OOMMF Tutorial
Part IV: Advanced Simulations and Post-processing

Michael J. Donahue

Applied and Computational Mathematics Division
National Institute of Standards and Technology
Gaithersburg, Maryland

9-June-2020

Special Thanks to

Online Spintronics Seminar

Professor Xin Fan (Univ. Denver)

Professor Kirill Belashchenko (Univ. Nebraska-Lincoln)

nanoHUB

Tanya Faltens (Purdue University)

IEEE Magnetics Society

DISCLAIMER: The mention of specific products, trademarks, or brand names is for purposes of identification only. Such mention is not to be interpreted in any way as an endorsement or certification of such products or brands by the National Institute of Standards and Technology. All trademarks mentioned herein belong to their respective owners.

OOMMF Tutorial Part IV: Advanced Simulations and Post-processing

Session schedule

- Thur, 21-May-2020: Intro to Micromagnetics
- Tues, 26-May-2020: OOMMF Basics
- Tues, 2-Jun-2020: Pitfalls, advanced MIF, writing an extension
- Tues, 9-Jun-2020: Advanced MIF, post-processing, images, movies, dispersion curves

All sessions start at 12:00 noon EDT.

Sample files for Session IV

Download link to sample files for this session on the page:

https://math.nist.gov/oommf/oommf_tutorial/tutorial.html

OOMMF Tutorial
Part IV: Advanced Simulations and Post-processing

Session 3 Homework

Using the equilibrium state from the Session II as the initial state, run a STT simulation using the **Anv_SpinTEvolve** extension with these parameters:

- u=100 m/s
- alpha=0.1
- beta=0.04

See the **Anv_SpinTEvolve** web page and sample problem to get started.

The skyrmion should move to the right, and slightly upward. Determine the speed of the skyrmion and the drift angle. Try varying alpha. For alpha = beta there should be no up or down drift. For alpha<beta the drift should be downward. For that condition flip the initial state using **Oxs_AffineOrientVectorField** and **Oxs_AffineTransformVectorField**.

Homework: Skyrmion motion

See sample file otprob1_2.mif:

```
Specify Anv_SpinTEvolve:evolver [subst {
 alpha $alpha
 u $u
 beta $beta
 method rkf54s
}]

# Select stopping_time and stage_count suitable for
# animation. 80 x 50 ps = 4 ns simulation time.
Specify Oxs_TimeDriver [subst {
 evolver :evolver
 stopping_time 0.05e-9
 stage_count 80
 mesh :mesh
 Ms $Ms
 m0 $m0 comment {Flipped or not depends on alpha<beta}
}]
```

Homework: Skyrmion motion, setup

- Oxs_AffineOrientVectorField: This class transforms the vector field $\mathbf{F}(r)$ into $\mathbf{F}(Mr+\mathbf{r}_{off})$, where M is a 3×3 matrix and $\mathbf{r}_{off}$ is an offset. This shifts, rotates, and/or flips the vector field *without* reorienting the field vectors themselves.
- Oxs_AffineTransformVectorField This class transforms the vector field $\mathbf{F}(r)$ into $M\mathbf{F}(r)+\mathbf{r}_{off}$, where M is a 3×3 matrix and $\mathbf{r}_{off}$ is an offset. This modifies each vector of the field *in place*.
- By using these two transforms together we can perform a mirror reflection of the initial magnetization state.
- One should also adjust (flip) pinning field.
- The deflection angle is about 7° for alpha = 0.1, 0° when alpha = beta = 0.04, and -3.5° for alpha = 0.01. The velocity is close to 100 m/s for all cases.

Homework: Skyrmion motion, setup

See sample file otprob1_2.mif:

```
Specify Oxs_FileVectorField:start [subst {
 file $start
 atlas :atlas
 norm 1.0
}]
set m0 :start ;# Unflipped initial state

if {$alpha < $beta} {
 Specify Oxs_AffineOrientVectorField:flipped_start [subst {
  field :start
  M {1 -1 1}
  offset {0 $ymax 0}
 }]
 Specify Oxs_AffineTransformVectorField:mirror_start {
  field :flipped_start
  M {1 -1 1}
 }
 set m0 :mirror_start ;# Flipped initial state
}
```

OOMMF Tutorial
Part IV: Advanced Simulations and Post-processing

MIF Destination and Schedule commands (Clarification)

- **Destination** and **Schedule** commands are executed when problem is loaded.
- **Schedule** commands overlay existing schedules (if any).
- **Schedule** commands are not “sticky.” Schedules can be reset via interactive interface.

Time varying applied field (Hping.mif)

```
proc Sinc { t } {
  if {abs($t)<1e-6} {
   set v [expr {1-$t*$t/6.}]
   set dv [expr {$t*$t*$t/-3.}]
  } else {
   set v [expr {sin($t)/$t}]
   set dv [expr {($t*cos($t)-sin($t))/($t*$t)}]
  }
  return [list $v $dv]
}
proc SincPulse { total_time } {
  global amp scale offset
  set st [expr {$scale*($total_time - $offset)}]
  set vals [Sinc $st]
  set Hy [expr {$amp*[lindex $vals 0]}]
  set dHy [expr {$amp*$scale*[lindex $vals 1]}]
  return [list 0 $Hy 0 0 $dHy 0]
}
```

Time varying applied field (Hping.mif, cont.)

```
Specify Oxs_ScriptUZeeman [subst {
  multiplier [expr {0.001/$mu0}]
  script SincPulse
  script_args total_time
}]

Specify Oxs_TimeDriver [subst {
  evolver :evolver
  mesh :mesh
  stopping_time $stage_time
  stage_count $number_of_stages
  Ms {Oxs_AtlasScalarField {
  atlas :atlas
  default_value 0.0
   values { ellipsoid 8e5 }
  }}
  m0 {1 0 0}
}]
```

Oxs_TransformZeeman

- The **Oxs_TransformZeeman** class produces an applied field that varies with both time and space. Given a vector field $\mathbf{F}(\boldsymbol{r})$, **Oxs_TransformZeeman** computes

$$M(t)\mathbf{F}(\boldsymbol{r})$$

  where $M(t)$ is a time varying 3×3 matrix provided by a Tcl script.

- As with **Oxs_ScriptUZeeman**, both $M(t)$ and its derivative wrt time must be provided by the Tcl script.

- Keep an eye on the time step when using either this or the **Oxs_ScriptUZeeman** classes; if dM/dt is wrong, then step size will collapse to $\approx 10^{-20}$ s.

Current pulse (spinping.mif)

```
proc Sinc { t } {
 if {abs($t)<1e-6} {
  set v [expr {1-$t*$t/6.}]
 } else {
  set v [expr {sin($t)/$t}]
 }
 return $v
}

proc SincPulse { total_time } {
 global pulse_scale pulse_offset
 set t [expr {$total_time - $pulse_offset}]
 set st [expr {$t*$pulse_scale}]
 return [Sinc $st]
}
```

Current pulse (spinping.mif, cont.)

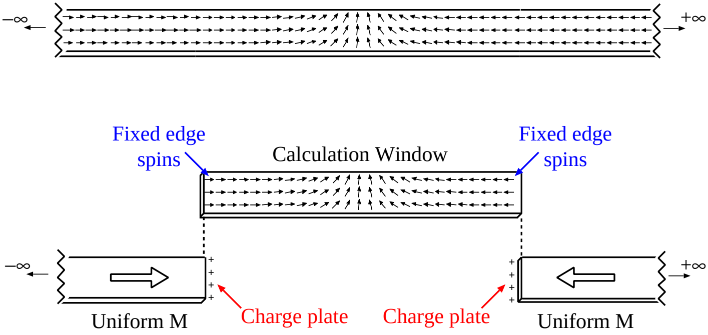
```
Specify Anv_SpinTEvolve [subst {
 do_precess 1
 gamma_LL 2.21e5
 method rkf54s
 alpha 0.005
 fixed_spins { atlas fixed }
 u $u_max
 u_profile SincPulse
 u_profile_args total_time
 beta 0.04
}]
```

⇒ Current density at point (x,y,z) is proportional to

$$u_{\text{profile}}(t)\cdot u(x,y,z).$$

Specify option $u = u(x,y,z)$ can be any scalar field.

Infinite strips



See spinping.mif for an example.

R.D. McMichael & M.J. Donahue, *IEEE Trans. Magn.*, **33**, 4167 (1997).

User specified data table outputs (1/2)

- The user_output **Specify** option is described in the MIF 2.1 section of the OUG.
- It adds an additional column to DataTable output.
- Output is

$$\sum_i W_{\text{select}}[i] \cdot V_{\text{source}}[i] \Big/ \sum_i |W_{\text{select}}[i]|$$

where W_{select} selects the volume and weightings, and V_{source} is an output, e.g., the magnetization field output.

User specified data table outputs (2/2)

Example use of user_output (cf. user_outputs.mif):

```
Specify Oxs_MinDriver [subst {
  ...
  user_output {
    name mx_top
    source_field Magnetization
    select_field TopX
    user_scaling [expr {1.0/1400e3}]
    units {Non Dim}
  }
  user_output {
    name mx_bottom
    source_field Magnetization
    select_field BottomX
    user_scaling [expr {1.0/1400e3}]
    units {Non Dim}
}]
```

OOMMF Tutorial
Part IV: Advanced Simulations and Post-processing

Command line aids

- Windows
  - Batch file **oommf.bat**:

    ```
    @tclsh C:\Users\fred\oommf\oommf.tcl %*
    ```

    Put **oommf.bat** in PATH.
  - Additional tools:

    grep, **sed**, **less**, **head**, **tail**

    from, e.g., GnuWin32.
- Unix
  - Bash shell wrapper, **oommf**:

    ```
    tclsh /home/barney/oommf/oommf.tcl "$*"
    ```

    Put **oommf** in $PATH and mark it **chmod** u+x.

odtcols (otprob1_2-alpha0.100.odt)

- **odtcols** can summarize ODT files, extract column subsets, and convert to other formats.
- **odtcols** operates as a filter, taking reading stdin and writing to stdout.
- To see a table summary:

```
oommf odtcols -s < otprob1_2-alpha0.100.odt
```

- To extract columns:

```
oommf odtcols 14 19 2 < otprob1_2-alpha0.100.odt
```

odtcols (2/2)

- To format output:

```
oommf odtcols 14 19 2 -w 20 -f %#10.6g \
  < otprob1_2-alpha0.100.odt
```

- To convert to csv format:

```
oommf odtcols 14 19 2 -w 20 -f %#10.6g -t csv \
  < otprob1_2-alpha0.100.odt
```

- Short form column headers:

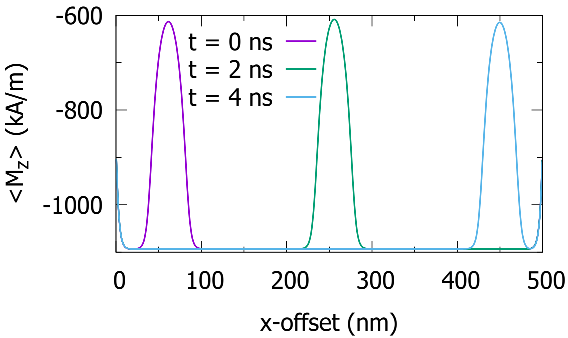
```
oommf odtcols 14 19 2 -w 20 -f %#10.6g -t csv \
  < otprob1_2-alpha0.100.odt                   \
  | sed "s/\(Oxs\|Anv\)[^ ]*://g"
```

See also **odtcalc** to create additional columns (e.g., to project hysteresis data to off-coordinate axes).

avf2odt

avf2odt extracts data from OVF files and converts to ODT format. In particular, **avf2odt** can extract averaged data from magnetization fields to produce line graphs:

```
oommf avf2odt -average plane -axis x \
  otprob1_2-stage39.omf  otprob1_2-stage79.omf  ...
```



Creating images from vector fields

1. Load OVF into mmDisp, configure, save .config.
2. Tweak .config:
   - Adjust arrow,antialias , misc,height, misc,width
   - Set misc,zoom = 0 to fill
   - Set misc,crop = 0 to match width $\times$ height
3. Run **avf2ppm** or **avf2ps** to make image file(s):

```
tclsh oommf.tcl avf2ppm -f -config foo.config  ^
 foo*.omf                                        ^
 -filter "tclsh oommf.tcl any2ppm -format png"   ^
 -opatexp "-[0-9]*\.omf" -opatsub .png
```

Converts, for example,

foo–Oxs_TimeDriver–Magnetization–13–0092005.omf

$\Rightarrow$ foo–Oxs_TimeDriver–Magnetization–13.png

Creating animation from images

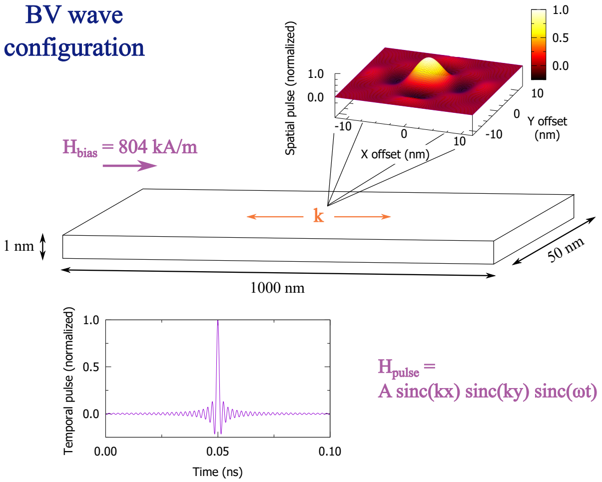
Combine a collection of images

```
foo-Oxs_TimeDriver-Magnetization-00.png
foo-Oxs_TimeDriver-Magnetization-01.png
foo-Oxs_TimeDriver-Magnetization-02.png
foo-Oxs_TimeDriver-Magnetization-03.png
 ...
```

using (for example) **ffmpeg**:

```
ffmpeg -r 10 -start_number 0                       ^
  -i foo-Oxs_TimeDriver-Magnetization-%02d.png ^
  -c:v libx264 -qmin 0 -qmax 32 -r 30              ^
  -an foo-Magnetization.mp4
```

Dispersion study



G. Venkat et al., "Proposal for a standard micromagnetic problem: Spin wave dispersion in a magnonic waveguide," *IEEE Trans. Magn.*, **49**, 524-529 (2013).

Dispersion curves (1/3)

- Run dispersion.mif simulation, saving magnetization field every ps.
- Use avf2odt to extract M_y along center line for **all** snapshots:

```
oommf avf2odt -average point                          \
 -defaultpos 0 -defaultvals 0 -headers none           \
 -valfunc my "" "$vy"                                 \
 -region - 24e-9 - - 26e-9 -                          \
 dispersion-backward-*-Magnetization-*-*.omf          \
 -onefile foo.dat -truncate 1
```

- File foo.dat is a single column of 500×5000 M_y values, with no header or trailer.

Dispersion curves (2/3)

- Load data into python:

```
import numpy as np
A=np.loadtxt("foo.dat")
A=np.reshape(A,(-1,500))
```

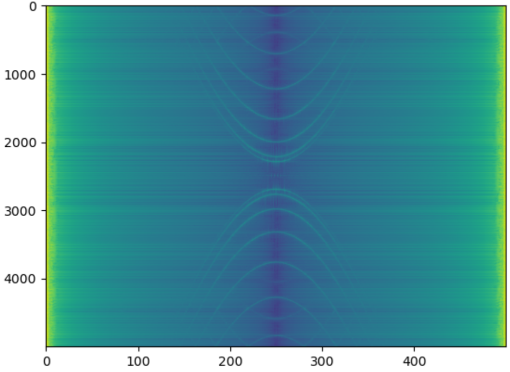
The last step creates a 2D array; we know the center line for each snapshot is 500 samples.

- Subtract initial (remanent) state from each snapshot, take 2D FFT, and display:

```
B=np.subtract(A,A[0])
C=np.fft.fft2(B)
import matplotlib.pyplot as plt
plt.imshow(np.log(np.abs(np.fft.fftshift(C))**2),
  aspect='auto')
plt.show()
```

Dispersion curves (3/3)

Example output:



Making a 3D quiver plot in MayaVi (1/3)

- Extract raw data from OVF file as 6-tuples, subsampled to a reasonable level

```
oommf avf2ovf stdprob3-vortex.omf          \
  -subsample 4 -format text -grid irreg \
  | grep -v "^#" > stdprob3-vortex.dat
```

- Output stdprob3-vortex.dat is six columns of data: x, y, z, Mx, My, Mz.

Making a 3D quiver plot in MayaVi (2/3)

- Load data into python:

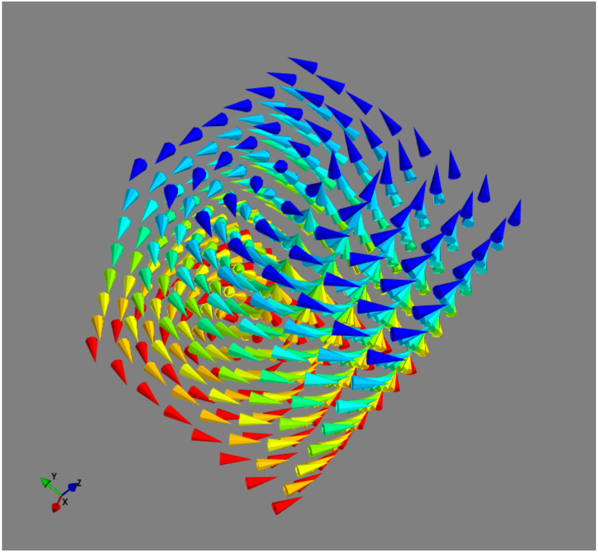
```
import numpy as np
A=np.loadtxt('stdprob3-vortex.dat')
x,y,z,mx,my,mz=np.hsplit(A,6)
x*=1e9 ; y*=1e9 ; z*=1e9
```

- Display data using MayaVi

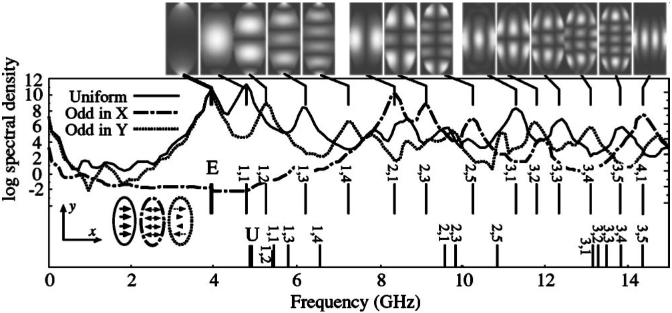
```
import mayavi.mlab as mlab
obj=mlab.quiver3d(x,y,z,mx,my,mz,mode='cone',
                  scalars=x)
obj.glyph.color_mode = 'color_by_scalar'
```

Making a 3D quiver plot in MayaVi (3/3)

This produces an image like this:



Normal mode diagrams



R.D. McMichael and M.D. Stiles, “Magnetic normal modes of nanoelements,” *J. Appl. Phys.* **97**, 10J901 (2005).

$$S_x(\boldsymbol{r}_k, f) = \left| \sum_j M_x(\boldsymbol{r}_k, t_j) e^{2\pi i f t_j} \right|^2$$

See also: M.J. Donahue and R.D. McMichael, “Micromagnetics on curved geometries using rectangular cells: Error correction and analysis,” *IEEE Trans. Magn.* **43**, 2878 (2007).