EE6506 - Computational Electromagnetics
Homework - 2

Srivenkat A (EE18B038), Srijan Gupta (EP17B009)

March 12, 2021

1 Question (1)

The given fuction to be analysed is:

$$f(x) = \frac{e^x}{\sqrt{x^2}}, \quad x \in [0.1, 4.9] \tag{1}$$

A) Plotting the function in the [0.1-4.9] interval gives:

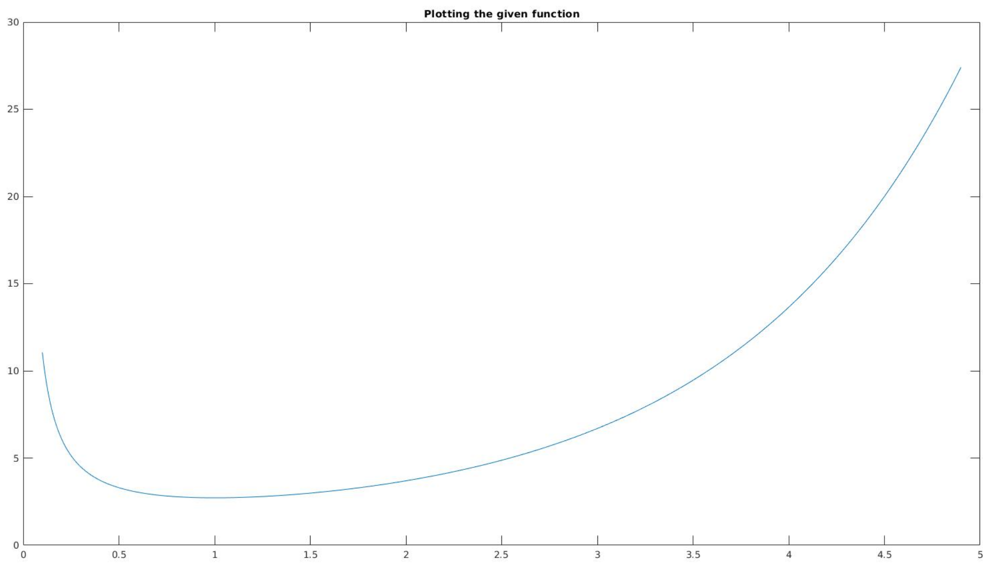

Figure 1: Plotting f(x)

Since in this question we only analyse for $x > 0$, we can replace the function as $f(x) = e^x/x$ for all practical purposes.

B) We use the Gauss quadrature rule to integrate the function in the interval [1,4]. The basic motivation behind Gauss quadrature is to minimise the number of points of function evaluations without compromising on accuracy. The tradeoff occurs in the rigidity imposed on choosing the points of function evaluation.

$$\int_{lim1}^{lim2} f(x) \quad = \quad \sum_{n=1}^{N} f(n) \cdot w(n) \tag{2}$$

To get an accuracy of order 2N-1, we need to evaluate the function at the roots of the Nth order Legendre Polynomial, suited to the particular interval of integration.

We use the inbuilt function `legendreP` in MATLAB to compute the legendre polynomial. But, the function only gives the value of the polynomial evaluated at the point in the input argument. So, to find the roots of the polynomial, we pass the symbolic variable 'x' as parameter and solve the symbolic polynomial expression to get the roots.

```
 %   lim1,lim2: limits of integration
 %   n: no. points of quadrature evaluation
 mean = (lim1 + lim2) ./ 2;    %scale [lim1,lim2] interval to [-1,1]
interval
 scale = (lim2 - lim1) ./2;

 syms x;
 pts_p = double(solve(legendreP(n,x) == 0));   %roots of the legendre
polynomial

 pts = (pts_p .* scale) + mean;   %shift and scale roots
```

The weights are computed as the analytical integrals of the corresponding lagrange interpolating polynomials in the integration interval.

The compuational advantage in quadrature rule is that, if we use the same order quadrature (say order n), the points of evaluation (the roots of n^{th} order legendre polynomial for Gauss quadrature) and weights can be pre-calculated, independent of the function whose integral is to be calculated. If the interval is different, the points just need to be scaled and shifted. So, we define 2 separate functions, `get_wts_pts_for_gauss` and `gauss_int` in our program. The former function is to pre calculate the quadrature evaluation points and weights for an interval. Then, we shift the points correspondingly and feed it into the latter function to compute the estimate of the integral over any shifted interval.

```
 wts = zeros(n,1);   %generate weights
 for i = 1:n
     lagrange_sym = 1;
     for j = 1:n
         if j ~= i
             lagrange_sym = lagrange_sym*(x - pts(j))/(pts(i) - pts(j));
%Construct lagrange polynomial
         end
     end
     lagrange_pol = coeffs(lagrange_sym,'all');    %get the coefficients of
lagrangeP with high precision
     int_lagrange_pol = poly2sym(polyint(lagrange_pol));    %integrate the
lagrangeP and convert into symbolic form
     wts(i) = subs(int_lagrange_pol,lim2) - subs(int_lagrange_pol,lim1);
  %calc weights
 end
```

The gauss_int function just implements the quadrature rule for a given set of weights and points. So, when we need to analytically integrate fucntions over numerous uniform discretisation units, we compute the set of points and weights from the get_wts_pts_for_gauss and just pass shifter points to the gauss_int function.

C) The analytical integration of the function from limits 1 to 4 gives a result of 17.7358 We plot the estimates obtained from the gauss quadrature rule and compare the results.

In our implementation, we use the analytical *integral* function to compute the integrals of the lagrange polynomials, that will be used as weights in the quadrature rule. We have also tried an implementation using the *polyint* function. But, due to the limited precision available when converting it into polynomial form, the integral of the corresponding polynomial blows up for large value of 'n' in a n point quadrature rule. Symbolically using the polynomial gets rid of this error because it uses Variable Precision Arithmetic and it is observed that the integrals obtained are accurate even for very large n.

So, we have also tried an intermediate implementation where we compute the lagrange polynomial symbolically and then convert it into a polynomial to be able to integrate it with its coefficients and then convert it back into symbolic form to retai its precison and then evaluate the integrated polynomial at the interval limits to get the weights.

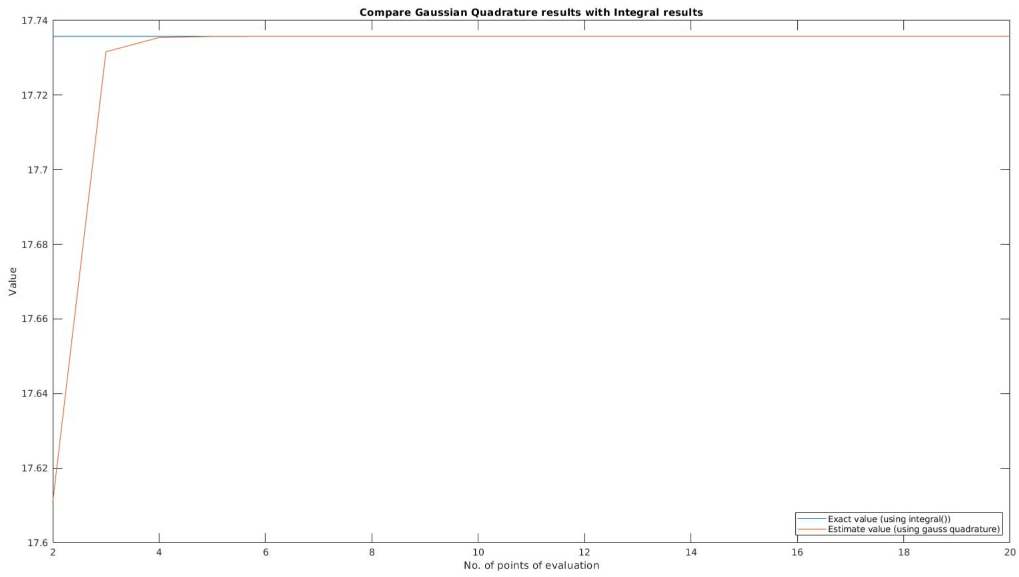

Figure 2: Accuracy of Gauss Quadrature

From the plot, it is clear that using a 4-pt quadrature rule gives a very accurate estimate and increasing the number of points beyond that don't give any improvement in the integral estimate for this function.

2 Question (2)

The given setup includes a thin metallic cylinder of length $L = 1$m placed along the y axis. It has a radius of $a = 0.01$m. The cylinder is connected to a 1V voltage cylinder and being a metallic object, the potential along its entire body will be 1V. We need to find the potential distribution caused by this cylinder in the regions surrounding it.

A) For ease of calculations, we assume that the surface charge distribution can be replaced by a linear charge and discretise it into smaller units of uniform lengths as:

$$\rho_l = \sum_{n=1}^{N} a_n g_n(y), \quad \rho_l = 2\pi a \rho_s \tag{3}$$

where

$$g_n(y) = \begin{cases} 1 & \text{if } y \in [(n-1)W, nW] \\ 0 & \text{o.w.} \end{cases} \tag{4}$$

Where W is the discretization length, $W = L/n$. We have the n matching points $(0, y_m, a)$ with $y_m = W(m - 0.5)$ $\forall m \in [1, n]$ on the surface of the cylinder. We can write the n equations corresponding to each matching point by equating the potential on the cylinder to 1:

$$4\pi\epsilon_0 = \sum_{n=1}^{N} a_n \int_{(n-1)W}^{nW} \frac{\mathrm{d}y'}{\sqrt{(y'-y_m)^2+a^2}} \quad \forall m \in [1,n] \tag{5}$$

The integrals can be solved using the Gauss quadrature.

Code for calculating the a_n's, i.e. ρ_l:

```
Nc = 100;    %number of discretisations of cylinder
L = 1;        %length of cylinder
a = 0.01;  %radius of cylinder
eps = 8.854e-12;
m = 3;       %m-point quadrature rule used

W = L/Nc;  %discretisation length
disc_l = (W/2:W:1 - W/2);    %discretised vector along length

%Points of evaluation and weights for 'm' point gauss quadrature
[pts, wts] = get_wts_pts_for_gauss(0, W, m);

% expressing eqn in Ax = B form
A = ones(Nc);
B = 4*pi*eps*ones(Nc,1);
%rho_l: unknown matrix of (Nc,1)

for i=1:Nc
    for j=1:Nc
        A(i,j) = gauss_int( @(yp) inv_dist( 0, W*(2*i - 1)/2, 0,  0, yp, a ),
    pts + (j-1)*W, wts );
    end
end

rho_l = A \ B; %solving for rho_l
rho_s = rho_l ./ (2*pi*a);
```

(Here inv_dist just calculates 1/distance)

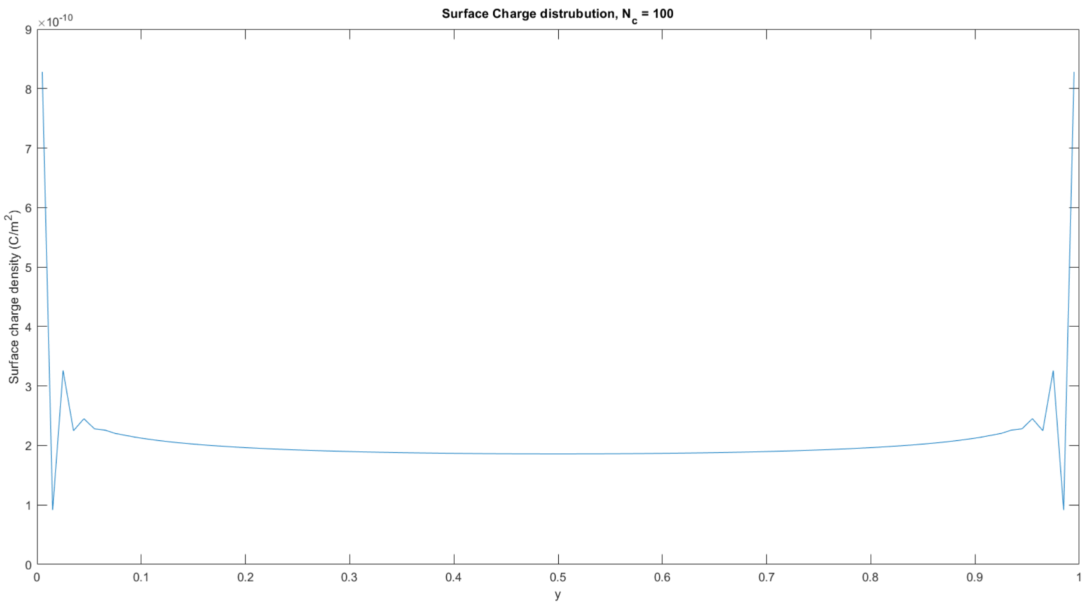

Figure 3: Surface charge density (n = 100)

B) The potential at a point $\vec{r} = (x, y, z)$ is given by,

$$V(\vec{r}) = \sum_{n=1}^{N} a_n \int_{(n-1)W}^{nW} \frac{\mathrm{d}y'}{|\vec{r} - (0, y', 0)|} \tag{6}$$

The integral can again be computed using Gauss quadrature.

Code for calculating the potential over a sphere of radius $R = 10$m centred at the centre of the cylinder:

```
Ns = 100;    %number of discretisations of sphere
R = 10;    %radius of sphere

% Defining points on sphere
phi = pi * (-1:(1/Ns):1);
theta = (pi/2) * (-1:(1/Ns):1)';
X = R .* cos(theta) * cos(phi);
Y = R .* cos(theta) * sin(phi);
Z = R .* sin(theta) * ones(size(phi));

% Matrix for the potential on sphere
V = zeros(2*Ns + 1);

% Calculating the potential over the sphere
for i=1:(2*Ns + 1)
    for j=1:(2*Ns + 1)
        for k=1:Nc
            V(i,j) = V(i,j) + rho_l(k) * gauss_int( @(yp) inv_dist( X(i,j), Y(i,j), Z(i,j),  0, yp, 0 ), pts + ((k-1)-Nc/2)*W, wts );
        end
    end
end
```

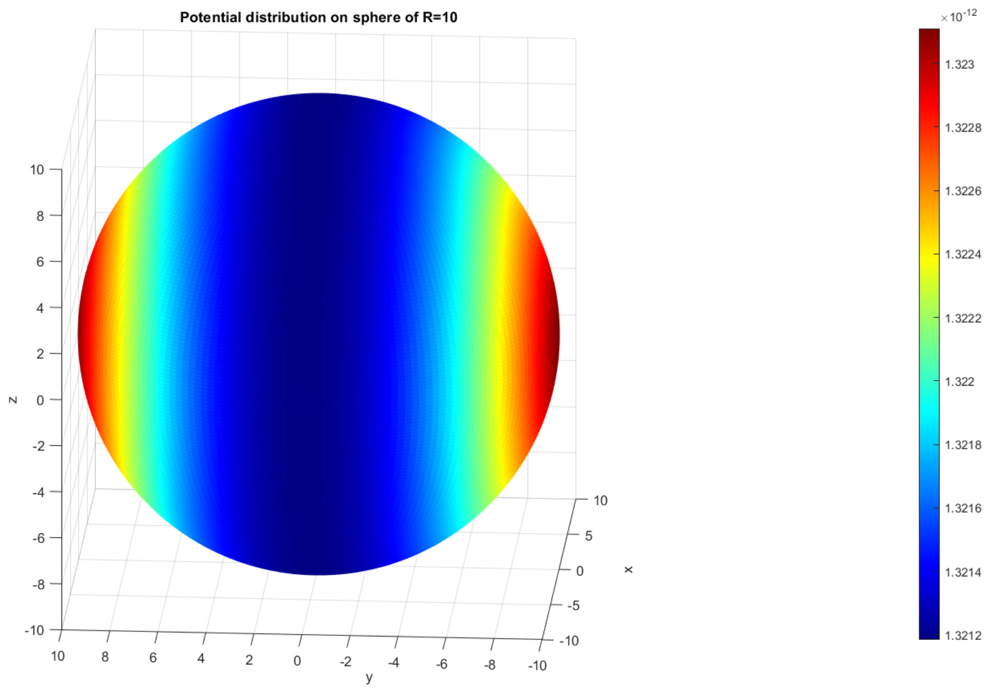

Figure 4: Potential over a sphere of radius 10m (in V)

Note that 3-point Gauss quadrature was used for computing each integral, and the points of evaluation and the weights were just computed once. The points of evaluation were just shifted according to the required interval (scaling wasn't required since interval length didn't change).

Justification for using 3-point Gauss quadrature:

A 3-point Gauss quadrature rule gives an accuracy of $O(5)$ polynomial. We have discretized a length of 1m into 100 parts, hence each integral is over an interval of 0.01m and the integrand is of the form $\sim 1/y$ which doesn't change much over such a small interval. Hence, an approximation by $O(5)$ polynomial should more than suffice, hence a 3-point Gauss quadrature rule is accurate enough.

3 Annexure

3.1 get_wts_pts_for_gauss

```
function [pts, wts] = get_wts_pts_for_gauss(lim1, lim2, n)
    %   lim1,lim2: limits of integration
    %   n: no. points of quadrature evaluation
    mean = (lim1 + lim2) ./ 2;    %scale [lim1,lim2] interval to [-1,1]
    interval
    scale = (lim2 - lim1) ./2;

    syms x;
    if n==3
        pts_p = [0, -0.7746, 0.7746];
    else
        pts_p = double(solve(legendreP(n,x) == 0));   %roots of the legendre
    polynomial
    end

    pts = (pts_p .* scale) + mean;   %shift and scale roots

    wts = zeros(n,1);   %generate weights
    for i = 1:n
        lagrange_sym = 1;
        for j = 1:n
            if j ~= i
                lagrange_sym = lagrange_sym*(x - pts(j))/(pts(i) - pts(j));
            end
        end
        lagrange_sym = matlabFunction(lagrange_sym);
        wts(i) = integral(lagrange_sym, lim1, lim2);
    end
end
```

3.2 gauss_int

```
function output = gauss_int(func, pts, wts)
%Calculates integral value using gaussian quadrature
    %   func: function whose integral is evaluated
    %   pts: points for evaluation

    n = length(pts);
    output = 0;
    for i=1:n
        output = output + func(pts(i)) * wts(i);
    end
end
```

3.3 hw2_q1

```
fun = @(x) exp(x) ./ sqrt(x.^2);
Nmax = 20;

figure;
a = [0.1:0.01:4.9];
plot(a, fun(a));
title("Plotting the given function")

crct_val = integral(fun,1,4); %The exact value
% Initializing
estimate_val = zeros(4,1);

for i=2:Nmax
    % computing weights and points for evaluation for gauss quadrature
    [pts, wts] = get_wts_pts_for_gauss(1, 4, i);
    % calculating the estimated integral
    estimate_val(i-1) = double( gauss_int(fun,pts, wts) );
end

figure;
plot([2:Nmax], crct_val*ones(Nmax-1,1));
hold on;
plot([2:Nmax], estimate_val);
legend("Exact value (using integral())","Estimate value (using gauss quadrature
    )",'Location','SouthEast');
xlabel('No. of points of evaluation')
ylabel('Value')
title("Compare Gaussian Quadrature results with Integral results");
```

3.4 hw2_q2

```
Nc = 100;   %number of discretisations of cylinder
Ns = 100;   %number of discretisations of sphere
L = 1;        %length of cylinder
a = 0.01;  %radius of cylinder
R = 10;    %radius of sphere
eps = 8.854e-12;
m = 3;      %m-point quadrature rule used

W = L/Nc;  %discretisation length
disc_l = (W/2:W:1 - W/2);    %discretised vector along length

%Calculating points of evaluation and weights for 'm' point gauss
%quadrature
[pts, wts] = get_wts_pts_for_gauss(0, W, m);

%rho_l = 2*pi*a*rho_s
% eqn used: 1 = integral( (rho_l * dl) /(4*pi*eps * r), 0, L )

% expressing eqn in Ax = B form
A = ones(Nc);
B = 4*pi*eps*ones(Nc,1);
%rho_l: unknown matrix of (Nc,1)

for i=1:Nc
    for j=1:Nc
```

```
        A(i,j) = gauss_int( @(yp) inv_dist( 0, W*(2*i - 1)/2, 0,  0, yp, a ), pts + (j-1)*W, wts );
    end
end

rho_l = A \ B; %solving for rho_l
rho_s = rho_l ./ (2*pi*a);

%Plotting
figure;
plot(disc_l, rho_s);
xlabel('y')
ylabel('Surface charge density (C/m^2)')
title(['Surface Charge distrubution, N_c = ',num2str(Nc)]);

figure;
imagesc(rho_s');
colormap(jet(Nc));
colorbar;
title(['Surface Charge distrubution, N_c = ', num2str(Nc)]);

% Defining points on sphere
phi = pi * (-1:(1/Ns):1);
theta = (pi/2) * (-1:(1/Ns):1)';
X = R .* cos(theta) * cos(phi);
Y = R .* cos(theta) * sin(phi);
Z = R .* sin(theta) * ones(size(phi));

% Matrix for the potential on sphere
V = zeros(2*Ns + 1);

% Calculating the potential over the sphere
for i=1:(2*Ns + 1)
    for j=1:(2*Ns + 1)
        for k=1:Nc
            V(i,j) = V(i,j) + rho_l(k) * gauss_int( @(yp) inv_dist( X(i,j), Y(i,j), Z(i,j),  0, yp, 0 ), pts + ((k-1)-Nc/2)*W, wts );
        end
    end
end

% Plotting
figure;
colormap(jet);
surf(X,Y,Z,V, 'EdgeColor','interp');
colorbar;
axis square;
xlabel('x')
ylabel('y')
zlabel('z')
title("Potential distribution on sphere of R=10");

function output = inv_dist(x1,y1,z1, x2,y2,z2)
    dist = sqrt( (x1 - x2)^2 + (y1 - y2)^2 + (z1 - z2)^2 );
    output = 1 / dist;
end
```