Ampere, Biot-Savart, Grassman and Weber Magnetic Force Laws with Code

Kurt Nalty

July 16, 2011

Abstract

Ampere derived a force law between current elements assuming force only pointed along the line joining the two elements. Biot-Savart provide a different force law, which was easier for manual calculations, and provided identical results provided one current path was a closed curve. Maxwell provided an explanation for multiple force laws providing same results integrated around a closed path, and provided four examples. Whittaker provided yet another presentation of equivalent force laws. This paper presents the work of these authors, along with simple C code verifying the closed loop equivalency of these forms using MKS magnetic units.

Ampere

During the years 1822 through 1825, Andre-Marie Ampere experimentally and mathematically studied electromagnetism. Maxwell, Article 502, describes Ampere's experimental null setup, and Ampere's results.

Quoting Whittaker, page 85, Ampere proved

1. The effect of a current is reversed when the current is reversed.

2. The effect of a current flowing in a circuit twisted into small sinuousites is the same as if the circuit were smoothed out.

3. The force exerted by a closed circuit on an element of another circuit is at right angles to the latter.

4. The force between two elements of circuits is unaffected when all linear dimensions are increased proportionally, with the current unaltered.

With the additional assumption that the force between two current elements was aligned with the vector separating those elements, Ampere determined the force law between two infinitesimal elements ds and ds', separated by a distance r. Expressed in MKS units, his force law is

$$\frac{d^2\vec{F}}{ds\ ds'} = \frac{\mu II'}{4\pi r^2}\left(2r\frac{\partial^2 r}{\partial s\ \partial s'} - \frac{\partial r}{\partial s}\frac{\partial r}{\partial s'}\right)\vec{a}_r$$

We integrate around our two paths to get the total force.

$$\vec{F} = \frac{\mu II'}{4\pi}\oint\oint\frac{1}{r^2}\left(2r\frac{\partial^2 r}{\partial s\ \partial s'} - \frac{\partial r}{\partial s}\frac{\partial r}{\partial s'}\right)\vec{a}_r\ ds\ ds'$$

To convert this to vector algebra notation, we note

$$\begin{aligned}
ds^2 &= dx^2 + dy^2 + dz^2 \\
ds'^2 &= dx'^2 + dy'^2 + dz'^2 \\
\vec{u} &= \frac{dx}{ds}\vec{a}_x + \frac{dy}{ds}\vec{a}_y + \frac{dz}{ds}\vec{a}_z \\
\vec{u}' &= \frac{dx'}{ds'}\vec{a}_x + \frac{dy'}{ds'}\vec{a}_y + \frac{dz'}{ds'}\vec{a}_z \\
r^2 &= (x-x')^2 + (y-y')^2 + (z-z')^2 \\
r &= \sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2}
\end{aligned}$$

Taking partial derivatives,

$$\begin{aligned}
\frac{\partial r}{\partial x} &= \frac{\partial}{\partial x}\sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2} \\
&= \frac{1}{2r}\frac{\partial}{\partial x}(x-x')^2 = \frac{x-x'}{r}
\end{aligned}$$

In a similar fashion,

$$\begin{aligned}
\frac{\partial r}{\partial x} &= \frac{x-x'}{r} \\
\frac{\partial r}{\partial y} &= \frac{y-y'}{r} \\
\frac{\partial r}{\partial z} &= \frac{z-z'}{r} \\
\frac{\partial r}{\partial x'} &= -\frac{x-x'}{r} \\
\frac{\partial r}{\partial y'} &= -\frac{y-y'}{r} \\
\frac{\partial r}{\partial z'} &= -\frac{z-z'}{r}
\end{aligned}$$

Using the chain rule for partial derivatives, we have

$$\begin{aligned}
\frac{\partial r}{\partial s} &= \frac{\partial r}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial r}{\partial y}\frac{\partial y}{\partial s} + \frac{\partial r}{\partial z}\frac{\partial z}{\partial s} \\
&= \frac{1}{r}\left[(x-x')\frac{\partial x}{\partial s} + (y-y')\frac{\partial y}{\partial s} + (z-z')\frac{\partial z}{\partial s}\right] \\
&= \vec{a}_r \cdot \vec{u} \\
\frac{\partial r}{\partial s'} &= \frac{1}{r}\left[-(x-x')\frac{\partial x'}{\partial s'} + -(y-y')\frac{\partial y'}{\partial s'} + -(z-z')\frac{\partial z'}{\partial s'}\right] \\
&= -\vec{a}_r \cdot \vec{u}'
\end{aligned}$$

For the double derivative, we have

$$\begin{aligned}
\frac{\partial^2 r}{\partial s'\,\partial s} &= \frac{\partial}{\partial s'}\left(\frac{1}{r}\left[(x-x')\frac{\partial x}{\partial s} + (y-y')\frac{\partial y}{\partial s} + (z-z')\frac{\partial z}{\partial s}\right]\right) \\
&= \frac{\partial}{\partial s'}\left(\frac{1}{r}\right)\left[(x-x')\frac{\partial x}{\partial s} + (y-y')\frac{\partial y}{\partial s} + (z-z')\frac{\partial z}{\partial s}\right] \\
&\quad + \frac{1}{r}\frac{\partial}{\partial s'}\left[(x-x')\frac{\partial x}{\partial s} + (y-y')\frac{\partial y}{\partial s} + (z-z')\frac{\partial z}{\partial s}\right] \\
&= -\left(\frac{1}{r^2}\frac{\partial r}{\partial s'}\right)\left[(x-x')\frac{\partial x}{\partial s} + (y-y')\frac{\partial y}{\partial s} + (z-z')\frac{\partial z}{\partial s}\right] \\
&\quad - \frac{1}{r}\left[\frac{\partial x'}{\partial s'}\frac{\partial x}{\partial s} + \frac{\partial y'}{\partial s'}\frac{\partial y}{\partial s} + \frac{\partial z'}{\partial s'}\frac{\partial z}{\partial s}\right]
\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 r}{\partial s'\,\partial s} &= -\left(\frac{1}{r}\frac{\partial r}{\partial s'}\right)(\vec{a}_r\cdot\vec{u}) - \frac{1}{r}(\vec{u}\cdot\vec{u}') \\ &= \frac{1}{r}\left[(\vec{a}_r\cdot\vec{u})(\vec{a}_r\cdot\vec{u}') - (\vec{u}\cdot\vec{u}')\right]\end{aligned}$$

Substituting these expressions in Ampere's force law, we have

$$\begin{aligned}\vec{F} &= \frac{\mu II'}{4\pi}\oint\oint\frac{1}{r^2}\left(2r\frac{\partial^2 r}{\partial s\;\partial s'} - \frac{\partial r}{\partial s}\frac{\partial r}{\partial s'}\right)\vec{a}_r\;ds\;ds' \\ &= \frac{\mu II'}{4\pi}\oint\oint\frac{1}{r^2}\left(2\left[(\vec{a}_r\cdot\vec{u})(\vec{a}_r\cdot\vec{u}') - (\vec{u}\cdot\vec{u}')\right] - (\vec{a}_r\cdot\vec{u})(-\vec{a}_r\cdot\vec{u}')\right)\vec{a}_r\;ds\;ds' \\ &= \frac{\mu II'}{4\pi}\oint\oint\frac{1}{r^2}\left[3(\vec{a}_r\cdot\vec{u})(\vec{a}_r\cdot\vec{u}') - 2(\vec{u}\cdot\vec{u}')\right]\vec{a}_r\;ds\;ds'\end{aligned}$$

Weber

In 1846, Weber published an electrodynamical force law generalized from Ampere, based upon potential theory of Gauss.

$$\vec{F} = \frac{q_i q_j}{4\pi\epsilon_0}\frac{\vec{r}_{ij}}{r_{ij}^3}\left(1 - \frac{\dot{r}_{ij}^2}{2c^2} + \frac{r_{ij}\ddot{r}_{ij}}{c^2}\right)$$

To obtain Ampere's law from this law, Weber used the Fechner hypothesis of equal and opposite currents flowing in opposite directions in a conductor. Weber's electrodynamics fell out of favor as Fechner's hypothesis was later disproved, and Helmholtz noted that run-away solutions existed for particle speed.

As it turns out, this is fairly unfortunate. Fechner's hypothesis was a sufficient, yet unnecessary condition. All that is required is neutral conductors to achieve Ampere's law from Weber's, as shown in Assis in *Weber's Electrodynamics*. Furthermore, Helmholtz's run-away solutions happen to be tachyonic solutions, where the particle speed exceeds that of light, and thus are currently non-physical. As an aside, Helmholtz's charged sphere accelerator looks fun to build, and certainly has science fiction appeal.

To get Ampere's law from Weber's, we add four contributions

1. Source electrons interacting with sensor electrons
2. Source nucleons (stationary) interacting with sensor electrons
3. Source electrons interacting with sensor nucleons (stationary)
4. Source nucleons (stationary) interacting with sensor nucleons (stationary) - no magnetic effects.

Biot-Savart

On October 30, 1820, Jean-Baptiste Biot and Felix Savart presented a magnetic force law between a magnet pole and a current flowing in a long straight wire. Quoting via Whittaker, Vol II, p 82,

> Draw from the pole a perpendicular to the wire; the force on this pole is at right angles to this line and to the wire, and its intensity is proportional to the reciprocal of the distance.

This implied a magnetic field was created by the current in the wire. The field was proportional to the current, but at right angles to the current and the distance from the current. The immediate results from the inverse radius observation became $\oint \vec{H} \cdot d\vec{s} = I_{\text{enclosed}}$. This led to $\vec{\nabla} \times \vec{B} = \mu \vec{J}$, which in turn, led to the definition below.

Define magnetic fields by

$$\begin{aligned} \vec{B} &= \frac{\mu}{4\pi} \oint \frac{\vec{I} \times \vec{a}_r}{r^2} ds \\ &= \frac{\mu I}{4\pi} \oint \frac{\vec{u} \times \vec{a}_r}{r^2} ds \end{aligned}$$

The differential force on another current elements is

$$\begin{aligned}
\frac{d\vec{F}}{ds'} &= I'\vec{u}' \times \vec{B} \\
\vec{F} &= \oint I'\vec{u}' \times \left(\frac{\mu I}{4\pi} \oint \frac{\vec{u} \times \vec{a}_r}{r^2} ds \right) ds' \\
&= \frac{\mu II'}{4\pi} \oint \oint \vec{u}' \times \left(\frac{\vec{u} \times \vec{a}_r}{r^2} \right) ds \; ds' \\
&= \frac{\mu II'}{4\pi} \oint \oint \frac{1}{r^2} \left[\vec{u} \left(\vec{u}' \cdot \vec{a}_r \right) - \vec{a}_r \left(\vec{u} \cdot \vec{u}' \right) \right] ds \; ds'
\end{aligned}$$

This became the Biot-Savart force law. Comparing the Ampere and Biot-Savart force laws, we have

$$\begin{aligned}
\vec{F}_a &= \frac{\mu II'}{4\pi} \oint \oint \frac{1}{r^2} \left[3 \left(\vec{a}_r \cdot \vec{u} \right) \left(\vec{a}_r \cdot \vec{u}' \right) - 2 \left(\vec{u} \cdot \vec{u}' \right) \right] \vec{a}_r \; ds \; ds' \\
\vec{F}_b &= \frac{\mu II'}{4\pi} \oint \oint \frac{1}{r^2} \left[\vec{u} \left(\vec{u}' \cdot \vec{a}_r \right) - \vec{a}_r \left(\vec{u} \cdot \vec{u}' \right) \right] ds \; ds'
\end{aligned}$$

Looking just at the terms in the square brackets, the other parts of the expression being equal, we see that each calculation for Ampere requires 15 multiplications and 7 additions, while Biot-Savart requires 12 multiplications and 7 additions. In a world before computers, this was a significant consideration when making calculations. Both calculations yield the same result. Consequently, for engineering calculations, Biot-Savart became the working formula, and Ampere's formula fell into neglect.

Maxwell

Maxwell, in A Treatise on Electricity and Magnetism, Vol. II, article 161 explains why multiple good working formulas for magnetic forces on closed circuits exist. Inside the loop integrals above, any potential can be added, which will self-cancel when the loop integral returns to the initial point.

Maxwell demonstrates several equivalent force formulas by using $Q(r)$ in the following form.

$$\frac{d^2\vec{F}}{ds \; ds'} = \frac{\mu II'}{4\pi r^2} \left[\left(2r \frac{\partial^2 r}{\partial s \; \partial s'} - \frac{\partial r}{\partial s} \frac{\partial r}{\partial s'} - r \frac{\partial^2 Q}{\partial s \; \partial s'} \right) \vec{a}_r + \frac{\partial Q}{\partial s'} \vec{u} - \frac{\partial Q}{\partial s} \vec{u}' \right]$$

Maxwell then gives three formulas, all of which provide the same macroscopic forces as Ampere and Biot-Savart.

Grassman - Eqn 43

The first form, Maxwell credits to Grassman, [Pogg., Ann. lxiv. p1](1845). Grassman assumed that two current elements in the same line have no mutual action.

$$\begin{aligned} Q &= -\frac{1}{2r} \\ F_r &= \frac{3}{2r}\frac{\partial^2 r}{\partial s\ \partial s'} \\ S &= \frac{1}{2r^2}\frac{\partial r}{\partial s'} \\ S' &= -\frac{1}{2r^2}\frac{\partial r}{\partial s} \end{aligned}$$

$$\begin{aligned} \vec{F} &= \frac{\mu I I'}{4\pi}\oint\oint\frac{1}{2r^2}\left[\left(3r\frac{\partial^2 r}{\partial s\ \partial s'}\right)\vec{a}_r + \frac{\partial r}{\partial s'}\vec{u} - \frac{\partial r}{\partial s}\vec{u}'\right] ds\ ds' \\ &= \frac{\mu I I'}{4\pi}\oint\oint\frac{1}{2r^2} \\ &\quad \left[\left(3\left(\vec{a}_r\cdot\vec{u}\right)\left(\vec{a}_r\cdot\vec{u}'\right) - \left(\vec{u}\cdot\vec{u}'\right)\right)\vec{a}_r - \left(\vec{a}_r\cdot\vec{u}'\right)\vec{u} - \left(\vec{a}_r\cdot\vec{u}\right)\vec{u}'\right] ds\ ds' \end{aligned}$$

Maxwell - Eqn 44

For the next example, Maxwell asserts that the attraction of the current elements is proportional to the cosine of the angle between them. This is equivalent to the Whittaker formula discussed later.

$$
\begin{aligned}
Q &= -\frac{1}{r} \\
F_r &= -\frac{1}{r^2}\vec{u}\cdot\vec{u}' \\
S &= \frac{1}{r^2}\frac{\partial r}{\partial s'} \\
S' &= -\frac{1}{r^2}\frac{\partial r}{\partial s}
\end{aligned}
$$

$$
\vec{F} = \frac{\mu I I'}{4\pi}\oint\oint \frac{1}{r^2}\left[(\vec{u}\cdot\vec{u}')\,\vec{a}_r - (\vec{a}_r\cdot\vec{u}')\,\vec{u} - (\vec{a}_r\cdot\vec{u})\,\vec{u}'\right] ds\; ds'
$$

Maxwell - Eqn 45

For the last case, Maxwell assumes the forces depent only on the angles the elements make with the lines joining them.

$$
\begin{aligned}
Q &= -\frac{2}{r} \\
F_r &= 3\frac{1}{r^2}\frac{\partial r}{\partial s}\frac{\partial r}{\partial s'} \\
S &= \frac{2}{r^2}\frac{\partial r}{\partial s'} \\
S' &= -\frac{2}{r^2}\frac{\partial r}{\partial s}
\end{aligned}
$$

$$
\vec{F} = \frac{\mu I I'}{4\pi}\oint\oint \frac{2}{r^2}\left[-\frac{3}{2}\left((\vec{a}_r\cdot\vec{u}')\,(\vec{a}_r\cdot\vec{u})\right)\vec{a}_r - (\vec{a}_r\cdot\vec{u}')\,\vec{u} - (\vec{a}_r\cdot\vec{u})\,\vec{u}'\right] ds\; ds'
$$

Whittaker

Whittaker, in *History of the Theories of Aether and Electricity* presents the multiplicity of indistinguishable force laws, this time from a more vector oriented point of view. Whittaker makes the observation that the most general

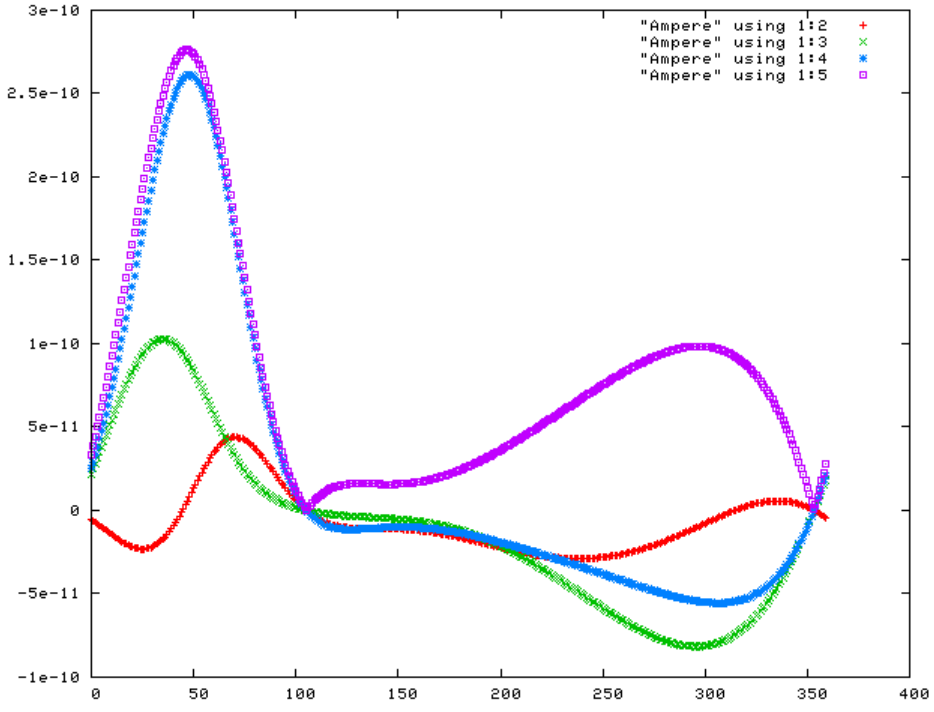
interaction between extended particles will consist of a force, change linear momentum, as well as a couple, changing internal angular momentum.

Whittaker's quaternion product force law is the same as Maxwell's Eqn. 44, above.

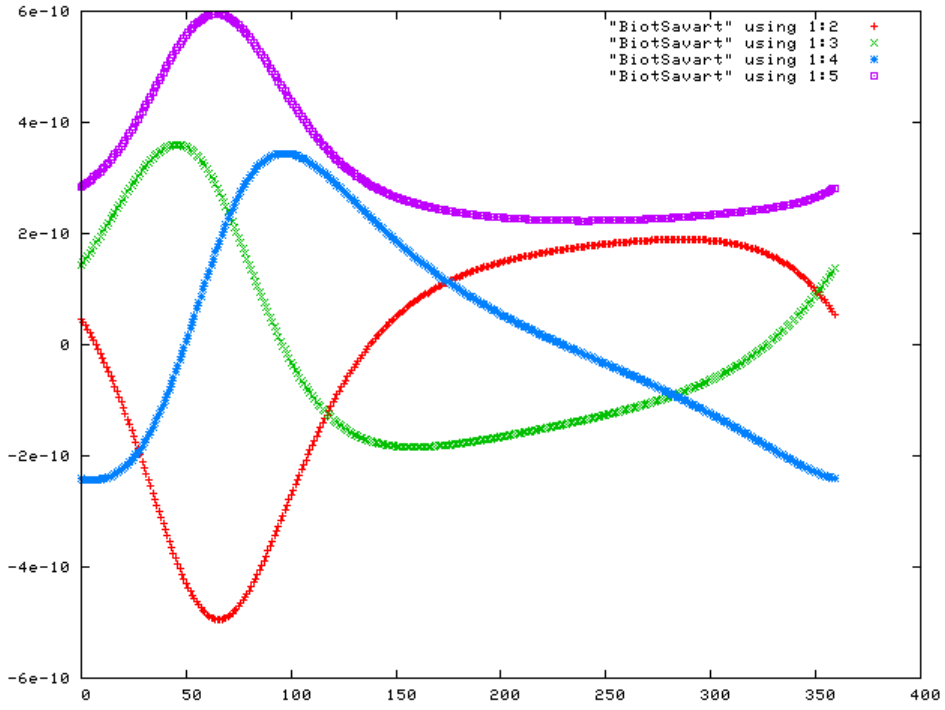
Graphical Comparision of Force Contributions

In the MagneticForce.c program later listed, we show that the accumulated force on a sensor current element due to a close source current is equal for the different force laws. We also show that the force profile, due to different source elements, is indeed different. We cannot, from macroscopic measurements, discern the actual force law for elemental currents.

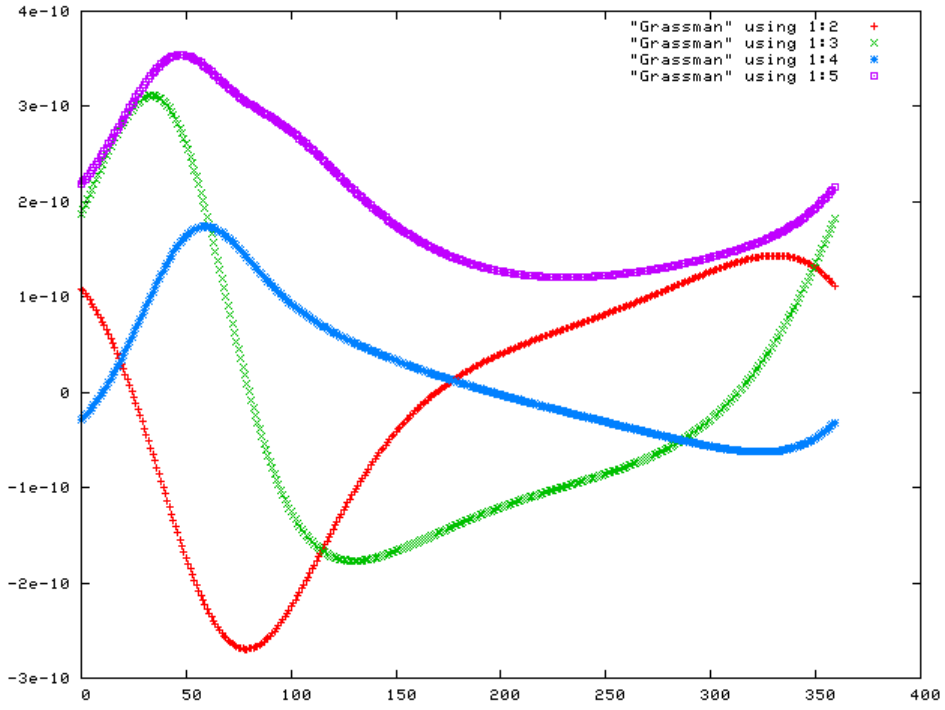
Following are the force contribution profiles from the different force laws presented above.



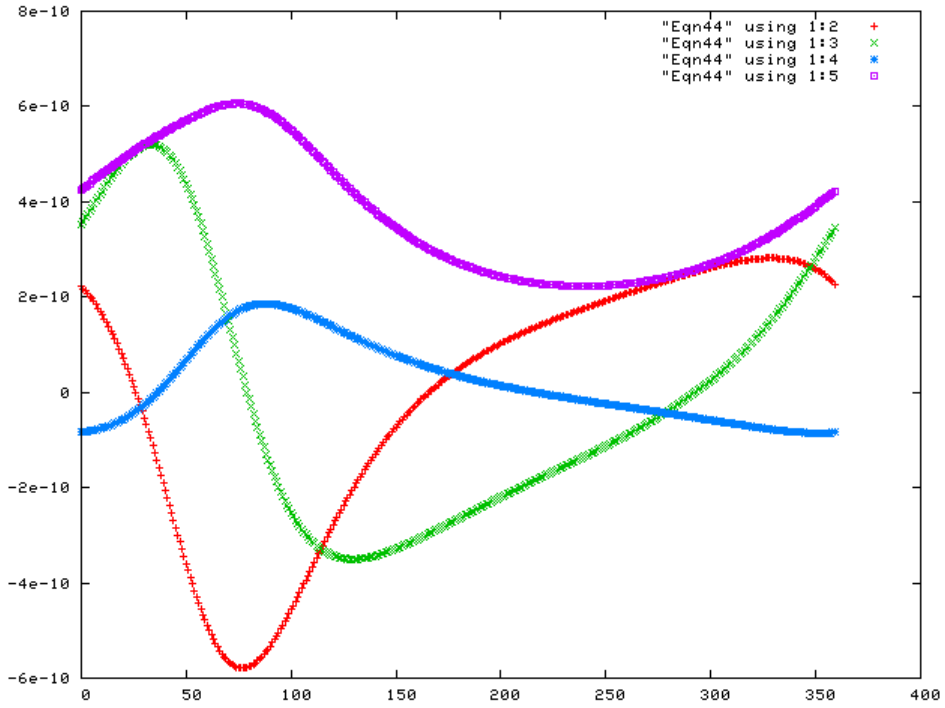
Ampere Force Contribution



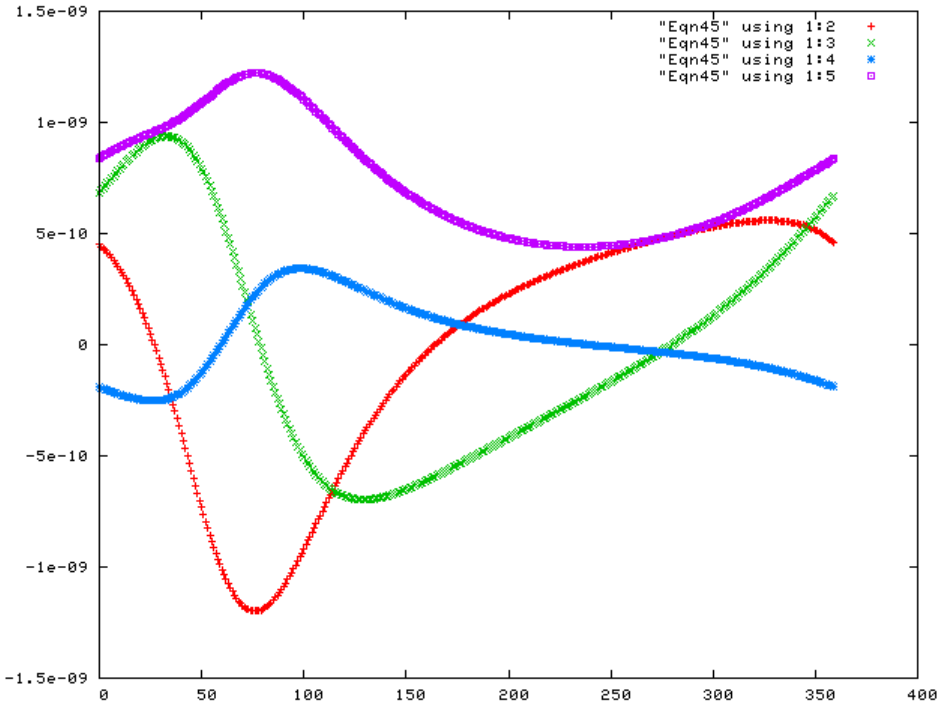
Biot-Savart Force Contribution



Grassman Force Contribution



Maxwell Eqn. 44 and Whittaker Force Contribution



Maxwell Eqn. 45 Force Contribution

Numerical Code

Here is the listing of the MagneticFields.c program, available at `http://www.kurtnalty.com/MagneticFields.c` .

```
// compile with tcc MagneticFields.c -o MagneticFields -lm
//          or  gcc MagneticFields.c -o MagneticFields -lm
// run with ./MagneticFields

#include <stdio.h>
#include <stdlib.h>
#include <math.h>



typedef struct
```

```
{
double x;
double y;
double z;
} Vector;

Vector Cross(Vector A, Vector B)
{
Vector 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);
};

void PrintVector(Vector A)
{
printf("( %e, %e, %e )",A.x,A.y,A.z);
}

Vector AddVector(Vector A, Vector B)
{
Vector C;
C.x = A.x + B.x;
C.y = A.y + B.y;
C.z = A.z + B.z;
return C;
}

Vector SubVector(Vector A, Vector B)
{
Vector C;
C.x = A.x - B.x;
C.y = A.y - B.y;
C.z = A.z - B.z;
return C;
}
```

```
Vector ScaleVector(Vector A, double scale)
{
Vector B;
B.x = A.x*scale;
B.y = A.y*scale;
B.z = A.z*scale;
return B;
}

float Dot(Vector A, Vector B)
{
return (A.x*B.x + A.y*B.y + A.z*B.z);
}


int Complete_Elliptic_K_and_E(char arg, double parameter, double* K, double* E)
{
/* Usage: the character arg is 'k', 'm', or 'a' to inform this routine of
               the  type of parameter being used. Pointers are used for K and E so
       so that we  can return multiple values without a struct.

               Complete_Elliptic_K_and_E('k',k,&K,&E);

        License: Freeware by Kurt Nalty, 2011.

        Credits: Algorithm based upon Handbook of Mathematical Functions,
                 Abramowitz and Stegun, National Bureau of Standards, pp 589-599

        Credit Coding Based After:
http:mymathlib.webtrellis.net/functions/elliptic_integrals.html
     (I very much liked his use of a letter parameter for function selection)


     returned code:  0 -> no problem
                    -1 -> bad parameter specification
                    -2 -> K sent to infinity by k=1, m=1, or alpha = 90 degrees

Accuracy - seems to be within 5 digits easily.
```

```
*/

double a[10],b[10],c[10];
int i;
double k, m, alpha;
double pi = 3.141592653589793;
double S, scale;
double tol = 1.0e-10,err; // initial tolerance

// Check arguments

switch (arg) {
case 'k':
k = parameter;
m = k*k;
alpha = asin(k);
break;
case 'm':
k = sqrt(parameter);
m = parameter;
alpha = asin(k);
break;
case 'a':
k = sin(parameter);
m = k*k;
alpha = parameter;
break;
                default:
*K = 1.0E30; // infinity approximated
*E = 1.0;
return (-2); // bad argument
}

// if (m > 0.99999999) {
// *K = 10.6; // infinity approximated
// *E = 1.0;
// return (-2); // bad argument
```

```
// }

/*
K(m) = /int ^{\pi/2} _{0} \left( 1 - m \sin^2 \theta \right)^{-1/2}
        d\theta
E(m) = /int ^{\pi/2} _{0} \left( 1 - m \sin^2 \theta \right)^{1/2}
 d\theta

For the AGM process

a[0] = 1.0     b[0] = cos(alpha) = sqrt(1-k*k)   c[0] = sin(alpha) = k

a[i] = 0.5*(a[i-1] + b[i-1])   b[i] = sqrt(a[i-1]*b[i-1])
                        c[i] = 0.5*(a[i-1] - b[i-1])

when c[i] is small enough,

K(alpha) = pi/(2 a[n])

S = 0.5*Sigma_i (2^i c_i^2) = Sigma_i (2^(i-1) c_i^2)
*/

a[0] = 1.0;  b[0] = cos(alpha);  c[0] = sin(alpha);  S = 0.5*c[0]*c[0];  scale = 1

for (i=1;i<9;i++) {
//max interation depth is 10. Leave early if within tolerance

a[i] = 0.5*(a[i-1] + b[i-1]);
b[i] = sqrt(a[i-1]*b[i-1]);
c[i] = 0.5*(a[i-1] - b[i-1]);
S += scale*c[i]*c[i];
scale *= 2.0;
err = (c[i]);
if (err < 0.0) err *= -1.0; // take magnitude
if(err <= tol) break;

}
*K = pi/(2.0*a[i]);
```

```
*E = *K - *K*S;
if (i==9) return(-2); // no convergence

return 0;

}



int main(void)
{
int i,j;

Vector A,B,dB,dF,dR,F,dFdl,R,a_R;

Vector SourceCurrent[360];
// current source as vector segments, 1A, tangential direction,
// offset 0.5 degree
Vector SourcePosition[360];
// current location, centered in segment, offset 0.5 degree

Vector SensorCurrent;
Vector SensorPosition;

Vector P, S, SP; // print components of force using P
FILE* BiotSavart;
FILE* Ampere;
FILE* Whittaker;
FILE* Eqn43;
FILE* Eqn44;
FILE* Eqn45;

double theta, dtheta, phi, a, r, r2, x, y, current,scale;
double pi = 3.141592653589793;
        double k, rho, z, Psi, CapitalPsi, B_rho, B_x, B_y, B_z, I, E, K;
double C,Alpha, Beta, Gamma, Pmag;
double drds, drdsp, d2rdsdsp;
```

```
FILE* OutputWireFrame;
FILE* EFile;
FILE* KFile;
FILE* ETimesK;

EFile = fopen("E(k)","w");
KFile = fopen("K(k)","w");
ETimesK = fopen("E^2*K","w");
for (i=0;i<100;i++) {
k = i/100.0;
Complete_Elliptic_K_and_E('k',k,&K, &E);
fprintf(ETimesK,"%g   %g  \n",k,E*E*K);
fprintf(EFile,"%g   %g  \n",k,E);
fprintf(KFile,"%g   %g  \n",k,K);
}
fclose(EFile);
fclose(KFile);
fclose(ETimesK);


OutputWireFrame = fopen("Path.xyz","w");


// Define the sensor current element and position

current = 1.0;
//these currents, positions, angles are arbitrary
phi = 30.0*pi/180.0;
theta = 50*pi/180.0;

SensorCurrent.x = current*cos(theta)*cos(phi);
SensorCurrent.y = current*sin(theta)*cos(phi);
SensorCurrent.z = current*sin(phi);

r = 2.0;
phi = 45.0*pi/180.0;
theta = 60*pi/180.0;
SensorPosition.x = r*cos(theta)*cos(phi);
```

```
SensorPosition.y = r*sin(theta)*cos(phi);
SensorPosition.z = r*sin(phi);

// Draw a vector from zero to the SensorPosition
fprintf(OutputWireFrame,"0 0 0 0 \n"); // moveto
fprintf(OutputWireFrame,"%f %f %f 90 \n",
  SensorPosition.x,SensorPosition.y,SensorPosition.z); // lineto

// Draw a vector from zero to the SensorPosition height
fprintf(OutputWireFrame,"0 0 0 0 \n"); // moveto
fprintf(OutputWireFrame,"0 0 %f 120 \n",
     SensorPosition.z); // lineto

// Draw a vector from SensorPosition.z to the SensorPosition
fprintf(OutputWireFrame,"0 0 %f 0 \n",SensorPosition.z); // moveto
fprintf(OutputWireFrame,"%f %f %f 150 \n",
   SensorPosition.x,SensorPosition.y,SensorPosition.z); // lineto

// Draw a vector from zero to a SourcePosition
fprintf(OutputWireFrame,"0 0 0 0 \n"); // moveto
fprintf(OutputWireFrame,"1 0 0 180 \n"); // lineto

// Draw a vector from SourcePosition to the SensorPosition
fprintf(OutputWireFrame,"%f %f %f 90 \n",
   SensorPosition.x,SensorPosition.y,SensorPosition.z); // lineto


// make a source current clockwise in a unit circle in the XY plane
a = r = 1.0;
current = 1.0;

for (i=0;i<360;i++) { //draw a wire frame for geometry sanity check
theta = (i+0.5)*pi/180.0;
SourcePosition[i].x = r*cos(theta);
SourcePosition[i].y = r*sin(theta);
SourcePosition[i].z = 0.0;

SourceCurrent[i].x = -current*sin(theta);
```

```
SourceCurrent[i].y =  current*cos(theta);
SourceCurrent[i].z =  0.0;

phi = i*pi/180.0;
x = r*cos(phi);
y = r*sin(phi);
z = 0.0;
fprintf(OutputWireFrame,"%f %f %f 0 \n",x,y,z); // moveto

phi = (i+1.0)*pi/180.0;
x = r*cos(phi);
y = r*sin(phi);
z = 0.0;
fprintf(OutputWireFrame,"%f %f %f 30 \n",x,y,z); // lineto

}

fclose(OutputWireFrame);

// Do a calculation of the force on the sensor element using BiotSavart

// First calculate B

B.x = 0.0; B.y = 0.0; B.z = 0.0; // initialize
dtheta = 2.0*pi/360.00;

for (i=0;i<360;i++) {
// dB = 1.0e-7*(Cross(SourceCurrent[i],a_R))/(dot(dR,dR);
dR = SubVector(SensorPosition,SourcePosition[i]);
   // sensor - source
r2 = Dot(dR,dR);
r = sqrt(r2);
a_R = ScaleVector(dR,(1.0/r));
dB = ScaleVector(Cross(SourceCurrent[i],a_R),
          (current*1.0e-7*a*dtheta/r2)); // u_0/(4 pi)
B = AddVector(B,dB);
}
// print B for comparison to integral calculation
```

```
printf("B calculated by summing 360 one degree segments\n");
printf("\nB = "); PrintVector(B); printf("\n\n");


// Now we calculate B using a current loop and Elliptic Integrals

/*
        Current loop in XY plane. Current I, radius a

k = 2 sqrt[(a rho)/(z^2 + (a + rho)^2 )]

Psi = (a^2 - rho^2 - z^2)/((a - rho)^2 - z^2)

B_rho = ((u_0 I k z)/(4 pi rho sqrt(a rho))) [-K(k) + Psi E(k)]
B_z = ((u_0 I k )/(4 pi sqrt(a rho))) [K(k) + Psi E(k)]


*/

// Cylindrical coordinates, reference Zhao (corrected by knalty)

I = current;

x = SensorPosition.x;
y = SensorPosition.y;
z = SensorPosition.z;

rho = sqrt(x*x + y*y);

a = 1.0; // previously defined
k = sqrt((4.0*a*rho)/(z*z + (a + rho)*(a + rho)));

Psi = (a*a - rho*rho - z*z)/((a - rho)*(a - rho) + z*z);
CapitalPsi = (a*a + rho*rho + z*z)/((a - rho)*(a - rho) + z*z);

Complete_Elliptic_K_and_E('k',k,&K, &E);

B_rho = ((1.0e-7*I*k*z)/(rho*sqrt(a*rho) )) * (- K + E*CapitalPsi);
```

```
B_z =   ((1.0e-7*I*k)  /(sqrt(a*rho)) ) * (K + E*Psi);

printf("(Elliptic Integral Zhao  )   B_rho = %g   B_z = %g  \n",
          B_rho, B_z);
B_x = B_rho*x/sqrt(x*x + y*y);
B_y = B_rho*y/sqrt(x*x +y*y);
printf("(Cyl to Cartesian of Above)  B_x = %g   B_y = %g   B_z = %g  \n",
B_x, B_y, B_z);

// Cylindrical coordinates, reference Julian Swinger,
//  Classical Electrodynamics, Problem 29.2, p334 (1998)

I = current;

x = SensorPosition.x;
y = SensorPosition.y;
z = SensorPosition.z;

rho = sqrt(x*x + y*y);

a = 1.0; // previously defined
k = sqrt((4.0*a*rho)/(z*z + (a + rho)*(a + rho)));


Complete_Elliptic_K_and_E('k',k,&K, &E);

B_rho = -((1.0e-7*I*2.0*z)/(rho*sqrt( (a+rho)*(a+rho) + z*z)))
        * (K - E*(a*a + rho*rho + z*z)/( (a - rho)*(a - rho) + z*z ));
B_z =   ((1.0e-7*I*2.0)  /(sqrt((a+rho)*(a+rho) + z*z)))
  * (K + E*(a*a - rho*rho - z*z)/((a - rho)*(a - rho) + z*z));

printf("(Numerical Summation Method) B.rho = %g   B.z = %g  \n",
          sqrt(B.x*B.x + B.y*B.y), B.z);
printf("(Elliptic Integral Method)   B_rho = %g   B_z = %g  \n",
            B_rho, B_z);
B_x = B_rho*x/sqrt(x*x + y*y);
B_y = B_rho*y/sqrt(x*x +y*y);
printf("(Cyl to Cartesian of Above)  B_x = %g   B_y = %g   B_z = %g  \n",
```

```
                B_x, B_y, B_z);

// Try another set of formulas

// James Simpson and others,
// Simple Analytic Expressions for the Magnetic Field of Circular Current Loop,
//  Feb 2001

C = 4.0e-7*I;
x = SensorPosition.x;
y = SensorPosition.y;
z = SensorPosition.z;

rho = sqrt(x*x + y*y);
r = sqrt(x*x + y*y + z*z);
Alpha = sqrt(a*a + r*r - 2.0*a*rho);
Beta = sqrt(a*a + r*r + 2.0*a*rho);
k = sqrt(1.0 - ((Alpha*Alpha)/(Beta*Beta)));
Gamma = SensorPosition.x*SensorPosition.x - SensorPosition.
    y*SensorPosition.y;
Complete_Elliptic_K_and_E('k',k,&K, &E);

B_x = ((C*x*z)/(2.0*Alpha*Alpha*Beta*rho*rho))*
        ( (a*a + r*r)*E - Alpha*Alpha*K   );
B_y = ((C*y*z)/(2.0*Alpha*Alpha*Beta*rho*rho))*
        ( (a*a + r*r)*E - Alpha*Alpha*K   );
B_z = (C/(2.0*Alpha*Alpha*Beta))*((a*a - r*r)*E
         + Alpha*Alpha*K    );
printf("(Elliptic Integral Method)   B_x = %g   B_y = %g   B_z = %g  \n",
   B_x, B_y, B_z);
printf("(Numerical Summation Method) B.x = %g   B.y = %g   B.z = %g  \n",
       B.x, B.y, B.z);

// calculate force dF = dl*Cross(SensorCurrent,B)

printf("\n\nThe purpose of the above calculations is to \
validate the conventional formula and piecewise approximation.\n\n");
```

```
dFdl = Cross(SensorCurrent,B);
printf("\ndFdl (Reference from dIXB) = "); PrintVector(dFdl);
 printf("\n");

printf("\nNow, we are going to reference Whittaker - \
History of the Theories of Aether and Electricity,\n");
printf("and Maxwell, Volume 2, article 527\n\n");

BiotSavart = fopen("BiotSavart","w");
Ampere = fopen("Ampere","w");
Whittaker = fopen("Whittaker","w");

Eqn43 = fopen("Grassman","w");
Eqn44 = fopen("Eqn44","w");
Eqn45 = fopen("Eqn45","w");


// Ampere's formula : dFdl = Constant*I_1*I_2 /oint /vec r
// ( 2*r^{-3} ds1.ds2 - 3*r^{-5} (ds1.r)(ds2.r)   ) d s_source

F.x = 0.0;  F.y = 0.0;  F.z = 0.0;
for (i=0;i<360;i++) {
dR = SubVector(SensorPosition,SourcePosition[i]);
// sensor - source
r2 = Dot(dR,dR);
r = sqrt(r2);
a_R = ScaleVector(dR,(1.0/r));
scale = -1.0e-7*((2.0*Dot(SourceCurrent[i],SensorCurrent)/r2) -
  (3.0*Dot(SourceCurrent[i],a_R)*Dot(SensorCurrent,a_R)/(r2)));
P = ScaleVector(a_R,scale*a*dtheta);
Pmag = sqrt(P.x*P.x + P.y*P.y + P.z*P.z);
fprintf(Ampere,"%d  %g  %g   %g    %g \n",
    i,P.x, P.y, P.z, Pmag);
F = AddVector(F,P);
}
printf("dFdl (Ampere Law form)            = "); PrintVector(F);
printf("\n");
fclose(Ampere);
```

```
F.x = 0.0;  F.y = 0.0;  F.z = 0.0;
for (i=0;i<360;i++) {
dR = SubVector(SensorPosition,SourcePosition[i]);
  // sensor - source
r2 = Dot(dR,dR);
r = sqrt(r2);
a_R = ScaleVector(dR,(1.0/r));

scale = -1.0e-7*((Dot(SourceCurrent[i],SensorCurrent)/r2));
F = AddVector(F,ScaleVector(a_R,scale*a*dtheta));
P = ScaleVector(a_R,scale*a*dtheta);

scale = 1.0e-7*(Dot(SensorCurrent,a_R)/(r2));
F = AddVector(F,ScaleVector(SourceCurrent[i],scale*a*dtheta));
P = AddVector(P,ScaleVector(SourceCurrent[i],scale*a*dtheta));
Pmag = sqrt(P.x*P.x + P.y*P.y + P.z*P.z);
fprintf(BiotSavart,"%d  %g  %g   %g   %g\n",
      i,P.x, P.y, P.z, Pmag);
}
printf("dFdl (Biot-Savart Law form)       = ");
     PrintVector(F); printf("\n");
fclose(BiotSavart);


F.x = 0.0;  F.y = 0.0;  F.z = 0.0;
for (i=0;i<360;i++) {
dR = SubVector(SensorPosition,SourcePosition[i]);
// sensor - source
r2 = Dot(dR,dR);
r = sqrt(r2);
a_R = ScaleVector(dR,(1.0/r));

scale = -1.0e-7*((Dot(SourceCurrent[i],SensorCurrent)/r2));
F = AddVector(F,ScaleVector(a_R,scale*a*dtheta));
P = ScaleVector(a_R,scale*a*dtheta);
```

```
scale = 1.0e-7*(Dot(SensorCurrent,a_R)/(r2));
F = AddVector(F,ScaleVector(SourceCurrent[i],scale*a*dtheta));
P = AddVector(P,ScaleVector(SourceCurrent[i],scale*a*dtheta));

scale = 1.0e-7*(Dot(SourceCurrent[i],a_R)/(r2));
F = AddVector(F,ScaleVector(SensorCurrent,scale*a*dtheta));
P = AddVector(P,ScaleVector(SensorCurrent,scale*a*dtheta));

Pmag = sqrt(P.x*P.x + P.y*P.y + P.z*P.z);
fprintf(Whittaker,"%d  %g  %g   %g   %g \n",
   i,P.x, P.y, P.z, Pmag);
}
printf("dFdl (Whittaker Riemann Law form) = "); PrintVector(F);
   printf("\n");
fclose(Whittaker);


F.x = 0.0;  F.y = 0.0;  F.z = 0.0;
for (i=0;i<360;i++) {
dR = SubVector(SensorPosition,SourcePosition[i]);
   // sensor - source
r2 = Dot(dR,dR);
r = sqrt(r2);
a_R = ScaleVector(dR,(1.0/r));

scale = 1.0e-7/r2;

drds = Dot(a_R,SourceCurrent[i]);
    // unit current => numerically equal tangent
drdsp = -Dot(a_R,SensorCurrent);
d2rdsdsp = (-drds*drdsp - Dot(SourceCurrent[i],SensorCurrent))/r;

P = ScaleVector(a_R,(2.0*r*d2rdsdsp - drds*drdsp));

P = ScaleVector(P,scale*a*dtheta);
F = AddVector(F,P);
Pmag = sqrt(P.x*P.x + P.y*P.y + P.z*P.z);
}
```

```
printf("dFdl (Ampere in Maxwell Form )    = "); PrintVector(F);
printf("\n");


F.x = 0.0;  F.y = 0.0;  F.z = 0.0;
for (i=0;i<360;i++) {
dR = SubVector(SensorPosition,SourcePosition[i]);
// sensor - source
r2 = Dot(dR,dR);
r = sqrt(r2);
a_R = ScaleVector(dR,(1.0/r));

scale = 1.0e-7/r2;

drds = Dot(a_R,SourceCurrent[i]);
// unit current => numerically equal tangent
drdsp = -Dot(a_R,SensorCurrent);
d2rdsdsp = (-drds*drdsp - Dot(SourceCurrent[i],SensorCurrent))/r;

P = ScaleVector(a_R,(1.5*r*d2rdsdsp)); // Radial Term
S = ScaleVector(SourceCurrent[i],-0.5*drdsp);
      // r2 term in later scale factor
SP = ScaleVector(SensorCurrent,0.5*drds);
P = AddVector(P,S); // S
P = AddVector(P,SP); // S

P = ScaleVector(P,scale*a*dtheta);
F = AddVector(F,P);
Pmag = sqrt(P.x*P.x + P.y*P.y + P.z*P.z);
fprintf(Eqn43,"%d  %g  %g   %g   %g \n",
        i,P.x, P.y, P.z, Pmag);
}
printf("dFdl (Grassman Law form)          = ");
        PrintVector(F); printf("\n");
fclose(Eqn43);


F.x = 0.0;  F.y = 0.0;  F.z = 0.0;
```

```
for (i=0;i<360;i++) {
dR = SubVector(SensorPosition,SourcePosition[i]);
      // sensor - source
r2 = Dot(dR,dR);
r = sqrt(r2);
a_R = ScaleVector(dR,(1.0/r));

scale = 1.0e-7/r2;

drds = Dot(a_R,SourceCurrent[i]);
       // unit current => numerically equal tangent
drdsp = -Dot(a_R,SensorCurrent);
d2rdsdsp = (-drds*drdsp - Dot(SourceCurrent[i],SensorCurrent))/r;

P = ScaleVector(a_R,-Dot(SourceCurrent[i],SensorCurrent));
S = ScaleVector(SourceCurrent[i], -drdsp);
     // r2 term in later scale factor
SP = ScaleVector(SensorCurrent, drds);
P = AddVector(P,S); // S
P = AddVector(P,SP); // S

P = ScaleVector(P,scale*a*dtheta);
F = AddVector(F,P);
Pmag = sqrt(P.x*P.x + P.y*P.y + P.z*P.z);
fprintf(Eqn44,"%d  %g  %g   %g   %g \n",
     i,P.x, P.y, P.z, Pmag);
}
printf("dFdl (Eqn44 Law form)             = "); PrintVector(F);
  printf("\n");
fclose(Eqn44);


F.x = 0.0;  F.y = 0.0;  F.z = 0.0;
for (i=0;i<360;i++) {
dR = SubVector(SensorPosition,SourcePosition[i]);
           // sensor - source
r2 = Dot(dR,dR);
r = sqrt(r2);
```

```
a_R = ScaleVector(dR,(1.0/r));

scale = 1.0e-7/r2;

drds = Dot(a_R,SourceCurrent[i]);
      // unit current => numerically equal tangent
drdsp = -Dot(a_R,SensorCurrent);
d2rdsdsp = (-drds*drdsp - Dot(SourceCurrent[i],SensorCurrent))/r;

P = ScaleVector(a_R,3.0*drds*drdsp);
S = ScaleVector(SourceCurrent[i], -2.0*drdsp);
  // r2 term in later scale factor
SP = ScaleVector(SensorCurrent, 2.0*drds);
P = AddVector(P,S); // S
P = AddVector(P,SP); // S

P = ScaleVector(P,scale*a*dtheta);
F = AddVector(F,P);
Pmag = sqrt(P.x*P.x + P.y*P.y + P.z*P.z);
fprintf(Eqn45,"%d  %g  %g   %g   %g \n",
i,P.x, P.y, P.z, Pmag);
}
printf("dFdl (Eqn45 Law form)             = "); PrintVector(F);
printf("\n");
fclose(Eqn45);

printf("\n");
}
```

References

[1] J. C. Maxwell, *Electricity and Magnetism, Vol II, Article 527* Cambridge University Press, New York. 1873 reprinted 2011.

[2] Sir Edmund Whittaker, *History of the Theories of Aether and Electricity*, Vol II, Thomas Nelson and Sons, New York 1951.

[3] Andre Kock Torres Assis, *Weber's Electrodynamics*, Kluwer Academic Publishers, Boston, 1994.

[4] Milton Abramowitz and Irene A. Stegun, *Handbook of Mathematical Functions with Formulas. Graphs and Mathematical Tables*, National Bureau of Standards, Washington D. C. 1964.

[5] Robert Von Helmholtz, Philosophical Magazine, xliv (1872) p. 530 (per Whittaker)

[6] Robert Von Helmholtz, Journal Fur Math, lxxv (1873), p. 35 (per Whittaker)