Isogeometric Analysis

Wintersemster 15/16

Report on HomeWork 1

Author:
Kumar Saurabh

1 Problem Description

The objective is to evaluate the following integral

$$I = \frac{1}{4\pi}\int_{C_1}\int_{C_2} \frac{(d\mathbf{C_1} \times d\mathbf{C_2}).(\mathbf{C_2} - \mathbf{C_1})}{||\mathbf{C_2} - \mathbf{C_1}||^3} \tag{1}$$

$$I = \frac{1}{4\pi}\int_{0}^{1}\int_{0}^{1} \frac{(d\mathbf{C_1}(\xi) \times d\mathbf{C_2}(\eta)).(\mathbf{C_2}(\eta) - \mathbf{C_1}(\xi))}{||\mathbf{C_2}(\eta) - \mathbf{C_1}(\xi)||^3} d\xi d\eta \tag{2}$$

where the curves $\mathbf{C_1}, \mathbf{C_2}$ represent the circle and the helix respectively, as illustrated in the Figure (1).

2 Steps Involved

- In the first step, the NURBS representation of the helix curve is coded in the Matlab File `generate_helix.m` (Code 2), which uses an open knot vector. A Matlab File `generate_circle.m` was used to create the circle.
- A File named `draw_fig.m` (Code 4) visualizes the geometry is created.
- The routine `quad_rule.m` from Exercise 4 is used to integrate Equation (2) by splitting each integral into integrals over knot spans in the routine `integrate_hw01` (Code 3).
- For debugging purposes the length of the curves is calculated to verify the integral.
- The effect of deformation or translation on the integral was investigated using the provided file `nurb_knot_refinement.m` to generate finer versions of curves, which leads to more accurate results.
- A main program `drive.m` (Code 1) is used to connect all the codes and run the program.

3 Visualization

Figure (1) shows the curves (splines) generated for the circle and the circular helix, with circle radius as 5, torus radius 5 and number of curls in circular helix as 17.

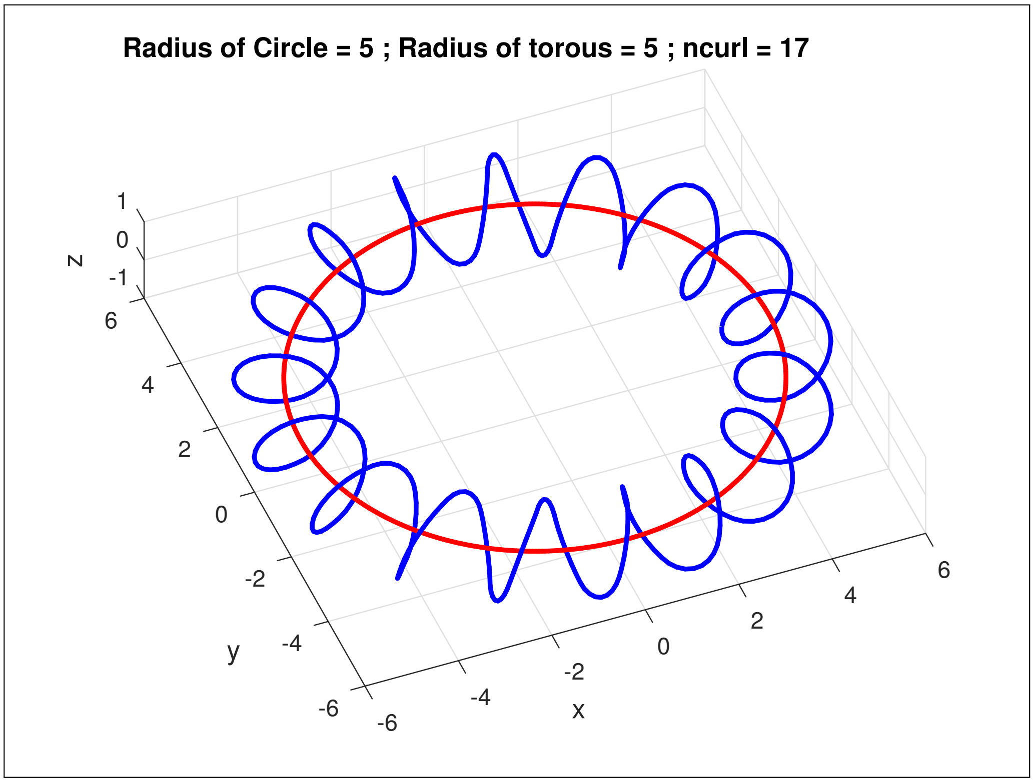

Figure 1: Circle and Helix

4 Significance of the Integral

This integral has a role in the field of electromagnetics. From the Biot Savart law

$$\overrightarrow{\mathbf{dB}} = \int \frac{\mu_0 I \overrightarrow{\mathbf{dl}} \times \overrightarrow{r}}{\| \overrightarrow{r}^3 \|} \tag{3}$$

where:

- dB represents the magnetic field
- $\mu_0 = 4\pi \times 10^{-7} H/m$ represents permeability of free space
- I is the current
- $\overrightarrow{r}$ position vector
- $\overrightarrow{dl}$ length element along the curve

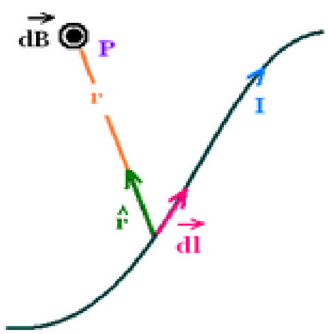

Figure 2: Biot Savart Law

From Ampere's law, we know that:

$$\oint B \overrightarrow{\mathbf{dl}} = \mu_0 I_{encl} \tag{4}$$

where I_{encl} represents the current enclosed by the closed path

Now consider a circular ring at a distance 'r' inside the torus region:

$$\begin{aligned}
\oint B\overrightarrow{\mathbf{dl_h}} &= \mu_0 NI &(5)\\
\frac{1}{\mu_0 I}\oint B\overrightarrow{\mathbf{dl_h}} &= N\\
\frac{1}{\mu_0 I}\oint\int \frac{\mu_0 i\overrightarrow{\mathbf{dl_c}}\times\overrightarrow{r}}{4\pi\|\overrightarrow{r}\|^3}.\overrightarrow{\mathbf{dl_h}} &= N\\
\frac{1}{4\pi}\int\int \frac{(\overrightarrow{\mathbf{dl_c}}\times\overrightarrow{\mathbf{dl_h}}).\overrightarrow{r}}{\|\overrightarrow{r}\|^3} &= N\\
\frac{1}{4\pi}\int_{C_1}\int_{C_2} \frac{(d\mathbf{C_1}\times d\mathbf{C_2}).(\mathbf{C_1}-\mathbf{C_2})}{\|\mathbf{C_1}-\mathbf{C_2}\|^3} &= N\\
\frac{1}{4\pi}\int_{C_1}\int_{C_2} \frac{(d\mathbf{C_1}\times d\mathbf{C_2}).(\mathbf{C_2}-\mathbf{C_1})}{\|\mathbf{C_2}-\mathbf{C_1}\|^3} &= -N &(6)
\end{aligned}$$

where:

- N represents the number of loops in the toroid.
- C_1 represents the circle.
- C_2 represents the helix.

5 Computation of the Integral

- The function `quad_rule.m` from Exercise 4 is used to integrate Equation (2) by splitting each integral into integrals over knot spans in the function `integrate_hw01.m` (Code 3).
- The integral is computed using the contributions from all the knotspans (for circle and helix).
- Consequently, the value of integral in (2) comes out to be -(number of curls in the helix) $\approx$ -17 (as seen in Figure (1)).

The above mentioned steps can be clearly seen in the codes attached below.

Code 1: drive.m

```
clc;
clear;
close all;

%% Adding paths
addpath('Exercise1');
addpath('Exercise2');
addpath('Exercise3');
%% Generation of Helix
tic;
torous_radius = 5;
ncurls = 17;
helix = generate_helix(torous_radius,ncurls,0);
draw_fig(helix,500);
hold on;
%% Generate Circle
circle_radius = 5;
circle = generate_circle(circle_radius,0);
draw_fig(circle,500);
toc;
title(['Radius of Circle = ',num2str(circle_radius) ' ; ...
    Radius of torous = ',...
     num2str(torous_radius) ' ; ncurl = ',num2str(ncurls)]);
%% Integration
val_integral = integrate_hw01(circle,helix);
fprintf('Value of the integral is %f\n',val_integral);
```

Code 2: Generation of helix

```
function nurb = generate_helix(torusradius, ncurl, refine)

nurb.number = [ 9 ];
nurb.order = [ 3 ];
p = nurb.order - 1;
nurb.knots{1} = [ 0 0 0 0.25 0.25 0.5 0.5 0.75 0.75 1 1 1];
nurb.coeffs = zeros(4,nurb.number);
s2 = 1.0/sqrt(2);
nurb.coeffs(1,1:nurb.number) = [-1 -s2 0 s2 1  s2  0 -s2 -1];
nurb.coeffs(3,1:nurb.number) = [ 0  s2 1 s2 0 -s2 -1 -s2  0];
nurb.coeffs(4,1:nurb.number) = [ 1  s2 1 s2 1  s2  1  s2  1];
% nurb.coeffs(4,:) = ones(2*nurb.number,1);
if (refine > 0)
  nurb = nurb_knot_refinement(nurb, refine);
```

```
end
nurb.coeffs_new = zeros(4,ncurl*(nurb.number - 1) + 1);
nurb.coeffs_new(1,1:nurb.number) = ...
    nurb.coeffs(1,1:nurb.number);
nurb.coeffs_new(3,1:nurb.number) = ...
    nurb.coeffs(3,1:nurb.number);
nurb.coeffs_new(4,1:nurb.number) = ...
    nurb.coeffs(4,1:nurb.number);

nurb.coeffs = nurb.coeffs_new;
% Remove premultiplied weights from control point coordinates
nurb.coeffs(1,1:nurb.number) = ...
    nurb.coeffs(1,1:nurb.number) ./ ...
    nurb.coeffs(4,1:nurb.number);
nurb.coeffs(3,1:nurb.number) = ...
    nurb.coeffs(3,1:nurb.number) ./ ...
    nurb.coeffs(4,1:nurb.number);
delta_Y = 1/(ncurl*(nurb.number - 1));
nurb.coeffs(2,:) = 1:-delta_Y:0;
% TODO: Add code to bend helix into circular shape.
U = nurb.knots{1};
U_tilda = U(p:nurb.number+2);
U_tilda = U_tilda./ncurl;
% Add premultiplied weights to new control point coordinates
x = nurb.coeffs(1,2:nurb.number);
z = nurb.coeffs(3,2:nurb.number);
w = nurb.coeffs(4,2:nurb.number);
x_tilda = x;
z_tilda = z;
w_tilda = w;
U = U_tilda;
for i = 2:ncurl
    x = [x,x_tilda];
    z = [z,z_tilda];
    w = [w,w_tilda];
    U = [U,(U_tilda(3:size(U_tilda,2)) +(i - 1)/ncurl)];
end
nurb.coeffs(1,:) = [nurb.coeffs(1,1),x];
nurb.coeffs(3,:) = [nurb.coeffs(3,1),z];
nurb.coeffs(4,:) = [nurb.coeffs(4,1),w];
nurb.knots{1} = [0,U,1];
r = torusradius + nurb.coeffs(1,:);
theta = 2*pi*nurb.coeffs(2,:);
nurb.coeffs(1,:) = r.*cos(theta);
nurb.coeffs(2,:) = r.*sin(theta);
```

```
nurb.coeffs(1,:) = nurb.coeffs(1,:) .* nurb.coeffs(4,:);
nurb.coeffs(2,:) = nurb.coeffs(2,:) .* nurb.coeffs(4,:);
nurb.coeffs(3,:) = nurb.coeffs(3,:) .* nurb.coeffs(4,:);
nurb.number = [(nurb.number - 1)*ncurl + 1];
end
```

Code 3: Evaluation of integral

```
function [ nurb_area ] = integrate_hw01( ...
    nurb_circle,nurb_helix )

%remove double entries in the knot arrays
xknots = unique(nurb_circle.knots{1});
yknots = unique(nurb_helix.knots{1});

[quad_weight_1d, quad_points_1d] = quad_rule();
nquad_1d = size(quad_weight_1d, 2);

nquad = nquad_1d * nquad_1d;
quad_weight = zeros(1,nquad);
quad_points = zeros(2,nquad);

nurb_area = 0.0;
% Loop over all elements, which are spaced from knot to knot
for ie = 1:size(xknots,2)-1
    xi1 = xknots(ie);
    del_Xi = xknots(ie+1) - xi1;
    for k = 1:nquad
        l = floor((k-1)/nquad_1d) + 1;
        quad_points(1,k) = 0.5*(del_Xi) *quad_points_1d(l) ...
            + 0.5*(xknots(ie+1) + xi1);
    end
    u_circle = quad_points(1,:);
     F_circle = nurb_eval(nurb_circle, nurb_circle.coeffs, ...
         3, u_circle);
        dF_circle = ...
            nurb_derv_eval(nurb_circle,nurb_circle.coeffs,...
             3,u_circle);
    for je = 1:size(yknots,2)-1
        % build quad rules
        eta1 = yknots(je);
        del_Eta = yknots(je+1) - eta1;
        area = del_Xi * del_Eta;
        for k = 1:nquad
```

```
            l = floor((k-1)/nquad_1d) + 1;
            m = mod(k-1,nquad_1d) + 1;
            quad_weight(1,k) = quad_weight_1d(l) * ...
                quad_weight_1d(m) * area/4;
            quad_points(2,k) = ...
                0.5*(del_Eta)*quad_points_1d(m) + ...
                 0.5*(yknots(je+1) + eta1);
        end


        u_helix  = quad_points(2,:);


        F_helix = nurb_eval(nurb_helix, nurb_helix.coeffs, ...
            3, u_helix);
        dF_helix = ...
            nurb_derv_eval(nurb_helix,nurb_helix.coeffs,3,...
             u_helix);

        cross_prod = cross(dF_circle,dF_helix);
        difference = F_helix-F_circle;
        val = zeros(1,nquad);
        for i=1:nquad
            val(1,i)=  dot(cross_prod(:,1,i), ...
                difference(:,i)) / ...
                ((norm(difference(:,i)))^3 );
        end
        nurb_area = nurb_area + sum(quad_weight.*val);
    end

end

nurb_area = (nurb_area)/(4*pi);
end
```

Code 4: Drawing of figure

```
function [] = draw_fig(nurb,points)
deltaX = 1/(points(1)-1);
%deltaY = 1/(points(2)-1);

[X] = meshgrid(0:deltaX:1,1);

u = zeros(1, points(1));
```

```
u(1,:) = reshape(X,1,[]);
S = nurb_eval(nurb,nurb.coeffs,3,u);
plot3(S(1,:),S(2,:),S(3,:),'LineWidth',2);
xlabel('x');
ylabel('y');
zlabel('z');
end
```

6 Translation and Deformation Effects

The magnetic field inside the toroid (Figure 3) is defined as:

$$B_{toroid} = \begin{cases} 0, \quad r < a \\ \frac{\mu_0 IN}{2\pi r}, \quad a \le r \le b \\ 0, r > b \end{cases} \tag{7}$$

From Eqn(5), for $r < a$ (Figure 4) and $r > b$, (Figure 5) we get the value of $B = 0$ as no current is enclosed within the surface.

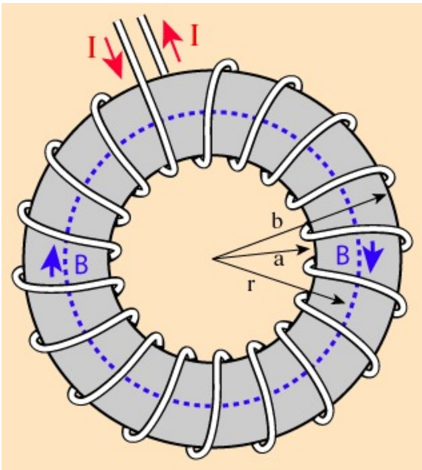

Figure 3: Magnetic Field inside a toroid

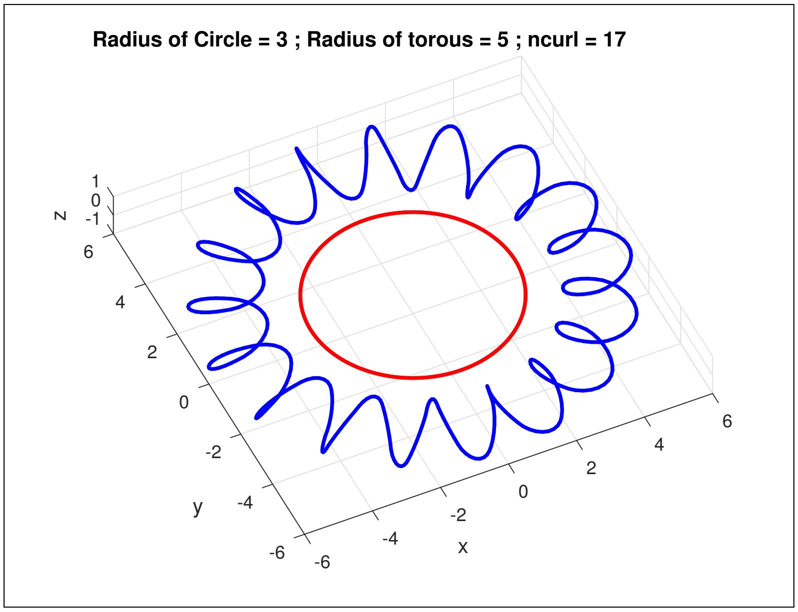

Figure 4: Circle inside the Circular Helix

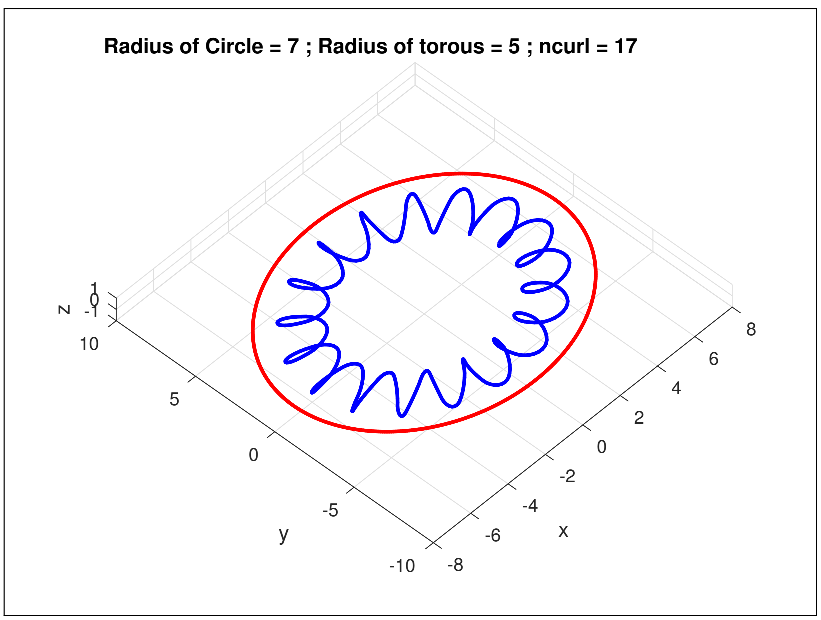

Figure 5: Circle outside the Circular Helix

7 Effect of Refining

- On increasing the level of refining, the number of control points as well as the size of the knot vector increases.
- As a result, the accuracy of the computed B-spline increases.
- Consequently, the accuracy of the integral value increases but the computational time and memory too increases.

Table (1) shows the effect of refinement on the integration value and computation time. The computation is carried on quad core 64-bit Intel(R) Core(TM) i3-4005U CPU.

Table 1: Table to compare value of integral and computation time for different refinement level

Helix Refinement Level	Circle Refinement Level	Integration Value (I)	Time taken (s)
0	0	-17.002824	0.040125
1	0	-17.001594	0.044445
2	0	-17.001588	0.044654
3	0	-17.001588	0.044742
4	0	-17.001586	0.047193
5	0	-17.001585	0.048125
0	1	-17.001257	0.041465
0	2	-17.001254	0.041762
0	3	-17.001253	0.041859
0	4	-17.001253	0.042104
0	5	-17.001252	0.042859