Cython Landau-Lifshitz Kernel

December 29, 2014

```
In [3]: %matplotlib inline
        from matplotlib import pyplot as plt
        import numpy as np
```

1 Cython Landau-Lifshitz Kernel

1.1 Python Kernel

Before developing the Cython Kernel, the Landau-Lifshitz equation will be solved using Python. This provides a benchmark to iterate from.

```
In [13]: from scipy.integrate import ode

         # Simulation parameters
         alpha = 0.01
         gamma = 2.1*9.2740e-21/(6.6261e-27 /(2.0*np.pi))
         Ms = 140 # emu/cc
         h_ext = np.array([0, 1000, 0], dtype=np.float32) / Ms
         dt = 5e-13*gamma*Ms/(1 + alpha**2.)

         # Initial conditions
         t0, m0 = 0., np.array([-0.999, 0.001, 0.001], dtype=np.float32)

         def evolve(t, m, h_ext):
             h_eff = h_ext
             hxm = np.cross(h_eff, m)
             mxhxm = np.cross(m, hxm)
             return [hxm + alpha*mxhxm]
```

```
In [5]: %timeit evolve(t0, m0, h_ext)
```

```
10000 loops, best of 3: 18.6 µs per loop
```

This gives a rate of 50,000 evolutions per second, considering only the execution of the Landau-Liftshitz function.

```
In [14]: def loop(m, h_ext, n, w):

             r = ode(evolve).set_integrator('dopri5')
             r.set_initial_value(m0, t0)
             r.set_f_params(h_ext)

             moments = np.zeros((w,3), dtype=np.float32)
             times = np.zeros((w,1), dtype=np.float32)
```

```
        i = 0

        while r.successful() and i < w*n:
            r.integrate(r.t + dt)
            if i % n == 0:
                #r.y = r.y/np.linalg.norm(r.y)
                moments[i/n], times[i/n] = r.y, r.t
            i += 1

        return times/(gamma*Ms/(1 + alpha**2.)), moments
```

```
In [15]: %timeit loop(m0, h_ext, 250, 1)
```

```
10 loops, best of 3: 67.3 ms per loop
```

This implies a rate of 3,570 evolutions per second, or 14 loops per second.

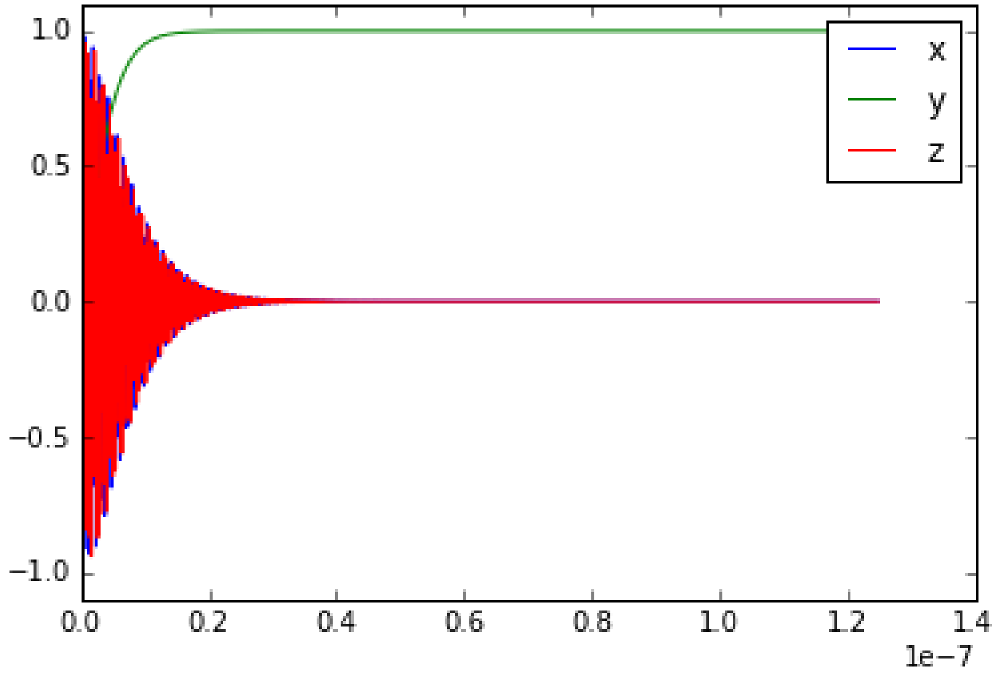
```
In [17]: %timeit loop(m0, h_ext, 250, 10)
```

```
1 loops, best of 3: 705 ms per loop
```

Ten loops requires 725 ms, which agrees with the rate expected from 1 loop.

```
In [18]: times, moments = loop(m0, h_ext, 250, 1000)
```

```
In [19]: plt.plot(times, moments[:,0], label='x')
         plt.plot(times, moments[:,1], label='y')
         plt.plot(times, moments[:,2], label='z')
         plt.ylim(-1.1, 1.1)
         plt.legend()
         plt.show()
```



The above behavior of the macrospin shows that the moment aligns to the external field as expected.

1.2 Cython Kernel

```
In [21]: %load_ext cythonmagic
```

```
In [44]: %%cython
         import numpy as np
         cimport numpy as np

         cdef:
             float CGS_GAMMA = 2.1*9.2740e-21/(6.6261e-27/(2.0*np.pi))
             float MS = 140 # emu/cc
             float ALPHA = 0.01
             float DT = 5e-13*CGS_GAMMA*MS/(1 + ALPHA**2.)

         def evolve(m, h_ext):
             h_eff = h_ext
             hxm = np.cross(h_eff, m)
             mxhxm = np.cross(m, hxm)
             m += DT*(hxm + ALPHA*mxhxm)

         def loop(m, h_ext, n, w):

             moments = np.zeros((w,3), dtype=np.float32)
             times = np.zeros((w,1), dtype=np.float32)

             for i in range(w):
                 for j in range(n):
                     evolve(m, h_ext)
                 m = m/np.linalg.norm(m)
                 moments[i] = m.copy()
                 times[i] = DT*n + times[i-1]

             return times/(CGS_GAMMA*MS/(1 + ALPHA**2.)), moments
```

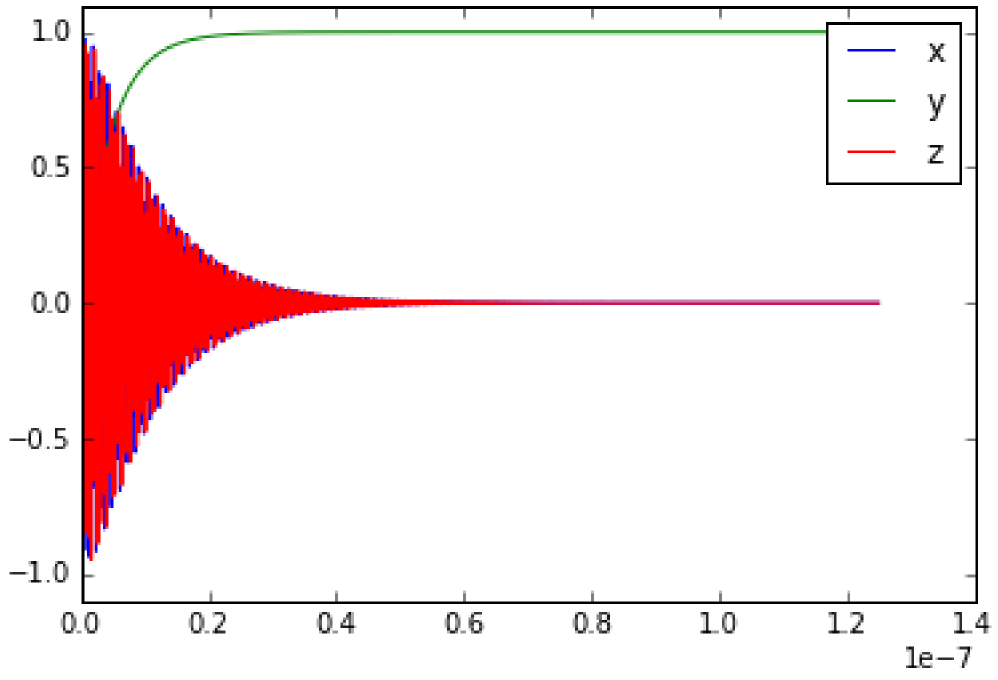
```
In [45]: Ms = 140
         m = np.array([-0.999, 0.001, 0.001], dtype=np.float32)
         h_ext = np.array([0, 1000, 0], dtype=np.float32) / Ms
```

```
In [50]: %timeit loop(m, h_ext, 250, 1)
```

```
100 loops, best of 3: 5.57 ms per loop
```

```
In [46]: times, moments = loop(m, h_ext, 250, 1000)
```

```
In [49]: plt.plot(times, moments[:,0], label='x')
         plt.plot(times, moments[:,1], label='y')
         plt.plot(times, moments[:,2], label='z')
         plt.ylim(-1.1, 1.1)
         plt.legend()
         plt.show()
```



Note that the oscillations in x and y now take longer to subside.

1.3 Improved Cython take 1

```
In [127]: %%cython
          import numpy as np
          cimport numpy as np

          cdef:
              float CGS_GAMMA = 2.1*9.2740e-21/(6.6261e-27/(2.0*np.pi))
              float MS = 140 # emu/cc
              float ALPHA = 0.01
              float DT = 5e-13*CGS_GAMMA*MS/(1 + ALPHA**2.)

          ctypedef struct float3:
              float x, y, z

          cpdef float3 make_float3(float[::1] m):
              cdef float3 r
              r.x, r.y, r.z = m[0], m[1], m[2]
              return r

          cdef float3 add(float3 a, float3 b):
              cdef float3 c
              c.x = a.x + b.x
              c.y = a.y + b.y
              c.z = a.z + b.z
              return c
```

```
cdef float3 cross(float3 a, float3 b):
    cdef float3 c
    c.x = a.y*b.z - a.z*b.y
    c.y = a.z*b.x - a.x*b.z
    c.z = a.x*b.y - a.y*b.x
    return c

cdef float3 mult(float3 a, float3 b):
    cdef float3 c
    c.x = a.x * b.x
    c.y = a.y * b.y
    c.z = a.z * b.z
    return c

cdef float3 times(float a, float3 b):
    cdef float3 c
    c.x = a*b.x
    c.y = a*b.y
    c.z = a*b.z
    return c

cdef float3 div(float3 a, float3 b):
    cdef float3 c
    c.x = a.x * b.x
    c.y = a.y * b.y
    c.z = a.z * b.z
    return c

cdef evolve(float3 m, float3 h_ext):
    h_eff = h_ext
    hxm = cross(h_eff, m)
    mxhxm = cross(m, hxm)
    m = add(m, times(DT, add(hxm, times(ALPHA, mxhxm))))

def loop(m, h_ext, int n, int w):

    moments = np.zeros((w,3), dtype=np.float32)
    times = np.zeros((w,1), dtype=np.float32)

    cdef:
        int i, j
        float3 mf3 = make_float3(m)
        float3 hf3 = make_float3(h_ext)

    for i in range(w):
        for j in range(n):
            evolve(mf3, hf3)
        m = m/np.linalg.norm(m)
        moments[i] = m.copy()
        times[i] = DT*n + times[i-1]

    return times/(CGS_GAMMA*MS/(1 + ALPHA**2.)), moments
```

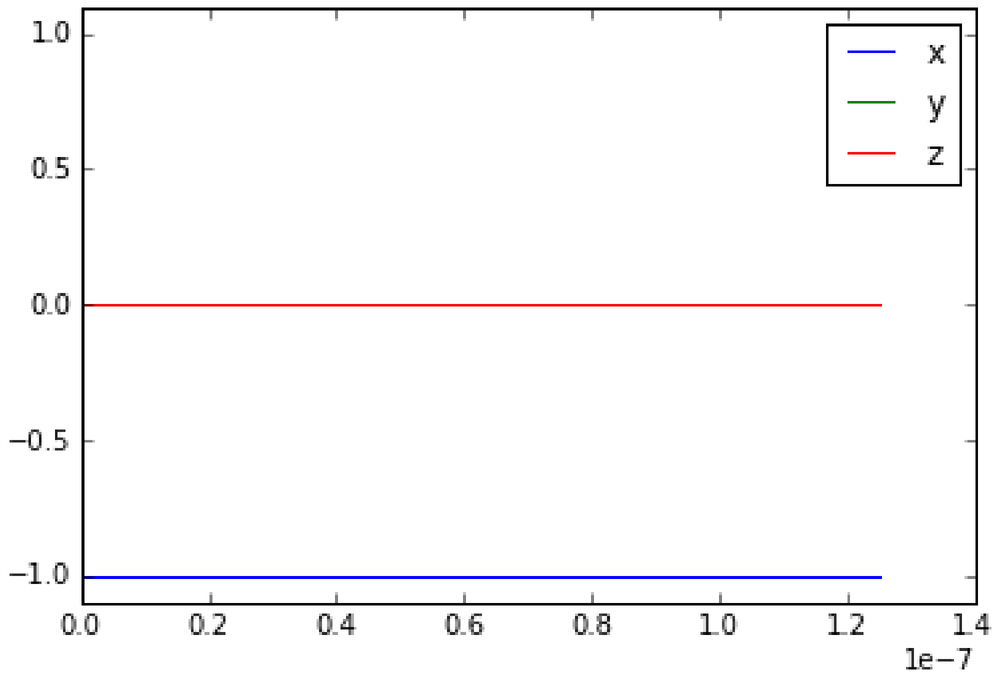
```
In [128]: Ms = 140
          m = np.array([-0.999, 0.001, 0.001], dtype=np.float32)
          h_ext = np.array([0, 1000, 0], dtype=np.float32) / Ms

In [129]: %timeit loop(m, h_ext, 250, 1)

10000 loops, best of 3: 26.9 µs per loop

In [130]: times, moments = loop(m, h_ext, 250, 1000)

In [131]: plt.plot(times, moments[:,0], label='x')
          plt.plot(times, moments[:,1], label='y')
          plt.plot(times, moments[:,2], label='z')
          plt.ylim(-1.1, 1.1)
          plt.legend()
          plt.show()
```


1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e−7
x
y
z