Assignment 3

111550142 尤瑋辰

I use MATLAB to calculate the result. Here is the result and code:

1. (a) Calculate the table by $f[x_0, x_1, .. x_n] = \frac{f[x_1, x_2, .. x_n] - f[x_0, x_1, .. x_{n-1}]}{x_n - x_0}$

   PS: The first column of divided-difference table is x, and the second column is f[x], and so forth.

```
% a
x = [-0.2, 0.3, 0.7, -0.3, 0.1];
f_x = [1.23, 2.34, -1.05, 6.51, -0.06];

table_a = DividedDifference(x,f_x);

disp('Divided-difference table:')
disp(table_a);
```

```
Divided-difference table:
   -0.2000    1.2300    2.2200  -11.8833 -103.5833   73.6111
    0.3000    2.3400   -8.4750   -1.5250  -81.5000         0
    0.7000   -1.0500   -7.5600   14.7750         0         0
   -0.3000    6.5100  -16.4250         0         0         0
    0.1000   -0.0600         0         0         0         0
```

```
function table = DividedDifference(x, f)
    % The first column of table is x, the second column of table is f[x],
    % the third column is f[xi, xi+1], and so forth.
    table = zeros(length(x), length(x)+1);

    table(:,1) = x;
    table(:,2) = f;

    for j = 3:length(x)+1
        for i = 1:length(x)-j+2
            table(i, j) = (table(i+1, j-1) - table(i, j-1))/(table(i+j-2,1) - table(i,1));
        end
    end
end
```

(b) Choose the first 3 points to interpolate.

```
% b
% Choose the first 3 points to interpolate
x = [-0.2, 0.3, 0.7];
f_x = [1.23, 2.34, -1.05];

table_b = DividedDifference(x,f_x);

x0 = 0.4;

result = table_b(1, length(x)+1);

for i = (length(x)):-1:2
    result = result*(x0 - x(i-1)) + table_b(1, i);
end

disp(['The interpolated value at x = 0.4 is: ', num2str(result)]);
```

```
Divided-difference table:
   -0.2000    1.2300    2.2200  -11.8833
    0.3000    2.3400   -8.4750         0
    0.7000   -1.0500         0         0

The interpolated value at x = 0.4 is: 1.849
```

(c) Choose the 3 points that are closest to the point x = 0.4, that is x = 0.3, 0.7, 0.1.

```
% c
% Choose the 3 points that are closest to the point x = 0.4
x = [0.3, 0.7, 0.1];
f_x = [2.34, -1.05, -0.06];

table_c = DividedDifference(x,f_x);

x0 = 0.4;

result = table_c(1, length(x)+1);

for i = (length(x)):-1:2
    result = result*(x0 - x(i-1)) + table_c(1, i);
end
disp('Divided-difference table:')
disp(table_c);
disp(['The interpolated value at x = 0.4 is: ', num2str(result)]);
```

```
Divided-difference table:
    0.3000    2.3400   -8.4750  -34.1250
    0.7000   -1.0500   -1.6500         0
    0.1000   -0.0600         0         0

The interpolated value at x = 0.4 is: 2.5162
```

2. According to the textbook P.172 to P.174. The end conditions 3 and 4 are described as follows:

3. Take $S_0 = S_1$, $S_n = S_{n-1}$. This is equivalent to assuming that the end cubics approach parabolas at their extremities.
4. Take S_0 as a linear extrapolation from S_1 and S_2, and S_n as a linear extrapolation from S_{n-1} and S_{n-2}. Only this condition gives cubic spline curves that match exactly to $f(x)$ when $f(x)$ is itself a cubic. For condition 4, we use these relations:

At left end: $\dfrac{S_1 - S_0}{h_0} = \dfrac{S_2 - S_1}{h_1}$, $\quad S_0 = \dfrac{(h_0 + h_1)S_1 - h_0 S_2}{h_1}$.

At right end: $\dfrac{S_n - S_{n-1}}{h_{n-1}} = \dfrac{S_{n-1} - S_{n-2}}{h_{n-2}}$,

$$S_n = \frac{(h_{n-2} + h_{n-1})S_{n-1} - h_{n-1}S_{n-2}}{h_{n-2}}.$$

This is called "not a knot condition."

Condition 3 $\quad S_0 = S_1, S_n = S_{n-1}$:

$$\begin{bmatrix} (3h_0 + 2h_1) & h_1 & & & \\ h_1 & 2(h_1 + h_2) & h_2 & & \\ & h_2 & 2(h_2 + h_3) & h_3 & \\ & & & \ddots & \\ & & & h_{n-2} & (2h_{n-2} + 3h_{n-1}) \end{bmatrix}.$$

Condition 4 $\quad S_0$ and S_n are linear extrapolations:

$$\begin{bmatrix} \dfrac{(h_0 + h_1)(h_0 + 2h_1)}{h_1} & \dfrac{h_1^2 - h_0^2}{h_1} & & & \\ h_1 & 2(h_1 + h_2) & h_2 & & \\ & h_2 & 2(h_2 + h_3) & h_3 & \\ & & & \ddots & \\ & & & \dfrac{h_{n-2}^2 - h_{n-1}^2}{h_{n-2}} & \dfrac{(h_{n-1} + h_{n-1})(h_{n-1} + 2h_{n-2})}{h_{n-2}} \end{bmatrix}.$$

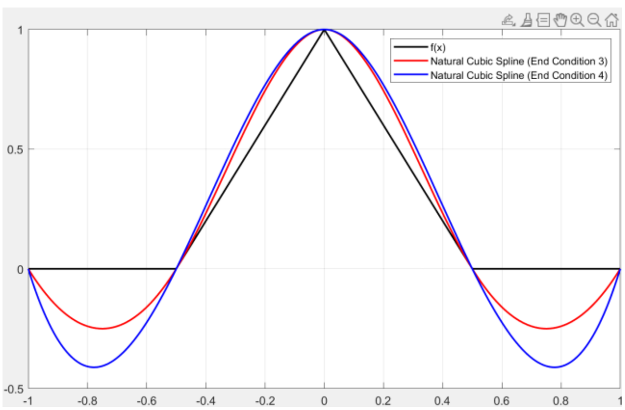
Based on the matrix H above, solve for S using the equation HS = Y.

```
% Define the five evenly spaced points and their function values
x = linspace(-1, 1, 5);
y = arrayfun(@f, x);

% Calculate the step sizes h = xi+1 - x
h = diff(x);

% Calculate the divided differences f[xi, xi+1] = (yi+1 - yi) / hi;
div_diff = diff(y) ./ h;

% Solve the system HS = Y
% End condition 3: S0 = S1, Sn-1 = Sn, End condition 4: Not a knot
n = length(x);
H_3 = zeros(n, n);
H_3(1,1) = 1;
H_3(n,n) = 1;

H_4 = zeros(n, n);
H_4(1,1) = 1;
H_4(n,n) = 1;
for i = 2:n-1
    H_3(i,i-1) = h(i-1);
    H_3(i,i) = 2 * (h(i-1) + h(i));
    H_3(i,i+1) = h(i);

    H_4(i,i-1) = h(i-1);
    H_4(i,i) = 2 * (h(i-1) + h(i));
    H_4(i,i+1) = h(i);
end
H_3(2,2) = 3*h(1) + 2*h(2);
H_3(n-1, n-1) = 2*h(end-1) + 3*h(end);

H_4(2,2:3) = [(h(1)+h(2))*(h(1)+2*h(2))/h(1), (h(2)^2-h(1)^2)/h(2)];
H_4(n-1, n-2:n-1) = [(h(end-1)^2-h(end)^2)/h(end-1),...
    (h(end-1)+h(end))*(h(end)+2*h(end-1))/h(end-1)];
```

```
Y = 6 * diff(div_diff)';
Y = [0; Y; 0];

S_3 = (H_3 \ Y)';
S_3(1) = S_3(2);
S_3(end) = S_3(end-1);

S_4 = (H_4 \ Y)';
S_4(1) = ((h(1)+h(2))*S_4(2) - h(1)*S_4(3))/h(2);
S_4(end) = ((h(end-1)+h(end))*S_4(end-1) - h(end)*S_4(end-2))/h(end-1);

% Calculate the coefficients of the cubic polynomials
% a = (Si+1 - Si) / 6hi
% b = Si/2
% c = (yi+1 - yi) / hi - (2hiSi + hiSi+1)/6
% d = yi;
a_3 = diff(S_3) ./ (6*h);
b_3 = S_3(1:n-1) / 2;
c_3 = div_diff - (2*S_3(1:n-1) + S_3(2:n)).*h/6;
d_3 = y(1:n-1);

a_4 = diff(S_4) ./ (6*h);
b_4 = S_4(1:n-1) / 2;
c_4 = div_diff - (2*S_4(1:n-1) + S_4(2:n)).*h/6;
d_4 = y(1:n-1);
```

After solving for S, calculate coefficients a, b, c, and d. Then, plot two cubic splines along with f(x).

```
% Plot graph
X = linspace(-1, 1, 400);
y = arrayfun(@f, X);
plot(X, y, 'k', 'LineWidth', 1.5); hold on;
for i = 1:n-1
    X = linspace(x(i), x(i+1), 100);
    p_3 = @(X) a_3(i)*(X - x(i)).^3 + b_3(i)*(X - x(i)).^2 + c_3(i)*(X - x(i)) + d_3(i);
    p_4 = @(X) a_4(i)*(X - x(i)).^3 + b_4(i)*(X - x(i)).^2 + c_4(i)*(X - x(i)) + d_4(i);

    plot(X, p_3(X),'r', 'LineWidth', 1.5);
    plot(X, p_4(X),'b', 'LineWidth', 1.5);
end
legend('f(x)', 'Natural Cubic Spline (End Condition 3)', ...
    'Natural Cubic Spline (End Condition 4)');
hold off;
grid on;

% Define the function f(x)
function y = f(x)
    if x < -0.5
        y = 0;
    elseif x > 0.5
        y = 0;
    else
        y = 1 - abs(2*x);
    end
```



3. (a) Use $x(u) = u^T M P p_x$ and $y(u) = u^T M P p_y$ to calculate the Bezier curve

(b) Since it needs to pass through the second and third point, we have following equation:

$$\begin{cases} y(u=\frac{1}{3}) = 0.3 \\ y(u=\frac{2}{3}) = 1.7 \end{cases} \Rightarrow$$

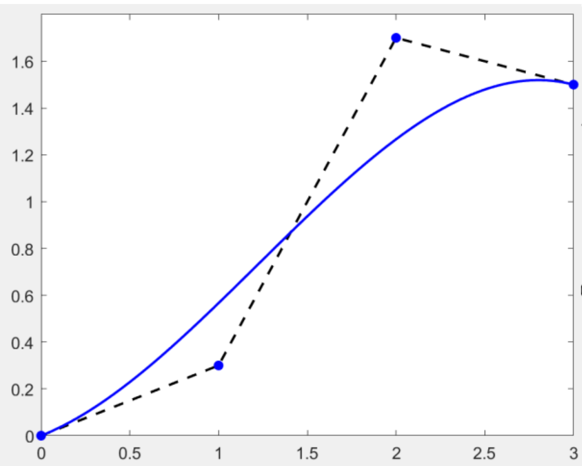
Assume two points are $(1, a)$, $(2, b)$, solve a, b

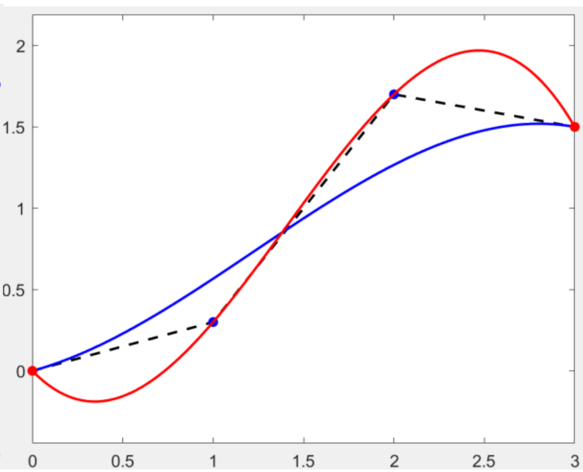
$$\begin{cases} u^T M P \begin{bmatrix} 0 \\ a \\ b \\ 1.5 \end{bmatrix} = 0.3 \\ u^T M P \begin{bmatrix} 0 \\ a \\ b \\ 1.5 \end{bmatrix} = 1.7 \end{cases}$$

```
% a
x = [0, 1, 2, 3];
y = [0, 0.3, 1.7, 1.5];

% Plot the zigzag line
plot(x, y, 'k--', 'LineWidth', 1.5); hold on;
scatter(x, y, 'filled', 'MarkerFaceColor', 'b');

% x(u) = u'MPx
% y(u) = u'MPy
M = [2,-2, 1, 1;
    -3, 3,-2,-1;
     0, 0, 1, 0;
     1, 0, 0, 0];

P = [1, 0, 0, 0;
     0, 0, 0, 1;
    -3, 3, 0, 0;
     0, 0,-3, 3];

x_u = M*P*(x');
y_u = M*P*(y');

u = linspace(0, 1, 100);
x_p = x_u(1)*u.^3 + x_u(2)*u.^2 + x_u(3)*u + x_u(4);
y_p = y_u(1)*u.^3 + y_u(2)*u.^2 + y_u(3)*u + y_u(4);

% Plot the function P(u) = (x(u), y(u))
plot(x_p, y_p, 'b', 'LineWidth', 1.5);
```

```
% b
% Solve a and b
u_a = [(1/3)^3, (1/3)^2, (1/3), 1];
u_b = [(2/3)^3, (2/3)^2, (2/3), 1];
vec_a = u_a*M*P;
vec_b = u_b*M*P;
A = [vec_a(2:3); vec_b(2:3)];
B = [0.3; 1.7] - 1.5*[37/999; 296/999];
result = A \ B;
a = result(1);
b = result(2);


% Calculate the new curve
x = [0, 1, 2, 3];
y = [0, a, b, 1.5];

scatter(x, y, 'filled', 'MarkerFaceColor', 'r');

x_u = M*P*(x');
y_u = M*P*(y');

u = linspace(0, 1, 100);

x_p = x_u(1)*u.^3 + x_u(2)*u.^2 + x_u(3)*u + x_u(4);
y_p = y_u(1)*u.^3 + y_u(2)*u.^2 + y_u(3)*u + y_u(4);

% Plot the function P(u) = (x(u), y(u))
plot(x_p, y_p, 'r', 'LineWidth', 1.5);
hold off;
```

$a = -1.15$, $b = 3.4$





4. First, calculate u and v by solving $\begin{cases} 2.8 = x(u,v) = u^T M X M^T v \\ 0.54 = y(u,v) = u^T M Y M^T v \end{cases}$

Then, you will derive $\begin{cases} 2.8 = 0.2667u^3 - 0.3u^2 + 0.9u + 2.4 \\ 0.54 = 0.0333v^3 - 0.05v^2 + 0.15v + 0.3833 \end{cases}$

Secondly, use u and v to calculte z by $z(u,v) = u^T M Z M^T v$

```
M = [-1, 3,-3, 1;
      3,-6, 3, 0;
     -3, 0, 3, 0;
      1, 4, 1, 0];

X = [1.3, 1.3, 1.3, 1.3;
     2.5, 2.5, 2.5, 2.5;
     3.1, 3.1, 3.1, 3.1;
     4.7, 4.7, 4.7, 4.7];

Y = [0.2, 0.4 ,0.5, 0.7;
     0.2, 0.4 ,0.5, 0.7;
     0.2, 0.4 ,0.5, 0.7;
     0.2, 0.4 ,0.5, 0.7];

Cx = 1/36 * M * X * (M');
vec_u = Cx(:,4)';
vec_u(4) = vec_u(4) - 2.8;
u_roots = roots(vec_u);
u = u_roots(imag(u_roots) == 0);

u_v = [u^3, u^2, u, 1];

Cy = 1/36 * M * Y * (M');
vec_v = Cy(4,:);
vec_v(4) = vec_v(4) - 0.54;
v_roots = roots(vec_v);
v = v_roots(imag(v_roots) == 0);

v_v = [v^3; v^2; v; 1];

test_x = 1/36 * u_v * M * X * (M') * v_v;
test_y = 1/36 * u_v * M * Y * (M') * v_v;

Z = [2.521 2.792 2.949 3.314;
     3.721 3.992 4.149 4.514;
     4.321 4.592 4.749 5.114;
     5.921 6.192 6.349 6.714];

% z(u, v) = 1/36 * u'MZM'v
z = 1/36 * u_v * M * Z * (M') * v_v;
disp(['z = ', num2str(z)]);
```

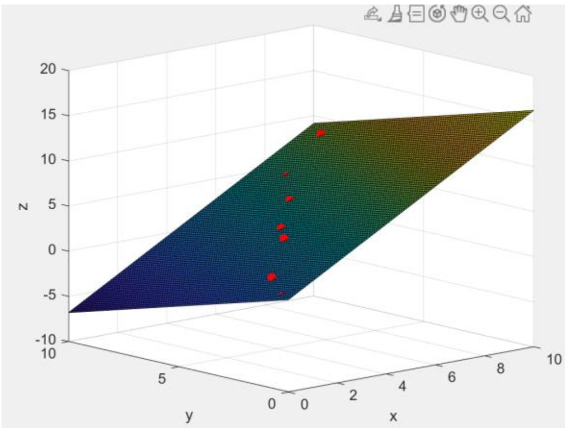
```
z = 4.5246
```

5. (a)

$$\begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ & \vdots & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

(b) Use least square method to solve a, b, and c. $A^T A x = A^T B, x = [a;\ b;\ c]$

```
% b
xi = [0.40 1.2 3.4 4.1 5.7 7.2 9.3];
yi = [0.70 2.1 4.0 4.9 6.3 8.1 8.9];
zi = [0.031 0.933 3.058 3.349 4.870 5.757 8.921];

% A = [xi, yi, 1...],  B = zi
A = [0.4 0.7 1;
     1.2 2.1 1;
     3.4 4.0 1;
     4.1 4.9 1;
     5.7 6.3 1;
     7.2 8.1 1;
     9.3 8.9 1];

B = [0.031; 0.933; 3.058; 3.349; 4.870; 5.757; 8.921];

% A'Ax = A'B, x = [a b c]
x = (A'*A)\(A'*B);
a = x(1); b = x(2); c = x(3);
disp(['a = ', num2str(a),', b = ', num2str(b),'c = ', num2str(c)]);
```




```
a = 1.5961, b = -0.70238c = 0.22067
```

(c) Calculate sum of the squares of the deviations by $SSD = \sum |d_i|$

```
% c
% SSD = sigma di^2 = sigma axi + byi + c - zi
SSD = 0;
for i = 1:length(xi)
    SSD = SSD + (a*xi(i) + b*yi(i) + c - zi(i))^2;
end
disp(['Sum of the squares of the deviations: ', num2str(SSD)]);
```

```
Sum of the squares of the deviations: 0.3194
```

6. First, I calculate the first 5 terms (truncated after 4th degree) of chebyshev series using the following formula:

$$f(x) = \sum_{i=0}^{4} a_i T_i(x)\,, where\ T_i(x) = 2xT_{i-1}(x) - T_{i-2}(x), T_0 = 1, T_1 = x$$

$$a_i = \frac{1}{N}\int_{-1}^{1} \frac{f(t)T_i(t)}{\sqrt{1-t^2}}dt, \quad \begin{cases} N = \pi\ , & i = 0 \\ N = \dfrac{\pi}{2}, & i > 0 \end{cases}$$

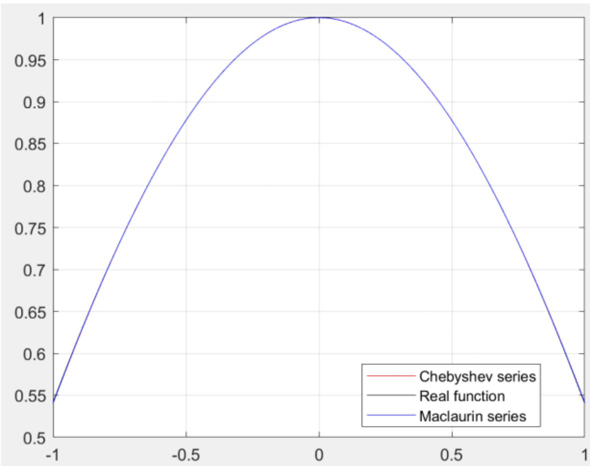
Since find the first few terms of the Chebyshev series for cos(x) by rewriting the Maclaurin series in terms is not precise, I use above formula to calculate the coefficient of T_i.

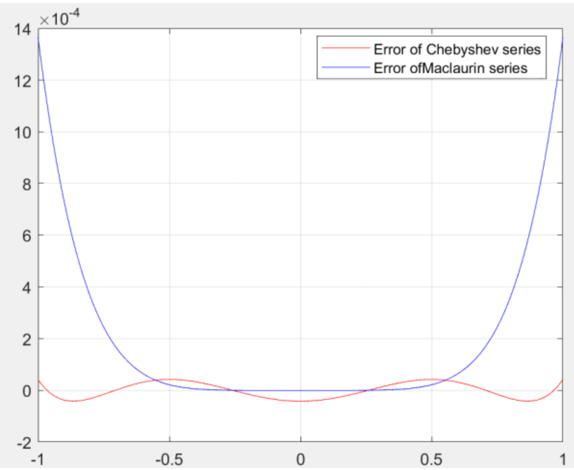
Then convert it back to power series of x:

```
0.99996 + -3.5339e-17x + -0.49924x^2 + 0x^3 + 0.039626x^4
```

```
% Define first N term and f function
N = 5;
f = @(x) cos(x);

% Chebyshev series
% Compute the first N Chebyshev polynomials
T = cell(1, N);
T{1} = @(x) 1; % T_0(x)
T{2} = @(x) x; % T_1(x)
for n = 3:N
    T{n} = @(x) 2 * x .* T{n-1}(x) - T{n-2}(x);
end

% Compute the coefficient a
integrand = cell(1, N);
for n = 1:N
    integrand{n} = @(t) f(t) .* T{n}(t) ./ sqrt(1 - t.^2);
end

a = cell(1, N);
for n = 1:N
    if n == 1
        a{n} = 1/pi * integral(@(t) integrand{n}(t), -1, 1);
    else
        a{n} = 2/pi * integral(@(t) integrand{n}(t), -1, 1);
    end
end

% Convert it back to power series of x
const = a{1} - a{3} + a{5};
x1 = a{2} - 3*a{4};
x2 = 2*a{3} - 8*a{5};
x3 = 4*a{4};
x4 = 8*a{5};
disp([num2str(const),' + ', num2str(x1),'x + ', num2str(x2),'x^2 + ', ...
    num2str(x3),'x^3 + ',num2str(x4),'x^4'])
```

```
% Plot both Chebyshev and Maclaurin series
X = linspace(-1, 1, 1000);
Y_C = const + x1*X + x2*X.^2 + x3*X.^3 + x4*X.^4;

% Maclaurin series
Y_M = zeros(size(X));
for i = 1:length(X)
    sigma = 0;
    for n = 0:N/2
        sigma = sigma + (-1)^n * X(i).^(2*n) / factorial(2*n);
    end
    Y_M(i) = sigma;
end

plot(X,Y_C,'r', 'DisplayName', 'Chebyshev series'); hold on;
plot(X, f(X),'k', 'DisplayName', 'Real function');
plot(X,Y_M,'b', 'DisplayName', 'Maclaurin series');
grid on;
legend('Location', 'best');
hold off;

% Plot error of both series
figure
error_C = Y_C - f(X);
error_M = Y_M - f(X);
plot(X, error_C, 'r', 'DisplayName', 'Error of Chebyshev series'); hold on;
plot(X, error_M, 'b', 'DisplayName', 'Error of Maclaurin series');
grid on;
legend('Location', 'best');
hold off
```





Conclusion, the Chebyshev series has smaller error near -1 and 1.

7. Calculate the A_0, A_n, B_n (first 10 terms) using the following formula:

$$A_0 = \frac{1}{3}\int_{-1}^{2} f(x)dx$$

$$A_n = \frac{2}{3}\int_{-1}^{2} f(x)\cos\left(\frac{2n\pi x}{3}\right)dx, \qquad n = 1, 2 \dots 10$$

$$B_n = \frac{2}{3}\int_{-1}^{2} f(x)\sin\left(\frac{2n\pi x}{3}\right)dx, \qquad n = 1, 2 \dots 10$$

```
An: (A1 to A10)
   -1.2829     0.2995     0.1013    -0.2352     0.1472     0.0253    -0.1274     0.0963     0.0113    -0.0873

Bn: (B1 to B10)
   -0.3123     0.4362    -0.3183     0.0700     0.1271    -0.1592     0.0521     0.0720    -0.1061     0.0398

A0:
   7.4015e-17
```

```
% Define the original function
f = @(x) x.^2 - 1;

% Define the period
T = 3;

% Calculate a0
a0 = (1/T)*integral(f, -1, 2);

% Number of terms in the Fourier series
N = 10;

% Generate X values for plotting
X0 = linspace(-1, 2, 90);
X = linspace(-10, 10, 600);

% Initialize Fourier series function
F = zeros(size(X));
F = F + a0/2;
% Compute the Fourier series approximation
a_n = zeros(1, N);
b_n = zeros(1, N);
for n = 1:N
    a_n(n) = (2/T)*integral(@(x) f(x).*cos(2*pi*n*x/T), -1, 2);
    b_n(n) = (2/T)*integral(@(x) f(x).*sin(2*pi*n*x/T), -1, 2);
    F = F + a_n(n) * cos(2*pi*n*X/T) + b_n(n) * sin(2*pi*n*X/T);
end
```

```
disp('An: (A1 to A10)')
disp(a_n);
disp('Bn: (B1 to B10)')
disp(b_n);
disp('A0:')
disp(a0);

% Compute Y for the original function
Y0 = f(X0);

% Plot the original function and its Fourier series approximation
plot(X, F, 'r', 'LineWidth', 2); % Fourier series approximation
hold on;
plot(X0, Y0, 'b', 'LineWidth', 2); % Original function
hold off;
xlabel('x');
ylabel('y');
title('Fourier Series Approximation of f(x)');
legend('Original Function', 'Fourier Series Approximation');
grid on;
```