Table of Contents


```
% EP501 - HW4 - Paul Yuska - 11/18/19

clc
clear
close all

% fprintf('===== Problem 1
 =============================================\n')

% define constants and grid sizing
I = 10; % amps
mu_0 = pi*4e-7; % Henrys/meter
a = 0.005; % meters

gridLim = 3*a; % for both x and y

% resolution; needs to be odd in order for linspace to generate a
 point
% exactly in the middle of the range.
nPts = 150;
fprintf('Number of simulation points: %d\n',nPts)
% midpoint = (nPts + 1)/2; % index of midpoint of any array
% midpoint = nPts/2; % index of midpoint of any array

% define grids
xgrid = linspace(-gridLim,gridLim,nPts);
ygrid = linspace(-gridLim,gridLim,nPts);

[X,Y] = meshgrid(xgrid,ygrid);

% initialize arrays for x and y components of B vector
Bx = zeros(nPts);
By = zeros(nPts);
Bmag = zeros(nPts);
```

Number of simulation points: 150

1a, 1b. Plot the vector magnetic field

```
for i = 1:nPts
    for j = 1:nPts
        mag = sqrt(xgrid(i)^2 + ygrid(j)^2); % magnitude of position
        coef = [mu_0*I/2/pi/mag;    ...
                mu_0*I/(2*pi*a^2)*mag]; % piecewise function
 coefficients
        if mag >= a
            Bx(i,j) = -coef(1)*ygrid(j)/mag;
            By(i,j) = coef(1)*xgrid(i)/mag;
        else % magnitude less than a
            Bx(i,j) = -coef(2)*ygrid(j)/mag;
            By(i,j) = coef(2)*xgrid(i)/mag;
        end % if - mag
        Bmag(i,j) = sqrt(Bx(i,j)^2 + By(i,j)^2);
    end % for - j
end % for - i
% have to add some transpose operators because of internal MATLAB...
 idiocy
% i kept getting x and y flipped and a quiver plot that wasn't even a
% circulation... god that was frustrating to figure out
```

1c. Compute the curl of del x B numerically

```
%{
del x F = (dFz/dy - dFy/dz)i + (dFx/dz - dFz/dx)j + (dFy/dx - dFx/dy)k

no z terms in F, so all partial derivs WRT z are zero, only non-zero
component of curl is z-component. partial derivatives of a function of
 2
variables are identical to a normal derivative of a function of only 1
variable. i.e. ?F(x,y)/?x = dF(x)/dx. there are no mixed partials, so
 we
can use standard centered difference formulas derived from expansion
 of
Taylor series. we can also assume equally spaced grids.
%}

dBydx = zeros(nPts); % partial WRT x
dBxdy = zeros(nPts); % partial WRT y

% set up boundaries
% standard 1st-order fwd/bwd derivatives: f' = (f1-f0)/dx
for i = 1:nPts
    for j = 1:nPts
        if i == 1 % upper boundary
            % j is column index, which represents a place along the x-
axis
            % vice versa for i
            dBxdy(i,j) = (Bx(i+1,j) - Bx(i,j))/(ygrid(end)-
ygrid(end-1));
```

```
        elseif j == 1 % left boundary
            dBydx(i,j) = (By(i,j) - By(i,j+1))/(xgrid(j+1)-xgrid(j));
        elseif i == nPts % lower boundary
            dBxdy(i,j) = (Bx(i-1,j) - Bx(i,j))/(ygrid(i-1)-ygrid(i));
        elseif j == nPts % right boundary
            dBydx(i,j) = (By(i,j-1) - By(i,j))/(xgrid(end-1)-
xgrid(end));
        end % if - i
    end % for - j
end % for - i

% compute interior values
for i = 2:nPts-1
    for j = 2:nPts-1
        % centered-difference: df/dx_i = (f_i+1 - f_i-1)/(2*dx)
        dBxdy(i,j) = (Bx(i,j+1) - Bx(i,j-1))/2/(ygrid(i+1) -
 ygrid(i));
        dBydx(i,j) = (By(i+1,j) - By(i-1,j))/2/(xgrid(j+1) -
 xgrid(j));
    end % for - j
end % for - i

% z-component of curl

% something is funky with the indices in my code. the curl is zero
 outside
% of the circle with radius a if the transpose operator is not
 present, but
% with the transpose, it matches the analytical solution very well.
 will
% come back to this if i have time.
numCurlB = dBydx-dBxdy';
```

1d. Compute the curl of B analytically

```
% del x B = mu_0*I/2/pi (2/a^2)k        for mag < a
% del x B = mu_0*I/2/pi [(y^2 - x^2)/(x^2 + y^2)^2]k for mag >= a

curlB = zeros(nPts);
%{
dBxdy_exact = zeros(nPts);
dBydx_exact = zeros(nPts);
%}
curlCoef = mu_0*I/2/pi;

for i = 1:nPts
    for j = 1:nPts
        mag = sqrt(xgrid(i)^2 + ygrid(j)^2);

        if mag < a
            curlB(i,j) = curlCoef*2/a^2;
            %{
            dBydx_exact(i,j) = curlCoef/a^2*xgrid(j);
```

```
                dBxdy_exact(i,j) = curlCoef/a^2*ygrid(i);
                %}
            else
                curlB(i,j) = 2*curlCoef*(ygrid(j)^2 - xgrid(i)^2)/...
                    (xgrid(i)^2 + ygrid(j)^2)^2;
                %{
                dBxdy_exact(i,j) = curlCoef*(xgrid(j)^2 - ygrid(i)^2)/...
                    (xgrid(i)^2 + ygrid(j)^2)^2;
                dBydx_exact(i,j) = curlCoef*(ygrid(j)^2 - xgrid(i)^2)/...
                    (xgrid(i)^2 + ygrid(j)^2)^2;
                %}
            end % if - mag
        end % for - j
    end % for - i
```

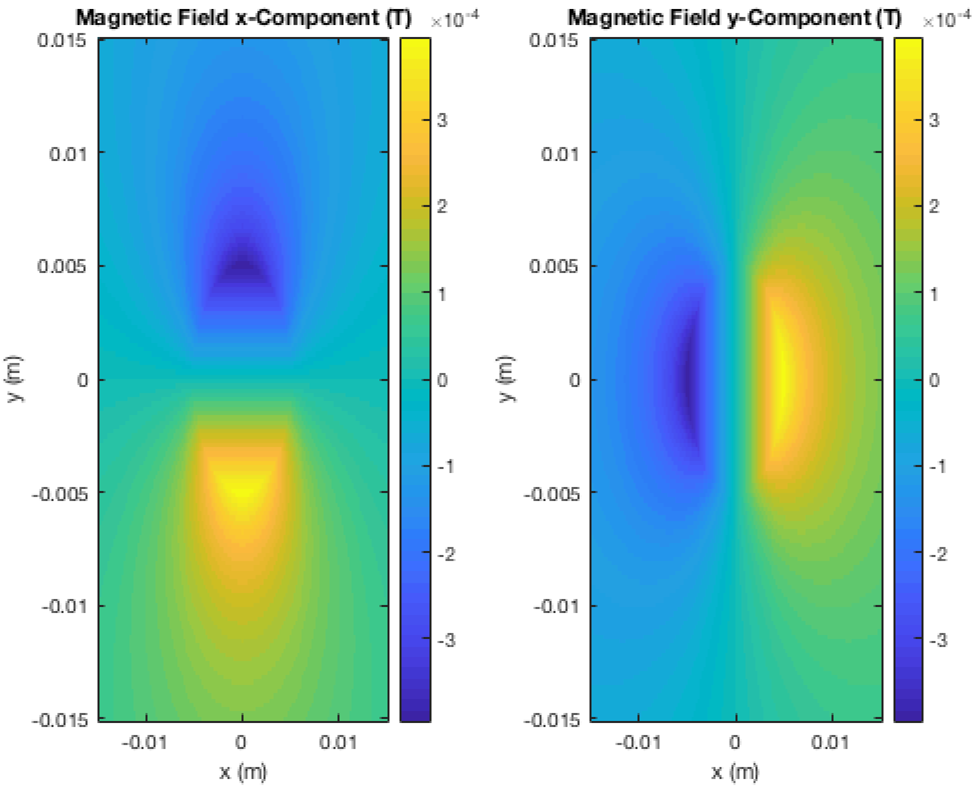
PLOT

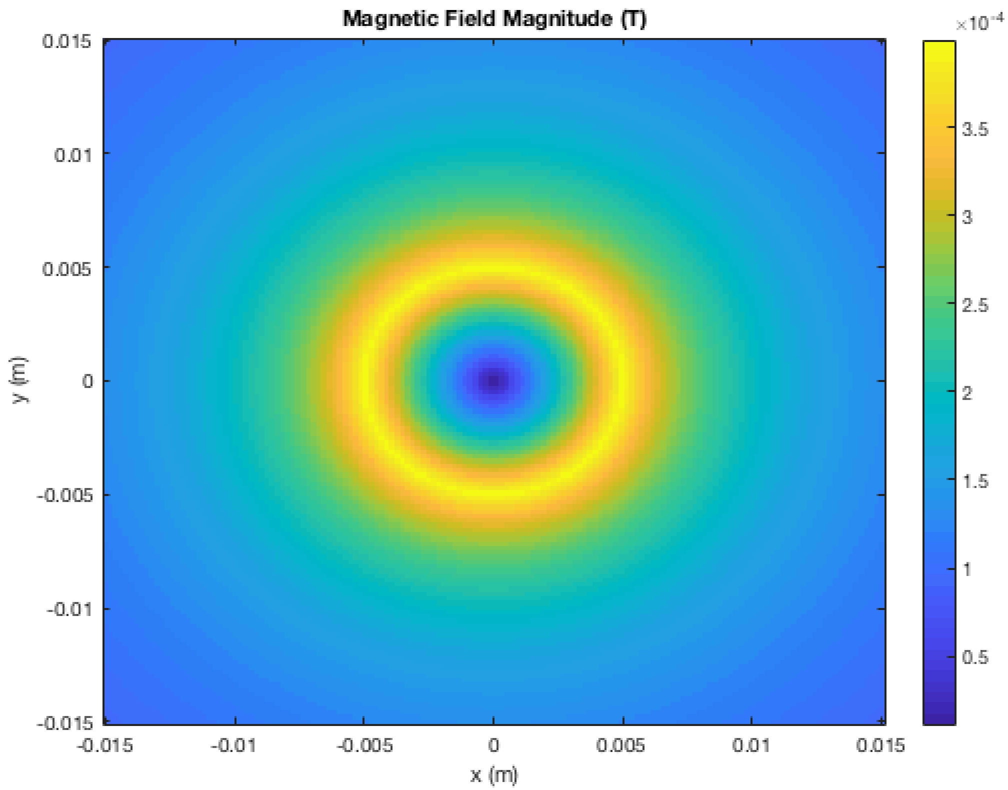
plot mag field components, magnitude, and vector field

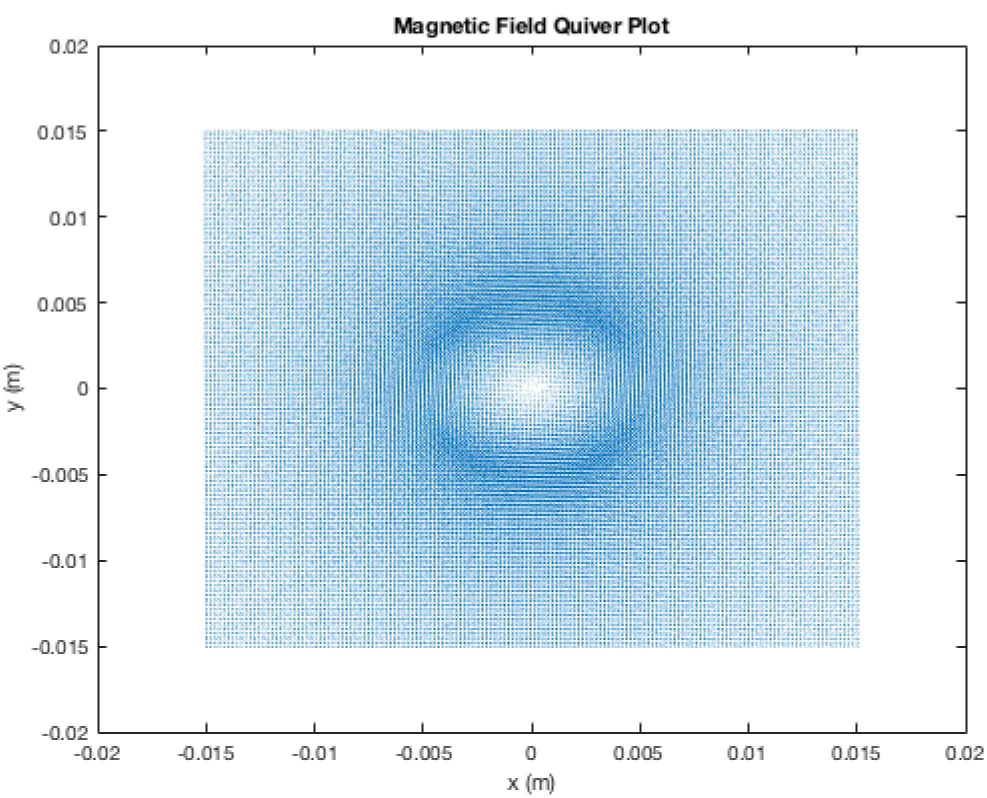
```
close all

figure
subplot(1,2,1)
imagesc(xgrid,ygrid,Bx')
title('Magnetic Field x-Component (T)')
xlabel('x (m)')
ylabel('y (m)')
colorbar
axis xy

subplot(1,2,2)
imagesc(xgrid,ygrid,By')
title('Magnetic Field y-Component (T)')
xlabel('x (m)')
ylabel('y (m)')
colorbar
axis xy

figure
imagesc(xgrid,ygrid,Bmag')
title('Magnetic Field Magnitude (T)')
xlabel('x (m)')
ylabel('y (m)')
colorbar
axis xy

% quiver plot
figure
quiver(xgrid,ygrid,Bx',By')
title('Magnetic Field Quiver Plot')
xlabel('x (m)')
ylabel('y (m)')
```

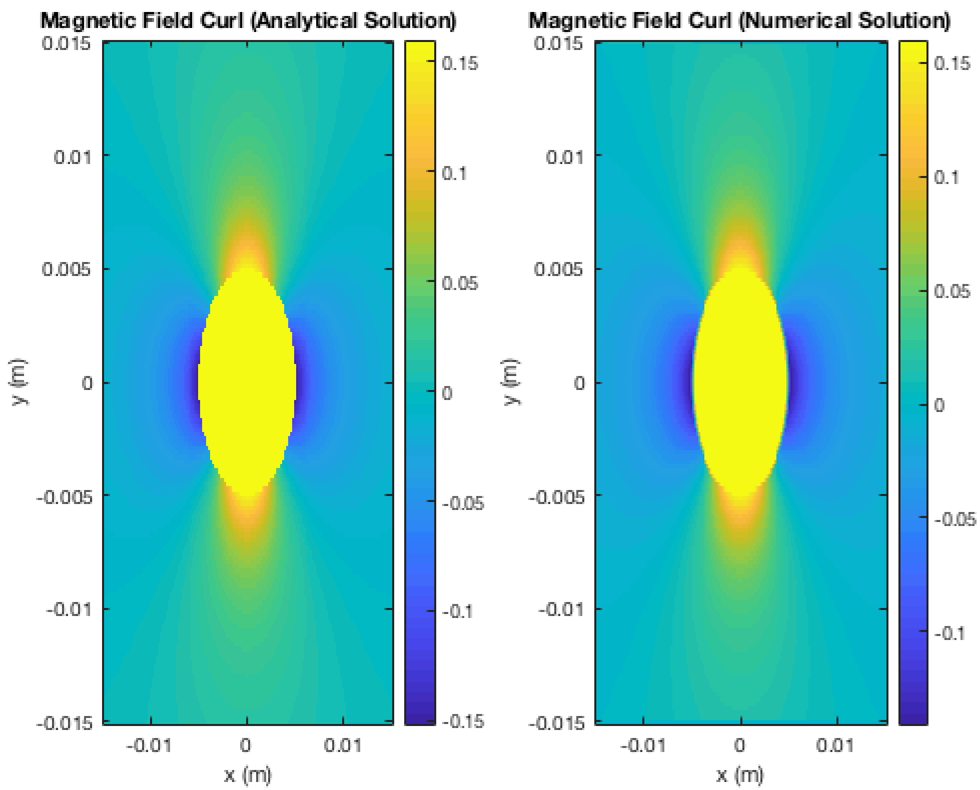
```
% plot numerical curl solution components (debug)
%{
figure
subplot(1,2,1)
imagesc(xgrid,ygrid,dBydx')
title('Magnetic Field Curl (dB_y/dx)')
xlabel('x (m)')
ylabel('y (m)')
colorbar
caxis([-0.08 0.08])
axis xy

subplot(1,2,2)
imagesc(xgrid,ygrid,dBxdy)
title('Magnetic Field Curl (dB_x/dy)')
xlabel('x (m)')
ylabel('y (m)')
colorbar
caxis([-0.08 0.08])
axis xy

debug plots
subplot(2,2,3)
imagesc(xgrid,ygrid,dBxdy_exact')
title('Magnetic Field Curl (dB_x/dy) Exact')
xlabel('x (m)')
ylabel('y (m)')
colorbar
axis xy

subplot(2,2,4)
imagesc(xgrid,ygrid,dBxdy_exact')
title('Magnetic Field Curl (dB_x/dy) Exact')
xlabel('x (m)')
ylabel('y (m)')
colorbar
axis xy
%}

% plot analytical and numerical curl solutions, compare
figure
subplot(1,2,1)
imagesc(xgrid,ygrid,curlB')
title('Magnetic Field Curl (Analytical Solution)')
xlabel('x (m)')
ylabel('y (m)')
colorbar
% caxis([-1e-3 1e-3])
axis xy

subplot(1,2,2)
imagesc(xgrid,ygrid,numCurlB')
title('Magnetic Field Curl (Numerical Solution)')
xlabel('x (m)')
```

```
ylabel('y (m)')
colorbar
% caxis([-1e-3 1e-3])
axis xy
```




Magnetic Field Magnitude (T)
×10^-4
3.5
3
2.5
2
1.5
1
0.5
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
y (m)
-0.015
-0.01
-0.005
0
0.005
0.01
0.015
x (m)


Magnetic Field Quiver Plot
0.02
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
-0.02
y (m)
-0.02
-0.015
-0.01
-0.005
0
0.005
0.01
0.015
0.02
x (m)



1e. Compute and plot the scalar field Phi

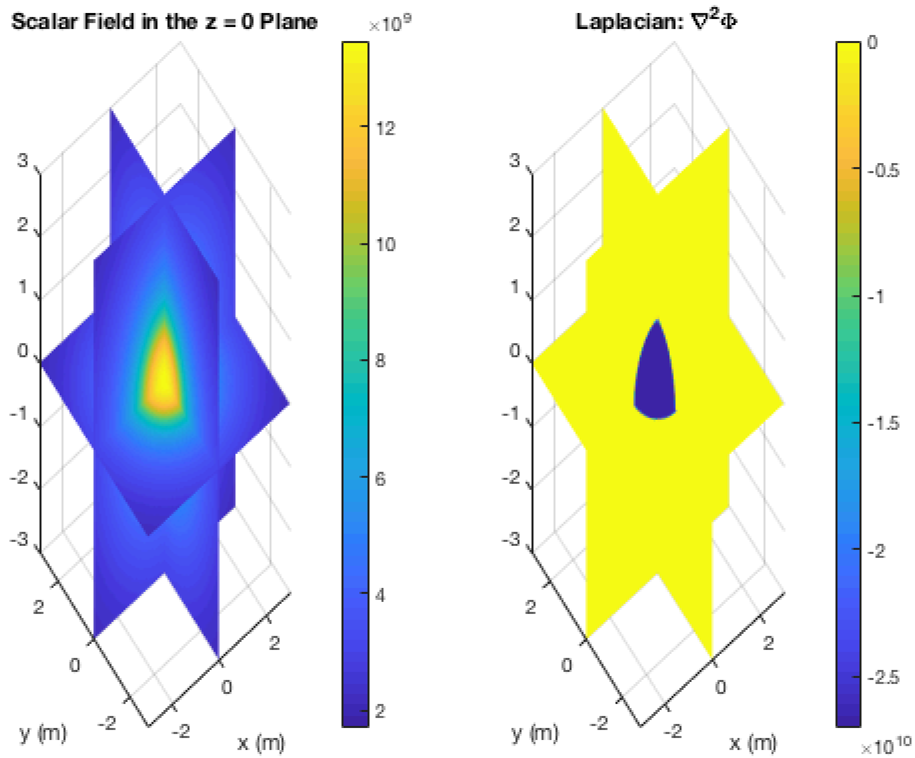
```
% define constants and allocate memory
Q = 1; % charge, Coulombs
a_scal = 1; % characteristic dimension, m
eps_0 = 8.854e-12; % permittivity of free space, F/m

phi = zeros(nPts,nPts,nPts);

scalgridlim = 3*a_scal;
scalxgrid = linspace(-scalgridlim,scalgridlim,nPts);
scalygrid = linspace(-scalgridlim,scalgridlim,nPts);
scalzgrid = linspace(-scalgridlim,scalgridlim,nPts);

[X2,Y2,Z2] = meshgrid(scalxgrid,scalygrid,scalzgrid);

for i = 1:nPts
    for j = 1:nPts
        for k = 1:nPts
            mag = sqrt(scalxgrid(i)^2 + scalygrid(j)^2 +
 scalzgrid(k)^2);

            if mag < a_scal
                phi(i,j,k) = Q/4/pi/eps_0/a_scal - (Q/8/pi/eps_0/
a_scal^3)...
```

```
                     * (scalxgrid(i)^2 + scalygrid(j)^2 +
 scalzgrid(k)^2 - a_scal^2);
            else
                phi(i,j,k) = Q/4/pi/eps_0/mag;
            end % if - a
        end % for - k
    end % for - j
end % for - i
```

1f. Numerically compute the Laplacian of Phi

```
% Laplacian in 3 dimensions is d^2 f/dx^2 + d^2 f/dy^2 + d^2 f/dz^2,
 so
% implement a 2nd-order accurate centered-difference formula given in
 the
% book by eqn 5.77

LaplPhi = zeros(nPts,nPts,nPts); % 3-dimensional array

% quick and dirty calculation of dx. we can work with equally spaced
 grids,
% so no need to make things complicated.
dx = scalxgrid(2)-scalxgrid(1);

% % interior
for i = 2:nPts-1
    for j = 2:nPts-1
        for k = 2:nPts-1
            LaplPhi(i,j,k) = (phi(i,j+1,k)-2*phi(i,j,k)+phi(i,j-1,k))/
dx^2 ...
                + (phi(i+1,j,k)-2*phi(i,j,k)+phi(i-1,j,k))/dx^2 ...
                + (phi(i,j,k+1)-2*phi(i,j,k)+phi(i,j,k-1))/dx^2;
        end % for - k
    end % for - j
end % for - i

figure

subplot(1,2,1)
s1=slice(X2,Y2,Z2,phi,0,0,0);
title('Scalar Field in the z = 0 Plane')
xlabel('x (m)')
ylabel('y (m)')
colorbar
% caxis([1e9 3e10])
s1(1).EdgeColor = 'none';
s1(2).EdgeColor = 'none';
s1(3).EdgeColor = 'none';
axis xy

subplot(1,2,2)
s2=slice(X2,Y2,Z2,LaplPhi,0,0,0);
title('Laplacian: \nabla^2\Phi')
```

```
xlabel('x (m)')
ylabel('y (m)')
colorbar
% caxis([-2.7e10 -2.5e10])
s2(1).EdgeColor = 'none';
s2(2).EdgeColor = 'none';
s2(3).EdgeColor = 'none';
axis xy

% fprintf('===== Problem 2
 ==============================================\n')
```



2a. Numerically compute the electrostatic energy in R

```
% have 3-dim. array in x,y,z (phi from 2e). hold y,z constant and
 evaluate
% integrals in x for several (constant) values of y. store this in a
 2D
% array (y and z). repeat the procedure for this array, holding z
 constant
% and integrating y for several values of z. store this in a vector.
% finally, integrate this vector.

yval = zeros(nPts); % 2D array for first round of integration results
```

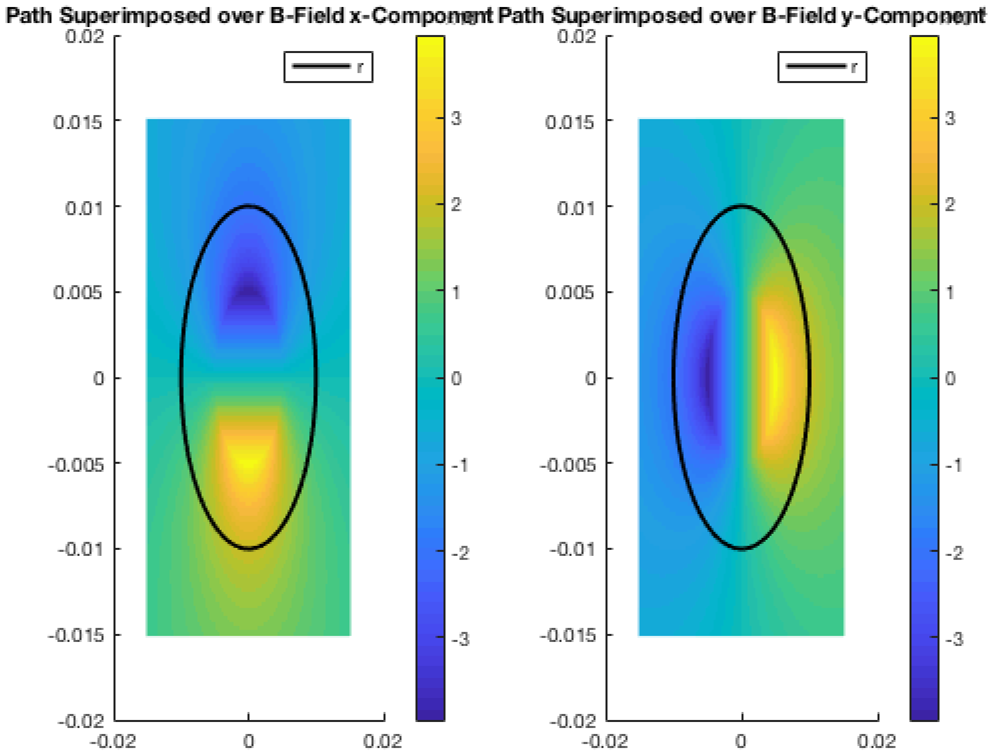
```
zval = zeros(nPts,1); % vector for second round of integration results
sumx = 0; %
sumy = 0;
sumz = 0;

integ = phi.*LaplPhi; % integrand

% build 2D array of integral summations along X for const Y,Z
for k = 1:nPts-1
    for j = 1:nPts-1
        for i = 1:nPts-1
            sumx = sumx + 0.5*dx*(integ(i,j,k) + integ(i+1,j,k));
        end % for - i
        yval(j,k) = sumx; % store the result of the integral in row j
        sumx = 0; % reset for next iteration
    end % for - j
end % for - k

% build vector of integral summations of rows of yval for const Z
for k = 1:nPts-1
    for j = 1:nPts-1
        sumy = sumy + 0.5*dx*(yval(j,k) + yval(j+1,k));
    end % for - j
    zval(k) = sumy; % store result of integral in index k
    sumy = 0; % reset for next iteration
end % for - k

for k = 1:nPts-1
    sumz = sumz + 0.5*dx*(zval(k) + zval(k+1));
end % for - k

W_E = -0.5*eps_0*sumz;

fprintf('The electrostatic energy in the region defined by R is: %.4g
 J\n',W_E)
```

The electrostatic energy in the region defined by R is: 5.392e+09 J

2b. Compute and plot a parametric path

```
% define grids and constants
r_0 = 2*a;
phiGrid = linspace(0,2*pi,nPts);

xPara = r_0*cos(phiGrid);
yPara = r_0*sin(phiGrid);

figure
subplot(1,2,1)
title('Path Superimposed over B-Field x-Component')
hold on
imagesc(xgrid,ygrid,Bx')
plot(xPara,yPara,'k','LineWidth',2)
```

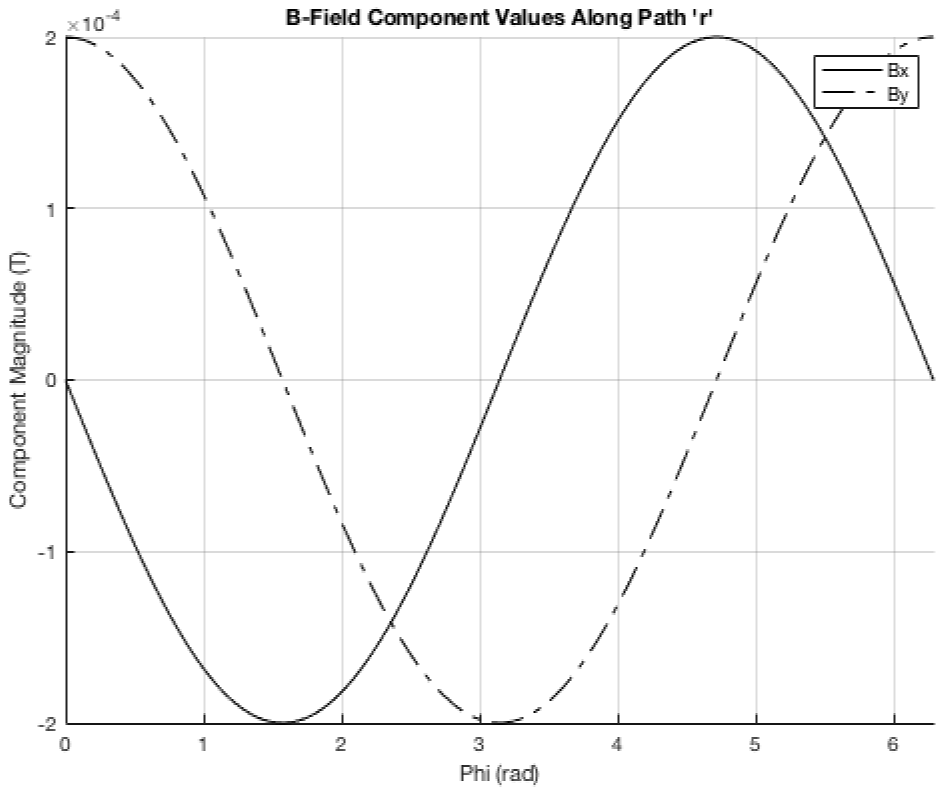
```
legend('r')
hold off
colorbar

subplot(1,2,2)
title('Path Superimposed over B-Field y-Component')
hold on
imagesc(xgrid,ygrid,By')
plot(xPara,yPara,'k','LineWidth',2);
legend('r')
hold off
colorbar
```



2c. Plot values of B along 'r'

```
% calculate Bx and By specifically at the coordinates given by xPara
 and
% yPara, defined in the previous part. use phi as the independent
 variable.
% magnitude is constant and equal to a.

% allocate
Bx_r = zeros(nPts,1);
By_r = zeros(nPts,1);

% define B-field components along path
```

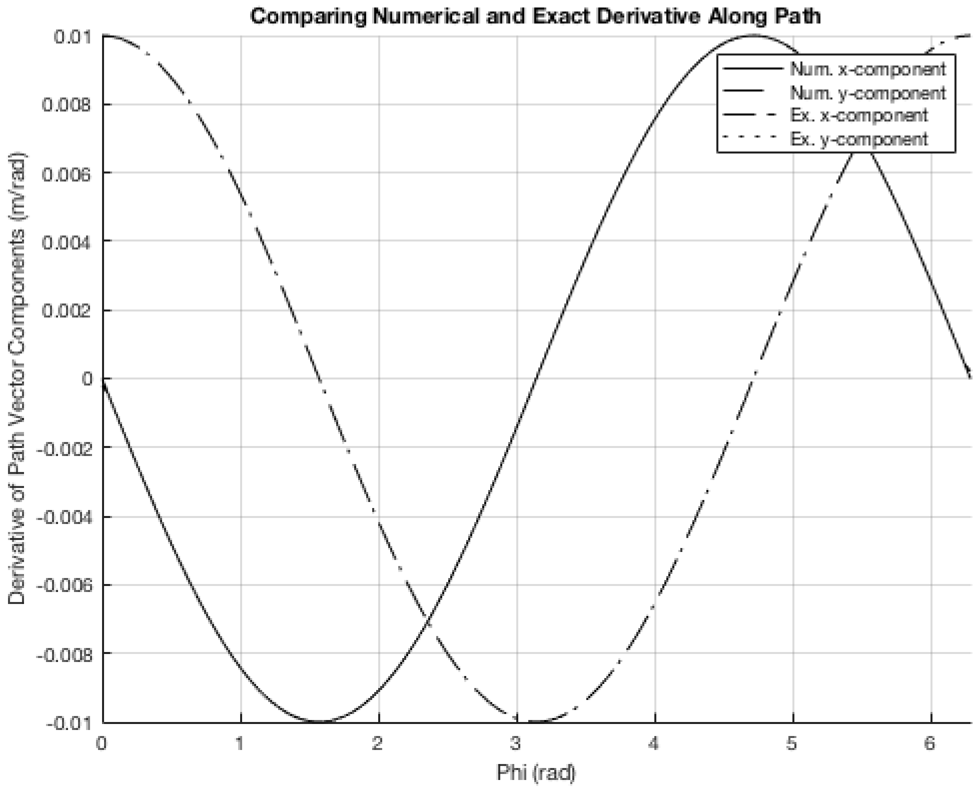
```
for p = 1:nPts
    Bx_r(p) = -mu_0*I/(2*pi)*yPara(p)/(xPara(p)^2 + yPara(p)^2);
    By_r(p) = mu_0*I/(2*pi)*xPara(p)/(xPara(p)^2 + yPara(p)^2);
end

figure
hold on
plot(phiGrid,Bx_r,'-k')
plot(phiGrid,By_r,'-.k')
hold off
title('B-Field Component Values Along Path ''r''')
legend('Bx','By')
xlabel('Phi (rad)')
ylabel('Component Magnitude (T)')
grid on
xlim([0 2*pi])

% DEBUG: plotting B-field vectors at points on circle
% figure
% hold on
% plot(xPara,yPara,'o')
% quiver(xPara,yPara,Bx_r',By_r',0.25)
% hold off
% grid on
```



2d. Numerically compute the tangent vector / derivative along 'r'

```
% r = r_0*cos(phi)i + r_0*sin(phi)j
% dr/d(phi) = r_0*(d(cos(phi)/d(phi)i + d(sin(phi)/d(phi)j)

% dr_x,i + dr_y,i is the tangent vector at point x,y
dr_x = zeros(nPts,1);
dr_y = zeros(nPts,1);
dphi = phiGrid(2)-phiGrid(1);

for p = 1:nPts
    if p == 1
        % first point, 1st-order derivative
        dr_x(p) = (xPara(p+1)-xPara(p))/dphi;
        dr_y(p) = (yPara(p+1)-yPara(p))/dphi;
    elseif p == nPts
        % last point, 1st-order derivative
        dr_x(end) = (xPara(end)-xPara(end-1))/dphi;
        dr_y(end) = (yPara(end)-yPara(end-1))/dphi;
    else
        % interior, centered difference
        dr_x(p) = (xPara(p+1)-xPara(p-1))/2/dphi;
        dr_y(p) = (yPara(p+1)-yPara(p-1))/2/dphi;
    end % if - p
end % for - p

% analytical derivatives
dr_xa = -r_0*sin(phiGrid);
dr_ya = r_0*cos(phiGrid);

figure
hold on
plot(phiGrid,dr_x,'-k')
plot(phiGrid,dr_y,'--k')
plot(phiGrid,dr_xa,'-.k')
plot(phiGrid,dr_ya,':k')
hold off
legend('Num. x-component','Num. y-component','Ex. x-component','Ex. y-
component')
title('Comparing Numerical and Exact Derivative Along Path')
xlabel('Phi (rad)')
ylabel('Derivative of Path Vector Components (m/rad)')
xlim([0 2*pi])
grid on

% DEBUG: plotting tangent vectors at points on circle
% figure
% hold on
% plot(xPara,yPara,'o')
% quiver(xPara,yPara,dr_x',dr_y',0.25)
% hold off
```

```
% grid on
```



2e. Integrate the B-field along the path

```
% calculated dl in part 2d, so just do dot product along path, and
% integrate. we know the equation of the path, and can analytically
 find
% the derivative, so no point in using the poorer-quality numerical
% approximation of the derivative

% create matrices to use with MATLAB dot() fn
% 1st row is x-components, 2nd row is y-components
B = vertcat(Bx_r',By_r');
dl = vertcat(dr_x',dr_y');
Bdl = dot(B,dl);
sumB = 0;

% integrate over phi using trapezoidal method
for p = 1:nPts-1
    sumB = sumB + 0.5*dphi*(Bdl(p)+Bdl(p+1));
end % for - p

sumB = sumB/mu_0;

fprintf('The line integral along the circle in B is: %.4f amps
\n',sumB)
```

```
The line integral along the circle in B is: 9.9970 amps
```

Published with MATLAB® R2018b