EPFL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Using DREAM

Mathias Hoppe, Oskar Vallhagen and Ola Embreus

- Overview of DREAM
- Installation
- The Python interface
  - Input
  - Output
- DREAM modelling
  - Modularity
  - Disruption modelling

Overview of DREAM

- Original idea: Go with kinetic hot-tail a'la Aleynikov (2017)
- Solver for $f_e\left(r, p, \xi_0\right)$
- …together with background plasma (T_e, n_i, $E_\parallel$, …)
  - Evolved self-consistently
  - …or according to prescribed data
- C++ code (in two libraries), with Python interface

$$\mu_0 \frac{j_\parallel}{B} \left\langle \boldsymbol{B} \cdot \nabla \phi \right\rangle = \frac{1}{V'} \frac{\partial}{\partial r} \left[V' \left\langle \frac{|\nabla r|^2}{R^2} \right\rangle \frac{\partial \psi}{\partial r} \right], \qquad \frac{\partial \psi}{\partial t} = -V_{\mathrm{loop}} + \frac{\partial}{\partial \psi_{\mathrm{t}}} \left(\psi_{\mathrm{t}} \mu_0 \Lambda \frac{\partial}{\partial \psi_{\mathrm{t}}} \frac{j_{\mathrm{tot}}}{B} \right),$$

(Ampère/Faraday)

$$\frac{\partial f_{\mathrm{hot/re}}}{\partial t} + eE_\parallel \frac{\partial f_{\mathrm{re/hot}}}{\partial p_\parallel} = C\left(f_{\mathrm{re/hot}}\right),$$

(Fokker–Planck)

$$\frac{\partial W_{\mathrm{cold}}}{\partial t} = \frac{j_\Omega}{B} \left\langle \boldsymbol{E} \cdot \boldsymbol{B} \right\rangle - \left\langle n_{\mathrm{cold}} \right\rangle \sum_i \sum_{j=0}^{Z_i - 1} n_i^{(j)} L_i^{(j)} + \left\langle Q_c \right\rangle + \frac{1}{V'} \frac{\partial}{\partial r} \left[V' \left(A_W W_{\mathrm{cold}} + D_W \frac{\partial W_{\mathrm{cold}}}{\partial r} \right) \right],$$

(Energy balance)

$$\left\langle Q_c \right\rangle = \int \mathrm{d}p \int_{-1}^{1} \mathrm{d}\xi_0 \frac{\mathcal{V}'}{V'} \Delta \dot{E}_{ee} f_{\mathrm{hot/re}} + \sum_i Q_{ei}, \quad Q_{ei} = \frac{\left\langle nZ^2 \right\rangle_k \left\langle nZ^2 \right\rangle_l e^4 \ln \Lambda_{kl}}{(2\pi)^{3/2} \epsilon_0^2 m_k m_l} \frac{T_k - T_l}{\left(\frac{T_k}{m_k} + \frac{T_l}{m_l} \right)^{3/2}}$$

(Collisional heating)

$$\frac{\partial n_i^{(j)}}{\partial t} = \left(I_i^{(j-1)} \left\langle n_{\mathrm{cold}} \right\rangle + \left\langle \sigma_{\mathrm{ion},i}^{(j-1)} v \right\rangle \right) n_i^{(j-1)} - \left(I_i^{(j)} \left\langle n_{\mathrm{cold}} \right\rangle + \left\langle \sigma_{\mathrm{ion},i}^{(j)} v \right\rangle \right) n_i^{(j)} +$$
$$+ R_i^{(j+1)} \left\langle n_{\mathrm{cold}} \right\rangle n_i^{(j+1)} - R_i^{(j)} \left\langle n_{\mathrm{cold}} \right\rangle n_i^{(j)}, \quad \left\langle \sigma_{\mathrm{ion},i}^{(j)} \right\rangle = \int \mathrm{d}p \int_{-1}^{1} \mathrm{d}\xi_0 \frac{\mathcal{V}'}{V'} v \sigma_{\mathrm{ion},i}^{(j)} f_{\mathrm{hot/re}},$$

(Ion balance)

Installation

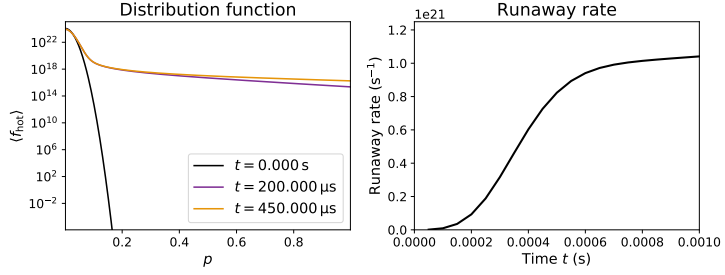
Details in DREAM documentation: https://ft.nephy.chalmers.se/dream

1. Make sure that dependencies are installed:
   - CMake ($\geq$ 3.12), C++17 compiler, GSL ($\geq$ 2.4), HDF5, Python 3
   - PETSc (https://petsc.org/; recommended config in DREAM docs)
   - **Recommended:** Intel MKL
   - **Python packages:** `h5py matplotlib numpy packaging scipy`
2. Download DREAM:
   ```
   git clone https://github.com/chalmersplasmatheory/DREAM.git
   ```
3. Configure and compile DREAM:
   ```
   cd /path/to/DREAM
   mkdir build
   cd build
   cmake ..
   make -j NTHREADS
   ```

(also check `setup/` for installation on various machines)

To check if your DREAM installation works, try to run the `runaway` example, located in `examples/runaway`:

```
$ cd /path/to/DREAM/examples/runaway
  Generate 'dream_settings.h5'
$ ./basic.py
  Run DREAM
$ ../../build/iface/dreami dream_settings.h5
  Visualize output
$ ./plotOutput.py
```



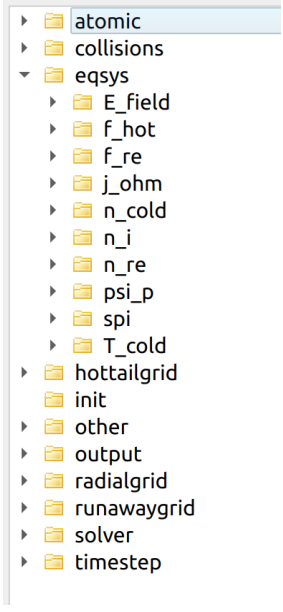
The Python interface

- Located under `py/DREAM`
- Command-line interface: `py/cli/cli.py`
- Two main classes for input/output:
  - `DREAMSettings` – Complete interface for configuring simulations.
  - `DREAMOutput` – Complete interface for postprocessing/visualization of simulations.
- Interface is object-oriented with many properties/methods shared

- Tree hierarchy with settings
  - Each node is a Python object
  - Preferably set settings with method calls
- Most model configuration done under `eqsys`

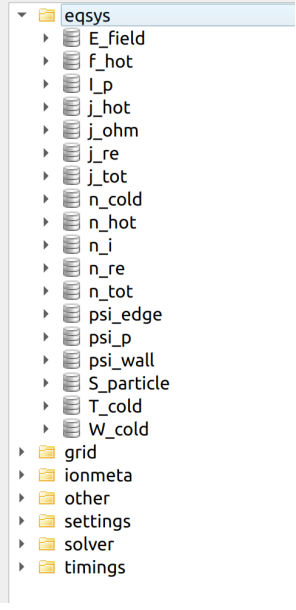
```
from DREAM import DREAMSettings
ds = DREAMSettings()

ds.eqsys.T_cold.setPrescribedData(1000)
ds.hottailgrid.setEnabled(True)
...
```

Detailed documentation online:
https://ft.nephy.chalmers.se/dream/frontend/api/Settings/DREAMSettings.html



- Powerful postprocessing/visualization interface
  - ...but criminally underdocumented...
- Tree structure similar to settings interface
- Interactive Python session set up by running `py/cli/cli.py`

```
>> f_hot.plot()
>> f_hot.plot(t=[0,1,2], r=4)
>> f_hot.animate(speed=50, r=4)
>> I_p.plot()
>> plt.plot(grid.t, j_re.current())
```



Examples of what the output interface can do for you:

- **Quick plotting/animation** of all quantities
- **Kinetic quantities:**
  - Angle average
  - Calculate moments: density, current, kinetic energy, synchrotron, user-specified...
- **Fluid quantities:**
  - Calculate total current (not so simple in toroidal geometry...)
  - Normalize electric field
  - Plot ion charge state evolution
  - Plot summary of runaway generation, energy balance

DREAM modelling

Quiescent runaways `examples/runaway/basic.py`

- Prescribed T_e, n_i, $E_\parallel$
- Solve kinetic equation

Disruption runaways `examples/Disruptions/generate.py`

- Fluid/isotropic/superthermal/fully kinetic
- Evolved in steps
  1. Initialize pre-disruption plasma (ohmic current)
  2. Inject impurities, trigger TQ, let ionize
  3. Evolve through CQ, plateau, …

SPI `examples/SPI/SPI.py`

- Basic JET-like example
- Single SPI shard as in Figure 1 of Gál *et al* (2008)