```
function pp  = math565_build_spline(xd,yd,ec,ed)

global xdata ydata end_cond end_data

% Set up global variables needed for other routines
xdata = xd;
ydata = yd;
end_cond = ec;
end_data = ed;

% Solve for the end point derivatives d = [d0,d1] using Newton's Method
d0 = [0;0];  % Starting value

% TODO : Compute the Jacobian J = F'(d), 2x2 matrix

J = zeros(2,2);

% Get column 1 of J :
 J(:,1) =  F(d0 + [1;0]) - F(d0);

% Get column two of J :
J(:,2) = F(d0 + [0; 1]) - F(d0);

% TODO : Setup Newton iteration to solve for d = [d0,d1]
dk = [0;0];  % Starting value

% Newton iteration :

dk = dk -J\F(dk);

d = dk;    % Change this to the correct value for d

% Compute derivatives using d=[d(1),d(2)] from above
ddata = compute_derivs(d);

% Get the spline coefficients
coeffs = spline_coeffs(ddata);

% Use Matlab function mkpp to set up spline
pp = mkpp(xdata,coeffs);

end

% Function F(d) :    Use Newton's method to solve F(d) = 0
function Fd = F(d)

global xdata ydata end_cond end_data

ddata = compute_derivs(d);

% Spline coefficients for this choice of end point values
coeffs = spline_coeffs(ddata);

% Four coefficients for the cubic in the first segment :
% p0(x) = a(1)*(x-x0)^3 + a(2)*(x-x0)^2 + a(3)(x-x0) + a(4)
a = coeffs(1,:);

% Four coefficients for the cubic in the last segment :
% p_{N-1}(x) = b(1)*(x-x_N-1)^3 + b(2)*(x-x_N-1)^2 + b(3)(x-x_N-1) + b(4)
b = coeffs(end,:);

% TODO : Use a and b to compute the first and second derivatives
% of the spline interplant at the endpoints  x_0 and x_N
```

```
p0_deriv_x0 = a(3);
p0_deriv2_x0 = 2*a(2);

 pNm1_deriv_xN = 3*b(1)*(xdata(end) - xdata(end-1))^2 + 2*b(2)*(xdata(end)-xdata(end-1)) + b(3);
 pNm1_deriv2_xN =  6*b(1)*(xdata(end) - xdata(end-1)) + 2*b(2);

% TODO : Define function F(d) :
Fd = [0;0];
switch end_cond
    case 'natural'
       Fd(1) = p0_deriv2_x0;
       Fd(2) = pNm1_deriv2_xN;
    case 'clamped'
        Fd(1) = d(1);
        Fd(2) = d(2);
end

end

% Given values of d = [d0,d1], compute derivatives
% at all nodes. This will require a linear solve.
function ddata = compute_derivs(d)

global xdata ydata

N = length(xdata) - 1;
h = diff(xdata);

% Compute the derivatives at the internal nodes

% TODO : Set up a linear system to solve for interval derivatives
 A = zeros(N-1);

 for j=1:N-1
    A(j,j)= 2*(1/h(j)+1/h(j+1));
end

for j=2:N-1
    A(j,j-1) = 1/j;
    A(j-1,j)=1/j;
end


% TODO : Set up right hand side vector b
 b = zeros(N-1,1);

 for i = 2:N
    b(i-1) = 3*(1/h(i-1))^2*(ydata(i)- ydata(i-1)) + 3*(1/h(i))^2*(ydata(i+1) - ydata(i));
 end

% TODO : Solve for dH.  Use either 'backslash', or one of the other
% routines you learned in class.

dH = A\b;

% Construct vectors needed for endpoint conditions
u0 = [h(1); zeros(N-2,1)];
uN = [zeros(N-2,1); h(end)];

% Augment internal nodes with endpoint derivatives
ddata = [d(1); dH - d(1)*u0 - d(2)*uN; d(2)];
end


% Spline coefficients for each segment.  See page 14 of Lecture
% notes on Piecewise Polynomials
```

```
function coeffs = spline_coeffs(ddata)

global xdata ydata

N = length(xdata) - 1;

h = diff(xdata);
C = [-2, 3, 0, -1; 2, -3, 0, 0; -1, 2, -1, 0; -1, 1, 0, 0];
coeffs = zeros(N,4);
for k = 1:N
    p = [ydata(k); ydata(k+1); ddata(k)*h(k); ddata(k+1)*h(k)];
    S = diag(1./h(k).^(3:-1:0));
    coeffs(k,:) = -p'*C*S;
end

end
```

```
Not enough input arguments.

Error in math565_build_spline (line 6)
xdata = xd;
```

Published with MATLAB® R2020a