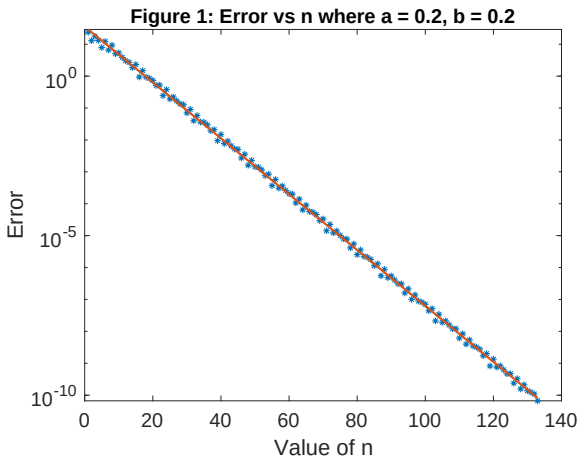
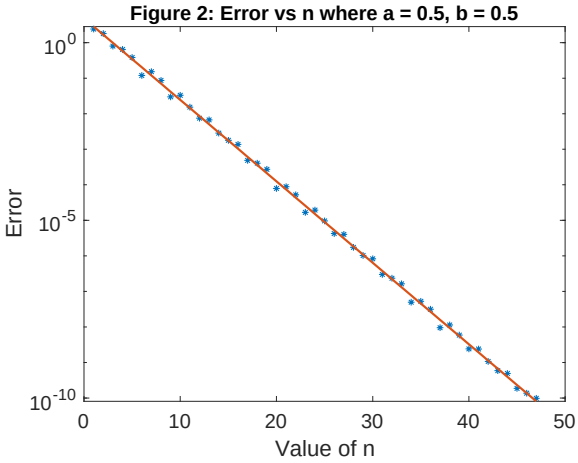
(a)

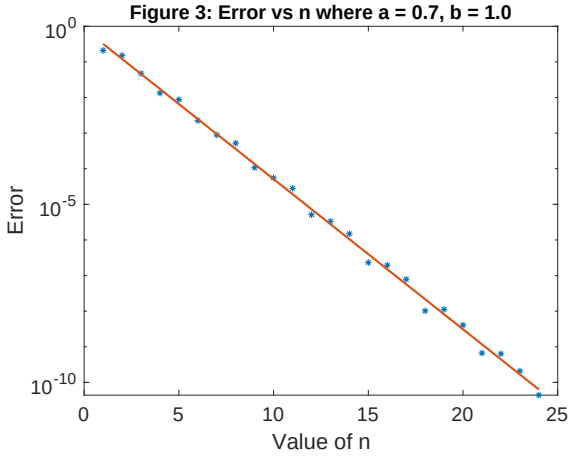


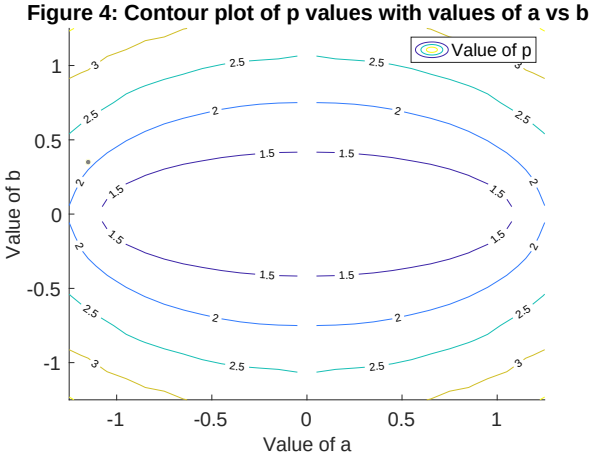


(b) When (a,b) = (0.2,0.2) the value of p is 1.2233. When (a,b) = (0.5,0.5), p becomes 1.6951 and when (a,b) = (0.7,1), p is 2.6410. I estimated the value of p by taking two consecutive error points on the best fit line (shown as a red line in Figure 1, 2 and 3) and then I took the ratio between them to get p. p is a measure of the upper bound of the error which is defined as Cp^{-n}, thus, the ratio between two consecutive error points at iteration n and n+1 gives us an accurate estimate of p because the constants C and the powers of -n cancel each other out.

(c) We use a range of n defined by a tolerance TOL because the error of polynomial interpolation at equally-spaced nodes increases when more nodes are taken into account in the calculation of the polynomial. The end points of the interval are not well-suited when taking the polynomial interpolation of equally-spaced nodes which reduces accuracy when we consider higher degree polynomial interpolations. This is Runge's phenomenon and setting a tolerance TOL to define the range of n instead of having a fixed maximum value of n in our algorithm combats this phenomenon. Additionally, defining the range of n with TOL can reduce the number of iterations n we need to get to within the error tolerance for two consecutive points in our polynomial interpolation of the original function.

(d)





(e) As depicted in Figure 4, the value of p increases whenever the absolute values of both a and b increase. The level curves of p are symmetrical between negative and positive a and b values. This means that we get the same value of p for a negative a value and negative b value that we get for a positive a value and positive b value. This also holds if a and b were to be different signs. Thus, p changes based on only the absolute magnitude of a and b and not their signs. Additionally, p is always greater than 1, which means its value is lower bounded by 1 for varying values of a and b. Because p is always greater than 1 this means that p cannot equal 1. This is illustrated in Figure 4 by the fact that there is no level curve for p = 1. The level curves we see are for p = 1.5, 2, 2.5 and 3.

MATLAB Code

```
%Figures for part A, B and C

err_grid = (linspace(-1,1,10000))'; % Equally-spaced grid of 10000 nodes to
compute error on
tol = 1e-10; % tolerance used to determined largest value of n

a = 0.2; b = 0.2; % Set values of a,b
f = @(x) 1./((x-a).^2+b.^2); % Define f(x)

f_grid = f(err_grid); % Evaluate f on the error grid
errvals = []; % Initialize an error vector
err = 1; % set the error initially to 1
n = 0; % initialize n

while err > tol
    n = n+1;

    % compute points
    x = (linspace(0,1,n+1))';
    x = cos(pi*x);

    % compute weights
    w = (-1).^((0:n)');
    w(1) = 1/2; w(n+1) = w(n+1)/2;

    y = f(x); % Evaluate f at the nodes
    p = baryinterp(x,w,y,err_grid); % Compute P(x) on the error grid
    err = max(abs(p - f_grid)); % Compute error
    errvals = [errvals err]; % Update error vector

end

figure(1);
semilogy(1:n,errvals,'*');
set(gca,'FontSize',18);
xlabel('Value of n','fontsize',20);
ylabel('Error','fontsize',20);
title('Figure 1: Error vs n where a = 0.2, b = 0.2','fontsize',18);
grid on;

xvals = 1:n;

poly = polyval(polyfit(xvals,log(errvals),1), xvals);

hold on;
semilogy(1:n,exp(poly),'linewidth',2);

g = exp(poly);
p2 = g(1)/g(2);
disp(p2);
```

```
%Figure for part D and E

err_grid = (linspace(-1,1,10000))'; % Equally-spaced grid of 10000 nodes to
compute error on
tol = 1e-10; % tolerance used to determined largest value of n

avals = 0.05:0.1:1.25;
bvals = 0.05:0.1:1.25;
pvals = zeros(length(avals),length(bvals));
disp(size(pvals));


f = @(x) 1./((x-a).^2+b.^2); % Define f(x)
f_grid = f(err_grid); % Evaluate f on the error grid
errvals = []; % Initialize an error vector
err = 1; % set the error initially to 1
n = 0; % initialize n

j = 0;
for i = 1:length(avals)
    for j = 1:length(bvals)
        aval = avals(i);
        bval = bvals(j);
        f = @(x) 1./((x-aval).^2+bval.^2);
        f_grid = f(err_grid); % Evaluate f on the error grid
        errvals = []; % Initialize an error vector
        err = 1; % set the error initially to 1
        n = 0; % initialize n

        while err > tol
            n = n+1;

            % compute points
            x = (linspace(0,1,n+1))';
            x = cos(pi*x);

            % compute weights
            w = (-1).^((0:n)');
            w(1) = 1/2; w(n+1) = w(n+1)/2;

            y = f(x); % Evaluate f at the nodes
            p = baryinterp(x,w,y,err_grid); % Compute P(x) on the error grid
            err = max(abs(p - f_grid)); % Compute error
            errvals = [errvals err]; % Update error vector

        end

        xvals = 1:n;

        poly = polyval(polyfit(xvals,log(errvals),1), xvals);

        g = exp(poly);
        pvals(j,i) = g(1)/g(2);
```

```
    end
end

hold on;
contour(avals,bvals,pvals, 'ShowText','on');
contour(-avals,-bvals,pvals, 'ShowText','on');
contour(-avals,bvals,pvals, 'ShowText','on');
contour(avals,-bvals,pvals, 'ShowText','on');
hold off;

set(gca,'FontSize',18);
xlabel('Value of a','fontsize',18);
ylabel('Value of b','fontsize',18);
title('Figure 4: Contour plot of p values with values of a vs b');
legend('Value of p');
```