with seven boundary files and a parameterized vacuum wall

```
bdryFile loop4.dat 21 0.4
bdryFile loop5.dat 25 0.4
bdryFile loop6.dat 17 0.2
bdryFile loop7.dat 19 0.1

useVacuum 1 9 0.1
vacuumParams 1.8 1.5 0.4 0.0 2.5

numFace 8
faceBdry  1 3 0.2
faceBdry  1 10 0.3
faceBdry  1 11 0.1
faceBdry  2 21 25 0.2
faceBdry  2 25 17 0.09
faceBdry  2 17 19 0.1
faceBdry  2 19 9 0.1
faceBdry  4 3 10 11 21 0.11

outFile input7
```

The figure 10 presents the mesh generated by the input file above.

3.2.6 Mesh with finite thickness wall and layers

Add text here

3.3 polar_meshgen

`polar_meshgen` requires an ascii file of arbitrary name that contains input parameters as the following:

- inFile: input file name containing equilibrium generation by jsolver
- outFile: output file name to save model and mesh
- meshSize: relative mesh size for each region (default 0.05)
- reorder: if 1, reorder PUMI mesh based on adjacency (default: 0) and generate vtk folders for mesh visualization. The mesh before and after reodering is saved in `original-mesh.vtk` and `reordered-mesh.vtk`, respectively. Note that the element order of Simmetrix mesh is not affected.

The following presents an example input file “`polar_input`”.

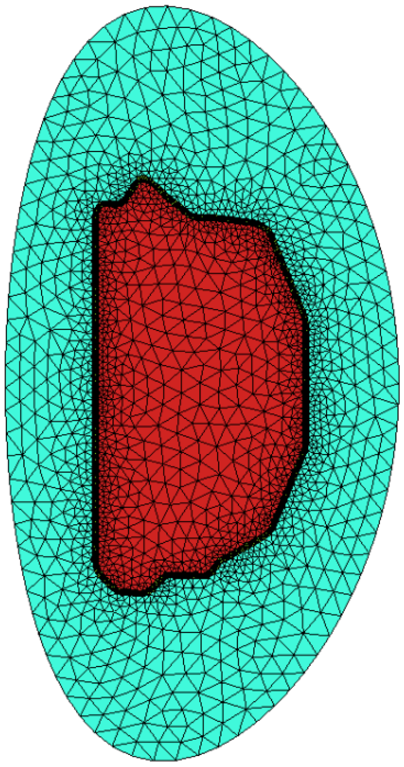


Figure 11: Add Caption Here

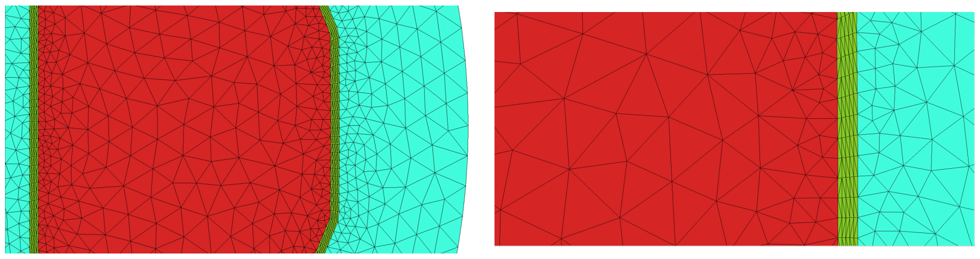


Figure 12: Add Caption Here

```
inFile POLAR
outFile polar
meshSize 0.04
```

To run `polar_meshgen`, place `polar_input` and `POLAR` in your work folder and do "`polar_meshgen polar_input`". The program will read `POLAR` and generate various model and mesh files starting with "polar". For instance, `polar-2K0.smb, pol-2K.sms, pol-2K.vtk, polar.dmg, polar.smd, polar.txt`. If the resulting mesh is too fine, increase the value of `meshSize`. If the resulting mesh is too coarse, decrease the value of `meshSize`. If `meshSize` is not specified in the input file, the default value is 0.05.

The program `read_jsolver` generates equilibrium and stores in the file `POLAR`. Given the input file `POLAR`, `m3dc1_meshgen` generates the following files:

- model.dmg: PUMI-readable model file
- model.txt: M3DC1-readable model file
- mesh0.smb: PUMI/M3DC1-readable mesh file
- mesh.vtk: Paraview data files
- norm_curv: ascii file containing nodes' normal/curvature information

3.4 Mesh Control with SimModeler

SimModeler is a graphical user interface to the Simmetrix geometry and mesh generation software. In cases where the currently available capabilities of m3dc1_meshgen do not provide a satisfactory mesh, SimModeler can be used to apply alternative mesh control information to the Tokamak cross section geometry to generate different meshes. The information below indicates the application of a subset of the mesh controls that can be applied. For additional information of the full range of SimModeler mesh control options see: ********** FILL IN POINTER TO SIMMETRIX DOCUMENTATION ***** (Contributed by D. Pfefferle on 4/27/16) On PPPL Portal, load a module `simmodeler` and run it.

1. From the menu `"File→Open Model"`, load a model file (`.smd`) generated by `m3dc1_meshgen`
2. In the upper panel, in the views section, click on `Front` to view the model, then go to `Meshing` tab
3. Select outer region, click `+` in `Mesh Attributes` and select `Mesh Size→relative`. Enter a value (typically 0.1)
4. Select wall region, click `+` in `Mesh Attributes` and select `Mesh Size→relative`. Enter a value (typically 0.02)
5. Select inner region, click `+` in `Mesh Attributes` and select `Mesh Size→relative`. Enter a value (typically 0.04). Here, one can already generate the mesh by clicking on `Generate Mesh` and verify if the mesh sizes are suitable

6. Select both inner and wall regions (holding shift key), click `+` in `Mesh Attributes` and select `Mesh Size`→`relative`. Enter a function, e.g. `0.01`×`abs($y+1.5)^2+0.004` to specify an anisotropic mesh density on top of previous settings
   There are many available parameters for fine-tuning the mesh density. For example, `Mesh Curvature Refinement` with parameter packs more elements near the edges of the resistive wall.

7. `Generate Mesh` and `Show Mesh` to view result in new windows

8. If the result is satisfactory, from the menu `File`→`Save Mesh`, give it a meaningful name with the extension `.sms`. The original model file `.smd` has been automatically saved by the program with your mesh modifications.

9. Close `simmodeler` then it will release a license. Until you quit Simmodeler, no one cannot run neither `m3dc1_meshgen` nor `simmodeler`.

10. Copy the `.txt, .smd` and `.sms` files to the simulation directory and run the following splitting routine to obtain PUMI-readable `.smb` mesh files.

    `/p/tsc/C1/m3dc1-sunfire.r6-1.5/bin/part_mesh.sh model_file.smd mesh_file.sms X`, where `X` is the number of parts you need in the `.smb` mesh.

11. Modify the `C1input` file accordingly

    ```
    mesh_filename = 'part.smb'
    mesh_model = 'filename.txt'
    ```

3.5 Mesh Partitioning

3.5.1 Splitting

The program `split_smb` increases the number of parts in a mesh from `P` to `N` (`P<N`). In each machine, the program `split_smb` is availble in `$SCOREC_UTIL_DIR` provided in `hostname.mk` file.

In order to split `P`-part mesh to `N` parts (`N>P`), run `"mpirun -np N ./split_smb input-mesh(.smb) output-mesh(.smb) X"`

- the file extension of input-mesh should be .smb
- the file extension of output-mesh should be .smb
- `N` is the number of parts in the output mesh
- For a `P`-part input mesh, `X` must be `N/P`
- For both input and output mesh, do not specify a number before the file extension
- `split_smb` will insert a number in the output mesh file. The number represents a global part ID.

- Make sure that the output mesh doesn't have any empty part. Otherwise, the program crashes with the following error message:
  ```
  APF warning: 1 empty parts
  split_smb: .../mds/mds.c:614: check_ent: Assertion ‘e >= 0’ failed
  ```

Examples on portal:

1. To split a serial (1-part) mesh to 6 parts, run
   `"mpirun -np 6 ./split_smb struct-curveDomain.smb part.smb 6"`
   - Input mesh: struct-curveDomain0.smb
   - Output mesh: part0.smb, part1.smb, part2.smb, part3.smb, part4.smb, part5.smb
2. To split a 2-part mesh to 6 parts, run `"mpirun -np 6 ./split_smb struct-curveDomain.smb part.smb 3"`
   - Input mesh: struct-curveDomain0.smb, struct-curveDomain1.smb
   - Output mesh: part0.smb, part1.smb, part2.smb, part3.smb, part4.smb, part5.smb

See `readme.split_smb` for detailed instructions and trouble shooting tips.

3.5.2 Mesh Merging

The program `collapse` decreases the number of parts in a mesh from `N` to `P` (`P`<`N`). In each machine, the program `collapse` is availble in `$SCOREC_UTIL_DIR` provided in `hostname.mk` file.

In order to merge `N`-part .smb mesh to `P` parts (`P`>`0`), run `"mpirun -np N ./collapse input-mesh(.smb) output-mesh(.smb) X"`

- the file extension of input-mesh should be .smb
- the file extension of output-mesh should be .smb
- `N` is the number of parts in the input mesh
- For a `P`-part output mesh, `X` must be `N/P`
- For both input and output mesh, do not specify a number before the file extension
- `collapse` will insert a number in the output mesh file. The number represents a global part ID.

Example on portal:
In order to merge 4-part mesh into a serial (1-part) mesh, run `"mpirun -np 4 ./collapse part.smb serial.smb 4"`

- Input mesh: part0.smb, part1.smb, part2.smb, part3.smb

- Output mesh: serial0.smb

See `readme.collapse` for detailed instructions and trouble shooting tips.

3.6 Miscellaneous

3.6.1 Verification

The program `check_smb` investigates an input mesh and prints any invalid aspects of the mesh. It prints out the mesh size (the number of global, local, and owned entities per dimension) at the end.

In order to run, do `"mpirun -np N ./check_smb input-mesh(.smb)"`.

4 Mesh Adaptation by Error Estimator

5 PETSc Option

5.1 2D

The petsc option to run 2D modes is default to superlu_dist. You can add the following line on your "srun" command line to change to to mumps:

```
-pc_factor_mat_solver_type mumps
```

5.2 3D

When running M3D-C^1 in the 3D nonlinear mode, you need to include PETSc Options file. There is a number "8" in the file below. It must be equal to the number of toroidal planes. It should be changed whenever you change the number of toroidal planes in the C1input file. The recommended *options_bjacobi* file is as follows:

```
-pc_type bjacobi
-pc_bjacobi_blocks 8
    (for 8 toroidal planes should be equal to nplanes in C1input)
-sub_pc_type lu
-sub_pc_factor_mat_solver_package superlu_dist
    (can exchange mumps for superlu_dist)
-mat_superlu_dist_rowperm NOROWPERM (only needed for superlu_dist)
-mat_mumps_icntl_14 50 (only needed for mumps)
    (50 means 50% of memory increase when needed.
     Users can make it 100 or more if encountering a runtime memory issue.)
-sub_ksp_type preonly
-ksp_type fgmres
-ksp_gmres_restart 220
-ksp_rtol 1.e-9
-ksp_max_it 10000
-on_error_abort

-hard_pc_type bjacobi
-hard_pc_bjacobi_blocks 8
    (for 8 toroidal planesshould be equal to nplanes in C1input)
-hard_sub_pc_type lu
-hard_sub_pc_factor_mat_solver_type superlu_dist
    (can change mumps for superlu_dist)
-mat_superlu_dist_rowperm NOROWPERM
    (only needed for superlu_dist)
-mat_mumps_icntl_14 50
    (only needed for mumps.)
    (50 means 50% of memory increase when needed.
     Users can make it 100 or more if encountering a runtime memory issue.)
-hard_sub_ksp_type preonly
-hard_ksp_type lgmres
-hard_ksp_lgmres_argument 4
-hard_ksp_gmres_restart 220
-hard_ksp_rtol 1.e-9
-hard_ksp_max_it 10000
```

5.3 More

More examples are in regtest/pellet/base/ directory, such as

```
options_bjacobi.type_superludist
options_bjacobi.type_mumps
```

The following are additional optional arguments:

```
-ksp_converged_reason
-ksp_view

-help

-options_table
-options_left

-trdump
-malloc_log
```

for diagnosing purpose.

6 Running Jobs

Users can find almost all of the needed example batch scripts and input files to run a job on available computing facilities from

```
unstructured/regtest/*/base/
```

directories.

6.1 Running 2D or Linear Jobs

In 2D, the run can be either linear or nonlinear, depending on the C1input parameter *linear*:

$linear = 0$ (non-linear run: must compile with the option RL=1)

$linear = 1$ (linear run: must compile with the option COM=1)

In both cases, set

$nplanes = 1$

For the linear case, use

$ntor = nn$

set the toroidal mode number.

To run your job on a scratch directory, copy the following files over:

```
executable (m3dc1_2d for non-linear run or m3dc1_2d_complex for linear run)
C1input
AnalyticModel (or MultiEdgeAnalyticModel)
struct-dmg.sms
(and geqdsk if needed)
```

To run non-linear job

```
mpirun np 8 ./m3dc1_2d
```

To run linear job

```
mpirun np 24 ./m3dc1_2d_complex -pc_factor_mat_solver_package mumps
```

6.2 Running 3D Nonlinear Jobs

For the 3D nonlinear run, set *linear* $= 0$ and set *nplanes* equal to the number of toroidal planes in *C1input* file. The number of bjacobi blocks in the PETSc options file must also be equal to *nplanes*. The total number of processors to request must be the product of nplanes and M (the number of processors per plane, which equals the number of mesh partitions per plane).

Files required to be present to the local directory are:

```
executable (m3dc1 or m3dc1_st)
C1input,
partnn.smb (one for each poloidal plane partition)
options_bjacobi
m3dc1.xml (if using ADIOS)
geqdsk (if needed)
```

To run linear job

```
mpirun np 16 ./ m3dc1 ipetsc options_file options_bjacobi (nplane=2, M=8)
```

See the previou section for the format of the PETSc option file options_bjacobi. In this example job, there are $M = 8$ mesh partition files:

```
part0.smb part1.smb part2.smb part3.smb part4.smb part5.smb part6.smb part7.smb
```

6.3 Graphics Files

The graphics files are of two types. There is a single file called: C1.h5 that contains all the timedependent scalar information. This must be saved and be present in the directory of a job so that it can be added to.

In addition to this file, each plot cycle will produce a file: time_nnn.h5, where nnn is the plot cycle number. The equilibrium is written into a file called equilibrium.h5. These must be stored in the same directory as the C1.h5 file.

6.4 Restarting Jobs

By default hdf5 files are written in every time step. Therefore jobs can be restarted from the hdf5 plot file, the same one that is used by the idl routines to make plots.

By default, the hdf5 files are written in single precision. If idouble_out is set to 1 in C1input file, hdf5 files are written in double precision.

6.4.1 Reading restart files for 2D real, 2D complex, or 3D real runs

To start a normal simulation with the hdf5 files, set the C1input parameter irestart to 1.

However, the files C1.h5 and the final time_nnn.h5 file must be present in the working directory. You may also restart from an intermediate time by setting irestart_slice=nn where nn is the nnth plot file. If this is not set, it will restart from the final plot file.

6.4.2 Reading real restart files to initialize 2D complex calculation

- Run 2D linear=0
- Copy 2D C1.h5 and the final time_nnnn.h5 to the working directory
- Run 2D (complex) linear=1. In the initial restart, the time and cycle number will start from t=0 and N=0 for the complex run

6.4.3 Running 3D real simulation from 2D real restart files

To start a 3D simulation with 2D restart files, do the following:

- Run 2D
- Copy 2D C1.h5 and the final time_nnn.h5 to the working directory
- In 3D work folder, set the C1input parameter irestart= 1 Regardless of the time step when the restart files were written, the 3D simulation starts with time step 1.

6.5 Monitoring Jobs

You can monitor the progress of your running job in several ways:

A. C1ke file. Each time step, one line will be added to the ASCII C1ke file in the run directory that you can open with a text editor. The first 4 fields are:

```
cycle time kinetic_energy growth_rate
```

B. C1.h5 file: You can monitor a time dependent run by using the idl utility described below. Especially useful is the

```
plot_scalar,ke
```

command and also

```
plot_scalar,ke,/growth
```

C. You can use a text editor to monitor the log file slurm-nnnn.out file (where nnnn is the job number assigned by SLURM)

6.6 Exporting Node/Vector/Matrix for Standalone Study

6.7 Archiving Data at PPPL

7 Boundary Conditions

In all cases, $f = 0$ on the boundary, and therefore also $\hat{\mathbf{t}} \cdot \nabla f = 0$. Some other boundary conditions that may be specified are as follows:

No normal flow (`inonormalflow=1`) Holds $\hat{\mathbf{n}} \cdot \mathbf{u}$ constant.

No poloidal flow (`inoslip_pol=1`) Holds $\hat{\mathbf{t}} \cdot \mathbf{u}$ constant.

No toroidal flow (`inoslip_tor=1`) Holds $\hat{\varphi} \cdot \mathbf{u}$ constant.

No normal current (`Winocurrent_pol inocurrent_norm=1`) Holds $\hat{\mathbf{n}} \cdot \mathbf{J}$ constant.

No poloidal current (`inocurrent_pol=1`) Holds $\hat{\mathbf{t}} \cdot \mathbf{J}$ constant.

No toroidal current (`inocurrent_tor=1`) Holds $\hat{\varphi} \cdot \mathbf{J}$ constant.

$$\begin{aligned}
\hat{\mathbf{n}} \cdot \mathbf{u} &= -R\hat{\mathbf{t}} \cdot \nabla U + \frac{1}{R^2}\hat{\mathbf{n}} \cdot \nabla \chi & (3)\\
\hat{\mathbf{t}} \cdot \mathbf{u} &= R\hat{\mathbf{n}} \cdot \nabla U + \frac{1}{R^2}\hat{\mathbf{t}} \cdot \nabla \chi & (4)\\
\hat{\varphi} \cdot \mathbf{u} &= R\omega & (5)
\end{aligned}$$

$$\begin{aligned}
\hat{\mathbf{n}} \cdot \mathbf{B} &= -\frac{1}{R}\hat{\mathbf{t}} \cdot \nabla \psi - \frac{1}{R^2}\hat{\mathbf{n}} \cdot \nabla f_\varphi & (6)\\
\hat{\mathbf{t}} \cdot \mathbf{B} &= \frac{1}{R}\hat{\mathbf{n}} \cdot \nabla \psi & (7)\\
\hat{\varphi} \cdot \mathbf{B} &= \frac{F}{R} & (8)
\end{aligned}$$

$$\begin{aligned}
\hat{\mathbf{n}} \cdot \mathbf{J} &= -\frac{1}{R}\hat{\mathbf{t}} \cdot \nabla F + \frac{1}{R^2}\hat{\mathbf{n}} \cdot \nabla \psi_\varphi & (9)\\
\hat{\mathbf{t}} \cdot \mathbf{J} &= \frac{1}{R}\hat{\mathbf{n}} \cdot \nabla (F + f_{\varphi\varphi}) + \frac{1}{R^2}\hat{\mathbf{t}} \cdot \nabla \psi_\varphi & (10)\\
\hat{\varphi} \cdot \mathbf{J} &= -\frac{1}{R}\Delta^* \psi & (11)
\end{aligned}$$

In the above definitions, $\hat{\mathbf{n}}$ is the unit vector normal to the boundary surface, and $\hat{\mathbf{t}} = \hat{\varphi} \times \hat{\mathbf{n}}$.

8 Discretization

8.1 Finite Elements

Each field is represented as a linear combination of N basis functions ν_i on the computational domain

$$U = \sum_{i=1}^{N} \nu_i U_i.$$

The finite element used in M3D-$C^1$is the reduced quintic element [?], in which the basis functions are fifth order polynomials. At each time step, the projection of the equations onto the basis functions are computed and solved. For example, the equation

$$\frac{\partial U}{\partial t} = F(U)$$

becomes the system of projection equations

$$\int dV \, \nu_i \frac{\partial U}{\partial t} = \int dV \, \nu_i F(U).$$

These projections equations are known collectively as the *weak form* of the equation. Solving the equation in this manner is known as the *Galerkin method.* Hereafter the index i will be dropped from ν_i.

Once the equations are cast in the weak form, integrations by parts may be carried out in order to reduce the order of the differential operators acting on the physical fields. For example,

$$\begin{aligned} \int dV \, \nu \nabla^2 U &= \int dV \, \nabla \cdot (\nu \nabla U) - \nabla \nu \cdot \nabla U \\ &= \oint d\mathbf{A} \cdot \nabla U \nu - \int dV \; \langle \nu, \nabla U \rangle \\ &= -\int dV \; \langle \nu, U \rangle \, . \end{aligned}$$

It is found that using integrations by parts to re-cast the equations into a form in which a roughly equal number of derivatives acts on the trial function as on the physical fields improves the numerical stability of methods for solving the equations. Thus, in the above example, the form $-\langle \nu, U \rangle$ is preferable to $\nu \nabla^2 U$.

8.1.1 Weak form of Physical Equations

`Integration Identities`
Rather than performing integrations by parts directly on each term in equations (**??**), it is simpler to begin directly from the vector form, equations (**??**) and use the following identities when applying the operations to extract the scalar equations:

$$\begin{aligned} -\int dV \, R^2 \nu \nabla \varphi \cdot \nabla \times \mathbf{A} &= -\int dV \, \mathbf{A} \cdot \left[\nabla (R^2 \nu) \times \nabla \varphi \right] \\ \int dV \, \nu \nabla \cdot \mathbf{A} &= -\int dV \, \nabla \nu \cdot \mathbf{A}. \end{aligned}$$

(Note that the torodal operator, $R^2\nu\nabla\varphi\cdot$, is not a differential operator and therefore the integration by parts cannot be performed *a priori.*)

Similary, useful identities for the operators that will act on the stress tensor Π are:

$$\begin{aligned}
R^2\nu\nabla\varphi\cdot\nabla\times(\nabla\cdot\Pi) &= R^2\partial_Z\nu\nabla\varphi\cdot\Pi\cdot\nabla\varphi-\nabla\nu\cdot\Pi\cdot\nabla Z \\
&\quad +r\nabla\varphi\cdot\left[\nabla\nabla(\nu r)\dot{\times}\Pi\right]+\nabla\cdot\mathbf{A}_1 && \text{(12a)}\\
-R^2\nu\nabla\varphi\cdot(\nabla\cdot\Pi) &= R^2\nabla\nu\cdot\Pi\cdot\nabla\varphi+\nabla\cdot\mathbf{A}_2 && \text{(12b)}\\
-\nu\nabla\cdot(\nabla\cdot\Pi) &= -\nabla\nabla\nu:\Pi+\nabla\cdot\mathbf{A}_3 && \text{(12c)}
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{A}_1 &= -R^2\nu\nabla\varphi\times(\nabla\cdot\Pi)-r\Pi\cdot[\nabla\varphi\times\nabla(r\nu)]+\nu\Pi\cdot\nabla Z\\
\mathbf{A}_2 &= -R^2\nu\Pi\cdot\nabla\varphi\\
\mathbf{A}_3 &= \nabla\nu\cdot\Pi-\nu\nabla\cdot\Pi.
\end{aligned}$$

(These identities hold for any symmetric tensor Π.) The total divergences vanish upon integration.

8.1.2 Physical Equations after Integrations by Parts

$$\begin{aligned}
\int dV\,N_n &= \int dV\,\left[N_{nU}+N_{n\chi}+N_{nD}\right] && \text{(13a)}\\
\int dV\,\left[U_{Un}+U_{\chi n}\right] &= \int dV\,\Big[U_{UUn}+U_{VVn}+U_{U\chi n}+U_{\chi\chi n} && \text{(13b)}\\
&\quad +U_{\psi\psi}+U_{FF}+U_{U\mu}+U_{\chi\mu}+U_g\\
&\quad +U_{UD}+U_{\chi D}+U_{\Pi_\parallel}+U_{\Pi_\times}\Big]\\
\int dV\,V_{Vn} &= \int dV\,\Big[V_{VUn}+V_{V\chi n}+V_{\psi F}+V_{V\mu} && \text{(13c)}\\
&\quad +V_{VD}+V_{\Pi_\parallel}+V_{\Pi_\times}\Big]\\
\int dV\,\left[X_{Un}+X_{\chi n}\right] &= \int dV\,\Big[X_{UUn}+X_{VVn}+X_{U\chi n}+X_{\chi\chi n} && \text{(13d)}\\
&\quad +X_p+X_{\psi\psi}+X_{FF}+X_{U\mu}+X_{\chi\mu}+X_g\\
&\quad +X_{UD}+X_{\chi D}+X_{\Pi_\parallel}+X_{\Pi_\times}\Big]\\
\int dV\,\Psi_\psi &= \int dV\,\left[\Psi_{\psi U}+\Psi_{\psi\chi}+\Psi_{\psi Fn}+\Psi_{\psi\eta}\right] && \text{(13e)}\\
\int dV\,F_F &= \int dV\,\big[F_{FU}+F_{\psi V}+F_{F\chi}+F_{\psi n}+F_{Fn} && \text{(13f)}\\
&\quad +F_{p_e n}+F_{F\eta}\big]\\
\int dV\,P_p &= \int dV\,\Big[P_{pU}+P_{p\chi}+P_{p_e Fn}+P_{\eta\psi}+P_{\eta F} && \text{(13g)}\\
&\quad +P_\kappa+P_{\kappa_\parallel}+P_{\kappa_\times}\Big]
\end{aligned}$$

The terms in the above equations are categorized and defined in the following sections. Each term has been integrated by parts to arrive at the simplest expression having for which the order of differentiation on the trial function is roughly equal to that on the physical fields.

`Basic Terms`
The terms in this section are the basic terms in the two-fluid equations, which do not depend on any specific choice of closure.

$$\begin{aligned}
N_n(\nu,\dot{n}) &= \nu\dot{n} \\
N_{nU}(\nu,n,U) &= \nu\left[U,n\right] \\
N_{n\chi}(\nu,n,\chi) &= n\left\langle \nu,\chi\right\rangle \\
N_{nD}(\nu,n,D) &= -D\left\langle \nu,n\right\rangle
\end{aligned} \tag{14}$$

$$\begin{aligned}
U_{Un}(\nu,\dot{U},n) &= -\tfrac{1}{R^2}n\left\langle R^2\nu,\dot{U}\right\rangle \\
U_{\chi n}(\nu,\dot{\chi},n) &= -R^2\nu\left[n,\dot{\chi}\right] \\
U_{UUn}(\nu,U,U,n) &= \tfrac{1}{R^2}n\Delta^*U\left[R^2\nu,U\right]+\tfrac{1}{2R^2}\left\langle U,U\right\rangle\left[R^2\nu,n\right] \\
U_{VVn}(\nu,V,V,n) &= \tfrac{1}{2R^2}\left[\nu,R^2\right]VVn \\
U_{U\chi n}(\nu,U,\chi,n) &= \tfrac{1}{R^2}n\Delta^*U\left\langle R^2\nu,\chi\right\rangle-\left[U,\chi\right]\left[R^2\nu,n\right] \\
U_{\chi\chi n}(\nu,\chi,\chi,n) &= \tfrac{1}{2}\left\langle \chi,\chi\right\rangle\left[R^2\nu,n\right] \\
U_{\psi\psi}(\nu,\psi,\psi) &= -\tfrac{1}{R^2}\left[R^2\nu,\psi\right]\Delta^*\psi \\
U_{FF}(\nu,F,F) &= -R^2\nu F\left[F,\tfrac{1}{R^2}\right] \\
U_{UD}(\nu,U,D) &= \tfrac{1}{R^2}\left\langle R^2\nu,U\right\rangle D \\
U_{\chi D}(\nu,\chi,D) &= -\left[R^2\nu,\chi\right]D
\end{aligned} \tag{15}$$

$$\begin{aligned}
V_{Vn}(\nu,V,n) &= \nu n\dot{V} \\
V_{VUn}(\nu,V,U,n) &= \nu n\left[U,V\right] \\
V_{V\chi n}(\nu,V,\chi,n) &= -\nu n\left\langle \chi,V\right\rangle \\
V_{\psi F}(\nu,\psi,F) &= \nu\left[F,\psi\right] \\
V_{VD}(\nu,V,D) &= -\nu VD
\end{aligned} \tag{16}$$

$$\begin{aligned}
X_{Un}(\nu,\dot{U},n) &= \nu\left[n,\dot{U}\right] \\
X_{\chi n}(\nu,\dot{\chi},n) &= -n\left\langle \nu,\dot{\chi}\right\rangle \\
X_p(\nu,p) &= \left\langle \nu,p\right\rangle \\
X_{UUn}(\nu,U,U,n) &= -\tfrac{1}{R^2}n\Delta^*U\left\langle \nu,U\right\rangle+\tfrac{1}{2}n\left\langle \nu,\tfrac{\langle U,U\rangle}{R^2}\right\rangle \\
X_{VVn}(\nu,V,V,n) &= \tfrac{1}{2}nVV\left\langle \tfrac{1}{R^2},\nu\right\rangle \\
X_{U\chi n}(\nu,U,\chi,n) &= \left(n\nabla^2\nu+\left\langle n,\nu\right\rangle\right)\left[U,\chi\right]+n\Delta^*U\left[\nu,\chi\right] \\
X_{\chi\chi n}(\nu,\chi,\chi,n) &= \tfrac{1}{2}n\left\langle \nu,\left\langle \chi,\chi\right\rangle\right\rangle \\
X_{\psi\psi}(\nu,\psi,\psi) &= \tfrac{1}{R^2}\Delta^*\psi\left\langle \nu,\psi\right\rangle \\
X_{FF}(\nu,F,F) &= \tfrac{1}{R^2}F\left\langle \nu,F\right\rangle \\
X_{UD}(\nu,U,D) &= \left[\nu,U\right]D \\
X_{\chi D}(\nu,\chi,D) &= \left\langle \nu,\chi\right\rangle D
\end{aligned} \tag{17}$$

$$\begin{array}{lll}
\Psi_\psi(\nu,\dot\psi) & = & \nu\dot\psi \\
\Psi_{\psi U}(\nu,\psi,U) & = & \nu\,[U,\psi] \\
\Psi_{\psi\chi}(\nu,\psi,\chi) & = & -\nu\,\langle\chi,\psi\rangle \\
\Psi_{\psi F n}(\nu,\psi,F,n) & = & d_i\nu\frac{1}{n}\,[\psi,F] \\
\Psi_{\psi\eta}(\nu,\psi,\eta) & = & -\frac{1}{R^2}\left\langle\psi,R^2\nu\eta\right\rangle
\end{array} \quad (18)$$

$$\begin{array}{lll}
F_F(\nu,\dot F) & = & \nu\dot F \\
F_{FU}(\nu,F,U) & = & R^2\nu\left[U,\frac{F}{R^2}\right] \\
F_{\psi V}(\nu,\psi,V) & = & R^2\nu\left[\frac{V}{R^2},\psi\right] \\
F_{F\chi}(\nu,F,\chi) & = & \frac{F}{R^2}\left\langle R^2\nu,\chi\right\rangle \\
F_{\psi n}(\nu,\psi,\psi,n) & = & d_i\frac{\Delta^*\psi}{R^2 n}\left[\psi,R^2\nu\right] \\
F_{Fn}(\nu,F,F,n) & = & d_iR^2\nu F\left[\frac{1}{R^2 n},F\right] \\
F_{p_e n}(\nu,p_e,n) & = & d_iR^2\nu\left[\frac{1}{n},p_e\right] \\
F_{F\eta}(\nu,F,\eta) & = & -\frac{1}{R^2}\eta\left\langle R^2\nu,F\right\rangle
\end{array} \quad (19)$$

$$\begin{array}{lll}
P_p(\nu,\dot p) & = & \nu\dot p \\
P_{pU}(\nu,p,U) & = & \nu\,[U,p] \\
P_{p\chi}(\nu,p,\chi) & = & \Gamma p\,\langle\nu,\chi\rangle+(\Gamma-1)\nu\,\langle p,\chi\rangle \\
P_{p_e,F,n}(\nu,p_e,F,n) & = & d_i\left(\frac{1}{n}\nu\,[p_e,F]+\Gamma\nu p_e\left[\frac{1}{n},F\right]\right) \\
P_{\eta,\psi}(\nu,\eta,\psi,\psi) & = & (\Gamma-1)\nu\frac{(\Delta^*\psi)^2}{R^2} \\
P_{\eta,F}(\nu,\eta,F,F) & = & (\Gamma-1)\nu\frac{F^2}{R^2}
\end{array} \quad (20)$$

`Gravity`
These terms are obtained assuming a gravitational force of the form given by equation (**??**).

$$\begin{array}{lll}
U_g(\nu,n) & = & g_r\nu\,[n,R]-g_Zr\nu\,\langle n,R\rangle \\
X_g(\nu,n) & = & \frac{n}{R^2}\left(g_r\,\langle\nu,R\rangle+g_Zr\,[\nu,R]\right)
\end{array} \quad (21)$$

`Heat Flux Terms`
These terms are obtained assuming a heat flux density of the form described in section **??**.

$$\begin{array}{lll}
P_\kappa(\nu,\kappa,T) & = & -(\Gamma-1)\kappa\,\langle\nu,T\rangle \\
P_{\kappa_\parallel}(\nu,\kappa_\parallel,T,\psi,\psi,B^{-2}) & = & -(\Gamma-1)\kappa_\parallel\frac{1}{B^2}\,[\psi,\nu]\,[\psi,T] \\
P_{\kappa_\times}(\nu,\kappa_\times,T,F,B^{-2}) & = & (\Gamma-1)\kappa_\times\frac{F}{B}\,[\nu,T]
\end{array} \quad (22)$$

$$\begin{array}{lll}
T & = & p/n \\
B^2 & = & \dfrac{1}{R^2}\left[\langle\psi,\psi\rangle+F^2\right]
\end{array}$$

`Isotropic Viscosity`

These terms result from isotropic viscosity of the form given by equation (**??**).

$$
\begin{aligned}
U_{U\mu}(\nu, U, \mu) &= \tfrac{1}{R^2}\left[\left(\left\langle \mu, R^2\nu\right\rangle + \mu\Delta^*(R^2\nu)\right)\Delta^* U\right. \\
&\quad \left. + \nabla^2\mu\left\langle R^2\nu, U\right\rangle + \Delta^*(R^2\nu)\left\langle \mu, U\right\rangle\right] \\
U_{\chi\mu}(\nu, \chi, \mu) &= -\nabla^2(R^2\nu)\left[\mu, \chi\right] - \Delta^*\mu\left[R^2\nu, \chi\right] \\
&\quad - \tfrac{1}{R^2}\Delta^*(R^2\chi)\left[R^2\nu, \mu\right] \\
V_{V\mu}(\nu, V, \mu) &= \left[\langle \nu, \mu\rangle + \tfrac{1}{R^2}\mu\Delta^*(R^2\nu)\right]V \\
X_{U\mu}(\nu, U, \mu) &= \nabla^2\nu\left[\mu, U\right] + \nabla^2\mu\left[\nu, U\right] + \Delta^* U\left[\nu, \mu\right] \\
X_{\chi\mu}(\nu, \chi, \mu, \mu_c) &= \nabla^2\nu\left\langle \mu, \chi\right\rangle + \nabla^2\mu\left\langle \nu, \chi\right\rangle + 2\mu_c\nabla^2\nu\nabla^2\chi
\end{aligned}
\tag{23}
$$

`Parallel Viscosity`
These terms are obtained assuming a parallel viscosity of the form given in equation (**??**). These equations were obtained using equations (12). For compactness, derivatives are written as subscripts in the following expressions (*i.e.* $\nu_Z = \partial_Z\nu$).

$$
\begin{aligned}
U_{\Pi_\parallel U}(\nu, U) &= \mu_{\parallel_U} D_U \\
U_{\Pi_\parallel V}(\nu, V) &= \mu_{\parallel_V} D_U \\
U_{\Pi_\parallel \chi}(\nu, \chi) &= \mu_{\parallel_\chi} D_U
\end{aligned}
\tag{24}
$$

$$
\begin{aligned}
V_{\Pi_\parallel U}(\nu, U) &= \mu_{\parallel_U} D_V \\
V_{\Pi_\parallel V}(\nu, V) &= \mu_{\parallel_V} D_V \\
V_{\Pi_\parallel \chi}(\nu, \chi) &= \mu_{\parallel_\chi} D_V
\end{aligned}
\tag{25}
$$

$$
\begin{aligned}
X_{\Pi_\parallel U}(\nu, U) &= \mu_{\parallel_U} D_X \\
X_{\Pi_\parallel V}(\nu, V) &= \mu_{\parallel_V} D_X \\
X_{\Pi_\parallel \chi}(\nu, \chi) &= \mu_{\parallel_\chi} D_X
\end{aligned}
\tag{26}
$$

$$
\begin{aligned}
D_U &= \frac{3}{B^2}\left\{-\frac{1}{2}R^2\left[\nu, \frac{\langle\psi,\psi\rangle}{R^2}\right] + \langle \psi, [\nu, \psi]\rangle - \frac{1}{R^2}F^2\nu_Z\right. \\
&\quad \left. - \frac{2}{R^2}\left[\nu_Z(\psi_Z^2 - \psi_R^2) + 2\nu_r\psi_r\psi_Z\right]\right\} \\
D_V &= -3\frac{F}{B^2}\left[\nu, \psi\right] \\
D_X &= -\nabla^2\nu\left(1 - \frac{3}{R^2}\frac{\langle\psi,\psi\rangle}{B^2}\right) \\
&\quad + \frac{3}{R^2B^2}\left(\frac{1}{2}R^2\left\langle \nu, \frac{\langle\psi,\psi\rangle}{R^2}\right\rangle - \langle\psi, \langle\nu, \psi\rangle\rangle + \frac{1}{R}F^2\nu_r\right)
\end{aligned}
$$

$$
\begin{aligned}
\mu_{\|U} &= \eta_0 \frac{p_i \tau_i}{R^2 B^2} \left(-\frac{1}{2} R^2 \left[U, \frac{\langle \psi, \psi \rangle}{R^2} \right] + \langle \psi, [U, \psi] \rangle - \frac{1}{R^2} F^2 U_Z \right) \\
\mu_{\|V} &= -\eta_0 p_i \tau_i \frac{F}{B^2} \left[\psi, \frac{V}{R^2} \right] \\
\mu_{\|\chi} &= \eta_0 \frac{p_i \tau_i}{R^2 B^2} \left(\frac{1}{2} R^2 \left\langle \chi, \frac{\langle \psi, \psi \rangle}{R^2} \right\rangle - \langle \psi, \langle \chi, \psi \rangle \rangle + \frac{1}{R} F^2 \chi_r \right. \\
&\qquad \left. + \nabla^2 \chi \langle \psi, \psi \rangle \right)
\end{aligned}
$$

`Gyroviscosity`
These terms are obtained using equations (12) assuming a gyroviscosity of the form given by equation (**??**).

$$
U_{\Pi_\times U}(\nu, U) = -\frac{p_i F}{2R^3 B^2}
\times \left\{ \begin{array}{l}
\left(1 + \frac{3}{2R^2} \frac{\langle \psi, \psi \rangle}{B^2}\right) \left[\begin{array}{l} \left([R^3 \nu_Z]_Z - [R^3 \nu_r]_r\right) \left(\left[\frac{U_R}{R}\right]_Z + \left[\frac{U_Z}{R}\right]_r \right) \\ -\left([R^3 \nu_r]_Z + [R^3 \nu_Z]_r\right) \left(\left[\frac{U_Z}{R}\right]_Z - \left[\frac{U_R}{R}\right]_r \right) \end{array} \right] \\
+ \frac{9}{2rB^2} \\
\times \left[\begin{array}{l} \left(\psi_Z^2 - \psi_R^2\right) \left(r\nu_Z \left[\left(\frac{U_Z}{R}\right)_Z - \left(\frac{U_R}{R}\right)_r \right] - \frac{1}{R^3} U_Z \left[(R^3 \nu_Z)_Z - (R^3 \nu_r)_r \right] \right) \\ \quad + 2\psi_r \psi_Z \left(r\nu_Z \left[\left(\frac{U_R}{R}\right)_Z + \left(\frac{U_Z}{R}\right)_r \right] - \frac{1}{R^3} U_Z \left[(R^3 \nu_r)_Z + (R^3 \nu_Z)_r \right] \right) \end{array} \right]
\end{array} \right\}
$$

$$
U_{\Pi_\times V}(\nu, V) = -\frac{p_i}{B^2}
\times \left\{ \begin{array}{l}
\frac{1}{4R^2} \left(1 - \frac{3F^2}{B^2 R^2}\right) \\
\times \left(\left\langle \frac{V}{R^2}, R^4 [\psi, \nu] \right\rangle - \left\langle \psi, R^4 \left[\nu, \frac{V}{R^2}\right] \right\rangle + \frac{1}{R^2} \left[\nu, R^6 \left\langle \frac{V}{R^2}, \psi \right\rangle \right] \right) \\
- \frac{3}{4B^2} \left[\psi, \frac{V}{R^2}\right] \\
\times \left(2 \langle \psi, \langle \psi, \nu \rangle \rangle - R^2 \left\langle \nu, \frac{\langle \psi, \psi \rangle}{R^2} \right\rangle - \Delta^* \nu \langle \psi, \psi \rangle + 6 \psi_Z [\nu, \psi] \right) \\
+ \frac{9F^2}{2B^2 R^2} \nu_Z \left\langle \psi, \frac{V}{R^2} \right\rangle
\end{array} \right\}
$$

$$
U_{\Pi_\times \chi}(\nu, \chi) = -\frac{p_i F}{2R^3 B^2}
\times \left\{ \begin{array}{l}
\left[(\chi_{RR} - \chi_{ZZ}) \left([R^3 \nu_r]_r - [R^3 \nu_Z]_Z\right) + 2\chi_{RZ} \left([R^3 \nu_r]_Z + [R^3 \nu_Z]_r\right) \right] \\
+ \frac{3}{R^2 B^2} \left[\begin{array}{l} \left(\Delta^* \chi [\psi_Z^2 - \psi_R^2] - \chi_{ZZ} \psi_Z^2 + \chi_{RR} \psi_R^2\right) \left([R^3 \nu_r]_r - [R^3 \nu_Z]_Z\right) \\ + 2\chi_{RZ} \left(\psi_Z^2 [R^3 \nu_r]_Z + \psi_R^2 [R^3 \nu_Z]_r\right) \\ - 2\psi_r \psi_Z \left(\left[\chi_{ZZ} - \frac{1}{R}\chi_r\right] [R^3 \nu_r]_Z + \left[\chi_{RR} - \frac{1}{R}\chi_r\right] [R^3 \nu_Z]_r \right) \end{array} \right]
\end{array} \right\}
$$

$$
V_{\Pi_\times U}(\nu, U) = \frac{p_i}{4rB^2}
\times \left\{ \begin{array}{l}
\left(1 - \frac{3}{R^2} \frac{F^2}{B^2}\right) \left(\langle \psi, R [U, \nu] \rangle + \langle \nu, R [U, \psi] \rangle - \frac{1}{R^3} \left[U, R^4 \langle \nu, \psi \rangle \right] + U_r [\nu, \psi] + \frac{2}{R} \psi_Z \langle \nu, U \rangle \right) \\
+ \frac{3}{RB^2} [\psi, \nu] \left(2 \langle \psi, \langle U, \psi \rangle \rangle - \Delta^* U \langle \psi, \psi \rangle - \frac{1}{R^2} \left\langle U, R^2 \langle \psi, \psi \rangle \right\rangle + \left[\psi, R^2\right] [\psi, U] \right) \\
- \frac{18}{R^2} \frac{F^2}{B^2} [U, R] \langle \nu, \psi \rangle
\end{array} \right\}
$$

$$V_{\sqcap_\times V}(\nu, V) = \frac{p_i F R^2}{4B^2}\left(1 - \frac{3}{R^2}\frac{\langle\psi,\psi\rangle - F^2}{B^2}\right)\left[\nu, \frac{V}{R^2}\right]$$

$$V_{\sqcap_\times \chi}(\nu, \chi) = \frac{p_i}{B^2} \times \left\{ \begin{array}{l} \left(\frac{1}{R^2}\left\langle\chi, R^2\langle\nu,\psi\rangle\right\rangle - \langle\nu,\langle\chi,\psi\rangle\rangle - \langle\psi,\langle\nu,\chi\rangle\rangle\right) \\ + \frac{3}{2rB^2}[\psi,\nu]\left(\langle\psi, R[\chi,\psi]\rangle - \frac{1}{2}r[\chi,\langle\psi,\psi\rangle]\right) \\ + \frac{3}{4R^2}\frac{F^2}{B^2}\left(\langle\psi,\langle\chi,\nu\rangle\rangle + \langle\nu,\langle\chi,\psi\rangle\rangle - \langle\chi,\langle\nu,\psi\rangle\rangle - 2\Delta^*\chi\langle\nu,\psi\rangle\right) \end{array} \right\}$$

$$X_{\sqcap_\times U}(\nu, U) = \frac{p_i F}{2R^2B^2} \times \left\{ \begin{array}{l} \langle\langle\nu,U\rangle\rangle - R^2[[\nu,U]] + \frac{1}{R}\left[U_r(\nu_{ZZ} - \nu_{RR}) - 2U_Z\nu_{RZ} - \frac{1}{R}U_r\nu_r\right] \\ + \frac{3}{RB^2}\left[\begin{array}{l} \left(\left[\frac{U_Z}{R}\right]_Z - \left[\frac{U_R}{R}\right]_r\right)\left(\nu_{ZZ}\psi_R^2 - \nu_{RR}\psi_Z^2 + \frac{1}{R}\nu_r[\psi_Z^2 - \psi_R^2]\right) \\ + 2\nu_{RZ}\left(\left[\frac{U_R}{R}\right]_Z\psi_R^2 + \left[\frac{U_Z}{R}\right]_r\psi_Z^2 - \frac{1}{R^2}U_Z[\psi_Z^2 - \psi_R^2]\right) \\ - 2\psi_r\psi_Z\left(\begin{array}{l} \left[\frac{U_R}{R}\right]_Z\nu_{RR} + \left[\frac{U_Z}{R}\right]_r\nu_{ZZ} - \frac{1}{R^2}U_Z[\nu_{ZZ} - \nu_{RR}] \\ - \frac{1}{R}\nu_r\left[\left(\frac{U_R}{R}\right)_Z + \left(\frac{U_Z}{R}\right)_r\right] \end{array} \right) \end{array} \right] \end{array} \right\}$$

$$X_{\sqcap_\times V}(\nu, V) = \frac{p_i}{4B^2} \times \left\{ \begin{array}{l} \left(1 - \frac{3}{R^2}\frac{F^2}{B^2}\right)\left(\frac{1}{R^2}\left\langle\nu, R^2\left\langle\frac{V}{R^2},\psi\right\rangle\right\rangle - \left\langle\psi,\left\langle\frac{V}{R^2},\nu\right\rangle\right\rangle - \left\langle\frac{V}{R^2},\langle\psi,\nu\rangle\right\rangle\right) \\ + \frac{6}{B^2}\left[\psi,\frac{V}{R^2}\right]\left(\frac{1}{R}\langle\psi, R[\nu,\psi]\rangle - \frac{1}{2}[\nu,\langle\psi,\psi\rangle]\right) \\ - 6\frac{F^2}{B^2}\left\langle\psi,\frac{V}{R^2}\right\rangle\left[\left(\frac{\nu_Z}{R^2}\right)_Z + \left(\frac{\nu_R}{R^2}\right)_r\right] \end{array} \right\}$$

$$X_{\sqcap_\times \chi}(\nu, \chi) = -\frac{p_i F}{B^2} \times \left\{ \begin{array}{l} \left(1 + \frac{3}{2R^2}\frac{\langle\psi,\psi\rangle}{B^2}\right)[\langle\nu,\chi\rangle] \\ + \frac{3}{2B^2}\left[\begin{array}{l} \left(-\frac{1}{2}[\chi,\langle\psi,\psi\rangle] + \frac{1}{R}\langle\psi, R[\chi,\psi]\rangle\right)\left(\left[\frac{\nu_R}{R^2}\right]_r + \left[\frac{\nu_Z}{R^2}\right]_Z\right) \\ -\left(-\frac{1}{2}[\nu,\langle\psi,\psi\rangle] + \frac{1}{R}\langle\psi, R[\nu,\psi]\rangle\right)\left(\left[\frac{\chi_R}{R^2}\right]_r + \left[\frac{\chi_Z}{R^2}\right]_Z\right) \end{array} \right] \end{array} \right\}$$

8.1.3 Spatial Integration

The integrals required to calculate the weak-form equations of the Galerkin method are computed numerically using a 79-point Gaussian quadrature. That is, the value of each field is calculated at 79 points for each triangular element, and a weighted sum of these values is computed to approximate the integral.

$$\int dA\ f(x) \simeq \sum_{i=1}^{79} w_i f(x_i),$$

where the integrand $f(x)$ is restricted to a single element. The sampling points and weights appropriate for a equilateral triangle are taken from ref. [**?**].

The coordinates of the sampling points are given in the "natural coordinates" (α, β, γ) in ref. [**?**]. These coordinates may be converted to cartesian coordinates (r, Z) for an equilateral triangle e having vertices

$$\left\{ \left(-\frac{\sqrt{3}}{2}, -\frac{1}{2} \right), \left(\frac{\sqrt{3}}{2}, -\frac{1}{2} \right), (0, 1) \right\}$$

using the linear transformation

$$\phi_{n \to e}(\alpha, \beta, \gamma) = \left(\frac{\sqrt{3}}{2}(\beta - \gamma), \frac{1}{2}(3\alpha - 1) \right).$$

The weights must be multiplied by the Jacobian of this transformation,

$$\mathcal{J}_{\phi_{n \to e}} = \frac{3\sqrt{3}}{4}.$$

To find the coordinates of the sampling points for a general triangle g having vertices $\{(-b, 0), (a, 0), (0, c)\}$, as in ref. [**?**], one may use the linear transformation

$$\begin{aligned}
\phi_{e \to g}(r, Z) &= \left(\frac{a+b}{\sqrt{3}} x + \frac{a-b}{3}(1-y), \frac{c}{3}(2y+1) \right) \\
\mathcal{J}_{\phi_{e \to g}} &= \frac{2c}{3\sqrt{3}}(a+b).
\end{aligned}$$

The transformation from natural coordinates to cartesian coordinates for a triangle having vertices $\{(-b, 0), (a, 0), (0, c)\}$ is therefore

$$\begin{aligned}
\phi_{n \to g}(r, Z) &= \left(\frac{1}{2}(a+b)(\beta - \gamma) + \frac{1}{2}(a-b)(1-\alpha), c\alpha \right) \\
\mathcal{J}_{\phi_{n \to g}} &= \frac{1}{2}(a+b)c.
\end{aligned}$$

The 79-point quadrature gives the exact results for integrands which are polynomials of degree 20 (or less). In the case of quintic finite elements, this means the integration is exact for terms involving products of three fields or fewer, not including the degree-five trial function ν. In cylindrical geometry, the presence factors of $1/R$ will cause the quadrature not to be exact, as $1/R$ is not in the form of a polynomial. The weights w_i must also be multiplied by R_i in cylindrical coordinates to account for the Jacobian of the transformation from cartesian to cylindrical coordinates.

8.2 Time Step

8.2.1 Implicit Time Advance

For the implicit time advance, equations (13) are evaulated at the θ-advanced time (*e.g.* $F(\psi) \to F(\psi + \theta \delta t \dot{\psi} + \cdots)$), linearized (*i.e.* $\mathcal{O}(\delta t^2)$ and higher are dropped), and then discretized temporally according to the chosen time integration method (*i.e.* $\dot{\psi} \to (\psi^{(n+1)} - \psi^{(n)})/\delta t$).

$$\begin{pmatrix} S^v_{11} & R^v_{11} & S^v_{12} & R^v_{12} & S^v_{13} & 0 & R^v_{14} & R^v_{13} \\ R^B_{11} & S^B_{11} & R^B_{12} & S^B_{12} & R^B_{13} & S^B_{13} & 0 & 0 \\ S^v_{21} & R^v_{21} & S^v_{22} & R^v_{22} & S^v_{23} & 0 & R^v_{24} & R^v_{23} \\ R^B_{21} & S^B_{21} & R^B_{22} & S^B_{22} & R^B_{23} & S^B_{23} & 0 & 0 \\ S^v_{31} & R^v_{31} & S^v_{32} & R^v_{32} & S^v_{33} & 0 & R^v_{34} & R^v_{33} \\ R^B_{31} & S^v_{31} & R^B_{32} & S^v_{32} & R^B_{33} & S^v_{33} & 0 & 0 \\ R^n_{31} & 0 & R^n_{32} & 0 & R^n_{33} & 0 & S^n & 0 \\ R^p_{31} & 0 & R^p_{32} & 0 & R^p_{33} & 0 & 0 & S^p \end{pmatrix} \begin{pmatrix} U \\ \psi \\ V \\ F \\ \chi \\ p_e \\ n \\ p \end{pmatrix}^{(n+1)} =$$

$$\begin{pmatrix} D^v_{11} & Q^v_{11} & D^v_{12} & Q^v_{12} & D^v_{13} & 0 & Q^v_{14} & Q^v_{13} \\ Q^B_{11} & D^B_{11} & Q^B_{12} & D^B_{12} & Q^B_{13} & D^B_{13} & 0 & 0 \\ D^v_{21} & Q^v_{21} & D^v_{22} & Q^v_{22} & D^v_{23} & 0 & Q^v_{24} & Q^v_{23} \\ Q^B_{21} & D^B_{21} & Q^B_{22} & D^B_{22} & Q^B_{23} & D^B_{23} & 0 & 0 \\ D^v_{31} & Q^v_{31} & D^v_{32} & Q^v_{32} & D^v_{33} & 0 & Q^v_{34} & Q^v_{33} \\ Q^B_{31} & D^v_{31} & Q^B_{32} & D^v_{32} & Q^B_{33} & D^v_{33} & 0 & 0 \\ Q^n_{31} & 0 & Q^n_{32} & 0 & Q^n_{33} & 0 & D^n & 0 \\ Q^p_{31} & 0 & Q^p_{32} & 0 & Q^p_{33} & 0 & 0 & D^p \end{pmatrix} \begin{pmatrix} U \\ \psi \\ V \\ F \\ \chi \\ p_e \\ n \\ p \end{pmatrix}^{(n)} + \begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \\ Q_7 \\ Q_8 \end{pmatrix} \qquad (27)$$

8.2.2 Split Time Step Method

Time is advanced using a split time-step method in which the velocity field is advanced first, then the density and total pressure fields are advanced separately, and finally the magnetic field and electron pressure are advanced together. Though the velocity and magnetic field are advanced separately, the Alfvén and magnetosonic waves are treated implicitly by using equations (**??**–**??**) to calculate analytically the advanced-time values of the pressure and magnetic field for use in the velocity time step.

$$\begin{pmatrix} S^v_{11} & S^v_{12} & S^v_{13} \\ S^v_{21} & S^v_{22} & S^v_{23} \\ S^v_{31} & S^v_{32} & S^v_{33} \end{pmatrix} \begin{pmatrix} U \\ V \\ \chi \end{pmatrix}^{(n+1)} \qquad (28)$$
$$= \begin{pmatrix} D^v_{11} & D^v_{12} & D^v_{13} \\ D^v_{21} & D^v_{22} & D^v_{23} \\ D^v_{31} & D^v_{32} & D^v_{33} \end{pmatrix} \begin{pmatrix} U \\ V \\ \chi \end{pmatrix}^{(n)} + \begin{pmatrix} Q^v_{11} & Q^v_{12} & Q^v_{13} \\ Q^v_{21} & Q^v_{22} & Q^v_{23} \\ Q^v_{31} & Q^v_{32} & Q^v_{33} \end{pmatrix} \begin{pmatrix} \psi \\ F \\ p \end{pmatrix}^{(n)}$$
$$+ \begin{pmatrix} O^v_1 \\ O^v_2 \\ O^v_3 \end{pmatrix}$$

$$
\begin{aligned}
S^n n^{(n+1)} &= D^n n^{(n)} + \begin{pmatrix} R_1^n & R_2^n & R_3^n \end{pmatrix} \begin{pmatrix} U \\ V \\ \chi \end{pmatrix}^{(n+1)} \\
&\quad + \begin{pmatrix} Q_1^n & Q_2^n & Q_3^n \end{pmatrix} \begin{pmatrix} U \\ V \\ \chi \end{pmatrix}^{(n)}
\end{aligned} \tag{29}
$$

$$
\begin{aligned}
S^p p^{(n+1)} &= D^p p^{(n)} + \begin{pmatrix} R_1^p & R_2^p & R_3^p \end{pmatrix} \begin{pmatrix} U \\ V \\ \chi \end{pmatrix}^{(n+1)} \\
&\quad + \begin{pmatrix} Q_1^p & Q_2^p & Q_3^p \end{pmatrix} \begin{pmatrix} U \\ V \\ \chi \end{pmatrix}^{(n)}
\end{aligned} \tag{30}
$$

$$
\begin{aligned}
&\begin{pmatrix} S_{11}^B & S_{12}^B & S_{13}^B \\ S_{21}^B & S_{22}^B & S_{23}^B \\ S_{31}^B & S_{32}^B & S_{33}^B \end{pmatrix} \begin{pmatrix} \psi \\ F \\ p_e \end{pmatrix}^{(n+1)} \\
&= \begin{pmatrix} D_{11}^B & D_{12}^B & D_{13}^B \\ D_{21}^B & D_{22}^B & D_{23}^B \\ D_{31}^B & D_{32}^B & D_{33}^B \end{pmatrix} \begin{pmatrix} \psi \\ F \\ p_e \end{pmatrix}^{(n)} + \begin{pmatrix} R_{11}^B & R_{12}^B & R_{13}^B \\ R_{21}^B & R_{22}^B & R_{23}^B \\ R_{31}^B & R_{32}^B & R_{33}^B \end{pmatrix} \begin{pmatrix} U \\ V \\ \chi \end{pmatrix}^{(n+1)} \\
&\quad + \begin{pmatrix} Q_{11}^B & Q_{12}^B & Q_{13}^B \\ Q_{21}^B & Q_{22}^B & Q_{23}^B \\ Q_{31}^B & Q_{32}^B & Q_{33}^B \end{pmatrix} \begin{pmatrix} U \\ V \\ \chi \end{pmatrix}^{(n)} + \begin{pmatrix} O_1^B \\ O_2^B \\ O_3^B \end{pmatrix}
\end{aligned} \tag{31}
$$

`Linear Calculations`

Linear calculations may be performed by calculating each matrix once, and recalculating the matrix-vector products each time step with the updated vectors. This method is very efficient because the S matrices need only be inverted once, and in all subsequent time steps the only matrix operations carried out are addition and matrix-vector multiplication. Non-linear simulations require all the matrices to be recalculated each time step, and the S matrices must be inverted each time step.

`Implementation of Electron Pressure Equation`

Because it is the electron pressure which appears in the generalized Ohm's law, equation (**??**), if the electron pressure equation is retained, it is solved with the magnetic field as the third row in equation (31) so as to keep the fast magnetosonic wave implicit. In this case, the full pressure is evolved independently in equation (30). If the electron pressure equation is not included, the third row in equation (31) is the total pressure equation, and the electron pressure is assumed to remain always at a specific fraction of the total pressure, which is determined by the initial conditions.

8.2.3 Crank-Nicholson

The Crank-Nicholson time step is defined by the following discretization:

$$\begin{aligned} \frac{\partial U}{\partial t} &\rightarrow \frac{U^{(n+1)} - U^{(n)}}{\delta t} \\ U &\rightarrow \theta U^{(n+1)} + (1-\theta) U^{(n)}. \end{aligned}$$

By Taylor expanding about U,

$$\begin{aligned} U^{(n+1)} &= U + \theta\, \delta t\, \dot{U} + \frac{1}{2}\theta^2 \delta t^2 \ddot{U} + \frac{1}{6}\theta^3 \delta t^3 \dddot{U} + \cdots \\ U^{(n)} &= U + (\theta - 1)\delta t\, \dot{U} + \frac{1}{2}(\theta-1)^2 \delta t^2 \ddot{U} + \frac{1}{6}(\theta-1)^3 \delta t^3 \dddot{U} + \cdots \end{aligned}$$

the trunctation error of the time-derivative operator can be calculated directly:

$$\begin{aligned} \Delta_{CN}(\delta t, \theta) &= \frac{U^{(n+1)} - U^{(n)}}{\delta t} - \dot{U} \\ &= \left(\theta - \frac{1}{2}\right) \delta t\, \ddot{U} + \frac{1}{2}\left(\theta^2 - \theta + \frac{1}{3}\right) \delta t^2 \dddot{U} + \cdots . \end{aligned}$$

When $\theta = 1/2$, the time differencing is "time-centered" because the two points involved in the time differencing are equidistant in the logical time coordinate from the point at which the field itself is evaluated. In this case, the leading-order truncation error is $\mathcal{O}(\delta t^2)$:

$$\Delta_{CN}(\delta t, \theta = 1/2) = \frac{1}{24}\delta t^2 \dddot{U} + \cdots . \tag{32}$$

8.2.4 BDF2

The BDF2 time step is defined by the following discretization:

$$\begin{aligned} \frac{\partial U}{\partial t} &\rightarrow \frac{3U^{(n+1)} - 4U^{(n)} + U^{(n-1)}}{2\,\delta t} \\ U &\rightarrow U^{(n+1)}. \end{aligned}$$

Taylor expanding about U:

$$\begin{aligned} U^{(n+1)} &= U \\ U^{(n)} &= U - \delta t\, \dot{U} + \frac{1}{2}\delta t^2 \ddot{U} - \frac{1}{6}\delta t^3 \dddot{U} + \cdots \\ U^{(n-1)} &= U - 2\,\delta t\, \dot{U} + 2\,\delta t^2 \ddot{U} - \frac{4}{3}\delta t^3 \dddot{U} + \cdots \end{aligned}$$

and the truncation error is

$$\Delta_{BDF2}(\delta t) = \frac{3U^{(n+1)} - 4U^{(n)} + U^{(n-1)}}{2\,\delta t} - \dot{U} = -\frac{1}{3}\delta t^2 \dddot{U} + \cdots . \tag{33}$$

9 Input Parameters

9.1 Model Options

Option	Default	Description
`numvar`	3	MHD model. 1: 2-field; 2: 4-Field; 3: 6-Field.
`linear`	0	1: linear (perturbation terms only, no matrix recalculation)
`eqsubtract`	0	1: remove equilibrium terms from equations
`icsubtract`	0	set to 1 if PF coils are in the domain. These are defined in the files "coils.dat" and "current.dat"
`extsubtract`	0	1: subtract fields from non-axisymmetric coils
`idens`	1	1: include density equation
`ipres`	0	1: include electron pressure equation
`ipressplit`	0	1: seperate pressure solve from the magnetic field solves when isplitstep=1. (ipressplit must be 0 for isplitstep=0)
`itemp`	0	1: Advance temperatures rather than pressures (for ipressplit=1 only)
`gyro`	0	1: include Braginskii gyroviscous term. (note: needs db to be nonzero also)
`igauge`	0	0: loop voltage applied to boundary psi only
`inertia`	1	1: Include $\mathbf{u} \cdot \nabla \mathbf{u}$ terms
`itwofluid`	1	1: Include $\mathbf{J} \times \mathbf{B}$ and ∇p_e terms in Ohm's law (electron form). 2: ion form (not recommended) 3: parallel pressure gradient in Ohm's law only (not recommended)
`ibootstrap`	0	1: include bootstrap current
`ibootstrap_model`	0	1: J_BS = alpha F ¡p,psi¿ B
`bootstrap_alpha`	0	alpha parameter in bootstrap current model
`imp_bf`	0	1: include implicit equation for f (recommended for 3D and 2D complex)
`nosig`	0	1: drop sigma terms from momentum equation
`itor`	0	0: cartesian; 1: cylindrical
`istatic`	0	1: Do not advance velocity
`iestatic`	0	1: Do not advance magnetic fields
`chiiner`	1.	factor to multiply the chi equation inertial terms
`ieq_bdotgradt`	1.	1: include equilibrium parallel T gradient
`no_vdg_T`	0	1: do notinclude V dot grad T in Temp equation (debug)
`iwall_is_limiter`	1	1: wall acts as limiter
`kinetic`	0	1: Use kinetic PIC for hot pressure, 2: Incompressible CGL, 3. Full CGL

Option	Default	Description
`iadiabat`	1	1: Corrects several problems with itemp=1 option
`irunaway`	0	1: include runaway electron model
`cre`	0	runaway speed
`imp_temp`	0	0: compute temperatures for isplitstep=0, itemp=0
`iohmic_heating`	1	1: Include Ohmic heating term in heating
`irad_heating`	1	1: Include radiation heat sink
`gravr`	0	gravitational acceleration in R-direction
`gravz`	0	gravitational acceleration in Z-direction

9.2 Initial Conditions Options

Option	Default	Description
`itaylor`	0	Pre-defined initial conditions. **for itor=1 (toroidal geometry)** 0: Tilting cylinder 1: Calls Grad-Shafranov solver 2: magneto-rotational equilibrium 3: rotational instability 40: Fixed boundary stellarator 41: Free boundary stellarator **for itor=0 (slab geometry)** 0: Tilting cylinder 1: Taylor Reconnection 2: Force-Free equilibrium (Taylor state) 3: GEM Reconnection 4: Wave Propagation 5: Gravitational Instability 6: Strauss equilibrium 7: circular field init 8,9: biharmonic 10,11,12,13:: analytic RWM test problem 14: 3D wave test 15: 3D diffusion test 16: FRS cylindrical equilibrium 17: ftz init 18: eigen init 19: ASDEX profiles similar to YU's 20: kstar profiles with multiple q=1 surfaces 21,22: fixed q(r) and p(r) profiles 23: Startsev equilibrium with $J = (1/R_0 q_0)(1 - r^2)$ 27: cylindrical test problem 29: basicJ profiles
`iupstream`	0	1: addsdiffusion term to convection-like upstream differencing
`magus`	5.e-2	magnitude of the upstream diffusion term
`iflip`	0	1: Flip coordinate system handedness
`iflip_b`	0	1: Flip sign of toroidal field
`iflip_j`	0	1: Flip sign of toroidal current
`iflip_v`	0	1: Flip sign of toroidal velocity
`iflip_z`	0	1: Flip equilibrium across z=0 plane

Option	Default	Description
`icsym`	0	Symmetry of random perturbations 0: No symmetry 1: Odd up-down symmetry (in U) 2: Even up-down symmetry (in U)
`bzero`	1	B_φ at `rzero`
`bx0`	0	Initial field in x-direction for some test problems
`vzero`	0	Initial toroidal velocity for some test problems
`phizero`	0	Initial velocity stream function for some test problems
`v0_cyl`	0	Central toroidal velocity for some test problems
`v1_cyl`	0	VZ=v0_cyl + v1_cyl*psin**beta
`idevice`	0	define coils for a particular device -1: reads coil.dat file 0: generic dipole configuration 1: CDX-U 2: NSTX 3: ITER 4: DIII-D
`iwave`	0	defines what wave to initialize in wave propagation test
`eps`	0.01	Size of random perturbation
`maxn`	200	Maximum wavenumber of initial random noise
`verzero`	0	magnitude of initial vertical velocity
`irmp`	0	1: apply nonaxisymmetric fields throughout plasma. reads rmp_coil.dat for (R,Z) of window pane coils. reads rmp_current.dat for (+-) currents in kA and phases in degrees. toroidal mode number of current specified by ntor. requires `type_ext_field` = 0. 2: apply nonaxisymmetric fields only at boundaries.
`type_ext_field`	-1	External field type 0: RMP or error field for tokamak geometry. 1: For free boundary stellarator only: FIELDLINES or MGRID.
`rmp_atten`	0	additional exponential decay of RMP field from r=1 for irmp=2
`iread_ext_field`	0	1: read external field
`beta`	0	parameter used in some model equilibrium initializations
`ln`	0	length scale parameter used in some model equilibrium
`elongation`	1	elongation used in Solovev equilibrium

Option	Default	Description
`isample_ext_field`	1	factor to down sample external field data toroidally
`isample_ext_field_pol`	1	factor do down sample external field data poloidally
`scale_ext_field`	1	factor to scale external field
`shift_ext_field`	0	toroidal shift (in deg) of external fields
`ibasicj_solvep`	0	0: uniform p, solve for F; 1: uniformF, solve for p
`basicj_nu`	1	exponent in basicj equilibrium
`basicj_j0`	1	On-axis current density in basicj equilibrium
`basicj_voff`	1	Radial extent of flat toroidal rotation in basicj equilibrium
`basicj_vdelt`	1	Width of velocity drop-off, as fraction of ln, in basicj equilibrium
`basicj_dexp`	1	parameter for basicj equilibrium
`basicj_dvac`	1	parameter for basicj equilibrium
`basicj_q0`	0	parameter for basicj equilibrium
`basicj_qa`	0	parameter for basicj equilibrium
`pf_shift`	0	(array) horizontal shift of PF coil
`pf_shift_angle`	0	(array) direction of PF shift in degrees
`pf_tilt`	0	(array) Angle of PF from vertical in degrees
`pf_tilt_angle`	0	(array) Axis of rotation for PF tilt in degrees
`tf_shift`	0	horizontal shift of TF coils
`tf_shift_angle`	0	direction of TF shift in degrees
`tf_tilt`	0	angle of TF from vertical in degrees
`tf_tilt_angle`	0	axis of rotation for TF tilt in degrees

9.3 Grad-Shafranov Solver Options

Option	Default	Description
`inumgs`	0	1: Use numerical def. of p and g from profile-p and profile-g files
`igs`	80	Max number of Picard iterations
`eta_gs`	1000.	factor for smoothing nonaxisymmetries in psi in 3D GS solve
`igs_pp_ffp_rescale`	0	1: rescale p' and FF' to match p and F
`nv1equ`	0	1:use numvar =1 equilibrium for numvar .GT. 1
`tcuro`	1	(scaled) plasma current in GS equilibrium
`xmag`	1	R-coordinate of magnetic axis
`zmag`	0	Z-coordinate of magnetic axis
`xmag0`	0	if nonzero, target magnetic axis R for feedback
`zmag0`	0	if nonzero, target magnetic axis Z for feecback
`xlim`	0	R-coordinate of limiter
`zlim`	0	Z-coordinate of limiter
`xlim2`	0	R-coordinate of limiter #2
`zlim2`	0	Z-coordinate of limiter #2
`rzero`	1	nominal major radius of device for itor=1
`libetap`	1.2	approximate value of $l_i/2 + \beta_P$ for free-boundary equ
`p0`	0.01	Pressure at magnetic axis
`pi0`	0.005	Ion pressure at magnetic axis
`p1`	0	$p'(\Psi)$ at magnetic axis
`p2`	0	$p''(\Psi)$ at magnetic axis
`pedge`	0	Pressure in vacuum region
`tedge`	0	temperature in vacuum region (if .GT. 0). Only used in GS solve. Boundary value of electron temp is $twall = pedge \times pefac/den\_edge$
`tiedge`	0	ion temperature in vacuum region
`expn`	0	Fraction of pressure gradient due to density gradient: $n = p^{\texttt{expn}}$.
`q0`	1	Safety factor at magnetic axis
`djdpsi`	0	$J'_\varphi(\Psi)$ at magnetic axis
`th_gs`	0.8	implicitness of GS Picard iterations
`tol_gs`	10^{-8}	convergence criteria for GS iteration
`pscale`	1.	factor multiplying pressure profile
`bscale`	1.0	Factor multipying toroidal field
`bpscale`	1.0	Factor multiplying F' (keeping F0 constant)
`vscale`	1.0	Factor multiplying toroidal rotation profile
`iread_bscale`	0	1: read profile_bscale for factor to scale F
`iread_pscale`	0	1: read profile_pscale for factor to scale p and p'

Option	Default	Description
batemanscale	1	Bateman scale the TF, keeping curent profile fixed
irot	0	1: include toroidal rotation in equilibrium calculatin
iscale_rot_by_p	1	see below
alpha0	0	α_0 in analytic rotation profile
alpha1	0	α_1 in analytic rotation profile
alpha2	0	α_2 in analytic rotation profile
alpha3	0	α_3 in analytic rotation profile

For iread_omega=0, the function $\alpha(\psi)$ is parameterized by:

$$\tilde{\alpha} = \alpha_0 + \alpha_1 s + \alpha_2 s^2 + \alpha_3 s^3$$

For iscale_rot_by_p = 0: $\alpha = \tilde{\alpha} \times n(\psi)/p(\psi)$.
For iscale_rot_by_p = 1: $\alpha = \tilde{\alpha}$.
For iscale_rot_by_p = 2: $\alpha = \left[\alpha_0 + \alpha_1 e^{-[(\psi-\alpha_2)/\alpha_3]^2}\right] \times n(\psi)/p(\psi)$
In all cases, the angular velocity is then determined by:

$$\omega = \left[\frac{2\alpha p(\psi)}{R_0^2 n(\psi)}\right]^{\frac{1}{2}}$$

idenfunc	0	0: $n = \text{den0} \times (p/p0)^{\text{expn}} + \text{denedge}$ 1: $n = \text{den0} \times \frac{1}{2} \times \left[1 + \tanh\left(\frac{\psi - (\psi_B + n_O(\psi_B - \psi_M))}{\Delta \times (\psi_B - \psi_M)}\right)\right]$ 2: $n = \text{den0} + \frac{1}{2}(\text{den\_edge} - \text{den0}) \times \left[1 + \tanh\left(\frac{\tilde{\psi} - n_O}{\Delta}\right)\right]$ 3: if $\tilde{\psi}$ .LT. n_O and $(\psi - \psi_M) \times [d\psi/dx(x - x_{MA}) + d\psi/dz(z - z_{MA})]$ .GT. 0, then n = den0. Else, n = den_edge. ($\psi_B = \text{psibound}, \psi_M = \text{psimin}, \tilde{\psi} = (\psi - \psi_M)/(\psi_B - \psi_M)$)
den_edge	0.0	edge density. If 0, set to den0*(pedge/p0)**expn
den0	1.0	(scaled) central density
denoff	1.0	n_O: offset for idenfunc=1,2,3
dendelt	0.1	Δ: width of transition region for idenfunc=1,2
divertors	0	Number of divertors (0–2)
divcur	0.1	Divertor current(s), as fraction of tcuro
xdiv	0	r-coordinate of divertor current(s)
zdiv	0	z-coordinate of divertor current. If divertors = 2, the second divertor has $z = -$zdiv.
xnull	0	Guess for r-coordinate of x-point
znull	0	Guess for z-coordinate of x-point
mod_null_rs	0	if 1: you can reset xnull and znull from C1input
xnull0	0	Target R-Coordinate of x-point for feedback
znull0	0	Target Z-Coordinate of x-point for feedback
xnull2	0	Guess for r-coordinate of inactive x-point
znull2	0	Guess for z-coordinate of inactive x-point
mod_null_rs2	0	if1: you can reset xnull2 and znull2 from C1input

Option	Default	Description
`gs_pf_psi_width`	0	width of psi smoothing into provate flux region
`gs_vertical_feedback`	0	proportional feedback of each coil to (zmag-zmag0) (array)
`gs_vertical_feedback_i`	0	integral feedback of each coil to (zmag-zmag0) (array)
`gs_vertical_feedback_x`	0	proportional feedback of each coil to (znull-znull0) (array)
`gs_vertical_feedback_x_i`	0	integral feedback of each coil to (znull-znull0) (array)
`gs_radial_feedback`	0	proportional feedback of each coil to (xmag-xmag0) (array)
`gs_radial_feedback_i`	0	integral feedback of each coil to (xmag-xmag0) (array)
`gs_radial_feedback_x`	0	proportional feedback of each coil to (xnull-xnull0) (array)
`gs_radial_feedback_x_i`	0	integral feedback of each coil to (xnull-xnull0) (array)
`igs_extend_p`	0	1: extend p past pls=1 using ne and Te profiles
`igs_feedfac`	1	proportionality factor for external field feedback
`igs_forcefree_lcfs`	-1	1: ensure that GS solution is force free at LCFS
`igs_start_xpoint_search`	0	number of GS iterations before searching for x-point
`sigma0`	0	width of Gaussian for initial current distribution for GS iteration
`igs_extend_diagmag`	1	1: extend diamagnetic rotation past psi=1
`adapt_qs`	0	Safety factor values to pack around (array)
`adapt_zlow`	0	Z-coordinate below which SOL adaption is coarse
`adapt_zup`	0	Z-coordinate above which SOL adaptation is coarse

9.4 Transport Coefficients

Option	Default	Description
`ivisfunc`	0	select viscosity function 0: visc = amu 1: visc = amu + $\frac{1}{2}$amu_edge $\times \left[1. + \tanh\left[\frac{\psi-(\psi_l+\nu_0(\psi_l-\psi_0))}{\nu_\Delta(\psi_l-\psi_0)}\right]\right]$ 2: visc = amu + $\frac{1}{2}$amu_edge $\times \left[1. + \tanh\left[\frac{\tilde{\psi}-\nu_0}{\nu_\Delta}\right]\right]$ or, if amuoff2 .ne. 0 and amudelt2.ne.0) visc = amu+$\frac{1}{4}$amu_edge$\times \left[2. + \tanh\left[\frac{\tilde{\psi}-\nu_0}{\nu_\Delta}\right] + \tanh\left[\frac{\tilde{\psi}-\nu_{02}}{\nu_{\Delta 2}}\right]\right]$ 3: visc = amu or amu_edge depending on criteria in define_fields
`amu`	0	core viscosity for ivisfunc =0,..,3
`amu_edge`	0	edge viscosity for ivisfunc = 1,..,3
`amuoff`	0	ν_0 in ivisfunc = 1,2
`amuoff2`		ν_{02} in ivisfunc = 1,2
`amudelt`	0	ν_Δ in ivisfunc = 1,2
`amudelt2`	0	$\nu_{\Delta 2}$ in ivisfunc = 1,2
`amuc`	0	Compressional viscosity coefficient
`amupar`	0	Parallel viscosity coefficient
`amue`	0	bootstrap viscosity coefficient
`iresfunc`	0	select resistivity function 0: eta = etar + eta0/Te**(3/2) 1: eta = etar + $\frac{1}{2}$eta0 $\times \left[1. + \tanh\left[\frac{\psi-(\psi_l+\text{etaoff}\times(\psi_l-\psi_0))}{\text{etadelt}\times(\psi_l-\psi_0)}\right]\right]$ 2: eta = etar + $\frac{1}{2}$eta0 $\times \left[1. + \tanh\left[\frac{\tilde{\psi}-\text{etaoff}}{\text{etadelt}}\right]\right]$ The following two options are applied in a way that they should not have negative values...even if the idl plots indicate otherwise 3: eta = etar for $\tilde{\psi}$ < etaoff othrwise eta0 4: Spitzer resistivity with offset. Define T_{wall} = pedge*pefac/den_edge for$T_e > T_{wall} - T_e^{off}, \eta = (T_e - T_e^{off})^{-3/2}$ for$T_e < T_{wall} - T_e^{off}, \eta = (T_{wall} - T_e^{off})^{-3/2}$ can be increased by inputing eta_fac > 1. 5: Simple neoclassical model: η = eta0 $\times (n_e/p_e)^{3/2}/(1. - 1.46(r/R)^{1/2})$
`etar`	0	see description of iresfunc
`eta0`	0	see description of iresfunc
`etaoff`	0	see description of iresfunc
`etadelt`	0	see description of iresfunc
`eta_te_offset`	0	T_e^{off} for iresfunc=4
`ikprad_te_offset`	0	if 1, T_e^{off} also used in kprad and ablation
`eta_fac`	0	for iresfunc=4, resistivity multiplied by eta_fac
`eta_mod`	0	if 1: remove d/dphi terms in resistivity

Option	Default	Description
`eta_max`	0	maximum resistivity in plasma (defaults to etavac)
`eta_min`	0	minimum resistivity in plasma
`ikappafunc`	0	select thermal conductivity function
		0: $\kappa = \text{kappat} + \text{kappa0} \times *(n^3/p)^{1/2}$
		1: $\kappa = \text{kappa0} \times \frac{1}{2}\left[1 + \tanh\left[\frac{\psi-\left(\psi_l+\kappa^{0ff}\times(\psi_l-\psi_0)\right)}{\kappa_\Delta\times(\psi_l-\psi_0)}\right]\right]$
		2: $\kappa = \text{kappa0} \times \frac{1}{2}\left[1 + \tanh\frac{\tilde{\psi}-\kappa^{off}}{\kappa_\Delta}\right] \text{for} \tilde{\psi} < 1$
		2: $\kappa = \text{kappa0} \times \frac{1}{2}\left[1 + \tanh\frac{2-\tilde{\psi}-\kappa^{off}}{\kappa_\Delta}\right] \text{for} \tilde{\psi} > 1$
		3: $\kappa = \text{kappat} + \text{kappa0} \times 1/(pn)^{1/2}$
		4: $\kappa = \text{kappat} + \text{kappa0} \times (1. + \text{kappadelt} \times \|\nabla T_e\|^2)$
		5: $\kappa = \text{kappat} + \text{kappa0}/T_e$ limited by kappa_max
		10: read from profile_kappa file in m^2/sec
		11: read from profile_kappa file in normalized units
		12: option to go with itaylor=27
`kappa_max`	0	if .NE. 0, max κ for ikappafunc=5
`kappai_fac`	1	ion thermal conduction is kappai_fac* kappa
`ikapscale`	0	if 1: kappar gets scaled by kappa
`ikappar_ni`	0	1: include 1/n terms in parallel heat flux
`kappaoff`	0	κ^{off} see ikappafunc
`kappadelt`	0	κ_Δ see ikappafunc
`kappat`	0	isotropic thermal conductivity
`kappa0`	0	see ikappafunc
`ikapparfunc`	0	select parallel thermal conductivity (PTC) function
		0: PTC = kappar
		1: PTC = kappar/ $\left[(T_{crit}/T)^{5/2} + 1\right]$
`kappar`	0	Parallel thermal conductivity
`tcrit`	0	T_{crit} for ikapparfunc = 1
`kappari_fac`	1	ion parallel thermal conductivity is kappari_fac x electron value
`kappax`	0	coefficient of $B \times \nabla T$ temperature diffusion
`kappah`	0	if nonzero, kappa = kappah $\times \tanh^2\left[(\tilde{\psi} - 1.)/2\right]$
`kappaf`	1	Factor multiplying kappa when $\nabla p < \nabla p_{crit}$
`kappag`	0	Thermal diffusion proportional to pressure gradient
`gradp_crit`	0	∇p_{crit} for kappaf,kappag model
`k_fac`	1	Factor by which TF is multiplied in denominator of kappa_par
`temin_q0`	0	Min temperature used in equipartition for ipres=1
`idenmfunc`	0	selects from of particle diffusion (PD)
		0: PD = denm
		1: PD = denm + denmt/Te
		10: read from file profile_denm in m^2/sec
		11: read from file profile_denm in normalized units
`denm`	0	see idenmfunc
`denmt`	0	multiplier of 1/Te for idenmfunc=1
`denmmin`	0	minimum value of denm
`denmmax`	1.E6	maximum value of denm

9.5 Hyper-Diffusivity

Option	Default	Description
`imp_hyper`	0	switch for evaluating hyper resistivity
		0: $\lambda_H \nabla^2 \mathbf{J}$ explicit for ψ implicit for F
		1: $\lambda_H \nabla^2 \mathbf{J}$ implicit for ψ implicit for F
		2: $(\mathbf{B}/B^2)\nabla \bullet \lambda_H \nabla \sigma$ implicit for ψ and F $(\sigma = \mathbf{J} \bullet \mathbf{B}/B^2)$
`deex`	1	scale length used in hyper
`hyper`	0	hyper coefficient for ψ equation
`hyperc`	0	hyper coefficient for poloidal velocity
`hyperi`	0	hyper coefficient for toroidal field
`hyperp`	0	hyper coefficient for pressure
`hyperv`	0	hyper coefficient for toroidal flow
`ihypdx`	2	hyper terms multiplied by deex**ihypdx
`ihypeta`	1	swithc for multipliers of hyper terms
	1	magnetic field hyper multiplied by eta
	2	magnetic field hyper multiplied by p
`ihypamu`	1	1: velocity hyper coefficient multiplied by amu
`ihypkappa`	1	1: pressure hyper coefficient multiplied by kappa

9.6 Unit Normalizations

Option	Default	Description
`n0_norm`	10^{14}	Density normalization (in cgs)
`b0_norm`	10^4	Magnetic field normalization (in cgs)
`l0_norm`	100	Length normalization (in cgs)

9.7 Boundary Conditions

Option	Default	Description
`isurface`	1	include surface terms in Galerkin method
`icurv`	2	if > 0, include curvature from mesh
`nonrect`	0	1: non-rectangular boundary
`ifixedb`	0	Set $\psi = 0$ on boundary
`inonormalflow`	1	1: No-normal-flow boundary
`inoslip_pol`	0	1: No-slip boundaries for poloidal velocity
`inoslip_tor`	1	1: No-slip boundaries for toroidal velocity
`inostress_tor`	0	1: No-normal-stress boundary for toroidal velocity
`iconst_bz`	1	1: Toroidal field held constant on boundary
`iconst_bn`	1	1: Hold normal field constant on boundary
`iconst_n`	0	1: Density held constant on boundary
`iconst_p`	1	1: Pressure held constant on boundary
`iconst_t`	0	1: Temperature held constant on boundary
`inograd_p`	0	1: No normal pressure gradient
`inograd_t`	0	1: No normal temperature gradient
`inograd_n`	0	1: No normal density gradient
`com_bc`	0	1: $\nabla^2\chi = 0$
`vor_bc`	0	1: $\Delta^* U = 0$
`inocurrent_pol`	0	1: no poloidal current on boundary
`inocurrent_tor`	0	1: no toroidal current on boundary
`inocurrent_norm`	0	1: no normal current on boundary
`ifbound`	-1	boundary condition on $f\prime$
		1: Dirichlet
		2: Neumann
`iconstflux`	0	1: conserves toroidal flux in nonlinear calculation
`tebound`	-1	boundary condition for electron temperature
`tibound`	-1	boundary condition for ion temperature
`iper`	0	1: Left/right boundaries periodic
`jper`	0	2: Top/bottom boundaries periodic

9.8 Time Step

Option	Default	Description
`dt`	0.1	Initial size of ime step
`ntimemax`	20	Total number of time steps
`integrator`	0	0: Crank-Nicholson (CN); 1: BDF2
`imp_mod`	0	0: θ-implicit 1: Implicit leapfrof (`isplitstep` = 1 only)
`thimp`	0.5	Implicitness parameter
`thimpsm`	1	Implicitness of the smoother functions
`isplitstep`	1	0: Fully implicit time step; 1: split time step.
`iteratephi`	0	1: Calculate transport coefficients after field advance, then recalculate field advance.
`idiff`	0	1: solve for difference between n and n+1 in B,p
`idifv`	0	1: solve for difference between n and n+1 for V
`irecalc_eta`	0	1:recalculate transport coefficients after density solve
`iconst_eta`	0	1" don't evolve resistivity
`itime_independent`	0	1:exclude d/dt terms
`harned_mikic`	0	coefficient of Harned-Mikic 2F stabilization term
`isources`	0	1:include source terms in velocity advance
`nskip`	1	number of time steps per matrix recalculation
`pskip`	1	number of timesteps the preconditioner is reused
`iskippc`	1	number of times preconditioner is reused
`ddt`	0	change in timestep per timestep
`frequency`	0	frequency in time-independent calculation
`dtmin`	4.0	minimum timestep for variable timestep calculation
`dtmax`	40.	maximum timestep for variable timestep calculation
`dtkecrit`	0	lower timestep if ekin es above this (0.01 typical)
`dtfrac`	.10	max fractional change of timestep in 1 cycle
`max_repeat`	3	max number time step is repeated for ksp_max iterations exceeded
`ksp_max`	10000	max number of ksp iterations before repeating time step
`ksp_min`	1200	increase dt if ksp ¡ ksp_min
`ksp_warn`	1600	decrease dt if ksp ¿ ksp_warn

9.9 Mesh

Option	Default	Description
`nplanes`	1	number of toroidal planes for 3D nonlinear
`xzero`	0	R-coordinate of bottom left corner of domain
`zzero`	0	z-coordinate of bottom left corder of domain
`tiltangled`	0	angle a rectangular mesh is tilted
`mesh_model`		model file name from which the mesh is generated
`mesh_filename`		mesh name of .smb files (without number)
`imatassemble`	0	1: use petsc matrix parallel assembly instead of scorec
`imulti_region`	0	1: Mesh has multiple regions that include resistive wall and vacuum. Wall resistivity is "eta_wall", vacuum resistivity is "eta_vac"
`toroidal_pack_angle`	0	toroidal angle of maximum mesh packing
`toroidal_pack_factor`	1	ratio of longest to shortest toroidal element
`iread_vmec`	0	1: read geometry from VMEC file
`vmec_filename`	geometry.nc	name of vmec data file
`nperiods`	1	number of field periods
`iread_planes`	0	Read positions of toroidal planes from plane_positions
`bloat_distance`	0	factor to expand VMEC domain
`bloat_factor`	0	factor to expand VMEC domain
`ifull_torus`	0	0: one field period, 1: full torus
`igeometry`	0	0: default, identity
`xcenter`	0	center of logical mesh (x)
`zcenter`	0	center of logical mesh (z)
`bound_type(i)`	0	Boundary conditions to apply on mesh loop `i`. 0 = None, 1 = First wall, 2 = Domain boundary
`zone_type(i)`	0	Physics model of mesh zone `i`. 1 = plasma, 2 = conductor, 3 = vacuum.

9.10 Solver

Option	Default	Description
`solver_type`	0	for PETSc only, 0: direct solve, 1: iterative solver. for Trilinios, iterative solver is used
`solver_tol`	1.e-9	solver tolerance
`num_iter`	100	maximum number of iterations

9.11 Mesh Adaptation (will be depricated soon)

Option	Default	Description
`iadapt`	0	0: no adaptation 1:adapt mesh from the flux field in equilibrium

9.12 Numerical Options

Option	Default	Description
`int_pts_main`	25	Sampling points for integrations in main time step matrices
`int_pts_aux`	25	Sampling points for integrations in calculations of auxiliary variables
`int_pts_diag`	25	Sampling points for integrations in diagnostic calculations
`int_pts_tor`	5	Max number of toroidal integration points
`ivform`	0	0: $\mathbf{u} = \nabla U \times \nabla\varphi + V\nabla\varphi + \nabla\chi$ 1: $\mathbf{u} = R^2\nabla U \times \nabla\varphi + R^2\omega\nabla\varphi + R^{-2}\nabla\chi$ Now depricated to ivform=1 only
`jadv`	0	1: Use toroidal current density equation instead of poloidal flux equation.
`max_ke`	1.0	Maximum value of kinetic energy before solution is rescaled in linear simulations. (0 = don't rescale)
`equilibrate`	0	1: scale trial function so L2 norm = 1
`regular`	0	regularization constant in chi equation
`iset_pe_floor`	0	1: do not let pe drop below pe_floor
`pe_floor`	0	minimum value for pe when iset_pe_floor=1
`iset_pi_floor`	0	1: do not let pi drop below pe_floor
`pi_floor`	0	minimum value for pi when iset_pi_floor=1
`iset_te_floor`	0	1: do not let te drop below te_floor
`te_floor`	0	minimum value for te when iset_te_floor=1
`iset_ti_floor`	0	1: do not let ti drop below ti_floor
`ti_floor`	0	minimum value for ti when iset_ti_floor=1
`iprecompute_metric`	0	1: precompute metric temsor

9.13 Input Options

Option	Default	Description
iread_eqdsk	0	1: Read EFIT g-file 'geqdsk' 2: Read psi from geqdsk, but use analytic profiles for p and F 3: read profiles from geqdsk, but not the psi
iread_dskbal	0	1: Read BAL file 'dskbal'
iread_jsolver	0	1: Read Jsolver file 'fixed'
iread_omega	0	1: reads in rotation profile
iread_omega_ExB	0	1: read ExB rotation
iread_omega_e	0	1: read electron rotation
iread_ne	0	1: read in electron density profile
iread_te	0	1: read in temperature profile
iread_p	0	1: read pressure profile from profile_p
iread_neo	0	1: read velocity profiles from NEO output
ineo_subtract_diamag	0	1: subtract diamagnetic term from input vel when reading neo velocity
iread_heatsource	0	1: read heat source profile (psi normalized) scaled by ghs_rate
iread_partilesource	0	1: read particle source profile (psi normalized) scaled with pellet_rate
iread_f	0	1: read R x BT from file
iread_j	0	1: read current density from a file

9.14 Output Options

Option	Default	Description
`ntimepr`	5	Number of timesteps per full field output
`iprint`	0	1: Print detailed info to stdout
`ifout`	-1	1: ourput f field
`idouble_out`	0	1: use double precision floating in putput hdf5 files
`itemp_plot`	0	1: output vdotgradt, deldotq_perp, deldotq_par, eta_jsq
`ibdgp`	0	optons for plotting partial terms for bdgp plot (1) $[\psi, \Phi]$, (2) $(f\prime, \Phi)$, (3) $-R^{-2}F\Phi\prime$
`iveldif`	0	option for plotting partial terms for veldif plot (1) $\|/psi, U\|$, (2) $R^{-3}(\psi, chi) + R^{-2}\|\chi, f\prime\|$ (3)$R^{-1}\Phi\prime$ (4) $R^{-1}\Phi\prime + R^{-1}FU\prime$ (5) $\|\psi, U\| + R(U, f\prime) - R^{-1}FU\prime$ (6) $R^{-2}\|/chi, f\prime\|$
`icalc_scalars`	1	1 : calculate scalar diagnostics
`ike_only`	0	1 " only calculate ke scalar diagnostic
`ike_harmonics`	0	number of toroidal harmonics of kinetic energy to be calculated for diagnostics
`ibh_harmonics`	0	number of toroidal harmonics of magnetic energy to be calculated for diagnostics
`irestart`	0	0: start from timestep 0 1: normal restart (can restart 3D from 2D) 3: start a 2D complex run from a 2D real restart
`irestart_slice`	-1	if set to an integer, restart from that time slice
`itimer`	0	1: output internal timeing data
`iwrite_transport_coefs`	1	1: output transport coefficients fields
`iwrite_aux_vars`	1	1:output auxiliary variable fields

9.15 Diagnostics

Option	Default	Description
`xray_detector_enabled`	0	1: enable xray detector
`xray_r0`	0	R coordinate of xray detector
`xray_phi0`	0	phi coordinate of xray detector
`xray_z0`	0	Z coordinate of xray detector
`xray_theta`	0	angle of xray detector chord (degrees)
`xray_sigma`	1	spread of xray detector chord (degrees)
`imag_probes`	0	number of magnetic probes
`mag_probe_x(i)`	0	R coordinate of magnetic probe i
`mag_probe_phi(i)`	0	phi coordinate of magnetic probe i
`mag_probe_z(i)`	0	Z coordinate of magnetic probe i
`mag_probe_nx(i)`	0	R component of normal vector of mag probe i
`mag_probe_nphi(i)`	0	phi component of normal vector of mag probe i
`mag_probe_nz(i)`	0	Z component of normal vector of mag probe i
`iflux_loops`	0	number of flux loops
`flux_loop_x(i)`	0	R coordinate of flux loop i
`flux_loop_z(i)`	0	Z coordinate of flux loop i
`ifixed_temax`	0	1: temax evaluated at (xmag0,0,zmag0)

9.16 Sources/Sinks

Option	Default	Description
`ibeam`	0	neutral beam source 1:include neutral beam particle, energy, and momentum source: $S = \frac{nb_n}{4r\pi^2 nb_{dr}} \exp - \left[(r - nb_r)^2 + (z - nb_z)^2\right] / 2nb_{dr}^2$ 2:include only particle and energy source 3:include only energy source 4:include only momentum and energy source 5:include only momentum source
`beam_x`	0	R coordinate of beam center (in m)
`beam_z`	0	Z coordinate of beam center (in m)
`beam_v`	1.e4	beam voltage (in volts)
`beam_rate`	0	ions/second deposited by beam
`beam_dr`	.1	dispersion of beam deposition
`beam_dv`	100.	dispersion of beam voltage (in volts)
`beam_fracpar`	1	cosine of beam angle relative to parallel (for momentum source)
`vloop`	0	initial loop voltage, NOTE: to change vloop at restart time, must have control_type = -1
`tcur`	0	target (scaled) plasma current for current control: $\mu_0 I_P$
`tcuri`	0	if tcuri .ne. tcurf, the target current is a function of time
`tcurf`	0	tcur = tcuri + (tcurf-tcuri) x .5 x (1. + tanh((t-tcur_t0)/tcur_tw))
`tcur_t0`	0	see tcurf
`tcur_tw`	0	see tcurf
`control_type`	-1.	0 old current control algorithm (not recommended) 1: standard PID control with the following control parameters
`control_p`	0	proportional control coefficient
`control_i`	0	integral control coefficient
`control_d`	0	derivative control coefficient
`ipellet`	0	density source if non-zero (3D part equals 1 for 2D runs). Double-digit values have volume integrals normalized to 1. Make netative for initial perturbation only. Define: $G_{2D} = \frac{1}{2\pi R V_P^2} \exp\left[-\frac{(R-R_P)^2+(Z-Z_P)^2}{2V_P^2}\right]$ 1: $S = G_{2D} \times \frac{R}{\sqrt{2\pi}V_t} \exp\left[-\frac{RR_P(1-\cos(\phi-\phi_P))}{V_t^2}\right]$ 2: $S = \text{den0} \times (\max(p, p_{edge})/p_0)^{expn}$ 2D and 3D 3: Gaussian sourrce proportional to pressure $S = p \times G_{2D} \times \frac{R}{\sqrt{2\pi}V_p} \exp\left[-\frac{RR_P(1-\cos(\phi-\phi_P))}{V_p^2}\right]$ 4: Same distribution as ipellet=1 in 3D $S = \sqrt{2\pi} R V_p \times G_{2D} \times \frac{1}{2\pi V_p V_t} \exp\left[-\frac{RR_P(1-\cos(\phi-\phi_P))}{V_t^2}\right]$

Option	Default	Description
		11: Same as #1 but numerically normalized 12: Spherical, Cartesian Gaussian numerically normalized 2D: $S = RG_{2D}$ 3D: $S = \exp\left[-\frac{(R\cos\phi - R_p\cos\phi_p)^2 + (R\sin\phi - R_p\sin\phi_p)^2 + (Z-Z_p)^2}{2V_P^2}\right]$ 13: Axisymmetric, toroidal Gaussian, numerically normalized 2D and 3D: $S = G_{2D}$ 14: Toroidal distribution is a blend of a von Mises and Cauchy distribution $S = G_{2D} \times \frac{R}{\sqrt{2\pi}V_p} \times \left[(1-f_c)\exp\left[-\frac{RR_p\left(1-\cos(\phi-\phi_p)\right)}{V_t^2}\right] + f_c \times \frac{\cosh(\frac{V_t}{\sqrt{RR_p}}) - cos(\phi_p)}{\cosh(\frac{V_t}{\sqrt{RR_p}}) - cos(\phi-\phi_p)}\right]$ 15: Toroidal von-Mises distribution with angular half-width $S = G_{2D} \times \exp\left[\cos(\phi-\phi_p)/V_p^2\right]$ Pellet_var_tor radians for ipellet=15, distance otherwise
`ipellet_z`	0	Atomic number of pellet (0 for main-ion species)
`ipellet_abl`	0	Turn on pellet ablation. (Recommend double-digit ipellet for particle conservation) 1: Include ablation model [Parks NF94] calibrated on DIII-D (Li) 2: Include new ablation model [Parks, 2015] for small pellets (Li) 3: Parks model developed 6/20/2017 (for Neon)
`temin_abl`	0	Minimum temperature at which ablation turns on
`iread_pellet`	0	0: Single pellet defined by scalar parameters below 1: Read pellet.dat, with one row per pellet. The 13 space-delimited columns are: pellet_r, pellet_phi, pellet_z, pellet_rate, pellet_var, pellet_var_tor, pellet_velr, pellet_vel_phi, pellet_vel_z, r_p, cloud_pel, pellet_mix, cauchy_fraction
`pellet_rate`	0	Particle number injection rate
`pellet_var`	1	Variance of injection profile
`pellet_var_tor`	0	toroidal spatial dispersion of pellet source (V_t). If zero, pellet_var_tor = pellet_var.
`pellet_R`	0	R-coordinate of injection profile
`pellet_z`	0	Z-coordinate of injection profile
`pellet_velr`	0	initial radial velocity of the pellet
`pellet_velphi`	0	initial toroidal velocity of pellet
`pellet_velz`	0	initial vertical velocity of pellet
`r_p`	1.e-3	initial pellet radius
`cloud_pel`	1	Parameter used to change the width of the density source if ablating. In this case, pellet_var = cloud_pel * r_p
`pellet_mix`	0	Molar fraction of diatomic main ion molecures in pellet (e.g., D_2)

Option	Default	Description
`irestart_pellet`	0	1: will read the following from C1input at restart (the rest from *.h5) pellet_rate, pellet_rate_D2, pellet_var_tor, pellet_var, cloud_pel, pellet_mix, cauchy_fraction
`abl_fac`	1.0	factor to multiply ablation rate from predefined formula
`n_control_type`	-1	-1: no density control 0: old density control algorithm (not recommended) 1: Standard PID control with the following parameters:
`n_control_p`	0	proportional feedback constant
`n_control_i`	0	integral feedback constant
`n_control_d`	0	derivative feedback constant
`igaussian_heat_source`	0	1: include Gaussian heat fource
`ghs_x`	0	R coordinate of Gaussian heat source
`ghs_z`	0	Z coordinate of Gaussian heat source
`ghs_rate`	0	amplitude of Gaussian heat source
`ghs_var`	0	variance of Gaussian heat source
`ghs_phi`	0	phi coordinate of Gaussian heat source
`ghs_var_tor`	0	toroidal variance of GAussian heat source
`ionization`	0	1: include neutral ionization source
`ionization_rate`	0	Ionization rate coefficient
`ionization_temp`	0.01	Ionization energy
`ionization_depth`	0.01	Temperature scale-length of neutral burn-out

current drive source		$\dot{\psi} = ... + \eta(\Delta^*\psi - J_{CD})$ $J_{CD} = J_0 \exp - \left[(R - R_{0cd})^2 + (Z - Z_{0cd})^2\right] / W_{cd} - \Delta_{cd}$
`icd_source`	0	1: include current drive
`J_0cd`	0	J_0: Magnitude of Gaussian
`R_0cd`	0	R_{0cd}: R coordinate of maximum
`Z_0cd`	0	Z_{0cd}: Z coordinate of maximum
`W_cd`	0	W_{cd}: Width of maximum
`delta_cd`	0	Δ_{cd}: shift of Gaussian
`isink`	0	number of density sinks: 0,1,or 2
`sink1_x`	0	R coordinate of first density sink
`sink1_z`	0	Z coordinate of first density sink
`sink1_rate`	0	rate of first density sink
`sink1_var`	1	variance of first density sink
`sink2_x`	0	R coordinate of second density sink
`sink2_z`	0	Z coordinate of second density sink
`sink2_rate`	0	rate of second density sink
`sink2_var`	1	variance of second density sink
`idenfloor`	0	1: density in vacuum pegged to den_edge
`alphadenfloor`	0	multiplier of (den_edge - den). Must be .lt. 1/DT
`ipforce`	0	1: include poloidal momentum source $F = f(\tilde{\psi})\nabla\psi \times \nabla\phi$ $f(\tilde{\psi}) = \alpha(1-\tilde{\psi})^N \frac{\delta^2}{(\tilde{\psi}-\psi_0)^2+\delta^2}$ 2: Include Luca Guzzatto form of momentum source

Option	**Default**	**Description**
`dforce`	0	δ
`xforce`	0	ψ_0
`nforce`	0	N
`aforce`	0	α
`iheat_sink`	0	1: special heat sink for itaylor=27
`coolrate`	0	S = coolrate*(pedge-p) for iheat_sink = 1
`iarc_source`	0	1: density source due to halo current
`arc_source_alpha`	0	parameter for arc_source
`arc_source_eta`	.01	parameter for arc_source

9.17 Resistive Wall

Option	**Default**	**Description**
`eta_vac`	1	resistivity of vacuum region
`eta_wall`	.001	resistivity of conducting wall regions
`eta_wallRZ`	.001	poloidal resistivity of wall region (if different from eta_wall)
`iwall_break`	0	number of wall breaks
`eta_break`	1	resistivity of wall break (array)
`wall_break_phimax`	0	max phi coordinate for break (array)
`wall_break_phimin`	0	min phi coordinate for break (array)
`wall_break_xmax`	0	max R coordinate for break (array)
`wall_break_xmin`	0	min R coordinate for break (array)
`wall_break_zmax`	0	max Z coordinate for break (array)
`wall_break_zmin`	0	min Z coordinate for break (array)
`iwall_regions`	0	number of resistive wall regions
`wall_region_eta()`	1.e-3	resistivity of each wall region
`wall_region_etaRZ()`	1.e-3	poloidal restivity (if different from wall_region_eta)
`wall_region_filename()`		file name will wall contour points
`eta_zone(i)`	eta_wall	Resistivity of mesh region `i`. Only applies if `zone_type(i)` = 2 (conductor).
`etaRZ_zone(i)`	eta_zone(i)	Poloidal resistivity of mesh region `i`. Only applies if `zone_type(i)` = 2 (conductor).
`eta_rekc`	0	resistivity of runaway electron killer coil (REKC)
`ntor_rekc`	0	toroidal mode number of REKC
`mpol_rekc`	0	poloidal mode number of REKC
`phi_rekc`	0	toroidal angle of fixed point of REKC
`theta_rekc`	0	poloidal angle of fixed point of REKC
`rzero_rekc`	0	R0 for computing theta of REKC
`zzero_rekc`	0	Z0 for computing theta of REKC
`isym_rekc`	0	if non-zero, coil is double helix with (+,-) mpol_rekc

9.18 Miscellaneous

Option	Default	Description
`( gam`	5/3	ratio of sepcific heatx
`db`	0	ion skin depth (overrides db_fac
`db_fac`	0	factor multiplying physical value of ion skin depth
`mass_ratio`	0	ratio of ion to electron mass
`lambdae`	0	lambdae
`z_ion`	1	Z-effective
`ion_mass`	1	ion mass in units of m_p
`lambda_coulomb`	17	Couloumb logarithm
`thermal_force_coeff`	0	coefficient of thermal force
`ntor`	0	toroidal mode number for 3D linear (complex)
`mpol`	0	poloidal mode number for certain test problem initializations

9.19 Deprecated

Option	Default	Description
`ipartitioned`	0	
`igs_method`	-1	
`ibform`	-1	
`delta_wall`	1.	

9.20 Trilinos Options

Option	Default	Description
`drop_tolerance`	0	ILU drop tolerance
`graph_fill`	0	graph fill level
`ilu_fill_level`	1	ILU fill level
`ilu_omega`	1	relaxation parameter for rILU
`krylov_solver`	gmres	Krylov solver
`poly_ord`	1	polynomial order for certain perconditioners
`preconditioner`	dom_decomp	preconditioner
`sub_dom_solver`	ilu	subdomain solver in preconditioner
`subdomain_overlap`	1	subdomain overlap

9.21 Simple Radiation Model

Option	Default	Description
iprad	0	1: call Prad radiation module with one impurity species $P_{rad} = n_e n_D L_D(T_e) + n_e n_Z L_Z(T_e)$ Cooling rate of deuterium is $L_D = 5.35 \times 10^{-37} T_e^{1/2} [keV] W - m$ $L_Z(T_e)$ taken from Post, et al, Atomic data and nuclear data tables, **20** pp. 397-439 (1977)
prad_fz	1	density of impurity species as fraction of n_e
prad_z	1	Z of impurity species: Z=6(C), 18(Argon), 26(Fe) are available
iread_prad	0	1: Read impurity density from profile_nz (units of $10^{20}/m^3$)

9.22 KPRAD Radiation Model

Option	Default	Description
ikprad	0	1: KPRAD module with one impurity species
ikprad_z	1	Z of impurity species in KPRAD module. Presently available: 2(He), 4(Be), 6 (C),10(Ne), 18 (Ar)
kprad_fz	0	density of neutrals as fraction of ne
kprad_nz	0	Density of neutral impurities
kprad_nemin	10^{-12}	Minimum normalized electron density for KPRAD evolution
kprad_temin	2×10^{-7}	Minimum normalized electron temperature for KPRAD evolution
ikprad_max_dt	0	Set max time step for KPRAD ionization 0: MHD time step dt 1: **RECOMMENDED:** dt/(kprad_z + 1) : ensures evolution through all charge states.
ikprad_evolve_internal	0	Update local temperature during KPRAD subcycling: 0: Te fixed before subcycling 1: **RECOMMENDED** Local ne and Te used for KPRAD ionization/radiation updated during subcycling each KPRAD time step due to density and energy changes
ikprad_evolve_neutrals	0	Determines how KPRAD neutrals evolve spatially: 0: Neither advect nor diffuse 1: **RECOMMENDED** Advect and diffuse like other charge states 2: Diffuse but do not advect
ikprad_min_option	1	Determine how KPRAD behaves below minimum density/temperature (kprad_nemin, kprad_temin) 1: No radiation/ionization/recombination (based on ne/te before subcycling) 2: **RECOMMENDED** Recombination but no radiation/ionization (based on ne/Te during subcycling) 3: No radiation/ionization/recombination (based on ne/Te during subcycling). NOTE: 1 and 3 behave the same if ikprad_evolve_internal = 0)
iread_lp_source	0	Read impurity source from Lagrangian Particle code cloud.txt (UNDER DEVELOPMENT)

9.23 Stellarator Geometry

Option	Default	Description
`type_ext_field`	-1	External fields to be read in.. 0: Axisymmetric only. For RMP and error fields. 1: Free-boundary stellarator (`itaylor=41`) data. 2: Free-boundary stellarator (`itaylor=41`) data with `extsubtract=1`.
`file_ext_field`		(string) Vacuum/external field to be subtracted for `itaylor=41`. Currently supported: FIELDLINES, MGRID. Must start with 'fieldlines' or 'mgrid'. Must have `type_ext_field=2`.
`file_total_field`		(string) Total field for free-boundary stellarator (`itaylor=41`). Currently supported: FIELDLINES, MGRID. Must start with 'fieldlines' or 'mgrid'. Must have `type_ext_field=1,2`.
`iread_vmec`	0	1: read VMEC file to determine geometry. Must be 1 for stellarator.
`vmec_filename`		(string) VMEC output .nc file
`bloat_factor`	0	Free boundary only: Scale factor to bloat computational boundary using input geometry.
`bloat_distance`	0	Free boundary only: Distance to expand computational boundary from input geometry.
`igeometry`	0	1: must be set to use stellarator version
`nperiods`	1	Number of field periods (stellarator geometry). Note nplanes should be equal to at least 2 x (number of toroidal modes per field period) x nperiods
`ifull_torus`	0	0: Solve on one field period 1: Solve on full torus
`nzer_factor`	-1	(integer) Scale factor for resolution of Zernike polynomial (used for interpolation of VMEC) -1: n_zeri= 2 x mpol (fixed boundary) and 1 x mpol (free boundary). Otherwise: n_zer = nzer_factor x mpol where mpol is poloidal resolution in VMEC
`nzer_manual`	-1	(int) Resolution of Zernike polynomial (mainly for testing)

A IDL Postprocessor

A.1 Introduction

The IDL routines described here have been created for the purpose of reading, displaying, and manipulating data written by M3D-C^1 to an HDF5 output file. These routines are contained within files stored in the subdirectory `trunk/unstructured/idl/` in the M3D-C^1 SVN repository.

`Invoking IDL`
First, IDL must be given access to the postprocessor routines. This may be accomplished either by copying the `*.pro` files in the repository to the working directory where IDL will be run, or to set the environment variable `IDL_PATH` to include the directory where these files are located.

The IDL module must first be loaded to run IDL on `portal.pppl.gov`. This is done by

```
module load idl
```

IDL may then be invoked by

```
idl
```

Once IDL is running, the postprocessor routines must be compiled. This is done by

```
.run plot_routines
.run power_spectrum
.run read_h5
```

The functions and procedures described in the following section are now ready to use.

`Help In IDL`

IDL has an on-line help system which may be invoked from the IDL command prompt by

```
?
```

Information about a specific intrinsic IDL routine, `<routine>`, may be obtained by

```
? <routine>
```

`Note on function/procedure descriptions`
Functions and procedures in IDL are differentiated by whether a value is returned (as with functions) or not (as with procedures). Both functions and procedures may take “arguments” and “keywords”

as command line parameters. In the following descriptions, the arguments of the function or procedure are shown with the command, and keywords are listed separately. Optional arguments are enclosed by square brackets. Keywords are always optional. (This notation differs from that in the on-line IDL help only in that keywords are not listed with the command.) Keywords are specified on the command line by

```
<keywordname>=<value>
```

Writing

```
/<keywordname>
```

has the same effect as

```
<keywordname>=1
```

and is therefore useful for boolean options. Some common examples are given below.

`Common Examples`
To display the field “psi” at time slice 1 of file “C1.h5”, sampled on a regular 100×100 grid, with an overlay of the LCFS and the mesh,

```
plot_field, 'psi', 1, filename='C1.h5', /iso, /lcfs, /mesh, points=100
```

To plot the time series of the kinetic energy of file “C1.h5” in domain $0 < t < 100$,

```
plot_scalar, 'ke', filename='C1.h5', xrange=[0,100]
```

To plot the flux-averaged temperature profiles of files “1/C1.h5” and “2/C1.h5” at time slice 1, versus the normalized poloidal flux,

```
plot_flux_average, 'T', 1, filename=['1/C1.h5', '2/C1.h5'], /norm
```

A.2 Functions/Procedures in `plot_routines.pro`

A.2.1 Procedure `contour_and_legend`

`contour_and_legend`, z [,x, y]

`Description`
This procedure draws a two-dimensional color contour plot of the data z, with horizontal and vertical coordinates x and y. nt frames are drawn.

`Arguments`

Name	I/O	Type	Description
z	I	`float`[nt,nx,nz]	The values of the field to plot
x	I	`float`[nx]	The values of the x-coordinate
y	I	`float`[nz]	The values of the y-coordinate

`Keywords`

Name	I/O	Type	Description
label	I	`string`[nt]	The IDL-formatted label of the color bar for each frame
title	I	`string`[nt]	The IDL-formatted title of the plot for each frame
range	I	`float`[2,nt]	An array of the ranges of z to plot in each frame
nlevels	I	`int`[nt]	The number of contour levels to plot for each frame
lines	I	`bool`[nt]	Whether to draw contour lines for each frame
zlog	I	`bool`[nt]	Whether to draw z on a log scale, for each frame
jpeg	I	`string`	Write the resulting .jpeg image to the file *jpeg*
isotropic	I	`bool`	Use an isotropic aspect ratio when plotting
color_table	I	`int`	Which intrinsic IDL color table to use when plotting

A.3 Functions/Procedures in `read_h5.pro`

A.3.1 Function `read_field`

field = `read_field`(*name*, r, z, t)

`Description`
This function reads the raw field data associated with *name* in the specified output file. The data is interpolated onto a uniform retangular grid, and returned in an array.

`Return Value`

`float`[nt, *points*, *points*] *field* containing the value of the field at *points* $\times$ *points* spatial sampling points at nt time slices.

`Arguments`

Name	I/O	Type	Description
name	I	`string`	The name of the field to read
r	O	`float`[*points*]	The values of the *r*-coordinate
z	O	`float`[*points*]	The values of the *z*-coordinate
t	O	`float`[*nt*]	The values of the *t*-coordinate

`Keywords`

Name	I/O	Type	Description
filename	I	`string`	The name of the HDF5 file to read
points	I	`int`	Number of sampling points per spatial dimension
rrange	I	`float`[2]	Range of *r*-coordinate to read
zrange	I	`float`[2]	Range of *z*-coordinate to read
h_symmetry	I	⟨ 1 | -1 ⟩	Return only left-right ⟨ symmetric | anti-symmetric ⟩ part of field
v_symmetry	I	⟨ 1 | -1 ⟩	Return only up-down ⟨ symmetric | anti-symmetric ⟩ part of field
diff	I	`bool`	If set, *name* should be an array with two filenames. The difference of the fields from the two files is returned.
slices	O	⟨ `int`| `int`[2] ⟩	⟨ Time slice | range of time slices ⟩ to read
mesh	O		

A.3.2 Function `flux_average`

result = `flux_average`(*field*, *slice*)

`Description`

This function finds the flux-surface average of the field *field*.

`Return Value`

`float`[*bins*] *result* contains the value of the flux-averaged field for a range of values of flux.

`Arguments`

Name	I/O	Type	Description
field	I	⟨ `string`| `float`[1,*points*,*points*] ⟩	If *field* is type `string`, then read field associated with name *field*. Otherwise, *field* is taken to contain the field data.
slice	I	`int`	If *field* is type `string`, then this is the time slice to read. Otherwise, *slice* is ignored.

`Keywords`

`flux_average` takes all of the optional arguments for `read_field`. In addition,

Name	I/O	Type	Description
filename	I	`string`	The name of the HDF5 file to read
bins	I	`int`	Number of bins to subdivide the flux
*psi**	I/O	`float`[1,*points*,*points*]	The flux field at the given time slice
x	I/O	`float`[*points*]	*r*-coordinate values
z	I/O	`float`[*points*]	*z*-coordinate values
t	I/O	`float`[*points*]	*t*-coordinate values
flux	O	`float`[*bins*]	The value of the flux for each bin
title	O	`string`	The IDL-formatted title of the field
symbol	O	`string`	The IDL-formatted symbol of the field
units	O	`string`	The IDL-formatted units of the field

* If *psi*, *x*, *z*, and *t* are all provided as input, the `flux_average` will not read the flux field itself.

A.3.3 Function `read_scalar`

result = `read_field`(*name*)

`Description`
This function reads the scalar quantity associated with *name* in the specified output file. The data is returned as an array .

`Return Value`

`float`[*nt*] *result* contains the value of the scalar at each time step.

`Arguments`

Name	I/O	Type	Description
name	I	`string`	The name of the scalar to read

`Keywords`

Name	I/O	Type	Description
filename	I	`string`	The name of the HDF5 file to read
time	O	`float`[*nt*]	The t-coordinate of each element
title	O	`string`	The IDL-formatted title of the scalar
symbol	O	`string`	The IDL-formatted symbol of the scalar
units	O	`string`	The IDL-formatted units of the scalar

A.3.4 Procedure `plot_field`

`plot_field`, *name*, *slice*, *r*, *z*, *t*

`Arguments`

Name	I/O	Type	Description
name	I	`string`	The name of the field to read
slice	I	`string`	The time slice to read
r	O	`float`[*points*]	The values of the r-coordinate
z	O	`float`[*points*]	The values of the z-coordinate
t	O	`float`[*nt*]	The values of the t-coordinate

`Keywords`

`plot_field` takes all of the optional arguments for `read_field` and `contour_and_legend`. In addition,

Name	I/O	Type	Description
xrange	I	`float`[2]	Range of r-coordinate to plot
yrange	I	`float`[2]	Range of z-coordinate to plot
lcfs	I	`bool`	Plot LCFS
mesh	I	`bool`	Plot mesh

A.3.5 Procedure `plot_flux_average`

`plot_flux_average`, *field*, *slice*

`Description`
Plots the flux-surface average of a field as a function of poloidal flux. Data from multiple files or multiple times may be plotted at once.

`Arguments`

Name	I/O	Type	Description
field	I	⟨ `string`| `float`[1,*points*,*points*] ⟩	If *field* is type `string`, then read field associated with name *field*. Otherwise, *field* is taken to contain the field data.
slice	I	`int`[*nt*]	If *field* is type `string`, then this is the time slice(s) to read. Otherwise, *slice* is ignored.

`Keywords`

Name	I/O	Type	Description
filename	I	`string`[*nfiles*]	Names of HDF5 file(s) to read
overplot	I	`bool`	Plot over previous plot
lcfs	I	`bool`	Plot LCFS
minor_radius	I	`bool`	Plot against flux-average minor radius
normalized_flux	I	`bool`	Plot against normalized flux
smooth	I	`int`	Boxcar-average final data over neighboring *smooth* bins

A.3.6 Procedure `plot_timings`

`plot_timings`

`Description`

This function plots the relative time spent in various subroutines of M3D-C^1. This data is only available if `itimer` = 1 was specified in the input file.

`Keywords`
`plot_flux_average` takes all of the optional arguments for `flux_average`. In addition,

Name	I/O	Type	Description
filename	I	`string`	The name of the HDF5 file to read
overplot	I	`bool`	Draws data over the

A.3.7 Procedure `write_geqdsk`

`write_geqdsk`

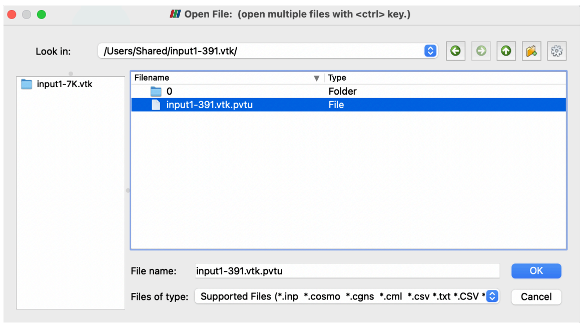
`Description`
This procedure writes equilibrium data do disk in the `geqdsk` format.

`Arguments`
None.

`Keywords`

`write_geqdsk` takes all of the optional arguments for `read_field` and `read_parameter`. In addition,

Name	I/O	Type	Description
eqfile	I	`string`	Filename to output geqdsk data
b0	I	`float`	Normalization of magnetic field, in Gauss
l0	I	`float`	Normalization of length scale, in cm


Open File: (open multiple files with <ctrl> key.)

Look in: /Users/Shared/input1-391.vtk/

input1-7K.vtk

Filename	Type
0	Folder
input1-391.vtk.pvtu	File

File name: input1-391.vtk.pvtu

Files of type: Supported Files (*.inp *.cosmo *.cgns *.cml *.csv *.txt *.CSV *

Figure 13: To visualize a mesh, select .pvtu file from Open File menu

B Mesh Visualization

Paraview is program created by Kitware, Inc. which can visualize meshes and fields on meshes. It is the program of choice for viewing meshes created by the PUMI libraries. The API *m3dc1_mesh_write* writes a mesh either in "smb" or "vtk".

```
// filename: output file name
// option: 0 vtk file with field; 1 smb file
m3dc1_mesh_write(char* filename, int *option)
```

If the first argument is "output"" and the second argument is 0, filename is output, it creates the files `output.pvtu`. Opening the `output.pvtu` file in Paraview will show users the mesh. Figures 13 and 14 illustrate "Open File" window and a mesh rendered in "Surface" mode by default.

Changing "Surface" to "Surface with Edges" will outline each visible element. Figures 15 and 16 illustrate how to make the decomposition visible.

Also, the mesh by default is rendered in one "Solid Color". There should be other options corresponding to the fields and numberings that were on this mesh at the time of file writing. There is an "apf_part" alternative for files written by APF, which allows users to see the parallel partitioning of the mesh in color.

Mesh generation program provides "gface" alternative, which allows users to see the geometric face of the mesh in color. Figure 17 depicts a Paraview window with a mesh rendered with "gface".

Figure 18 illustrates the "Color Palette" tab of "Settings" menu which allows the users to change the colors of "Render View" such as edges, background, etc..

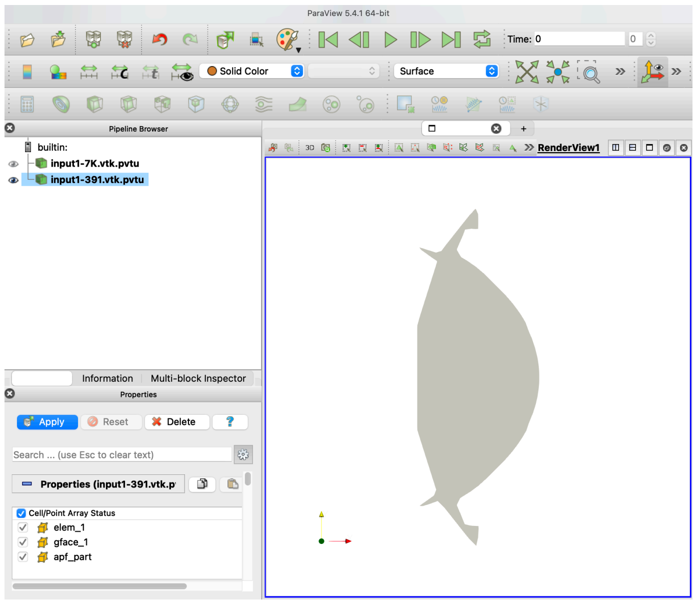


Figure 14: Initial Paraview window with a mesh

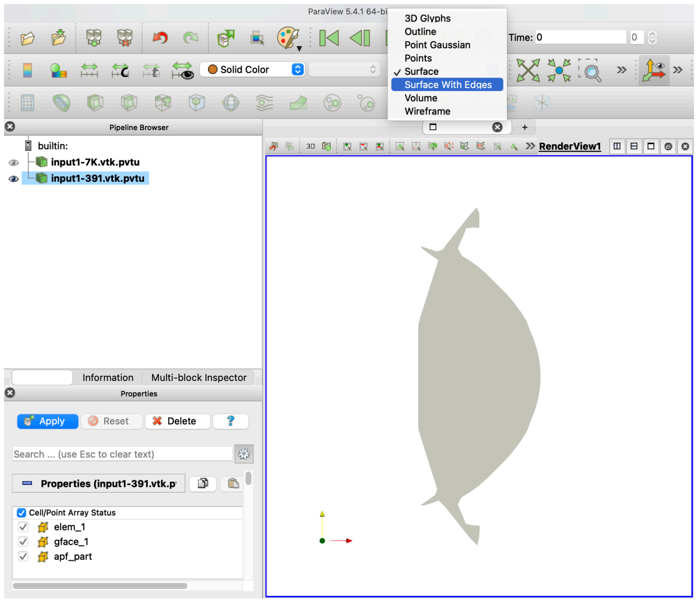


Figure 15: To render mesh decomposition, choose “Surface With Edge”

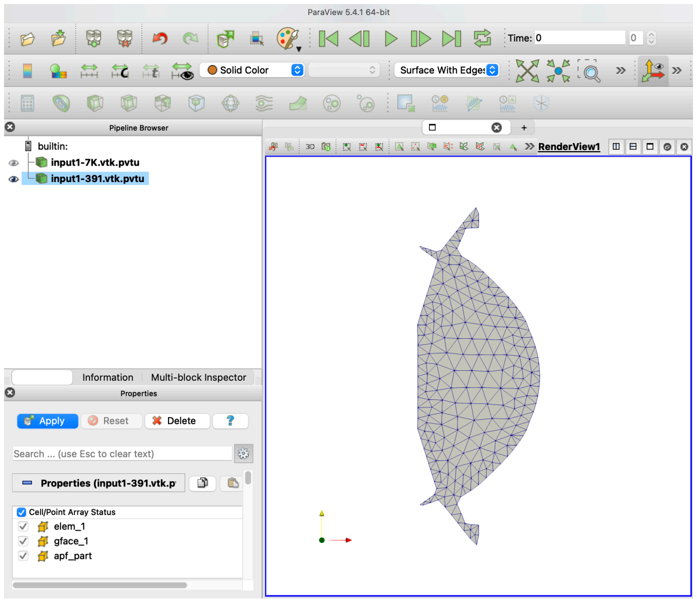
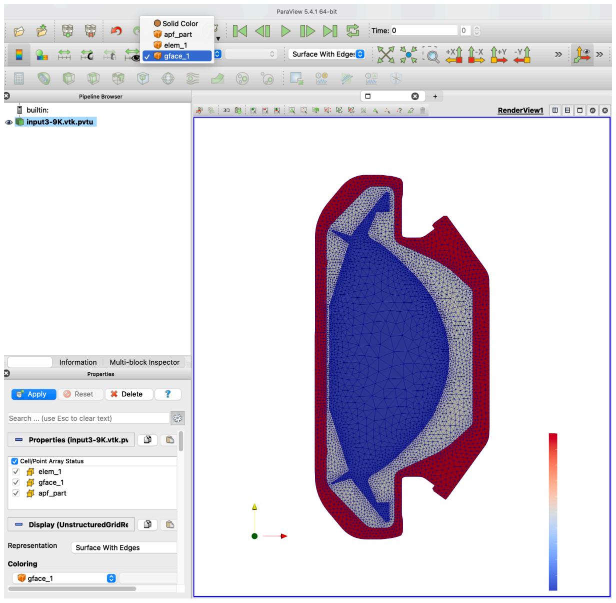


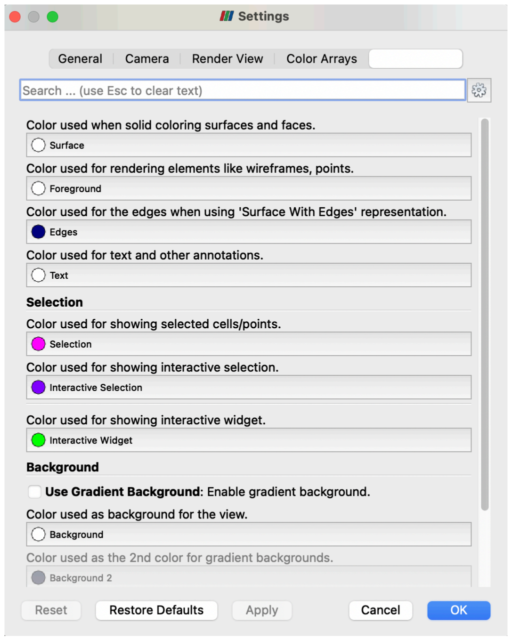
Figure 16: A mesh with "Surface With Edge"



Representation Surface With Edges

Coloring gface_1

Figure 17: A mesh rendered with "gface"



Color used when solid coloring surfaces and faces.
○ Surface
Color used for rendering elements like wireframes, points.
○ Foreground
Color used for the edges when using 'Surface With Edges' representation.
● Edges
Color used for text and other annotations.
○ Text

Selection

Color used for showing selected cells/points.
● Selection
Color used for showing interactive selection.
● Interactive Selection

Color used for showing interactive widget.
● Interactive Widget

Background

☐ **Use Gradient Background**: Enable gradient background.

Color used as background for the view.
○ Background
Color used as the 2nd color for gradient backgrounds.
● Background 2

Figure 18: “Color Palette” tab of “Settings” menu”