**P3.2.2** Write a recursive function

analogous to. Use the same interval acceptance tests as in. The string is name of the derivative function.

Use both and to produce approximations to on the interval. Fix. Print a table that shows the number of partition points computed by and for.

+ Function:

function [x,y,s] = pwCAdapt(fname,fpname,L,fL,DfL,R,fR,DfR,delta,hmin)

% [x,y,s] = pwCAdapt(fname,fpname,L,fL,DfL,R,fR,DfR,delta,hmin)

% Adaptively determines interpolation points for a piecewise cubic hermite

% approximation of a specified function.

%

% fname is a string that specifies a function of the form y = f(u).

% fpname is a string that names a function that is the derivative of f.

% L is real and fL = f(L) and DfL = f'(L).

% R is real and fR = f(R) and DfR = f'(R).

% delta is a positive real

% hmin is a positive real

%

% x is a column n-vector with the property that

% L = x(1) < ... < x(n) = R

% each subinterval is either <= hmin in length or

% has the property that at its midpoint m, |f(m) - q(m)| <= delta,

% where q(x) is the cubic hermite interpolant of (L,fL,DfL) and (R,fR,DfR).

%

% y is a column n-vector with the property that y(i) = f(x(i)).

%

% s is a column n-vector with the property that s(i) = f'(x(i)).

if (R-L) <= hmin

% Subinterval is acceptable

x = [L;R];

y = [fL;fR];

s = [DfL;DfR];

else

mid = (L+R)/2;

fmid = feval(fname,mid);

% Compute the cubic hermite interpolant and evaluate at the midpoint:

% function HCubic in page 117

[alfa,beta,gamma,eta] = HCubic(L,fL,DfL,R,fR,DfR);

% function pwCEval in page 119 and 120

qeval = pwCEval(alfa,beta,gamma,eta,[L;R],mid);

if abs(qeval - fmid) <= delta

% Subinterval acceptable

x = [L;R];

y = [fL;fR];

s = [DfL;DfR];

else

% Produce left and right partitions, then synthesize without

% redundancy.

smid = feval(fpname,mid);

[xLeft,yLeft,sLeft] = pwCAdapt(fname,fpname,L,fL,DfL,mid,fmid,smid,delta,hmin);

[xRight,yRight,sRight] = pwCAdapt(fname,fpname,mid,fmid,smid,R,fR,DfR,delta,hmin);

x = [ xLeft;xRight(2:length(xRight))];

y = [ yLeft;yRight(2:length(yRight))];

s = [ sLeft;sRight(2:length(sRight))];

end

end

+ Script P3\_2\_2

% P3.2.2

% Compare pwLAdapt and pwCAdapt on f(x) = sqrt(x) on [.001, 9]

% dfdx(x) = .5./sqrt(x)

hmin = .001;

L = .001; fL = f(L); DfL = dfdx(L);

R = 9.000; fR = f(R); DfR = dfdx(R);

disp(' pwL pwC')

disp(' delta length(x) length(x)')

disp('--------------------------------------')

close all

for delta = [.1 .01 .001 .0001 .00001]

[xl,yl] = pwLAdapt('f',L,fL,R,fR,delta,hmin);

nL = length(xl);

[xc,yc,sc] = pwCAdapt('f','dfdx',L,fL,DfL,R,fR,DfR,delta,hmin);

nC = length(xc);

fprintf(' %7.5f %3.0f %3.0f\n',delta,nL,nC);

figure

plot(xl,yl,'ro',xc,yc,'\*')

legend('Linear','Cubic',2)

pause

end

+ Result:

>> p3\_2\_2

pwL pwC

delta length(x) length(x)

--------------------------------------

0.10000 7 9

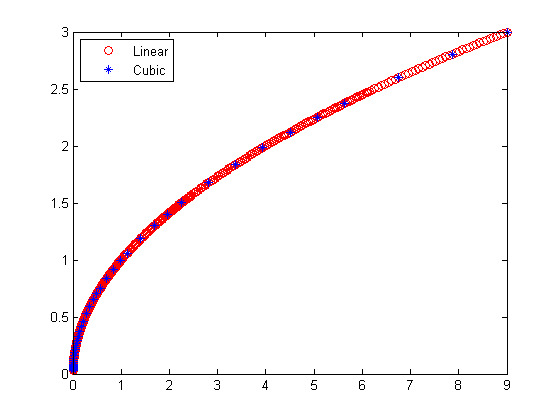
0.01000 17 12

0.00100 51 15

0.00010 159 25

0.00001 470 42

Last figure:



**P3.3.9** Complete function:

+ Function:

function [a,b,c,d] = SmallSpline(z,y)

% z is a scalar and y is 3-vector

% a,b,c,d are column 2-vectors with the property that if

%

% S(x) = a(1) + b(1)(x-z) + c(1)(x-z)^2 + d(1)(x-z)^3 on [z-1,z]

% and

% S(x) = a(2) + b(2)(x-z) + c(2)(x-z)^2 + d(2)(x-z)^3 on [z,z+]

% then

% (a) S(z-1) = y(1), S(z) = y(2), S(z+1) = y(3),

% (b) S''(z-1) = S''(z+1) = 0

% S'(x) = 0 + b(1) + 2c(1)(x-z) + 3d(1)(x-z)^2

% S''(x) = 0 + 0 + 2c(1) + 6d(1)(x-z)

% (c) S, S', and S'' are continuous on [z-1,z+1]

% => must be continuous on x = z

S = zeros(8,1);

% S(z-1) = a(1) - b(1) + c(1) - d(1) = y(1)

T = [1 -1 1 -1 0 0 0 0];

S(1) = y(1);

% S(z) = a(1) + 0 + 0 + 0 = y(2)

T = [T ; 1 0 0 0 0 0 0 0];

S(2) = y(2);

% S(z+1) = a(2) + b(2) + c(2) + d(2) = y(3)

T = [T; 0 0 0 0 1 1 1 1];

S(3) = y(3);

% S''(z-1) = 0 + 0 + 2c(1) - 6d(1) = 0

T = [T; 0 0 2 -6 0 0 0 0];

S(4) = 0;

% S''(z+1) = 0 + 0 + 2c(2) + 6d(2) = 0

T = [T; 0 0 0 0 0 0 2 6];

S(5) = 0;

% S continuous at x = z, S1(z) = S2(z) => a(1) = a(2) or a(1) - a(2) = 0

T = [T; 1 0 0 0 -1 0 0 0];

S(6) = 0;

% S' continuous at x = z, S1'(z) = S2'(z) => b(1) = b(2) or b(1) - b(2) = 0

T = [T; 0 1 0 0 0 -1 0 0];

S(7) = 0;

% S'' continuous at x = z, S1''(z) = S2''(z) => c(1) = c(2) or c(1) - c(2) = 0

T = [T; 0 0 1 0 0 0 -1 0];

S(8) = 0;

% u = [a(1) b(1) c(1) d(1) a(2) b(2) c(2) d(2)]

u = T\S;

a = u([1 5]);

b = u([2 6]);

c = u([3 7]);

d = u([4 8]);

% Review: z is not used

+ Script P3\_3\_9:

% Problem P3\_3\_9

% Illustrate small spline.

y = [-2 ; -10 ; -3];

z = 2;

[a,b,c,d] = SmallSpline(z,y);

xL = linspace(z-1,z)';

zL = xL - z;

CL = ((d(1)\*zL + c(1)).\*zL + b(1)).\*zL + a(1);

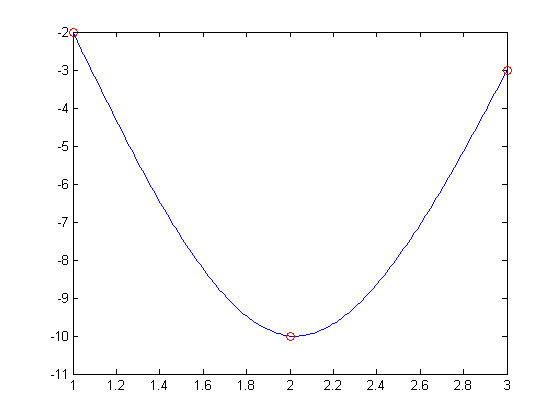
xR = linspace(z,z+1)';

zR = xR - z;

CR = ((d(2)\*zR + c(2)).\*zR + b(2)).\*zR + a(2);

plot([xL;xR],[CL;CR],[z-1;z;z+1],y,'ro')

+ Result of running script P3\_3\_9:



**P3.3.12** Suppose *x* and *y* are column *n*-vectors with. If *z* is a column *m*-vector, then is a column *m*-vector with property that, where *S* is the not-a-knot spline interpolant of . Let

be the not-a-knot spline interpolant of at

be the not-a-knot spline interpolant of at

be the not-a-knot spline interpolant of at

be the not-a-knot spline interpolant of at

Write a MATLAB script that plots in a single window these four splines across the interval. The plots should be based one-hundred, equally-spaced evaluations. Avoid unnecessary function calls. You do not have to exploit any trigonometric or exponential identities.

+ Script P3\_2\_12:

% Problem P3\_3\_12

% Illustrate not-a-knot spline interpolants using build-in function spline.

% Default: 100 points

z = linspace(0,2);

x = linspace(0,2,21);

y1 = sin(x);

y2 = exp(x);

y3 = y1.\*y2;

y4 = 2\*y1 + 3\*y2;

sval1 = spline(x,y1,z);

sval2 = spline(x,y2,z);

sval3 = spline(x,y3,z);

sval4 = spline(x,y4,z);

plot(z,sval1,'g',z,sval2,'b',z,sval3,'r',z,sval4,'m',x,y1,'go',x,y2,'bo',x,y3,'ro',x,y4,'mo');

hleg = legend('S\_1(x) = sin(x)','S\_2(x) = e^x','S\_3(x) = sin(x) + e^x','S\_4(x) = 2sin(x) + 3e^x');

set(hleg,'Location','Best');

+ Result of running script P3\_3\_12:

