## Common workflow

Puffin outputs 3 types of files, identified by their naming convention. The general file-naming convention is <*basename*>_<*filetype*>_<*stepnumber*>.h5, where <*basename*> is the name of the main input file input to Puffin (without the '.in' extension), <*filetype*> indicates one of the 3 filetypes, and <*stepnumber*> indicates the stage in the simulation at which the file was written. File types are as follows:

- Integrated, with naming convention <*basename*>_integrated_<*number*>.h5. This contains data derived from the 'raw' data representation in Puffin, such as the beam current and bunching, and the field power.
- Field mesh, with naming convention <*basename*>_aperp_<*number*>.h5, containing the 'raw' field values at the nodes of the field mesh, along with metadata describing the layout of the mesh.
- Electron macroparticles, with naming convention <*basename*>_electrons_<*number*>.h5, containing the raw electron electron macroparticle positions and charge weights.

The latter two types, containing the raw data, can be written out with a different frequency than the integrated files. Depending on the type of run, it may make more sense to dump the raw data more often.

There are multiple modes in which Puffin can run. Users can choose between 1D or 3D, and fully temporal or periodic. For a 3D, fully temporal run, the raw data files will likely be very large (potentially >100's of GB's), and it makes more sense to do very infrequent writes of the mesh and macroparticle data. On the other hand, 1D runs will likely be very small, and writing out the raw data frequently makes more sense.

All 3 types include VizSchema metadata so that they can be used in e.g. VisIt. Due to how the data is stored in Puffin, the field mesh files in both 1D and 3D cases are stored in Fortran ordering, and have to be rearranged into C ordering to work in Visit. This can be done by using the ReorderColMajorFtoColMinorC.py script.

The use cases in the 1D and 3D modes are quite different - in 1D the data dumps are very small, and so you can write the full data dumps often - say, every undulator period. In contrast, the 3D data dumps can be huge, so you probably want to avoid doing a lot if you can, but it may be unavoidable for your situation depending on what you want to see! In the 3D case usually only a few full full dumps will typically be made through the run, and the integrated data is dumped more often - again, perhaps at every undulator period.

## 1D Case

- 1D case
    - Run Puffin
        - Typically use frequent dumps of field and macroparticle data
    - Run the convert2C script on the aperp files
    - Run powPrep if desired
    - Py plotting routines can be used on the mesh and macroparticle dumps

For 1D mode, the data sizes are small enough so that dumps can be done frequently on modern machines. Runs may typically be done locally, or they may be moved from remote locations due to relatively small file size, removing the need for remote visualization software.

In the 3D case, the integrated data is automatically high-pass filtered (due to the diffraction algorithm) so that the integrated data immediately looks familiar to a FEL physicist. The 1D case is NOT filtered in the simulation, so the integrated data files include the low-frequency emission, and this obscures the FEL gain in the data until typically around halfway to saturation.

Because of this, usually many mesh and electron data dumps are used in 1D mode, and Puffin comes with a few Python scripts to analyse the data from the ‘raw’ dumps. The integrated data is not used so much in this use case.

This allows one to choose, for example, to measure the gain at a certain radiation frequency quickly and easily from the 1D files. If only the integrated data were used, this information would be lost.

## 1D run example workflow

Run Puffin:

```
mpirun -np 2 puffin f1main.in
```

Reorder the field mesh files from Fortran to C ordering:

```
python $PUFFDIR/bin/post/ReorderColMajorFtoColMinorC.py f1main
```

The python scripts can plot field spectrum, power, etc from the *aperp_C* files:

```
python $PUFFDIR/bin/pyPlotting/plotPowVsZ2.py f1main_aperp_C_6900.h5
python $PUFFDIR/bin/pyPlotting/plotSpecPow.py f1main_aperp_C_6900.h5
```

and so forth. The `plotPowVsZ2` script plots the temporal power. The filtered power can be plotted by supplying same script with an additional 2 numbers describing the filter, in units of the 'central' or reference frequency. To filter around the fundamental, do

```
python $PUFFDIR/bin/pyPlotting/plotPowVsZ2.py f1main_aperp_C_6900.h5 1 0.2
```

The power as a function of z (distance through the undulator) can be plotted by supplying the 'basename', and the script automatically detects and uses all the relevant files according to the naming convention above (basename + aperp_C_step.h5):

```
python $PUFFDIR/bin/pyPlotting/plotPowVsZ.py f1main
```

The filtered power can be plotted by supplying same script with an additional 2 numbers describing the filter, in units of the 'central' or reference frequency. To filter around the fundamental, do

```
python $PUFFDIR/bin/pyPlotting/plotPowVsZ.py f1main 1 0.2
```

Visit can be used to view the bunching, current, etc from the integrated files. Python scripts to plot these, similar to the field plotting routines above, will be added soon.

### 3D Case

- 3D case
    - Run Puffin
        - Typically use infrequent dumps of field mesh and electron macroparticles, but frequent integrated writes
    - Run the convert2C script on the reduced aperp files
    - Run the powPrep script on the integrated data files
    - Py plotting routines and Visit can be used on these processed files
    - Optionally, run the reduce script on the aperp files

Typically you'll use less data dumps for the run as in 1D, as the files are huge. For CLARA, a single 3D field mesh file is around 350GB, and to write enough to get a nice gain curve from this can easily then take up multiple 10's or 100's of TB's per run.

This poses interesting problems from the data reduction point of view. Ultimately, the more data dumps, the better, but Puffin currently outputs enough integrated data (power and so on) so that you can see if the FEL is working. The diffraction algorithm in Puffin includes a high-pass filter, so most of the low frequency CSE effects in the power curves should not be there (the filter may be adjusted in the Puffin input file, and you could conceivably get the low frequency content back into the run if you wanted). So the radiated energy curves as output in the integrated files look fine without having to worry about low frequencies, as opposed to the 1D case.

If you are interested in the power growth at 2 or more distinct frequencies, then the energy (gain) curve will include all of these frequencies, and they cannot be retrieved from the integrated data. For now, many dumps should be written out if you really want to get into the detail of how these separate colours evolve through the undulator. There may be options added in the future to make Puffin filter some field energy/power output for you, but this may be computationally unfeasible for a few different reasons.

More likely, Puffin will have an option to write out a reduced mesh, and allow the selection of an inner mesh for writing, where the inner, say 30x30 nodes in the transverse mesh are dumped. This dramatically reduces the mesh size in the files.

## 3D run example workflow

Run Puffin:

```
mpirun -np 2 puffin f1main.in
```

Reorder the field mesh files from Fortran to C ordering:

```
python $PUFFDIR/bin/post/ReorderColMajorFtoColMinorC.py f1main
```

The python scripts can plot field spectrum, filtered 1D power, etc from the *aperp_C* files:

```
python $PUFFDIR/bin/pyPlotting/plotPowVsZ2.py f1main_aperp_C_6900.h5
python $PUFFDIR/bin/pyPlotting/plotSpecPow.py f1main_aperp_C_6900.h5
```

and so forth. The power can be plotted by supplying the ‘basename’, and the script automatically detects and uses all the relevant files according to the naming convention above (basename + aperp_C_step.h5):

```
python $PUFFDIR/bin/pyPlotting/plotPowVsZ.py f1main
```

The filtered power can be plotted by supplying same script with an additional 2 numbers describing the filter, in units of the ‘central’ or reference frequency. To filter around the fundamental, do

```
python $PUFFDIR/bin/pyPlotting/plotPowVsZ.py f1main 1 0.2
```

Some of these scripts may not work if the files are too large. In which case, you would need to either write your own parallel scripts to open the files up across many nodes, or use the data in the integrated files.

By using the ‘powPrep’ script, the power data in the integrated files are collected into one single power file. Give it the basename of the run, so:

```
python $PUFFDIR/bin/post/powPrep.py f1main
```

will create the <*basename*>_integrated_all.vsh5 file, which can be viewed in visit, or accessed by other means (it also calculates and stores the total radiated energy as a function of z).

Plot the radius as a function of z, with

```
python $PUFFDIR/bin/pyPlotting/plotBeamRVsZ.py f1main
```

which uses the integrated files to plot the radius.

Optional: ‘Reduce’ the field mesh files to the center nodes in the transverse plane. This will make it easier to process the files yourself:

```
for i in $(ls f1main_aperp_*.h5); do  python $PUFFDIR/bin/post/reduceField.py $i; done
```

This will create the additional smaller files indicated with the *small* label.