Week Nine
Python for Electromagnetism

Physical laws are expressed as mathematical relations which can then be solved under desired boundary conditions to obtain the results. Consider for example the relationship between electric and magnetic fields as represented by Maxwell's equations:

$$\nabla \times E = -\frac{\partial B}{\partial t}; \nabla \times B = \mu_0 J + \mu_0 \epsilon_0 \frac{\partial E}{\partial t}; \nabla \cdot B = 0$$

In order to find out the field distribution inside a resonator cavity, we need to solve these equations by applying suitable boundary conditions dictated by the geometry of the cavity and electromagnetic properties of the material used for making it. For simple geometries, analytic solutions are possible but in most cases numerical methods are essential and that is one reason why computers are important for physics. In this weeks exercise we will be looking at ways in which to visulaise the fields encountered in electromagnetism and have a first look at some of the numerical tools available to solve these equations. Note: *Scipy* contains modules for solving first order ordinary differnetial equations scipy.integrate.ode (a Nth order equation can also be solved using SciPy by transforming it into a system of first order equations)scipy.integrate.odeint provides a module for solving a system of first order ODEs. Partial differential equation solvers are not included in the Enthought distribution and are beyond the scope of this introductory course but modules such as FiPy (Finite volume approach) and SfePy (Simple Finite Element approach) are capable of solving this class of equation.

1 Numerical Differentiation

The following code will calculate and display graphically the path of a charged particle (with initial position and velocity) under the influence of external magnetic and electric fields. Download the example code from moodle and observe the effects of changing the initial parameters. In this example code the position x, y, z at $t + dt$ is obtained using the simplest of algorithms, Euler's method. This is a method for solving ordinary differential equations using the formula:

$$y_{n+1} = y_n + hf(x_n, y_n)$$

which advances a solution from x_n to $x_{n+1} = x_n + h$. Note that the method increments a solution through an interval h while using derivative information from only the beginning of the interval. As a result, the step's error is $O(h^2)$. A better solution can be obtained by using a method that calculates the average derivative over the interval such as the Runge-Kutta described in Appendix A of the mathematical methods notes. A fourth order Runge-Kutta is a method that evaluates the derivative four times; once at the initial point, twice at two trial midpoints and once at trial a endpoint. The final value is evaluated from these derivatives using the equations:

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2})$$

$$k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2})$$

$$k_4 = hf(x_n + h, y_n + k_3)$$

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

1.1 Wk9Ex1:

Modify the field_simulation.py code to use a 4th order Runge-Kutta method, plot and compare with the Euler's method result. Try increasing the total time interval to see how the errors vary with time.

```
from numpy import *
from pylab import *
from mpl_toolkits.mplot3d import Axes3D
"""
The following example traces the movement of a charged particle under the
    influence of electric and
magnetic fields. You can edit the program to change the components of the
    fields to see their effect on
the trajectory.
"""
Ex = 0.0 # Components of applied Electric Field
Ey = 2.0
Ez = 0.0
Bx = 0.0 # Components of applied Magnetic field
By = 0.0
Bz = 5.0
m = 2.0 # Mass of the particle
q = 5.0 # Charge
x = 0.0 # initial position x
y = 0.0 # initial position y
z = 0.0 # initial position z
vx = 20.0 # initial velocity vx
vy = 0.0 # initial velocity vy
vz = 2.0 # initial velocity vz
a = []
b = []
c = []
t = 0.0
dt = 0.01
while t < 6: # trace path until time reaches value
    Fx = q * (Ex + (vy * Bz) - (vz * By) )
    vx = vx + Fx/m * dt # Acceleration = F/m; dv = a.dt
    Fy = q * (Ey - (vx * Bz) + (vz * Bx) )
    vy = vy + Fy/m * dt
    Fz = q * (Ez + (vx * By) - (vy * Bx) )
```

```
        vz = vz + Fz/m * dt
        x = x + vx * dt
        y = y + vy * dt
        z = z + vz * dt
        a.append(x)
        b.append(y)
        c.append(z)
        t = t + dt

fig=figure()
ax = Axes3D(fig)
ax.set_title("Path of charged particle under influence of electric and
    magnetic fields")
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.plot3D(a,b,c, color='blue', label='path')
ax.legend(loc='lower left')

show()
```

field_simulation.py

2 Vector Fields

The code vectorfield.py is a basic script for generating a simple vector field around a point charge using Mayavi. Download the code from moodle and run it. Exlpore the functionality of the Mayavi toolbar, the first button opens a dialog for the Mayavi pipeline that allows you to control many aspects of the view.

Figure 1: The Mayavi toolbar, the first button launches the Mayavi pipeline

2.1 Wk9Ex2:

Modify the code such that the vector field is only calculated and plotted in the x,y plane. Add axis labels and a title to the plot and save a .png output of the plot.

```
from numpy import mgrid, array, sqrt
from mayavi.mlab import quiver3d
# this creates a 3D grid which is ranges from -100 to 100 in all
# coordinates, with intersections at every 20th value
# (-100, -80, -60, ...). See 'help mgrid' for details.
x,y,z = mgrid[-100.:101.:20., -100.:101.:20., -100.:101.:20.]
# We will now store the coordinates of our charge as a
```

```
# vector. All vectors in Python should be stored as arrays.
# Note that we are placing this charge in between the grid points.
# This is to get the best possible symmetry in this specific field
# If you place the charge _on_ the grid, you get a division of zero
# when calculating the E-field.
qpos = array([10.,10.,10.])
# the magnitude of our charge. This could be any number
# at the moment.
qcharge = 1.0
# Create a grid for the electric field. This has the size of
# x,y and z, but all the values are now set to zero.
Ex, Ey, Ez = x*0, y*0, z*0

# Calculate the x, y and z distance to the charge at every point in the grid:
rx = x - qpos[0]
ry = y - qpos[1]
rz = z - qpos[2]

# Calculate the distance at every point in the grid:
r = sqrt(rx**2 + ry**2 + rz**2)

# Calculate the field for each component at every point in the grid:
Ex = (qcharge / r**2) * (rx / r)
Ey = (qcharge / r**2) * (ry / r)
Ez = (qcharge / r**2) * (rz / r)

# Draw the vector field in 3D
quiver3d(x,y,z,Ex,Ey,Ez)
```

vectorfield.py

2.2 Wk9Ex3:

The expressions needed for question 3 are contained within the moodle pdf "EM Standing Waves in Resonant Cavities". Consider a rectangular cavity with perfectly reflecting walls (dimensions a x b x d).

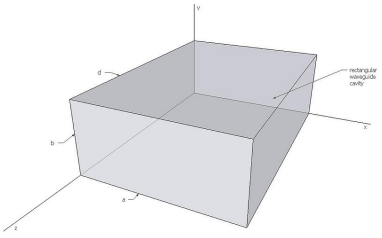

Figure 2: Rectangular resonant cavity

- Write code to calculate the resonant frequencies as a function of the mode numbers, use reduced units scale everything to a/v (v speed of light)

- Make a list of the 10 first frequencies for a cavity 36 x 33 x 23 cm. This is about the size of my microwave oven, comment on the typical operating frequency of a microwave.
- Plot the E, B field lines for the two first TE modes
- Plot the current intensity on the wall, find the nodes location
- Repeat the last two plots with the 2 first TM modes.

TE modes (Transverse Electric) no electric field in the direction of propagation. TM modes (Transverse Magnetic) no magnetic field in the direction of propagation.

An alternative description of the physics can be found at: http:cas.web.cern.chcasDenmark-2010LecturesWolski-2.pdf (page 32)