

MATH 335: Numerical Analysis

Problem Set 12, Solutions

★20 (a) First,

$$A_i(x_j) = [1 - 2(x_j - x_i)L'_i(x_i)][L_i(x_j)]^2 = [1 - 2(x_j - x_i)L'_i(x_i)]\delta_{ij} = \delta_{ij}.$$

Next, we differentiate

$$A'_i(x) = 2[1 - 2(x - x_i)L'_i(x_i)]L_i(x)L'_i(x) - 2L'_i(x_i)[L_i(x)]^2$$

from which it follows that

$$A'_i(x_j) = 2\delta_{ij} \{ [1 - 2(x_j - x_i)L'_i(x_i)]L'_i(x_j) - L'_i(x_i) \} = 0.$$

Now to the B_i 's. We have

$$B_i(x_j) = (x_j - x_i)\delta_{ij},$$

which is plainly zero both when $j = i$ and when $i \neq j$. Differentiating, we have

$$B'_i(x) = [L_i(x)]^2 + 2(x - x_i)L_i(x)L'_i(x),$$

so

$$B'_i(x_j) = \delta_{ij} + 2(x_j - x_i)\delta_{ij}L'_i(x_j) = \delta_{ij}.$$

(b) First, we note that, since each cardinal function L_i is a polynomial of degree $(n - 1)$, the resulting A_i and B_i are both polynomials of degree of $(2n - 1)$. Thus, the polynomial defined by

$$p(x) = \sum_{j=1}^n [c_{j0}A_j(x) + c_{j1}B_j(x)]$$

is a polynomial of degree at most $(2n - 1)$. Moreover,

$$p(x_i) = \sum_{j=1}^n [c_{j0}A_j(x_i) + c_{j1}B_j(x_i)] = \sum_{j=1}^n c_{j0}\delta_{ji} = c_{i0}.$$

To see that the derivative conditions are met, we note that

$$p'(x_i) = \sum_{j=1}^n [c_{j0}A'_j(x_i) + c_{j1}B'_j(x_i)] = \sum_{j=1}^n c_{j1}\delta_{ji} = c_{i1}.$$

- (c) The key to understanding my algorithm is to see that a cardinal function $L_i(x)$ is the unique polynomial of degree $(n - 1)$ which passes through the points $(x_1, \delta_{1i}), \dots, (x_n, \delta_{ni})$. So, we can employ code like

```
n = length(x);
I = eye(n);
cardFnj = [cardFns; polyfit(x, I(:, j), n-1)];
```

to build a row vector containing the coefficients of of the j^{th} cardinal function L_j . My code, then, looks like this:

```
function vals = hermite (x, y, yp, xi)
% function vals = hermite (x, y, yp, xi)
%
% Routine finds an interpolating polynomial p that, for
% each i between 1 and length(x), satisfies
%   p(x(i)) = y(i)   and   p'(x(i)) = yp(i).
%
% INPUTS:
%   x   vector of nodes
%   y   vector of values p should take at nodes
%   yp  vector of values p' should take at nodes
%   xi  vector of numbers at which the values of p are desired
% OUTPUTS:
%   The values of this interpolating polynomial p at numbers in xi

% check the shape of the inputs
if (rows(x) == 1), x = x'; end
if (rows(y) == 1), y = y'; end
if (rows(yp) == 1), yp = yp'; end

n = length(x);
I = eye(n);
p = zeros(1, 2*n)
for j = 1:n;
    cardFnj = polyfit(x, I(:, j), n-1);
    cardPrimej = polyder(cardFnj);
    Aj = conv([0 1]-2*polyval(cardPrimej, x(j))*[1 -x(j)], ...
              conv(cardFnj, cardFnj));
    Bj = conv([1 -x(j)], conv(cardFnj, cardFnj));
    p = p + y(j)*Aj + yp(j)*Bj;
end
```

```
vals = polyval(p, xi);
end
```

(d) Since $c_{10} = c_{11} = c_{21} = 0$ in this problem, we have

$$p(x) = c_{20}A_2(x) = 2 \cdot \frac{1}{16} x^2 \left(3 - \frac{1}{2} x\right) = \frac{1}{16} x^2(6 - x).$$

★21 (b) I'll answer (b) first, as this will provide a method for (a). A polynomial p of degree k has the zero function as its $(k + 1)^{\text{st}}$ derivative. Thus, by Taylor's theorem,

$$p(x) = \sum_{j=0}^k \frac{p^{(j)}(x_0)}{j!} (x - x_0)^j + \frac{f^{(k+1)}(\xi)}{(k+1)!} (x - x_0)^{k+1} = \sum_{j=0}^k \frac{p^{(j)}(x_0)}{j!} (x - x_0)^j,$$

which is a polynomial of that is "offset" by x_0 .

(a) Applying the method above to $p(x) = x^3 + x - 1$, we have

$$\begin{aligned} p(x) &= x^3 + x - 1 &\Rightarrow & p(2) = 9 \\ p'(x) &= 3x^2 + 1 &\Rightarrow & p'(2) = 13 \\ p''(x) &= 6x &\Rightarrow & p''(2) = 12 \\ p'''(x) &= 6 &\Rightarrow & p'''(2) = 6 \end{aligned}$$

Thus,

$$x^3 + x - 1 = (x - 2)^3 + 6(x - 2)^2 + 13(x - 2) + 9.$$

★22 (c) My program to do those things required of parts (a)–(c):

```
function vals = myspl3 (x, y, xi, z1=0, zn=0)
# function vals = myspl3 (x, y, xi, z1=0, zn=0)
#
# Routine computes values at xi of the cubic spline
# S computed for interpolation points with abscissas
# in x, ordinates in y. For the two additional
# conditions it employs
# S''(x(1)) = z1 and S''(x(end)) = zn.
#
# Inputs:
# x      vector of nodes
# y      ordinates of interpolating cubic spline at nodes
# xi     vector of numbers at which value of cubic spline is sought
```

```

# z1      2nd derivative value of cubic spline at left endpt
# zn      2nd derivative value of cubic spline at right endpt
# Outputs:
# vals    vector of cubic spline values computed at xi

# check shape of input vectors
if (rows(x) == 1), x = x'; end
if (rows(y) == 1), y = y'; end
if (rows(xi) == 1), xi = xi'; end

n = length(x);
h = diff(x);
fdivdiffs = diff(y) ./ h;

# compute vector of 2nd derivative values for S
A = diag([1; 2*h(1:n-2); 1]) + diag([0; 2*h(2:n-1); 0]) ...
    + diag([h(1:n-2); 0], -1) + diag([0; h(2:n-1)], 1);
q = [z1; 6*diff(fdivdiffs); zn];
z = A \ q;

# compute coefficients of piecewise cubics
a = y(1:n-1);
b = fdivdiffs - h.*(2 * z(1:n-1) + z(2:n)) / 6;
c = z(1:n-1) / 2;
d = diff(z) ./ h / 6;

coeffs = [d(:), c(:), b(:), a(:)];
pp = mkpp(x, coeffs);
vals = ppval(pp, xi);
end

```

- (d) From class notes, we have the individual pieces of the cubic spline S are $S_i(x) = d_i(x - x_i)^3 + c_i(x - x_i)^2 + b_i(x - x_i) + a_i$ where, for $i = 1, \dots, n - 1$, the coefficients b_i, c_i, d_i satisfy the equations

$$\begin{aligned}
 b_i &= f[x_i, x_{i+1}] - \frac{h_i}{6}(2z_i + z_{i+1}), \\
 c_i &= \frac{1}{2}z_i, \\
 d_i &= \frac{1}{6h_i}(z_{i+1} - z_i).
 \end{aligned}$$

Let y'_1, y'_n denote the user-specified values for $S'(x_1)$ and $S'(x_n)$. We have

$$y'_1 = S'(x_1) = S'_1(x_1) = b_1 = f[x_1, x_2] - \frac{h_1}{6}(2z_1 + z_2).$$

Paying attention only to the extreme ends of this equation, we may rearrange it to say

$$(2z_1 + z_2)h_1 = 6(f[x_1, x_2] - y'_1).$$

The other condition translates as

$$\begin{aligned} y'_n &= S'(x_n) = S'_{n-1}(x_n) = 3d_{n-1}h_{n-1}^2 + 2c_{n-1}h_{n-1} + b_{n-1} \\ &= \frac{1}{2}(z_n - z_{n-1})h_{n-1} + z_{n-1}h_{n-1} + f[x_{n-1}, x_n] - \frac{h_{n-1}}{6}(2z_{n-1} + z_n) \end{aligned}$$

$$\begin{aligned} \Rightarrow 6(y'_n - f[x_{n-1}, x_n]) &= [3(z_n - z_{n-1}) + 6z_{n-1} - (2z_{n-1} + z_n)]h_{n-1} \\ &= (z_{n-1} + 2z_n)h_{n-1}. \end{aligned}$$

Thus, our routine should start out by solving the tridiagonal matrix system $\mathbf{Az} = \mathbf{q}$, where $z = (z_1, z_2, \dots, z_n)$,

$$\mathbf{A} = \begin{bmatrix} 2h_1 & h_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ h_1 & 2(h_1 + h_2) & h_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & h_2 & 2(h_2 + h_3) & h_3 & & 0 & 0 & 0 \\ \vdots & & \ddots & \ddots & \ddots & & & \\ 0 & 0 & 0 & 0 & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix},$$

and

$$\mathbf{q} = 6 \begin{bmatrix} f[x_1, x_2] - y'_1 \\ f[x_2, x_3] - f[x_1, x_2] \\ f[x_3, x_4] - f[x_2, x_3] \\ \vdots \\ f[x_{n-1}, x_n] - f[x_{n-2}, x_{n-1}] \\ y'_n - f[x_{n-1}, x_n] \end{bmatrix}.$$

My program, then, looks like this:

```
function vals = clampedSpl3 (x, y, xi, yprime1, yprimen)
# function vals = myspl3 (x, y, xi, yprime1, yprimen)
#
# Routine computes values at xi of the cubic spline
# S computed for interpolation points with abscissas
```

```

# in x, ordinates in y. For the two additional
# conditions it employs
#   S'(x(1)) = yprime1 and S'(x(end)) = yprimen.
#
# Inputs:
# x      vector of nodes
# y      ordinates of interpolating cubic spline at nodes
# xi     vector of numbers at which value of cubic spline is sought
# yprime1 1st derivative value of cubic spline at left endpt
# yprimen 1st derivative value of cubic spline at right endpt
# Outputs:
# vals   vector of cubic spline values computed at xi

# check shape of input vectors
if (rows(x) == 1), x = x'; end
if (rows(y) == 1), y = y'; end
if (rows(xi) == 1), xi = xi'; end

n = length(x);
h = diff(x);
fdivdiffs = diff(y) ./ h;

# compute vector of 2nd derivative values for S
A = diag([0; 2*h]) + diag([2*h; 0]) + diag(h, -1) + diag(h, 1);
q = 6*[fdivdiffs(1)-yprime1; diff(fdivdiffs); yprimen-fdivdiffs(n-1)];
z = A \ q;

# compute coefficients of piecewise cubics
a = y(1:n-1);
b = fdivdiffs - h.*(2 * z(1:n-1) + z(2:n)) / 6;
c = z(1:n-1) / 2;
d = diff(z) ./ h / 6;

coeffs = [d(:), c(:), b(:), a(:)];
pp = mkpp(x, coeffs);
vals = ppval(pp, xi);
end

```