THE UNIVERSITY of York

Using PETSc solvers in BOUT++

Ben Dudson

York Plasma Institute, University of York, UK

benjamin.dudson@york.ac.uk

4th September 2013

Compiling BOUT++ for workshop

If you haven't yet compiled BOUT++ with PETSc:

1) Log into Hopper

2) Delete and re-download BOUT++

```
rm -rf BOUT-2.0
git clone
   https://github.com/boutproject/BOUT-2.0.git
```

3) Re-download and run workshop configuration script:

```
cd BOUT-2.0
source configure.workshop
```

To check what BOUT++ is configured with, check **make.config**

PETSc and MUMPS solvers

External solvers are used for two components:

1) Time integration
    - Currently fixed timestep methods available
    - See Wednesday's talk on preconditioning

2) Inversion of Laplacian or Helmholtz type problems
    - Solution of linear spatial PDEs which arise in drift-reduced models (vorticity or polarisation equations)
    - Optional replacements for built-in solver

Motivation

Many plasma models of interest involve an equation of the form

$$\nabla \cdot (n \nabla_\perp \phi) = U$$

Where n is the total density, φ is electrostatic potential, and U is the vorticity.

The built-in solvers in BOUT++ make the following assumptions:

1) Parallel derivatives are small, so y derivatives are ignored
   → 2D solves in (X,Z)

2) Coefficient n is constant in Z (toroidal angle) so that FFTs can be used in Z to decompose into toroidal modes

Motivation

Many plasma models of interest involve an equation of the form

$$\nabla \cdot (n \nabla_\perp \phi) = U$$

Where n is the total density, φ is electrostatic potential, and U is the vorticity.

The built-in solvers in BOUT++ make the following assumptions:

1) Parallel derivatives are small, so y derivatives are ignored
   → 2D solves in (X,Z)
   → Work in progress. See Dudson poster on Friday

2) Coefficient n is constant in Z (toroidal angle) so that FFTs can be used in Z to decompose into toroidal modes
   → PETSc and MUMPS solvers do not make this assumption

Preconditioning of PETSc Laplace

- MUMPS is a direct solver, so doesn't need preconditioning
  → Tends to be more robust (will converge)
  → Generally slow: 10 – 100x slower than built-in solver

- PETSc can use direct solvers (including MUMPS), but primary use is for iterative solvers (KSP component)
  → May not converge if starting solution isn't “close enough”
  → Can be much more efficient (< 10 x built-in)

Preconditioning of PETSc Laplace

- MUMPS is a direct solver, so doesn't need preconditioning
  → Tends to be more robust (will converge)
  → Generally slow: 10 – 100x slower than built-in solver

- PETSc can use direct solvers (including MUMPS), but primary use is for iterative solvers (KSP component)
  → May not converge if starting solution isn't “close enough”
  → Can be much more efficient (< 10 x built-in)

- Many “black box” preconditioning methods are available in PETSc e.g. jacobi, sor, gamg

```
[laplace]
type = petsc
pctype = sor
rightprec = true
```

- There is another way...

Preconditioning of PETSc Laplace

- Reminder: Aim is to solve a linear problem of the form:

$$\mathbb{A}\mathbf{x} = \mathbf{b}$$

A preconditioner is a fast approximate inverse of A:

$$\mathbb{P} \simeq \mathbb{A}^{-1}$$

For left preconditioning, the iterative (PETSc) solver is now solving

$$\underbrace{(\mathbb{P}\mathbb{A})}_{\mathbb{A}'}\mathbf{x} = \underbrace{\mathbb{P}\mathbf{b}}_{\mathbf{b}'}$$

- Most of the time the built-in solver is a good approximation to the full problem
  → Use the approximate solver to precondition PETSc solver
  → If n is constant in Z then preconditioner is inverse of A (apart from differences in Z discretisation, FFT vs FD)

Preconditioning of PETSc Laplace

- Set laplace solver and preconditioner types

```
[laplace]
type = petsc
pctype = user
rightprec = true
```

"user" → use a second solver
Right preconditioning by default

- Set the options for the preconditioner in a subsection

```
[laplace:precon]
filter      = 0.
flags       = 49152
```

Leave type to default (tri or spt)
Don't filter, or preconditioner is singular
Set boundary to identity

- Use the new(er) object interface to pass 3D fields

See test-laplace2 and test-petsc-laplace examples

Preconditioning of PETSc Laplace

Timings for **test-laplace2** with n varying in Z:

```
n = sin(x) * gauss(x-0.5) * (1.0 + 0.9*cos(z))
```

Inversion time in seconds (iterations)

preconditioner \ **Resolution (X x Z)**	40 x 32	40 x 32	40 x 32	80 x 64
Z perturb	10%	50%	90%	90%
none	0.157 (319)	0.110 (226)	0.142 (299)	
jacobi	0.162 (318)	0.113 (224)	0.137 (299)	
sor	0.022 (30)	0.048 (35)	0.048 (40)	0.610 (178)
user	0.013 (4)	0.015 (5)	0.024 (9)	0.110 (8)

FFT solver: ~ 0.002s

Tests and applications

- **test-laplace2** is a case which just runs two different solvers and compares the results. Useful for quick checks and timing
- **test-petsc-laplace** is a more complicated case which compares different order methods (2^{nd}, 4^{th}) against analytical solutions

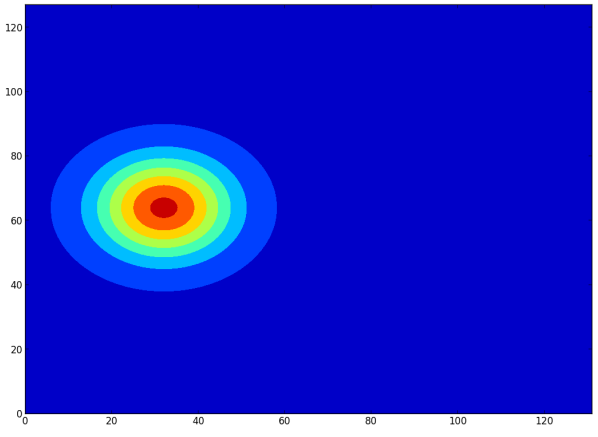
- **blob2d**: 2D Blob dynamics example
  → A simplified model for density blobs/holes in the SOL
  → Can be used to test the impact of the Boussinesq behavior
  → Can be extended in many ways for physics studies

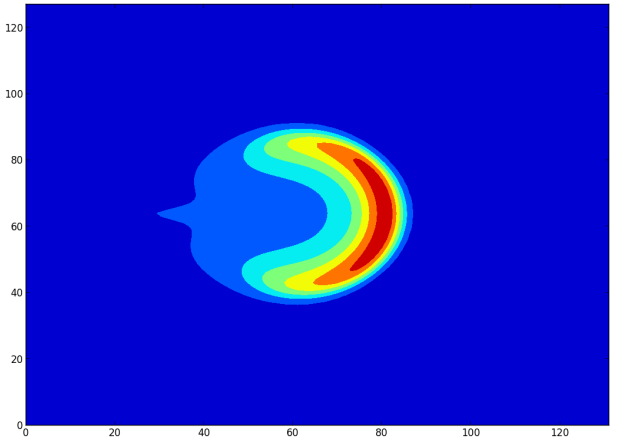
2D blob problem

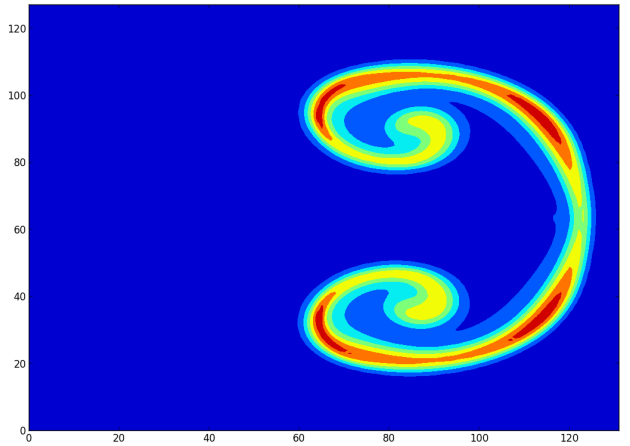
- Slab simulation of a plasma 'blob' moving in the SOL (drift plane)
- A physical system with large density inhomogeneities (100%)

$$\frac{\partial n}{\partial t} = -\mathbf{v}_E \cdot \nabla n + 2\frac{\rho_s}{R_c}\frac{\partial n}{\partial z} + D_n \nabla_\perp^2 n$$

$$\frac{\partial \omega}{\partial t} = -\mathbf{v}_E \cdot \nabla \omega + 2\frac{\rho_s}{R_c}\frac{1}{n}\frac{\partial n}{\partial z} + D_\omega \frac{1}{n}\nabla_\perp^2 \omega$$







2D blob problem

- Vorticity equation solved for electrostatic potential:

$$\nabla \cdot (n \nabla_\perp \phi) = \omega$$

- The Laplacian class solves boundary-value problems of form:

$$D\nabla_\perp^2 \phi + \frac{1}{C} \nabla_\perp C \cdot \nabla_\perp \phi + A\phi = \omega$$

$$D = 1 \qquad C = None \qquad A = 0$$

→ In **examples/blob2d**:

```
Laplacian *phiSolver;
```

Pointer to a solver

```
phiSolver = Laplacian::create(Options::getRoot()
                              ->getSection("phiSolver"));
```

Create a solver, using options from BOUT.inp

```
phiSolver->setCoefC(n);
```

Change the coefficients and solve

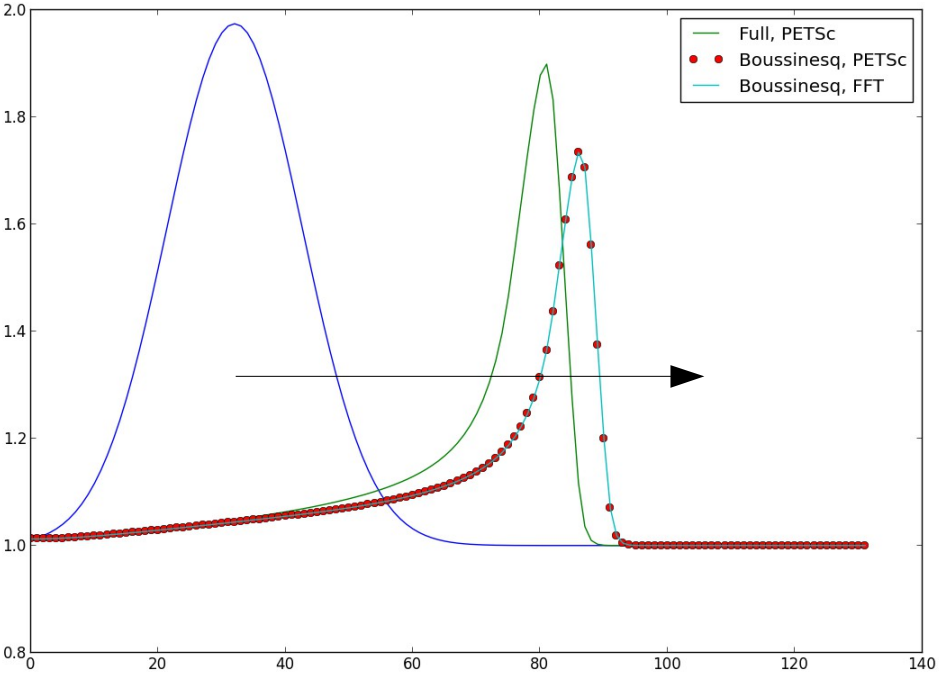
```
phi = phiSolver->solve(omega / n, phi);
```

2D blob problem

- The Boussinesq approximation can be switched on and off in BOUT.inp input file

```
boussinesq = false
```

- If Boussinesq = true, options in `[phiBoussinesq]` are used
  → Defaults to FFT methods



- If Boussinesq = false, options in [phiSolver] are used
  → PETSc + FFT precon

- Some differences seen in radial velocity
  → Still needs investigation

2D blob problem

- Quick (flawed) timing comparison using 128 x 128 mesh, T=5e3

  Boussinesq (FFT method) : 8m 23s

  Boussinesq (PETSc 4th-order): 47 m 37 s

  Full (non-Boussinesq, PETSc 4th-order): 41 m 12 s

- Vorticity inversion went from ~14% of run-time to ~80%

- Depending on your problem, this extra cost may or may not be worthwhile

- Simple to try (same interface). See examples e.g. blob2d