Verification using the Method of Manufactured Solutions

Ben Dudson (UoY),
Jens Madsen (DTU), John Omotani (CCFE)

York Plasma Institute, Department of Physics,
University of York, Heslington, York YO10 5DD, UK

BOUT++ Workshop

16th September 2014

Outline

- Verification and MMS
- Verification of BOUT++
- Implementation
- Example

- **Verification**: Checks that a chosen set of partial differential equations is solved correctly and consistently
  - As the spatial and temporal mesh is refined the solution converges to a solution of the continuum equations
  - Order of convergence should be the accuracy expected of the numerical scheme used
- **Validation**: Checks that the correct set of equations has been chosen
  - Usually involves comparison against experiment

Verification methods

- Inspection of the output for “reasonable” features
- Comparison against analytic solutions
- Convergence to an analytic solution
- Code cross-comparisons
- Convergence to a manufactured solution

Manufactured solutions

How do you test convergence to an exact solution, when no analytical solution can be found for your equations?
$\rightarrow$ Change your equations!

- If you are solving a set of equations to solve, for quantities $\underline{f}$:

$$\frac{\partial \underline{f}}{\partial t} = F\left(\underline{f}\right)$$

- Add a source term S:

$$\frac{\partial \underline{f}}{\partial t} = F\left(\underline{f}\right) + S\left(t\right)$$

- Now choose a (manufactured) solution $\underline{f}^M$ and calculate S

$$S = \frac{\partial \underline{f}^M}{\partial t} - F\left(\underline{f}^M\right)$$

- S can be calculated analytically, and evaluated to machine precision
- When the modified equations are solved numerically, any error must come from the discretisation of F or time integration.

Manufactured solutions: Example

By inserting a known source into the equations, a solution can be chosen for an arbitrarily complex set of equations [1]

e.g. Viscid Burger's equation:

$$\frac{\partial u}{\partial t} = \underbrace{-u\frac{\partial u}{\partial x} + \nu\frac{\partial^2 u}{\partial x^2}}_{F} \underbrace{+S}_{\text{Added source}}$$

Choose a solution for u

$$u = \sin\left(x - t\right)$$

Insert it into the equation to calculate the source S

$$S = \underbrace{-\cos\left(x - t\right)}_{\partial u/\partial t} + \underbrace{\sin\left(x - t\right)\cos\left(x - t\right)}_{u \cdot \partial u/\partial x} + \underbrace{\nu \cdot \sin\left(x - t\right)}_{-\nu\partial^2 u/\partial x^2}$$

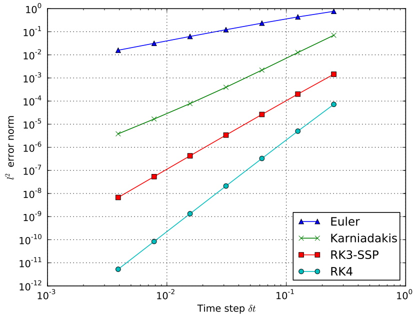
[1] With some restrictions

Manufactured solutions: Example

- The simulation code is now modified slightly, adding a time-dependent source to the equations
- Start at $t = 0$ with the manufactured solution
- Run the simulation for a short time Δt
- The difference between the exact and numerical solution $\epsilon = f(\Delta t) - f^M(\Delta t)$ at $t = \Delta t$ is due to numerical error
- This error should decrease towards machine precision as the resolution (time and spatial) of the simulation is increased (i.e. converge).
- The rate of convergence at high resolution ("asymptotic" regime) should agree with the expected rate e.g. $\epsilon \propto \delta x^2$ for second-order central differencing.

BOUT++ has been recently tested using this method[2]

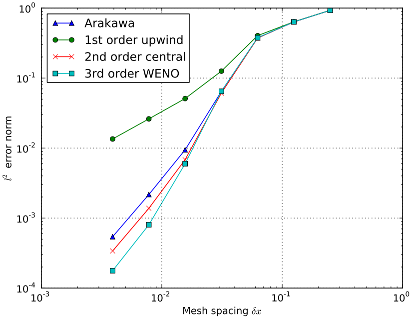
- Time integration schemes



[2] Enabling Research project CfP-WP14-ER-01/Swiss Confederation-01

BOUT++ has been recently tested using this method[2]

- Time integration schemes
- Advection schemes



Note: Limited to second order accurate by boundary conditions and calculation of velocity field

[2]Enabling Research project CfP-WP14-ER-01/Swiss Confederation-01

Verification of BOUT++

BOUT++ has been recently tested using this method[2]

- Time integration schemes
- Advection schemes
- Boundary conditions (particular thanks to Jens, John, and Luke)

Boundary conditions require additional modifications for MMS testing

e.g Neumann boundary conditions

$$\frac{\partial f}{\partial x} = 0$$

must be modified since in general

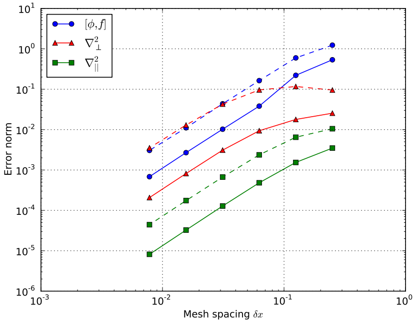
$$\frac{\partial f^M}{\partial x} = N(t) \neq 0$$

Boundaries now located half-way between cells

[2] Enabling Research project CfP-WP14-ER-01/Swiss Confederation-01

BOUT++ has been recently tested using this method[2]

- Time integration schemes
- Advection schemes
- Boundary conditions (particular thanks to Jens, John, and Luke)
- Toroidal coordinates, and shifted metric procedure

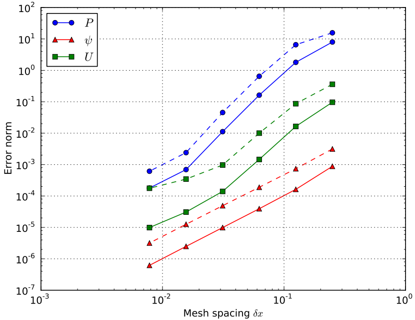


[2] Enabling Research project CfP-WP14-ER-01/Swiss Confederation-01

Verification of BOUT++

BOUT++ has been recently tested using this method[2]

- Time integration schemes
- Advection schemes
- Boundary conditions (particular thanks to Jens, John, and Luke)
- Toroidal coordinates, and shifted metric procedure
- 3-field (p, U, Ψ) reduced MHD for ELM simulations

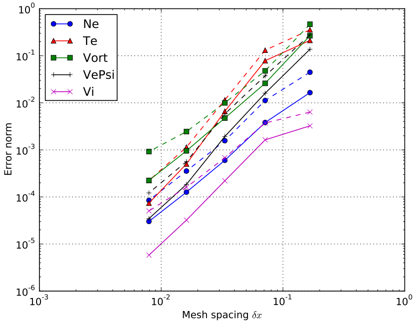


[2] Enabling Research project CfP-WP14-ER-01/Swiss Confederation-01

Verification of BOUT++

BOUT++ has been recently tested using this method[2]

- Time integration schemes
- Advection schemes
- Boundary conditions (particular thanks to Jens, John, and Luke)
- Toroidal coordinates, and shifted metric procedure
- 3-field (p, U, Ψ) reduced MHD for ELM simulations
- 5-field $(n, T_e, \omega, v_{\|e}, v_{\|i})$ equations for turbulence simulations



---

[2]Enabling Research project CfP-WP14-ER-01/Swiss Confederation-01

MMS testing requires three new features:

- Calculating manufactured solutions f^M from analytic expressions
  $\rightarrow$ Initialisation, and calculation of errors
- Calculating exact sources S from analytic solutions
  $\rightarrow$ Added to time-derivatives
- Calculation of boundary condition values from analytic expressions
  $\rightarrow$ Modify boundary conditions by adding source

Analytic expressions can become extremely large
$\rightarrow$ All of these expressions must be copied from Mathematica or Python into BOUT++ inputs without modification.

Implementation in BOUT++

BOUT++ contains a parser which can evaluate analytic expressions

- Used previously to set initial conditions
- Significantly enhanced, allowing more complex expressions
- Made consistent with definitions of boundary locations, sufficiently exact for MMS

Parses and evaluates a string:

```
#include <field_factory.hxx>

FieldFactory *factory = FieldFactory::get();

Field3D f = factory->create3D("cos(y-z)");
```

Can include input options in expressions:

```
Field3D f = factory->create3D("ne:solution")
```

which could be set in BOUT.inp:

```
[ne]
solution = x^2 + sin(y)
```

or on the command line

```
$ mpirun -np 4 ./mycode ne:solution=cosh(x)
```

Implementation in BOUT++: Solutions and sources

This `FieldFactory` is used by the time integration `Solver` class to evaluate analytic expressions

- Enabled by setting `solver:mms` to `true`
- For each evolving variable, `Solver` looks up a `solution` and `source` option

```
[n]

solution = 0.9*xl + 0.2*sin(5.0*xl^2 - 2*zl)*cos(10*t) + 0.9

source   = 0.9*x - 1.0*(0.5*x - sin(3.0*x^2 - 3*z)*cos(7*t))*sin(pi*x)
          - 1.0*(-20.0*x^2*sin(5.0*x^2 - 2*z) + 2.0*cos(5.0*x^2 - 2*z))
          *cos(10*t) + 0.4*(pi*(0.5*x - sin(3.0*x^2 - 3*z)*cos(7*t))
          *cos(pi*x) + (-6.0*x*cos(7*t)*cos(3.0*x^2 - 3*z) + 0.5)*sin(pi*x))
          ...
```

- Whenever the user's RHS function $F(\underline{f}, t)$ is called, the source term is evaluated at the time t, and added to the time-derivative given to the integration code (e.g RK4, CVODE, PETSc, ...)

Implementation in BOUT++: Boundary conditions

The same mechanisms are used to set boundary conditions, which now depend on position and time.

- Boundary conditions are usually set in the input

```
[ne]
bndry_all = dirichlet_o2
```

- These can be given optional input expressions:

```
[ne]
bndry_all = dirichlet_o2(ne:solution)
```

- To apply the boundary condition, this expression will be evaluated for each point on the boundary, every time F is calculated

Implementation in BOUT++

- For every evolving variable, the solver will add another output, appending "E_" to the name.
- This can be collected as usual, and used to calculate an error
- The testing procedure can be automated. See Python scripts in the `examples/MMS` subdirectories

Some issues to be aware of:

- Solutions should be chosen to exercise the relevant physics, and so that the magnitude of
- Solutions should be smooth, obey periodicity constraints, and physical constraints such as $n_e > 0$
- Input expressions use normalised x, y and z coordinates, which are uniform in grid cell number. Transforming between coordinates is a common source of error
- The `FieldFactory` code was not written with efficiency in mind, so this can be a little slow on large sets of equations

Conclusions

- The Method of Manufactured Solutions (MMS) provides a rigorous way to test that a set of equations is implemented correctly
- BOUT++ has been modified to make testing easier, with little to no modification to user code
- Large parts of BOUT++ have now been tested
- Several bugs and inconsistencies found and fixed. Fortunately none of them serious.
- It is highly recommended that MMS tests are used for all present and future implementations in BOUT++. It will save time for the deveoper by discovering errors before the code is used to ”do physics”

- Getting everything right can be a long and frustrating experience
- The end result (straight lines) doesn’t make for exciting talks!