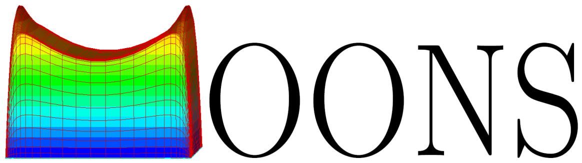

Magnetohydrodynamic Object-Oriented Numerical Simulation (MOONS)

C. Kawczynski

Department of Mechanical and Aerospace Engineering

University of California Los Angeles, USA

December 29, 2014

Abstract

UCLA has developed a Magnetohydrodynamic Object-Oriented Numerical Simulation (MOONS) finite difference code. This report is an attempt to document 1) the governing equations that MOONS solves 2) techniques that MOONS implements when solving solving these equations 3) ideas that have been used to develop MOONS 4) benchmarks that MOONS has performed. 5) provide a low barrier to entry to new users.

1 Introduction

The purpose of MOONS was to compare a small research code with a commercial code, HIMAG, developed by HyperComp with help from UCLA. A new formulation was being implemented in HIMAG, and MOONS was developed to compare the old and new version of HIMAG for simple test cases that MOONS could handle. There are derivations of selected equations in the documentation section of MOONS.

2 Governing Equations

2.1 Momentum

The momentum equation is solved using the same numerical techniques as described in [1]. Non-dimensionalizing the momentum equation with

$$U_i^* = U_i/U_c \qquad \nabla^* = L_c\nabla \qquad t^* = t/t_c \qquad t_c = L_c/U_c$$

Yields the dimensionless momentum equation. Removing the asterisks for simplicity, the momentum equation is

$$\frac{\partial \boldsymbol{u}}{\partial t} = -(\boldsymbol{u} \bullet \nabla)\boldsymbol{u} - \nabla p + Re^{-1}\nabla^2 \boldsymbol{u} + \frac{L_c}{\rho_c U_c^2}\boldsymbol{f} + \frac{Ha^2}{Re}\boldsymbol{j} \times \boldsymbol{B} \tag{1}$$

Or, using index notation, the ith component is

$$\partial_t u_i = -u_j \partial_j u_i - \partial_i p + Re^{-1}\partial_j \partial_j u_i + \frac{L_c}{\rho_c U_c^2} f_i + \frac{Ha^2}{Re}\epsilon_{imn} j_m B_n \tag{2}$$

$$Re = \frac{U_c L_c}{\nu_c} \qquad\qquad Ha = B_c L_c \sqrt{\sigma_c/\mu_c} \tag{3}$$

Where f_i is some dimensional force per unit volume along the ith direction. The advective term can be treated in one of two ways.

1) Conservative, Donor-cell form (neglecting sign):

$$\partial_j (u_j u_i) \tag{4}$$

2) Advective form (neglecting sign)

$$u_{j,ave}\partial_j u_i \tag{5}$$

This can be specified in user/simParams.f90

2.2 Full Induction

Solving for the electric field in Ohm's law, and the current in Ampere's law, and combining them with Faraday's law yields the induction equation. Non-dimensionalizing the induction equation with the same characteristic values from above as well as

$$B_i^* = B_i/B_c, \qquad \sigma^* = \sigma/\sigma_c, \qquad \mu^* = \mu/\mu_c, \qquad E_i^* = E_i/(U_c B_c), \qquad j_i^* = \frac{j}{B_i/(\mu_c L_c)}$$

Yields the dimensionless induction equation. Removing the asterisks for simplicity we have

$$\frac{\partial \boldsymbol{B}}{\partial t} = \nabla \times (\boldsymbol{u} \times \boldsymbol{B}) - \frac{1}{Re_m}\nabla \times \left(\frac{1}{\sigma}\nabla \times \frac{\boldsymbol{B}}{\mu}\right) \tag{6}$$

With the constraint

$$\nabla \bullet \boldsymbol{B} = \boldsymbol{0} \tag{7}$$

Or

$$\frac{\partial \boldsymbol{B}}{\partial t} = -\nabla \times \boldsymbol{E}, \qquad \boldsymbol{E} = \frac{1}{Re_m}\frac{\boldsymbol{j}}{\sigma} - \boldsymbol{u} \times \boldsymbol{B} \tag{8}$$

Using index notation and the Kronecker Delta identity, we may rewrite our diffusion and advection terms:

$$\nabla \times (\boldsymbol{u} \times \boldsymbol{B}) = -\partial_j(u_j B_i - u_i B_j), \qquad \frac{1}{Re_m}\nabla \times \left(\frac{1}{\sigma}\nabla \times \frac{B}{\mu}\right) = \frac{1}{Re_m}\partial_j\left(\frac{1}{\sigma}\left[\partial_i \frac{B_j}{\mu} - \partial_j \frac{B_i}{\mu}\right]\right)$$

So the ith component of the induction equation is

$$\partial_t B_i = -\partial_j(u_j B_i - u_i B_j) - \frac{1}{Re_m}\partial_j\left(\frac{1}{\sigma}\left[\partial_i \frac{B_j}{\mu} - \partial_j \frac{B_i}{\mu}\right]\right) \tag{9}$$

Where

$$Re_m = \frac{U_c L_c}{(\mu_c \sigma_c)^{-1}} \tag{10}$$

Note that the subscript c indicates a 'characteristic' uniform, time-independent, constant. Also note, the dimensionless material properties, σ and μ, in these equations may vary, in general, in space and or time.

2.3 Low Magnetic Reynolds Number approximation

Multiplying 9 by Re_m and setting $Re_m$to zero results in only the diffusion term for the induced field and only the advection term for the applied field. Combining these two results, and introducing a pseudo time-stepping, we may write this as

$$\partial_s B_i = -\partial_j(u_j B_i^0 - u_j B_i^0) - \frac{1}{Re_m}\partial_j\left(\frac{1}{\sigma}\left[\partial_i \frac{B_j}{\mu} - \partial_j \frac{B_i}{\mu}\right]\right) \tag{11}$$

Where B and B^0 are the induced and applied magnetic fields and s is a pseudo time step used to solve the steady state BVP. For uniform properties, this may simplify, by applying $\partial_j B_j = 0$ to the diffusion term after exchanging the order of derivatives, to the Poisson equation

$$\partial_j \partial_j B_i = \partial_j(u_j B_i^0 - u_i B_j^0) \tag{12}$$

2.4 Divergence of the Magnetic Field

The divergence B constraint, 7, is not naturally satisfied when solving the full induction equation and requires special care to be enforced. MOONS offers several options to handle this constraint and they are discussed in the next subsections.

2.4.1 Constrained Transport (CT) method

The CT method enforces 7 within machine accuracy at every time step so long the applied magnetic field also satisfies 7. This is accomplished by solving 8 where the electric field is carefully defined such that there is a perfect numerical cancellation of the divergence of B [2].

2.4.2 Brackbill - Projection method

One cleaning procedure that has been implemented was performed by [3]. This method is a projection method where the solution of the induction equation is a sub-step, and then subtracting the gradient of potential field, ϕ results in the 'cleaned' magnetic field:

$$\boldsymbol{B}^{n+1} = \boldsymbol{B}^* - \nabla\phi \tag{13}$$

Where ϕ satisfies Poisson equation:

$$\nabla^2\phi = \nabla \bullet \boldsymbol{B}^* \tag{14}$$

And $\boldsymbol{B}^*$ is the magnetic field solved from 6 or 9.

2.4.3 No treatment

In some cases, the numerical divergence of the Magnetic field is small and has no significant effect on the fluid flow. In these cases, it may be more advantageous to avoid any sort of cleaning or staggered variables to enforce 7 at all.

2.5 Material properties

To treat the interface between the liquid and solid domain, the approach in [4] was adopted. At the interface, a harmonic weighting function was used to described the transition of the changing material properties:

$$\sigma^*_{interface} = \frac{\sigma}{\sigma_l} = \frac{2\sigma_w}{\sigma_w + \sigma_l}$$

2.6 Boundary conditions

MOONS implements a class called BCs to package BCs for any particular solver. Then, the routine applyAllBCs is used to assign the appropriate BCs. BCs are implemented for 1st and 2nd order accuracy for collocated and staggered grids. If a variable is not coincident with the wall where you wish to enforce a particular BC, then it includes the interpolation such that the BC is satisfied at the wall.

2.6.1 Velocity Field

MOONS has a simple interface for enforcing the following BCs

No slip / no flow through (Dirichlet) and Fully developed (Neumann)

$$\boldsymbol{u} = \boldsymbol{u}_b \qquad \boldsymbol{u} = \boldsymbol{0} \qquad \partial_n \boldsymbol{u}_n = 0$$

2.6.2 Magnetic Field

By default, MOONS implements Fluent-Type BCs for the magnetic field:

$$\left(\frac{\partial \boldsymbol{B}_{normal}}{\partial n}\right)_{boundary} = 0 \qquad \boldsymbol{B}_{tangential} = 0$$

If a cleaning procedure is used to enforce $\nabla \bullet B = 0$ then the BCs for the scalar potential are

$$\left(\frac{\partial \phi}{\partial n}\right)_{boundary} = 0$$

2.6.3 applying BCs in MOONS

In order to assign BCs for the Ufield or Bfield, navigate to the user/initialize/momentum or /user/initialize/induction respectively. This is where the BCs are assigned for the velocity and magnetic field. The module computations/applyBCs.f90 specifies the stencils for all boundary conditions. They include stencils for collocated and staggered grids, meaning that if a variable is not coincident with the wall where you wish to enforce a particular BC, then it includes the interpolation such that the BC is satisfied at the wall. Looking at /static/BCs.f90 and applyBCs.f90 should help understand how to create the derived type, 'BCs', which are used in all of the solvers.

Since the velocity field is solved on a staggered grid, the BCs for wall-normal and wall-tangent components are different.

3 Solvers

3.1 Momentum

3.2 Induction

3.3 Poisson

3.3.1 SOR / Gauss Seidel

The relaxation parameter was determined by

$$\omega_{opt} = \frac{2}{1+\sqrt{1-\rho^2}} \tag{15}$$

Where ρ is the spectral radius of the iteration matrix of the Jacobi method [5]. For a 1D and 2D grids, the optimal parameter may be calculated as [6]

$$\omega_{opt,1D} = \frac{2}{1+\sqrt{1-\left(\frac{\cos\left(\frac{\pi}{N_x+1}\right)}{1}\right)^2}} \tag{16}$$

$$\omega_{opt,2D} = \frac{2}{1+\sqrt{1-\left(\frac{\cos\left(\frac{\pi}{N_x+1}\right)+\cos\left(\frac{\pi}{N_y+1}\right)}{2}\right)^2}} \tag{17}$$

respectively. The following has not been found in literature, but MOONS implements the following optimal parameter for the 3D simulations:

$$\omega_{3D} = \frac{2}{1+\sqrt{1-\left(\frac{\cos\left(\frac{\pi}{N_x+1}\right)+\cos\left(\frac{\pi}{N_y+1}\right)+\cos\left(\frac{\pi}{N_z+1}\right)}{3}\right)^2}} \tag{18}$$

There is a logical parameter in SOR.f90 to use Gauss Seidel, which sets $\omega = 1$.

3.3.2 Multigrid

MOONS has an unfinished geometric multigrid. There are many restrictions and prolongations required for the MG method, and implementing this routine has not been my top priority.

3.4 Root Solver

MOONS implements a root solver in order to estimate matching the interior cell size with the first exterior wall cell size. This is done by matching the interior stretching factor, β_i, with the stretching factor in the wall β_w. Currently, the root solver currently defines a vector of the roots, takes the absolute value of the vector, then locates the minimum. I believe it may be better to implement Newton's method in the future. The only issue I see with using Newtons method is that there are asymptotic bounds in the root that may cause problems with using Newton's method. The current method avoids this by ensuring that not too large and not too small values are chosen. Also, the root solver need not be exact, since it is only used to determine the cell nodes (corners) after the first exterior cell.

4 Modules

5 Mesh

The mesh created was using a Robert's stretching function defined in [7].

MOONS automatically ensures that the size of the first exterior (wall) cell is the same exact size as the first interior fluid cell. This is important because MOONS extrapolates any data using a linear extrapolation, which assumes that the cells are the same size. This means special treatment would be required for extrapolated data to be consistent with, for example, the forcing term in the Poisson equation for pressure or for the magnetic field if the low Re_mapproximation is being used.

MOONS also uses a few checks to make sure that prescribed inputs are consistent. For example, if the wall thickness is zero, then the number of cells in the wall better be zero. If inconsistent problems like these are found, MOONS stops the simulation before it even starts and reports the error.

6 Errors

The MOONS error module calculates several errors including L_1,L_2 and L_∞ norms. The following sections show the details of these calculations. They are performed in /computations/myError.f90

6.1 Absolute

$$L_n(f_{exact}, f) = \sqrt[n]{\sum_{i,j,k} |(f_{exact})_{i,j,k} - f_{i,j,k}|^n} \qquad L_\infty(f_{exact}, f) = \max((f_{exact}) - f) \tag{19}$$

6.2 Relative

$$R_n(f_{exact}, f) = \frac{L_n(f_{exact}, f)}{L_n(f_{exact}, 0)} \qquad R_\infty(f_{exact}, f) = \frac{L_\infty(f_{exact}, f)}{L_\infty(f_{exact}, 0)} \tag{20}$$

Note that when f_{exact} is less than a tolerance of 10^{-6} everywhere, 1 is added to the denominator to avoid division by zero, which results in

$$R_n(f_{exact}, f) = L_n(f_{exact}, f) \qquad R_\infty(f_{exact}, f) = L_\infty(f_{exact}, f) \tag{21}$$

7 Parallelization

MOONS is openMP parallel ready. In gfortran, the -fopenmp flag must be used. In addition to the compiler flag, useOpenMP must be set to .true. in user/simParams.f90 to solve the Poisson Equation in parallel. The reason for this is that the Poisson loop was split into two loops for parallelization, and the single loop may run faster in serial.

MOONS has parallelization for the following routines:

```
interpO2()          (/computations/vectorOps.f90)
myDel1()            (/computations/myDel.f90)
mySOR()             (/solvers/steadyState/SOR.f90)
myError()           (/computations/myError.f90)
```

8 Directionality

MOONS computes derivatives, interpolates, and is indexed in the same way for all three directions. Implementing directionless routines was a highly sought task when implementing MOONS and has paid off greatly in the simplicity of implementation. Furthermore, the explanation of these routines will be described in 1D for succinctness.

9 Derivatives, Interpolations and BC formulas

9.1 Derivatives

9.2 Interpolations

9.3 BC formulas

10 Indexing

MOONS indexes starting from 1. This was a carefully chosen decision because with the indexing starting from 1, the Fortran intrinsic `shape()` can be used to obtain the size of the incoming data in a subroutine and internally decide where the data lives based on the dimensions.

The index convention has been drawn in a diagram below.

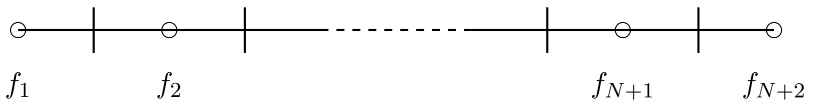

Figure 1: Index convention for cell center / edge data

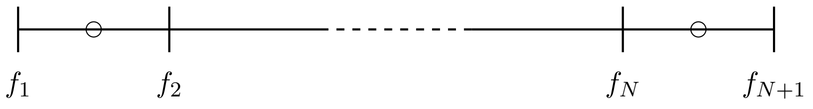

Figure 2: Index convention for node / face centered data

Where N is the number of *cells* in the domain (interior or total depending on the field in question).

11 Running MOONS

In order to run MOONS, you'll need to call MOONS from a project directory. The only input to calling MOONS is the output directory.

```
include 'includes.f90'
program runMoons
use MOONS_mod
implicit none
call MOONS('out\LidDrivenCavity\')
end program
```

Before running MOONS, your project directory may look something like this:

```
Project/
├── bin
└── main.f90
```

After running MOONS, your directory tree should looks like this

```
Project/
├── bin
├── main.f90
└── out
    └── LidDrivenCavity
        ├── BField
        ├── JField
        ├── material
        ├── parameters
        ├── UField
        └── CPU TIME INFO.dat
```

11.1 Post Processing

Once MOONS has finished running, navigate to /MATLAB/MOONS/main.m and run with the appropriate settings. Once doing this, each field folder in your directory tree should have a new folder named 'figs' which will contain the output plots of MOONS.

12 Running a simple simulation

Let's set up the benchmark problem 1. A Lid Driven Cavity flow for Re=400, Ha=0. The following table shows which parameters are in which files and the values you must set them to.

griddata.f90

12.1 Setting parameters

12.2 Compile

12.3 Run

12.4 Post Processing

13 Benchmarks

13.1 Lid Driven Cavity Flow

13.2 Square Duct Flow

14 Navigating MOONS

A large effort in developing MOONS was to ensure that the code, organization, and logic is highly transparent. Look at the directory tree, it shouldn't take more than a few seconds to find which folder the momentum solver is located. The idea of transparency also holds in the individual files themselves. If a subroutine or function seems opaque, there should be documentation explaining what it's doing, otherwise, it should be relatively self-explanatory by the name of the subroutine.

14.1 Directory Tree

Below is the MOONS directory tree, showing the location of the modules and solvers for the user to set parameters for a simulation.

```
MOONS/
├── documentation
│   ├── MOONS.pdf (this file)
│   └── Derivations
│       └── Non-uniform Grid Stencil.pdf
├── solvers
│   ├── steadyState
│   │   ├── Poisson.f90
│   │   ├── SOR.f90
│   │   └── MG.f90 (not finished)
│   └── transient
│       ├── MHDSolver.f90
│       ├── ADI.f90 (not finished)
│       ├── momentum
│       │   └── momentumSolver.f90
│       └── induction
│           └── inductionSolver.f90
└── static ...  Auxiliary files, no settings exist here
```

```
|
|__matlab
|   |__plotResults...  matlab plotting files
|   |__tools...  matlab tools for plotting
|   |__main.m
|   |__benchmarking.m
|__user
|   |__initialize
|   |   |__momentum
|   |   |   |__initializeUBCs.f90
|   |   |   |__initializeUField.f90
|   |   |__induction
|   |       |__initializeBBCs.f90
|   |       |__initializeBField.f90
|   |       |__initializeSigmaMu.f90
|   |__constants.f90 - sets precision etc
|   |__griddata.f90 - sets geometry, N cells, t_w, stretching etc
|   |__MOONS.f90 - sets Re, Ha, Re_m etc
|   |__rundata.f90 - sets Δt etc
|   |__simParams.f90 - sets equations to solve, how to solve them (coupled/uncoupled) etc
|__includes.f90 - specifies files for MOONS to include
```

References

[1] Michael Griebel, Thomas Dornseifer, and Tilman Neunhoeffer. *Numerical simulation in fluid dynamics: a practical introduction*, volume 3. Siam, 1997.

[2] Gábor Tóth. The divergence Constraint in Shock-Capturing MHD Codes. *Journal of Computational Physics*, 161(2):605–652, July 2000.

[3] J.U. Brackbill. The Effect of Nonzero V . B on the Numerical Solution of MHD Equations. *JCP*, 430:426–430, 1980.

[4] Neil Franco De Carvalho. METHODS FOR CALCULATING THE THERMAL CONDUCTIVITY AT THE CONTROL-VOLUME SURFACES. (Cobem), 2013.

[5] Shiming Yang and Matthias K. Gobbert. The optimal relaxation parameter for the SOR method applied to the Poisson equation in any space dimensions. *Applied Mathematics Letters*, 22(3):325–331, March 2009.

[6] J M Mcdonough. LECTURES on COMPUTATIONAL NUMERICAL of PARTIAL DIFFERENTIAL EQUATIONS. 2008.

[7] Richard H Pletcher, John C Tannehill, and Dale Anderson. *Computational fluid mechanics and heat transfer*. CRC Press, 2012.