

Progress on embedded turbulence computations in Facets

A. Hakim, S. Vadlamani and J. Carlsson

Facets Winter Meeting, Feb 24th 2011

Core transport equations are coupled to equilibrium

$$\begin{aligned}\left[\frac{1}{V'} \frac{\partial}{\partial t} V' + \dot{\rho} \frac{\partial}{\partial \rho} \right] n + \frac{1}{V'} \frac{\partial}{\partial \rho} (V' \Gamma_n) &= S_n \\ \left[\frac{1}{V'^{5/3}} \frac{\partial}{\partial t} V'^{5/3} + \dot{\rho} \frac{\partial}{\partial \rho} \right] \frac{nT}{\gamma - 1} + \frac{1}{V'} \frac{\partial}{\partial \rho} (V' \Gamma_E) &= S_E \\ \left[\frac{1}{V'} \frac{\partial}{\partial t} V' + \dot{\rho} \frac{\partial}{\partial \rho} \right] L_\Omega + \frac{1}{V'} \frac{\partial}{\partial \rho} (V' \Gamma_\Omega) &= S_\Omega \\ \frac{\partial \psi}{\partial t} - \frac{\eta_{\parallel}^{\text{nc}}}{\mu_0} \Delta^+ \psi &= S_\psi\end{aligned}$$

These equations are coupled to 2D equilibrium equation

$$\Delta^* \psi = -FF' - R^2 \mu_0 p'$$

There are $N_i + N_i + 1 + 1 + 1$ 1D equations coupled to 2D equilibrium equation.

Core transport equations can be formulated as a system of balance laws

Consider the one-dimensional system of m balance laws

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} = \mathbf{S}$$

Here $0 < x < x_e$, $\mathbf{Q}(x, t) \in \mathbb{R}^m$, $\mathbf{F}(\mathbf{Q}, \partial \mathbf{Q} / \partial x)$ is the flux $\mathbf{S}(\mathbf{Q}, x, t)$ are source terms. Geometric terms are ignored.

- ▶ Boundary conditions $\mathbf{F} = 0$ at $x = 0$ and either \mathbf{Q} or \mathbf{F} is specified at $x = x_e$.

Example: coupled turbulent evolution of electron density and energy

Consider a system of two coupled equations (density and heat flux), i.e. $\mathbf{Q} = [n, 3nT/2]^T$, with fluxes given by $\mathbf{F} = [F_1, F_2]^T$ where

$$F_1 = -D_{11}\partial n/\partial x - D_{12}\partial T/\partial x$$

$$F_2 = -D_{21}\partial n/\partial x - nD_{22}\partial T/\partial x$$

where the transport matrix \mathbf{D} is positive definite, i.e. $\det \mathbf{D} > 0$. (Horton, Hu, Dong and Zhu, *New Journal of Physics*, 2003)

Example continued: three types of flux-gradient relation with off-diagonal diffusivities

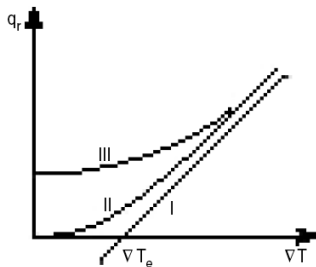


Figure: Three types of transport processes: Type I starts above a critical gradient, in Type II the flux vanishes when the gradient vanishes and in Type III flux is finite when the gradient vanishes.

In FACETS we presently use a finite-volume scheme to discretize the transport equations

$$\mathbf{Q}_i^{n+1} = \mathbf{Q}_i^n - \frac{1}{\Delta x_i} \left(\mathbf{F}_{i+1/2}^{n+\theta} - \mathbf{F}_{i-1/2}^{n+\theta} \right) + \Delta t \mathbf{S}_i^{n+\theta}$$

here $\mathbf{S}_i^{n+\theta} = \mathbf{S}(\mathbf{Q}_i^{n+\theta}, x, t)$ and

$$\mathbf{F}_{i-1/2}^{n+\theta} = \mathbf{H}(\mathbf{Q}_{i-1}^{n+\theta}, \mathbf{Q}_i^{n+\theta})$$

is a *numerical flux* function, and $Q^{n+\theta} = (1 - \theta)Q^n + \theta Q^{n+1}$. The numerical flux must satisfy the *consistency condition*

$$\lim_{\epsilon \rightarrow 0} \mathbf{H}(\mathbf{Q}(x - \epsilon, t), \mathbf{Q}(x + \epsilon, t)) = \mathbf{F}(\mathbf{Q}, \partial \mathbf{Q} / \partial x)$$

This ensures convergence of scheme as $\Delta t, \Delta x \rightarrow 0$.

In FACETS we presently use a finite-volume scheme to discretize the transport equations

$$\mathbf{Q}_i^{n+1} = \mathbf{Q}_i^n - \frac{1}{\Delta x_i} \left(\mathbf{F}_{i+1/2}^{n+\theta} - \mathbf{F}_{i-1/2}^{n+\theta} \right) + \Delta t \mathbf{S}_i^{n+\theta}$$

here $\mathbf{S}_i^{n+\theta} = \mathbf{S}(\mathbf{Q}_i^{n+\theta}, x, t)$ and

$$\mathbf{F}_{i-1/2}^{n+\theta} = \mathbf{H}(\mathbf{Q}_{i-1}^{n+\theta}, \mathbf{Q}_i^{n+\theta})$$

is a *numerical flux* function, and $Q^{n+\theta} = (1 - \theta)Q^n + \theta Q^{n+1}$. The numerical flux must satisfy the *consistency condition*

$$\lim_{\epsilon \rightarrow 0} \mathbf{H}(\mathbf{Q}(x - \epsilon, t), \mathbf{Q}(x + \epsilon, t)) = \mathbf{F}(\mathbf{Q}, \partial \mathbf{Q} / \partial x)$$

This ensures convergence of scheme as $\Delta t, \Delta x \rightarrow 0$.

Boundary conditions for the discrete system can lead to additional equations

- ▶ At $x = 0$ we set $\mathbf{F}_{1/2} = 0$.
- ▶ If flux is specified at $x = x_e$ then we set $\mathbf{F}_{N+1/2} = \mathbf{F}_e(t)$.
- ▶ If time-dependent values are specified, then we introduce a “ghost cell” the same size at the last cell C_N . Then, we set $\mathbf{Q}_{N+1} = 2\mathbf{Q}_e(t) - \mathbf{Q}_N$. The flux at the last edge is then

$$\mathbf{F}_{N+1/2}^{n+\theta} = \mathbf{H}(\mathbf{Q}_N^{n+\theta}, 2\mathbf{Q}_e(t + \theta\Delta t) - \mathbf{Q}_N^{n+\theta}).$$

There are several open questions as we improve our core solver

- ▶ How to implement grid sequencing more flexibly?
- ▶ How to we minimize the number of flux evaluations? TGLF and GYRO are *very* expensive.
- ▶ What time-stepping schemes to use? Currently FACETS computes *increments* $\Delta Q \equiv Q^{n+1} - Q^n$ which allows implementation in input files of different time-stepping schemes. Should we let Petsc do the time-stepping using the Time-Stepper layer?
- ▶ What is the optimum infrastructure to handle parallel flux computations? (Tom E's work is a start)
- ▶ How to deal with the fact that fluxes from GYRO are noisy? Are there robust algorithms to handle this?

Not directly related: do we make TGLF parallel?

There are several open questions as we improve our core solver

- ▶ How to implement grid sequencing more flexibly?
- ▶ How to we minimize the number of flux evaluations? TGLF and GYRO are *very* expensive.
- ▶ What time-stepping schemes to use? Currently FACETS computes *increments* $\Delta Q \equiv Q^{n+1} - Q^n$ which allows implementation in input files of different time-stepping schemes. Should we let Petsc do the time-stepping using the Time-Stepper layer?
- ▶ What is the optimum infrastructure to handle parallel flux computations? (Tom E's work is a start)
- ▶ How to deal with the fact that fluxes from GYRO are noisy? Are there robust algorithms to handle this?

Not directly related: do we make TGLF parallel?

There are several open questions as we improve our core solver

- ▶ How to implement grid sequencing more flexibly?
- ▶ How to we minimize the number of flux evaluations? TGLF and GYRO are *very* expensive.
- ▶ What time-stepping schemes to use? Currently FACETS computes *increments* $\Delta Q \equiv Q^{n+1} - Q^n$ which allows implementation in input files of different time-stepping schemes. Should we let Petsc do the time-stepping using the Time-Stepper layer?
- ▶ What is the optimum infrastructure to handle parallel flux computations? (Tom E's work is a start)
- ▶ How to deal with the fact that fluxes from GYRO are noisy? Are there robust algorithms to handle this?

Not directly related: do we make TGLF parallel?

There are several open questions as we improve our core solver

- ▶ How to implement grid sequencing more flexibly?
- ▶ How to we minimize the number of flux evaluations? TGLF and GYRO are *very* expensive.
- ▶ What time-stepping schemes to use? Currently FACETS computes *increments* $\Delta Q \equiv Q^{n+1} - Q^n$ which allows implementation in input files of different time-stepping schemes. Should we let Petsc do the time-stepping using the Time-Stepper layer?
- ▶ What is the optimum infrastructure to handle parallel flux computations? (Tom E's work is a start)
- ▶ How to deal with the fact that fluxes from GYRO are noisy? Are there robust algorithms to handle this?

Not directly related: do we make TGLF parallel?

There are several open questions as we improve our core solver

- ▶ How to implement grid sequencing more flexibly?
- ▶ How to we minimize the number of flux evaluations? TGLF and GYRO are *very* expensive.
- ▶ What time-stepping schemes to use? Currently FACETS computes *increments* $\Delta Q \equiv Q^{n+1} - Q^n$ which allows implementation in input files of different time-stepping schemes. Should we let Petsc do the time-stepping using the Time-Stepper layer?
- ▶ What is the optimum infrastructure to handle parallel flux computations? (Tom E's work is a start)
- ▶ How to deal with the fact that fluxes from GYRO are noisy? Are there robust algorithms to handle this?

Not directly related: do we make TGLF parallel?

There are several open questions as we improve our core solver

- ▶ How to implement grid sequencing more flexibly?
- ▶ How to we minimize the number of flux evaluations? TGLF and GYRO are *very* expensive.
- ▶ What time-stepping schemes to use? Currently FACETS computes *increments* $\Delta Q \equiv Q^{n+1} - Q^n$ which allows implementation in input files of different time-stepping schemes. Should we let Petsc do the time-stepping using the Time-Stepper layer?
- ▶ What is the optimum infrastructure to handle parallel flux computations? (Tom E's work is a start)
- ▶ How to deal with the fact that fluxes from GYRO are noisy? Are there robust algorithms to handle this?

Not directly related: do we make TGLF parallel?

Example of GYRO diffusivities computed for shot 118897, 3400 ms into discharge

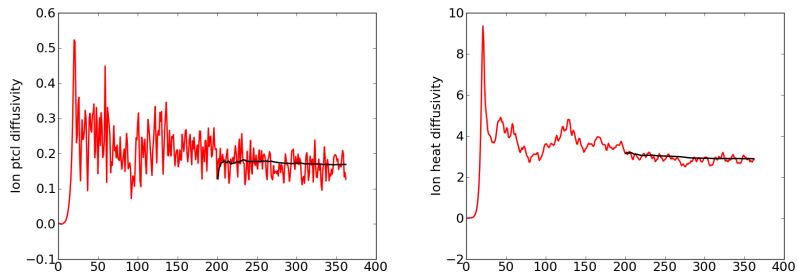


Figure: Ion particle diffusivity (left) and heat diffusivity (right). The red line shows diffusivity from GYRO and the black line a running average starting from 200th time-step.

Current progress in improved core solver

- ▶ Created a new core component class that allows for arbitrary number of species and any combination of profiles (density and temperature).
- ▶ Ability to specify some profiles as “kinematic”, for example, from experimental measurements.
- ▶ Added ability to specify non-uniform meshes. Allows refinement in pedestal.
- ▶ Created a C++ class wrapping PetSc SNES. Allows different solver classes to provide a C++ method implementing function to find roots and Jacobian evaluation. Tested with linear and non-linear diffusion solvers.

Next step: Figure out how to implement grid sequencing and integrate load balancing code.

Current progress in improved core solver

- ▶ Created a new core component class that allows for arbitrary number of species and any combination of profiles (density and temperature).
- ▶ Ability to specify some profiles as “kinematic”, for example, from experimental measurements.
- ▶ Added ability to specify non-uniform meshes. Allows refinement in pedestal.
- ▶ Created a C++ class wrapping PetSc SNES. Allows different solver classes to provide a C++ method implementing function to find roots and Jacobian evaluation. Tested with linear and non-linear diffusion solvers.

Next step: Figure out how to implement grid sequencing and integrate load balancing code.

Current progress in improved core solver

- ▶ Created a new core component class that allows for arbitrary number of species and any combination of profiles (density and temperature).
- ▶ Ability to specify some profiles as “kinematic”, for example, from experimental measurements.
- ▶ Added ability to specify non-uniform meshes. Allows refinement in pedestal.
- ▶ Created a C++ class wrapping PetSc SNES. Allows different solver classes to provide a C++ method implementing function to find roots and Jacobian evaluation. Tested with linear and non-linear diffusion solvers.

Next step: Figure out how to implement grid sequencing and integrate load balancing code.

Current progress in improved core solver

- ▶ Created a new core component class that allows for arbitrary number of species and any combination of profiles (density and temperature).
- ▶ Ability to specify some profiles as “kinematic”, for example, from experimental measurements.
- ▶ Added ability to specify non-uniform meshes. Allows refinement in pedestal.
- ▶ Created a C++ class wrapping PetSc SNES. Allows different solver classes to provide a C++ method implementing function to find roots and Jacobian evaluation. Tested with linear and non-linear diffusion solvers.

Next step: Figure out how to implement grid sequencing and integrate load balancing code.

Current progress in improved core solver

- ▶ Created a new core component class that allows for arbitrary number of species and any combination of profiles (density and temperature).
- ▶ Ability to specify some profiles as “kinematic”, for example, from experimental measurements.
- ▶ Added ability to specify non-uniform meshes. Allows refinement in pedestal.
- ▶ Created a C++ class wrapping PetSc SNES. Allows different solver classes to provide a C++ method implementing function to find roots and Jacobian evaluation. Tested with linear and non-linear diffusion solvers.

Next step: Figure out how to implement grid sequencing and integrate load balancing code.