Accurate Formulas for $\vec{A}$ and $\vec{B}$ due to Circular Current Loops with Code

Kurt Nalty

September 17, 2012

Geometry

Assume a current loop in the XY plane at $Z = 0$ of radius R carrying a current I. The observation point (ρ, ϕ, z) is in cylindrical coordinates, with $\rho = \sqrt{x^2 + y^2}$, and z being the height above the coil.

$\vec{A}$ Field

$$\begin{aligned} \vec{A}(\rho,\phi,z) &= \vec{a}_\phi \frac{\mu I R}{4\pi} \oint \frac{\cos\theta d\theta}{\sqrt{z^2+\rho^2+R^2-2R\rho\cos\theta}} \\ A_\phi &= \frac{\mu I R\sqrt{a+b}}{\pi b}\left[\left(1-\frac{k^2}{2}\right)K(k)-E(k)\right] \end{aligned}$$

where K and E are complete elliptic integrals, and

$$\begin{aligned} a &= z^2+\rho^2+R^2 \\ b &= 2R\rho \\ k &= \sqrt{\frac{2b}{a+b}} = \sqrt{\frac{4R\rho}{z^2+(R+\rho)^2}} \end{aligned}$$

$\vec{B}$ Field

$$\begin{aligned} B_r &= -\frac{2\mu I z}{4\pi\rho\sqrt{z^2+(R+\rho)^2}}\left(K(k)-E(k)\frac{R^2+\rho^2+z^2}{(R-\rho)^2+z^2}\right) \\ B_\phi &= 0 \\ B_z &= \frac{2\mu I}{4\pi\sqrt{z^2+(R+\rho)^2}}\left(K(k)+E(k)\frac{R^2-\rho^2-z^2}{(R-\rho)^2+z^2}\right) \end{aligned}$$

C Program for Field Calculation

The following program is at `http://www.kurtnalty.com/B_and_A.c` as of July 5, 2011.

The elliptic integrals are calculated using the algorithms in AMS 55, Ch 19. The calling conventions for the complete integrals are based upon the excellent routines at `http://www.mymathlib.webtrellis.net/functions/elliptic_integrals.html`.

```
/*
    B_and_A.c is a little demo program in C illustrating the use
    of complete elliptic integrals to evaluate magnetic fields and
    potentials.
```

```
    This program is totally freeware, enjoy.

    Compiling using GCC -
        gcc B_and_A.c -o B_and_A -lm
        ./B_and_A
*/

#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 that we  can return multiple
     values without a struct.

       Example Call: err = 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 = 1.0;

    for (i=1;i<9;i++) {  // max interation depth is 10.
                         // Break 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,dA,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;

    double theta, dtheta, phi, a, r, r2, x, y, current,scale;
    double pi = 3.141592653589793;
        double k, rho, z, A_theta, B_rho, B_x, B_y, B_z, I, E, K;
    double A_x, A_y, A_z;
    double C,Alpha, Beta, Gamma;

    FILE* OutputWireFrame;
    FILE* EFile;
    FILE* KFile;

// First, we provide a table of value for E(k) and K(k)
// for comparison against references.

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



// 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);

// Create a wireframe model for the current source,
// and indicate the sensor position

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

    // 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);

// First calculate B and A by piecewise integration for reference

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

    for (i=0;i<360;i++) {
        // dA = 1.0e-7*(SourceCurrent[i])/(r);
        // 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));
        dA = ScaleVector(SourceCurrent[i],(current*1.0e-7*a*dtheta/r));
        dB = ScaleVector(Cross(SourceCurrent[i],a_R),
             (current*1.0e-7*a*dtheta/r2)); // u_0/(4 pi)
        A = AddVector(A,dA);
        B = AddVector(B,dB);
    }

// print A for comparison to integral calculation
    printf(
"A calculated by summing 360 one degree segments (Cartesian format)\n");
    printf("\nA = "); PrintVector(A); printf("\n\n");

// print B for comparison to integral calculation
    printf(
"B calculated by summing 360 one degree segments (Cartesian format)\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 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 source current radius

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

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

    A_theta = ((4.0e-7*I*a*sqrt(z*z + rho*rho + a*a + 2.0*a*rho))
                           /(2.0*a*rho))*( (1.0 - 0.5*k*k)*K - E  );
    printf("(Numerical Summation Method) A.theta = %g \n",
                         sqrt(A.x*A.x + A.y*A.y));
    printf("(Elliptic Integral Method)   A.theta = %g  \n",A_theta);
    A_x = -A_theta*y/sqrt(x*x + y*y);
    A_y =  A_theta*x/sqrt(x*x + y*y);
    A_z =  0.0;
    printf(
 "(Cyl to Cartesian of Above)  A_x = %g   A_y = %g   A_z = %g  \n\n"
             ,A_x, A_y, A_z);


    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);


    printf("\n");
}

/************* Results ****************************

A calculated by summing 360 one degree segments (Cartesian format)

A = ( -4.100909e-08, 2.367661e-08, 0.000000e+00 )

B calculated by summing 360 one degree segments (Cartesian format)
```

```
B = ( 2.564194e-08, 4.441314e-08, 2.956286e-08 )

(Numerical Summation Method) A.theta = 4.73532e-08
(Elliptic Integral Method)   A.theta = 4.73532e-08
(Cyl to Cartesian of Above)  A_x = -4.10091e-08   A_y = 2.36766e-08   A_z = 0

(Numerical Summation Method) B.rho = 5.12839e-08   B.z = 2.95629e-08
(Elliptic Integral Method)   B_rho = 5.12839e-08   B_z = 2.95629e-08
(Cyl to Cartesian of Above)  B_x = 2.56419e-08   B_y = 4.44131e-08   B_z = 2.95629e-08

*/
```

Bibliography

[1] J. C. Maxwell, *Electricity and Magnetism, Vol II, Article 701* Dover Reprint 1954. 1873

[2] Edward B. Rosa and Louis Cohen, "Formulae and Tables for the Calculation of Mutual and Self-Inductance", Bulletin of the Bureau of Standards, Vol 5, No. 1, pp6, August, 1908.

[3] Alexander Russell, "The Magnetic Field and Inductance Coefficients of Circular, Cylindrical and Helical Currents", Procedings of the Royal Academy of Science, Vol 20, 1906. p 476

[4] William H. Beyer, *CRC Standard Math Tables, 27th Edition*, CRC Press, Boca Raton Florida, 1984.

[5] I. Gradshteyn and I. Ryzhik, *Table of Integrals, Series and Products*, Academic Press, New York, 1980

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

[7] Frederick W. Grover, *Inductance Calculations Working Formulas and Tables*, Instrument Society of America, Research Triangle Park, N. C., 1973

[8] George Arfken, *Mathematical Methods for Physicists*, Third Edition, Academic Press, Orlando Florida, 1985 pp 321-326.

[9] Arthur Erdelyi, *Higher Transcendal Functions* , Krieger Publishing, Malabar Florida, 1985 Ch. 13 in Vol II