Table of Contents

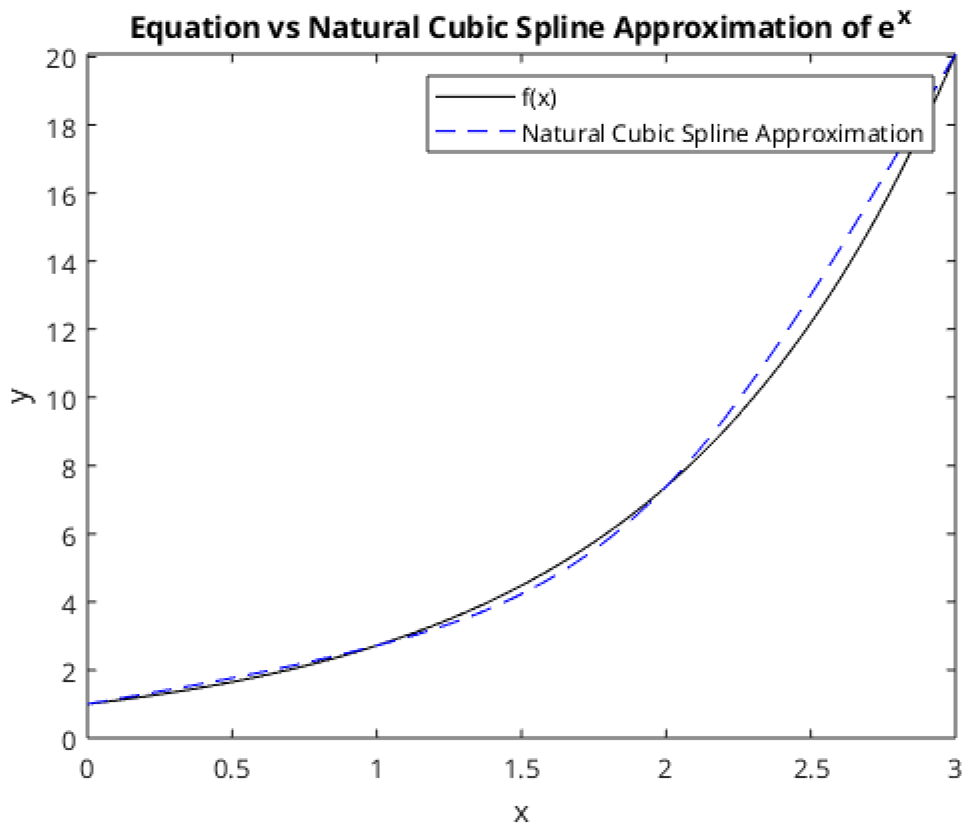
Part 1

Form a natural cubic spline of e^x

```
fprintf("Part 1: Form a natural cubic spline of e^x\n");

% Input points here
x = [0; 1; 2; 3];
% Input equation here
a = exp(x);

h = x(2:end) - x(1:end-1);

matrixAplus1 = diag([0; h(2:end)], 1); % Finds diagonal+1 of A
matrixAmiddle = diag([1; (2*(h(1:end-1) + h(2:end))); 1], 0); % Finds
 diagonal of A
matrixAminus1 = diag([h(1:end-1); 0], -1); % Finds diagonal-1 of A
matrixA = matrixAplus1 + matrixAmiddle + matrixAminus1;

matrixB = [0; (3./h(2:end)).*(a(3:end)-a(2:end-1))-(3./
h(1:end-1)).*(a(2:end-1)-a(1:end-2)); 0];

c = linsolve(matrixA, matrixB);

b = (a(2:end)-a(1:end-1))./h(1:end) -
 (c(2:end)+2.*c(1:end-1)).*(h(1:end)./3);

d = (c(2:end)-c(1:end-1))./(3.*h(1:end));

% z is the value to estimate, y is which piecewise equation to use
S = @(z, y) a(y) + (b(y) * (z-x(y))) + (c(y) * (z-x(y))^2) + (d(y) *
 (z-x(y))^3);

w = linspace(x(1), x(end), 100);
Sw = zeros(size(w, 2), 1);
i = 1;
Sw(1) = S(w(1), 1);
for j = 2:size(w, 2)
    Sw(j) = S(w(j), i);
    % w must contain all the nodes (x)
    if w(j) == x(i+1)
        i = i + 1;
    end
end
figure;
```

```
plot(w, exp(w), '-k');
hold on;
plot(w, Sw, '--b');
hold off;
xlabel('x');
ylabel('y');
title('Equation vs Natural Cubic Spline Approximation of e^x');
legend('f(x)', 'Natural Cubic Spline Approximation');
```

```
Part 1: Form a natural cubic spline of e^x
```



Part 2

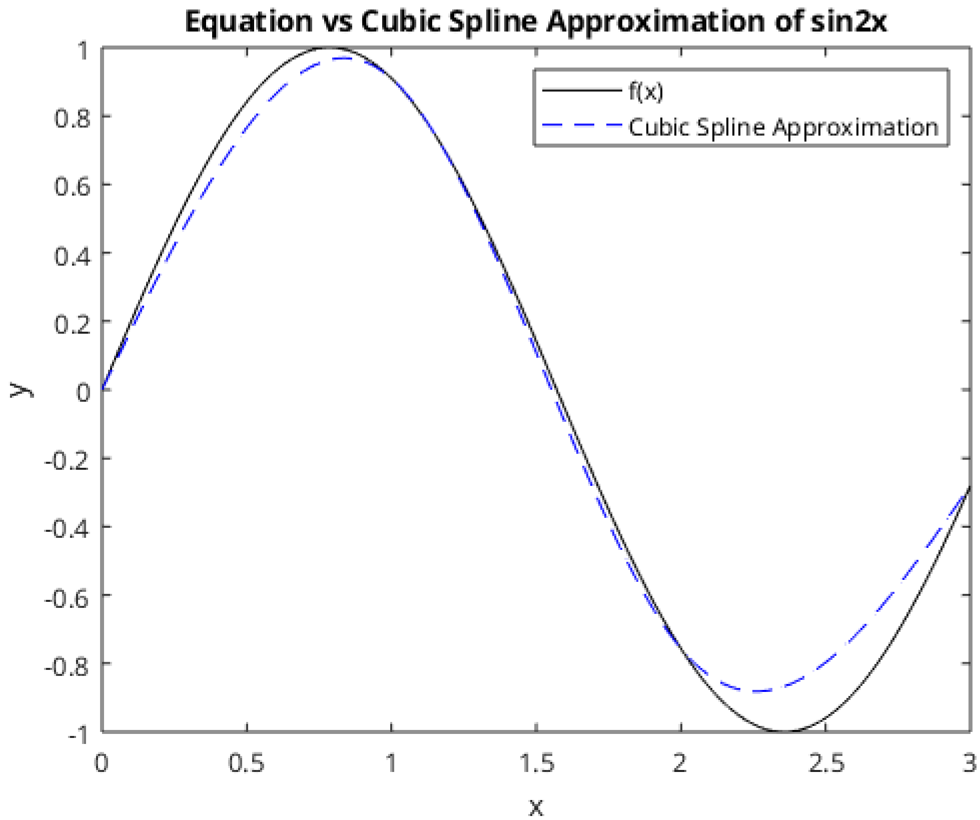
Form a natural cubic spline of sin2x

```
fprintf("Part 2: Form a natural cubic spline of sin2x\n");

% Input points here
n = randi([3 10], 1, 1);
x = 0:n;
fprintf("N = %.f\n", n);
x = x';
% Input equation here
a = sin(2*x);

h = x(2:end) - x(1:end-1);
```

```
matrixAplus1 = diag([0; h(2:end)], 1); % Finds diagonal+1 of A
matrixAmiddle = diag([1; (2*(h(1:end-1) + h(2:end))); 1], 0); % Finds
 diagonal of A
matrixAminus1 = diag([h(1:end-1); 0], -1); % Finds diagonal-1 of A
matrixA = matrixAplus1 + matrixAmiddle + matrixAminus1;

matrixB = [0; (3./h(2:end)).*(a(3:end)-a(2:end-1))-(3./
h(1:end-1)).*(a(2:end-1)-a(1:end-2)); 0];

c = linsolve(matrixA, matrixB);

b = (a(2:end)-a(1:end-1))./h(1:end) -
 (c(2:end)+2.*c(1:end-1)).*(h(1:end)./3);

d = (c(2:end)-c(1:end-1))./(3.*h(1:end));

% z is the value to estimate, y is which piecewise equation to use
S = @(z, y) a(y) + (b(y) * (z-x(y))) + (c(y) * (z-x(y))^2) + (d(y) *
 (z-x(y))^3);

% need to multiply by n then add 1 to ensure that the nodes are points
 in w
w = linspace(x(1), x(end), 1 + (n*100));
Sw = zeros(size(w, 2), 1);
i = 1;
Sw(1) = S(w(1), 1);
for j = 2:size(w, 2)
    Sw(j) = S(w(j), i);
    % w must contain all the nodes (x)
    if w(j) == x(i+1)
        i = i + 1;
    end
end
figure;
plot(w, sin(2*w), '-k');
hold on;
plot(w, Sw, '--b');
hold off;
xlabel('x');
ylabel('y');
title('Equation vs Cubic Spline Approximation of sin2x');
legend('f(x)', 'Cubic Spline Approximation');
```

Part 2: Form a natural cubic spline of sin2x

N = 3



Part 3

Predict Kentuckey Derby times and speeds

```
fprintf("Part 3: Predict Kentuckey Derby times and speeds\n");

% Input points here
x = [0; 1/4; 1/2; 1; 5/4];
% Input equation here
a = [0; 23.04; 47.37; 97.45; 123.66];

h = x(2:end) - x(1:end-1);

matrixAplus1 = diag([0; h(2:end)], 1); % Finds diagonal+1 of A
matrixAmiddle = diag([1; (2*(h(1:end-1) + h(2:end))); 1], 0); % Finds
 diagonal of A
matrixAminus1 = diag([h(1:end-1); 0], -1); % Finds diagonal-1 of A
matrixA = matrixAplus1 + matrixAmiddle + matrixAminus1;

matrixB = [0; (3./h(2:end)).*(a(3:end)-a(2:end-1))-(3./
h(1:end-1)).*(a(2:end-1)-a(1:end-2)); 0];

c = linsolve(matrixA, matrixB);

b = (a(2:end)-a(1:end-1))./h(1:end) -
 (c(2:end)+2.*c(1:end-1)).*(h(1:end)./3);
```

```
d = (c(2:end)-c(1:end-1))./(3.*h(1:end));

fprintf("Matrix of cubic coefficients\n");
answerMatrix = [a(1:end-1), b, c(1:end-1), d]
% z is the value to estimate, y is which piecewise equation to use
S = @(z, y) a(y) + (b(y) * (z-x(y))) + (c(y) * (z-x(y))^2) + (d(y) *
 (z-x(y))^3);

w = linspace(x(1), x(end), 100);
Sw = zeros(size(w, 2), 1);
i = 1;
Sw(1) = S(w(1), 1);
for j = 2:size(w, 2)
    Sw(j) = S(w(j), i);
    % w must contain all the nodes (x)
    if w(j) == x(i+1)
        i = i + 1;
    end
end

fprintf("Predicted 3/4 mile time: %.2f\n", S(3/4, 3));
fprintf("Relative Error: %.8f\n", abs((71.8-S(3/4, 3))/71.8));
% At the start x = 0 so the derived equation will be equal to b
% b is seconds per mile so 1/b will be miles per second
fprintf("Predicted Start Speed: %.4f miles per second\n", 1/b(1));
% derivative done by hand for this equation
endspeed = b(4)*(5/4) + 2*c(4)*(5/4) - 2*c(4) + 3*d(4)*(5/4)^2 -
 6*d(4)*(5/4) + 3*d(4);
fprintf("Predicted End Speed: %.4f miles per second\n", 1/endspeed);
```

```
Part 3: Predict Kentuckey Derby times and speeds
Matrix of cubic coefficients

answerMatrix =

         0   90.8695         0   20.6479
   23.0400   94.7410   15.4859  -20.6793
   47.3700   98.6066   -0.0236    6.2610
   97.4500  103.2787    9.3679  -12.4905

Predicted 3/4 mile time: 72.12
Relative Error: 0.00442886
Predicted Start Speed: 0.0110 miles per second
Predicted End Speed: 0.0076 miles per second
```

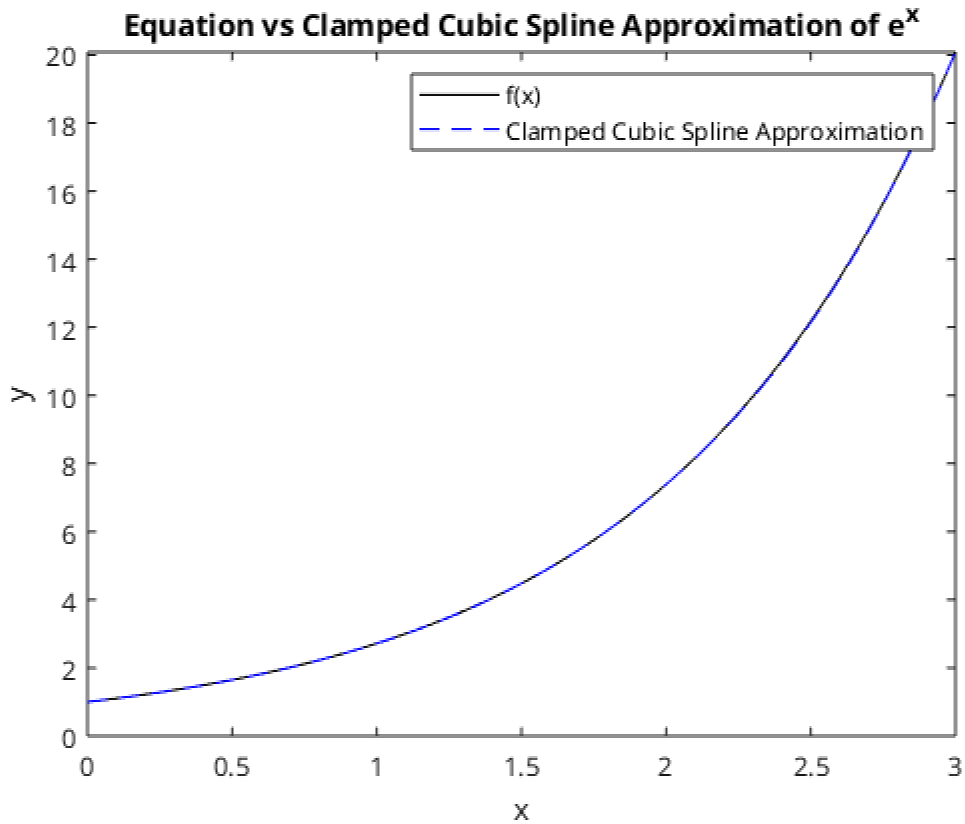
Part 4

Form a clamped cubic spline of e^x

```
fprintf("Part 4: Form a clamped cubic spline of e^x\n");

% Input points here
```

```
x = [0; 1; 2; 3];
% Input equation here
a = exp(x);

h = x(2:end) - x(1:end-1);

matrixAplus1 = diag(h(1:end), 1); % Finds diagonal+1 of A
matrixAmiddle = diag([2*h(1); (2*(h(1:end-1) + h(2:end))); 2*h(end)],
 0); % Finds diagonal of A
matrixAminus1 = diag(h(1:end), -1); % Finds diagonal-1 of A
matrixA = matrixAplus1 + matrixAmiddle + matrixAminus1;

matrixB = [(3/h(1)*(a(2)-a(1))) - 3*(exp(x(1))); (3./
h(2:end)).*(a(3:end)-a(2:end-1))-(3./h(1:end-1)).*(a(2:end-1)-
a(1:end-2)); 3*(exp(x(end))) - (3/h(end)*(a(end)-a(end-1)))];

c = linsolve(matrixA, matrixB);

b = (a(2:end)-a(1:end-1))./h(1:end) -
 (c(2:end)+2.*c(1:end-1)).*(h(1:end)./3);

d = (c(2:end)-c(1:end-1))./(3.*h(1:end));

% z is the value to estimate, y is which piecewise equation to use
S = @(z, y) a(y) + (b(y) * (z-x(y))) + (c(y) * (z-x(y))^2) + (d(y) *
 (z-x(y))^3);

w = linspace(x(1), x(end), 100);
Sw = zeros(size(w, 2), 1);
i = 1;
Sw(1) = S(w(1), 1);
for j = 2:size(w, 2)
    Sw(j) = S(w(j), i);
    % w must contain all the nodes (x)
    if w(j) == x(i+1)
        i = i + 1;
    end
end
figure;
plot(w, exp(w), '-k');
hold on;
plot(w, Sw, '--b');
hold off;
xlabel('x');
ylabel('y');
title('Equation vs Clamped Cubic Spline Approximation of e^x');
legend('f(x)', 'Clamped Cubic Spline Approximation');
```

Part 4: Form a clamped cubic spline of e^x



Published with MATLAB® R2020a