Katherine Olson

998992204

MAT 128A

Due 11/9/16

MAT 128A Project II: Estimating Derivatives

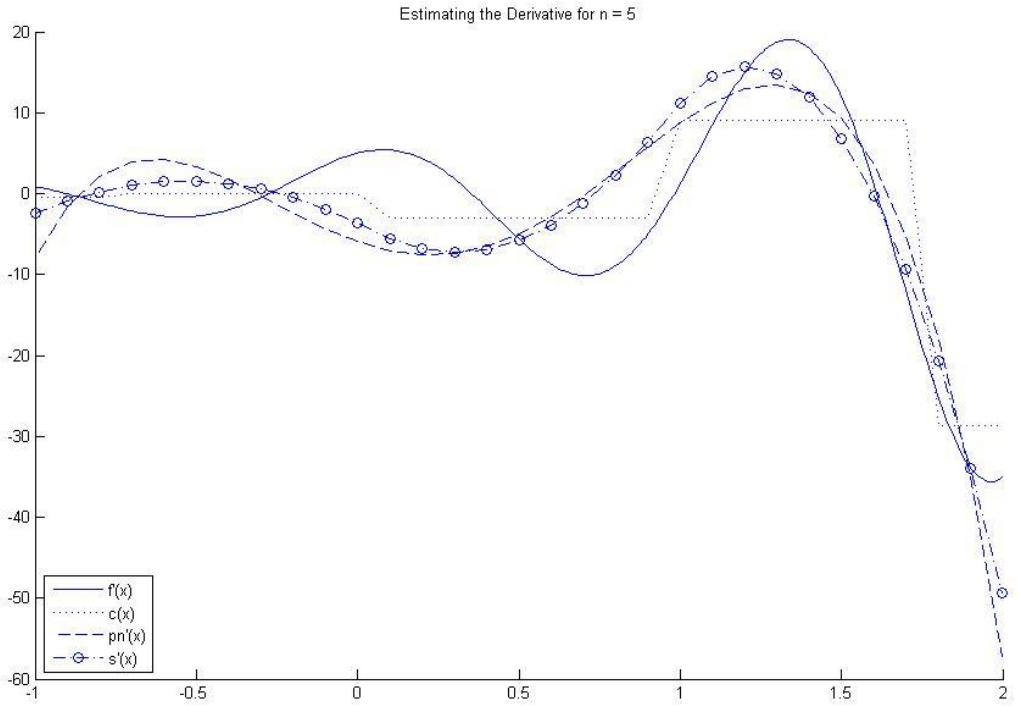
c(x) = the simple solution. pn'(x) uses a polynomial of degree n. s'(x) uses a spline polynomial.

n = 5

x	f'(x)	c(x)	pn'(x)	s'(x)
-1	8.745359576372145e-01	-5.246716814879859e-01	-8.034857368610361e+00	-2.482326239914257e+00
-0.5	-2.792582532694237e+00	-1.915057015627597e-02	3.348994875513513e+00	1.513319573792451e+00
0	5.000000000000000e+00	-1.915057015627597e-02	-5.941132081593030e+00	-3.674873907882845e+00
1	1.248742390143258e+00	9.067957670504891e+00	8.692064188507960e+00	1.118659649585994e+01
2	-3.501953550417126e+01	-2.867393229388423e+01	-5.725215416433501e+01	-4.938356795879422e+01

x	| f'(x) - c(x)|	| f'(x) - pn'(x)|	| f'(x) - s'(x)|
-1	1.399207639125200e+00	8.909393326247574e+00	3.356862197551471e+00
-0.5	2.773431962537961e+00	6.141577408207750e+00	4.305902106486688e+00
0	5.019150570156276e+00	1.094113208159303e+01	8.674873907882844e+00
1	7.819215280361633e+00	7.443321798364702e+00	9.937854105716683e+00
2	6.345603210287031e+00	2.223261866016375e+01	1.436403245462297e+01

Looking at the plot below, all three functions do a reasonable job of estimating the derivative for n = 5. c(x) resulted in a very square looking estimation, but it does a good job of peaking where the function peaks. s'(x) and pn'(x) give about the same estimation with a better curve shape, but less accurate for peaking in the right places.

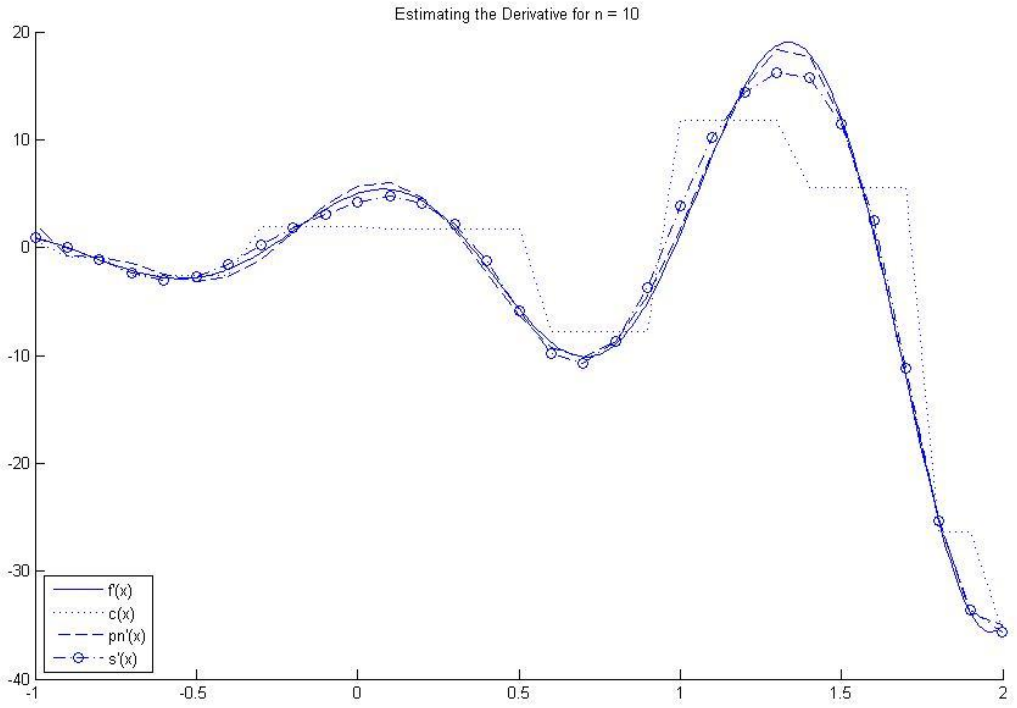


---

n = 10

x	f'(x)	c(x)	pn'(x)	s'(x)
-1	8.745359576372145e-01	5.705305450635049e-01	2.218725001336789e+00	8.935738138837546e-01
-0.5	-2.792582532694237e+00	-2.551298423618298e+00	-3.116651904244976e+00	-2.742904352948967e+00
0	5.000000000000000e+00	1.990373344139357e+00	5.720788488973929e+00	4.174280287788108e+00
1	1.248742390143258e+00	1.180595593164605e+01	1.723874252913852e+00	3.867149742653975e+00
2	-3.501953550417126e+01	-3.542814723904549e+01	-3.499203213774152e+01	-3.560134496080646e+01

x	| f'(x) - c(x)|	| f'(x) - pn'(x)|	| f'(x) - s'(x)|
-1	3.040054125737096e-01	1.344189043699575e+00	1.903785624654009e-02
-0.5	2.412841090759388e-01	3.240693715507388e-01	4.967817974526989e-02
0	3.009626655860643e+00	7.207884889739287e-01	8.257197122118924e-01
1	1.055721354150279e+01	4.751318627705943e-01	2.618407352510718e+00
2	4.086117348742278e-01	2.750336642974105e-02	5.818094566351988e-01

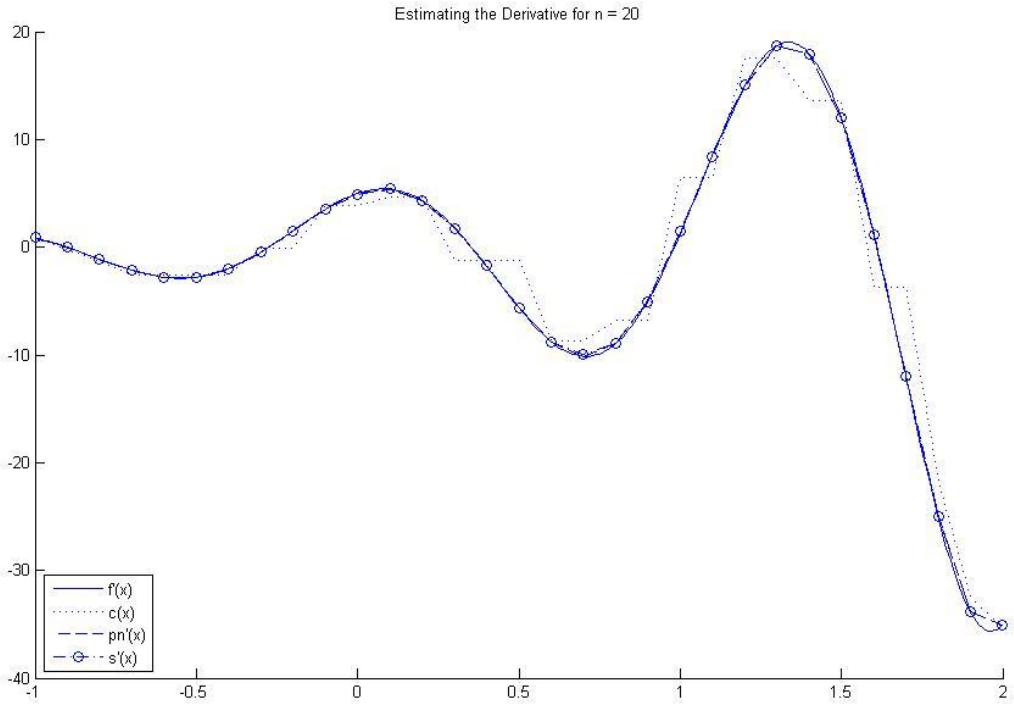


Looking at the plot above, pn'(x) and s'(x) now do a much better job of estimating the derivative with the doubled value of n, but not yet perfect. pn'(x) does a better job than s'(x). c(x) also improves, but is more off than the other two and results in a square estimation again.

n = 20

x	f'(x)	c(x)	pn'(x)	s'(x)
-1	8.745359576372145e-01	8.038904882876280e-01	8.745363820495831e-01	8.750089811994209e-01
-0.5	-2.792582532694237e+00	-2.540660536049228e+00	-2.792582552565905e+00	-2.799023894469834e+00
0	5.000000000000000e+00	3.882521455477203e+00	4.999999931101510e+00	4.937598854797884e+00
1	1.248742390143258e+00	6.453554797381901e+00	1.248741956253405e+00	1.455932549586021e+00
2	-3.501953550417126e+01	-3.528250361368088e+01	-3.501953723262420e+01	-3.502445356228202e+01

x	| f'(x) - c(x)|	| f'(x) - pn'(x)|	| f'(x) - s'(x)|
-1	7.064546934958649e-02	4.244123685825940e-07	4.730235622064649e-04
-0.5	2.519219966450090e-01	1.987166742267732e-08	6.441361775596732e-03
0	1.117478544522797e+00	6.889849046842755e-08	6.240114520211648e-02
1	5.204812407238643e+00	4.338898529354651e-07	2.071901594427632e-01
2	2.629681095096217e-01	1.728452943439152e-06	4.918058110760626e-03

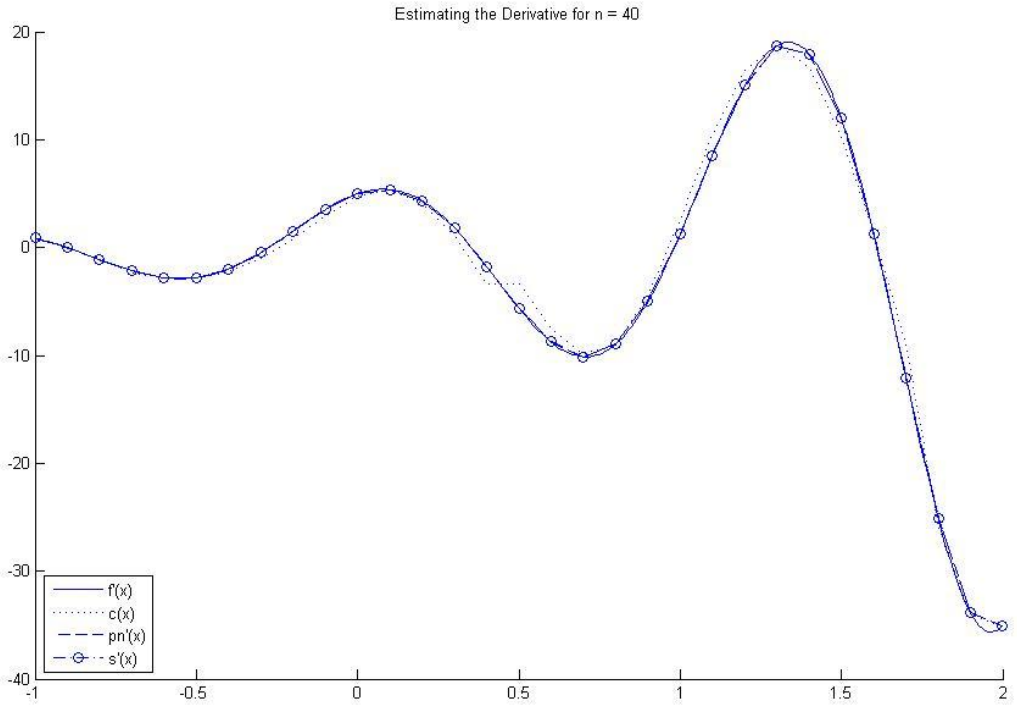


Looking at the plot, everything has improved with n doubling again. pn'(x) and s'(x) almost overlap perfectly with the function and c(x) is still a bit off and square, but it has gotten closer. Looking at the errors, pn'(x) largely outshines the other two methods with much smaller errors and s'(x) outshines c(x).

n = 40

x	f'(x)	c(x)	pn'(x)	s'(x)
-1	8.745359576372145 e-01	8.572407504582557 e-01	8.745359576372527 e-01	8.745426266809122 e-01
-0.5	-2.792582532694237 e+00	-2.823003267080257 e+00	-2.792582532694236 e+00	-2.792194024255538 e+00
0	5.000000000000000 e+00	4.742304069138739 e+00	5.000000000000001 e+00	4.995671528043381 e+00
1	1.248742390143258 e+00	2.623971738061652 e+00	1.248742390143262 e+00	1.262927855955148 e+00
2	-3.501953550417126 e+01	-3.509575763138733 e+01	-3.501953550417036 e+01	-3.501959829647231 e+01

x	| f'(x) - c(x)|	| f'(x) - pn'(x)|	| f'(x) - s'(x)|
-1	1.729520717895883e-02	3.819167204710539e-14	6.669043697726806e-06
-0.5	3.042073438601944e-02	1.332267629550188e-15	3.885084386991977e-04
0	2.576959308612610e-01	8.881784197001252e-16	4.328471956618785e-03
1	1.375229347918394e+00	4.440892098500626e-15	1.418546581189073e-02
2	7.622212721607014e-02	9.023892744153272e-13	6.279230105121769e-05



Looking at the plot for n = 40, the doubling of n has again improved all of the estimations. pn'(x) and s'(x) almost completely overlap with f'(x). The only difference that is clear is at the end near x = 2 and at the last peak around x = 1.4. c(x) improved, but is still a little off, especially in the middle. pn'(x) still has way smaller errors than the other two and s'(x) is still beating c(x).

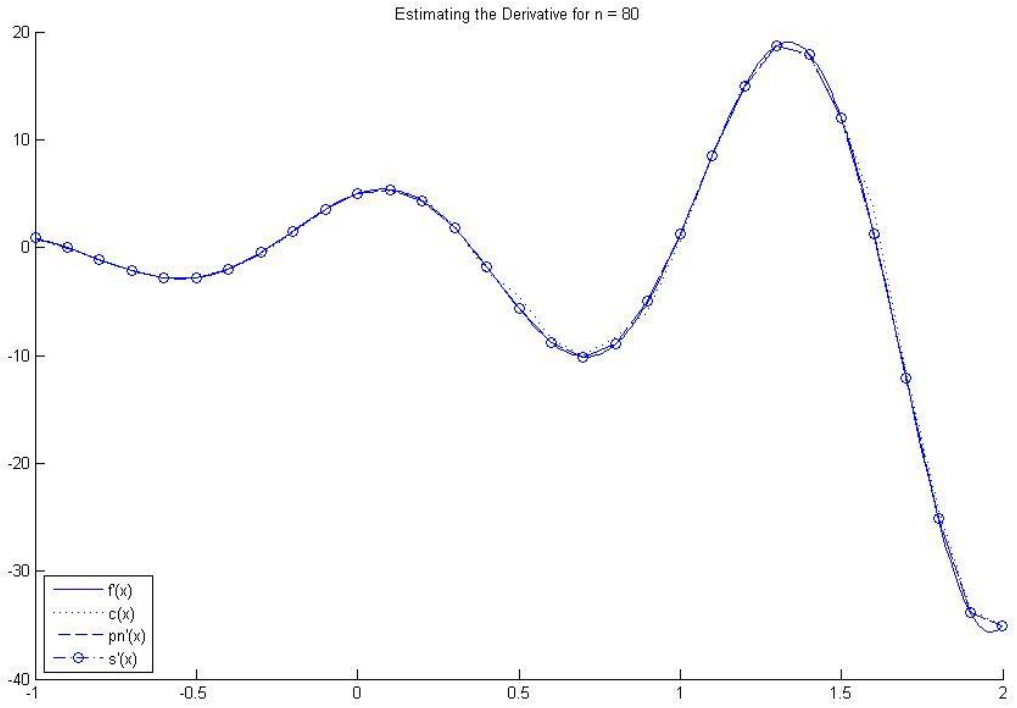
n = 80

x	f'(x)	c(x)	pn'(x)	s'(x)
-1	8.745359576372145e-01	8.702354922244022e-01	8.745359576370440e-01	8.745360589450784e-01
-0.5	-2.792582532694237e+00	-2.770805780763892e+00	-2.792582532694234e+00	-2.792614233067023e+00
0	5.000000000000000e+00	5.067618145400889e+00	5.000000000000014e+00	5.000503783324358e+00
1	1.248742390143258e+00	6.608441943543891e-01	1.248742390143244e+00	1.247068672092272e+00
2	-3.501953550417126e+01	-3.503925194784352e+01	-3.501953550416522e+01	-3.501953642819532e+01

x	| f'(x) - c(x)|	| f'(x) - pn'(x)|	| f'(x) - s'(x)|
-1	4.300465412812260e-03	1.705302565824240e-13	1.013078638756326e-07
-0.5	2.177675193034556e-02	3.552713678800501e-15	3.170037278588112e-05
0	6.761814540088906e-02	1.421085471520200e-14	5.037833243575207e-04
1	5.878981957888684e-01	1.354472090042691e-14	1.673718050985151e-03
2	1.971644367225878e-02	6.039613253960852e-12	9.240240572694347e-07

Looking at the plot on the next page, pn'(x) and s'(x) look about the same, but there was not much to improve on from the previous n. c(x) looks better, a can barely be seen at a few sections. Overall, all three estimates do a really good job.

Conclusion: For all three methods of estimating the derivative, as n got larger, the methods did a better job with error getting smaller and smaller. pn'(x) did the best job with the smallest error, followed by s'(x) and then c(x).



Code Appendix

The main script:

```
clear;
format long e;
% Getting points:
n = 80; % Just change the n here for all other ns
a = -1;
b = 2;
for i = 1:(n + 1)
  xn(i) = ((1/2) * (a + b)) + ((1/2) * (b - a) * cos((pi * (n - (i - 1))) /
n));
end
func = @(x) exp(x) * sin(5 * x);
for i = 1:(n + 1)
  yn(i) = func(xn(i));
end

% Getting all the derivative values
x = [-1 -0.5 0 1 2];
funcPrime = @(x) exp(x) * (sin(5 * x) + 5 * cos(5 * x));
for i = 1:length(x)
    real(i) = funcPrime(x(i));
end
for i = 1:length(x)
    cx(i) = Simple(x(i), xn, yn);
end
```

```
errorC = abs(real - cx);
for i = 1:length(x)
    pn(i) = Pn(x(i), xn, yn);
end
errorP = abs(real - pn);
for i = 1:length(x)
    sx(i) = SplinePrime(x(i), xn, yn);
end
errorS = abs(real - sx);

% The plots:
xVal = -1:0.1:2;
hold on;
fplot(funcPrime, [a,b])
for i = 1:length(xVal)
    cPlot(i) = Simple(xVal(i), xn, yn);
    pPlot(i) = Pn(xVal(i), xn, yn);
    sPlot(i) = SplinePrime(xVal(i), xn, yn);
end
plot(xVal, cPlot, ':')
plot(xVal, pPlot, '--')
plot(xVal, sPlot, '-.o')
legend('f''(x)', 'c(x)', 'pn''(x)', 's''(x)','Location','southwest')
title('Estimating the Derivative for n = 80')
hold off;
```

The c(x) function:

```
function [ fPrime ] = Simple( x, xn, yn )
% This function gives an approximation to f'(x) with the simplest solution
% xn and yn are length n + 1 going from n = 0, 1, ..., n
n = length(xn) - 1;
for j = 2:(n + 1)
    if ((x <= xn(j)) && (x >= xn(j - 1)))
      fPrime = (yn(j) - yn(j - 1)) / (xn(j) - xn(j - 1));
      break;
    end
end % for j

end
```

The pn'(x) function:

```
function [ fPrime ] = Pn( x, xn, yn)
% This function computes the approximation pn'(x)
% Note: I had to remove the weights from the input and calculate them in
% this function since they are not given.

n = length(xn) - 1;
wn = ones(n + 1, 1);

% Getting the weights: (from project I)
  for i = 1:(n + 1)
```

```
      for j = 1:(n + 1)
          if (j ~= i)
              wn(i) = wn(i) * (xn(i) - xn(j));
          end % if
      end % for j
      wn(i) = 1 / wn(i);
  end % for i

% Finding j:
min = 1000;
smallJ = 1000;
for j = 1:(n+1)
    left = abs(x - xn(j));
    for i = 1:(n+1)
      right = abs(x - xn(i));
      if(right < min)
           min = right;
      end
    end % for i
    if(left == min)
        newSmallJ = j;
        if(newSmallJ < smallJ)
            smallJ = newSmallJ;
        end
    end
    min = 1000;
end % for j

% Now we have the smallest j that satisfied the condition and need to plug
% into formula for pn'(x)
dj = x - xn(smallJ);
qj = 0;
for i = 1:(n+1)
  if(i ~= smallJ)
    qj = qj + ((wn(i) * yn(i)) / (x - xn(i)));
  end % if
end % for i
rj = 0;
for i = 1:(n+1)
  if(i ~= smallJ)
    rj = rj + ( wn(i) / (x - xn(i)));
  end % if
end % for i
sj = 0;
for i = 1:(n+1)
  if(i ~= smallJ)
    sj = sj + ((wn(i) * yn(i)) / ((x - xn(i))^2));
  end % if
end % for i
tj = 0;
for i = 1:(n+1)
  if(i ~= smallJ)
    tj = tj + ( wn(i) / ((x - xn(i))^2) );
  end % if
end % for i
j = smallJ;
```

```
a = wn(j) * (qj - (yn(j) * rj) + (dj * ((yn(j) * tj) - sj)));
b = dj^2 * ((qj * tj) - (rj * sj));
c = (wn(j) + (dj * rj))^2;
pnPrime = (a + b) / c;
fPrime = pnPrime;

end
```

The s'(x) function:

```
function [ fPrime ] = SplinePrime( x, xn, yn )
% This function approximates f'(x) through s'(x) the derivative of the
% not-a-knot spline
pp = spline(xn, yn);
coefs = pp.coefs;
j = 1;
k = x;
for i = (pp.order - 1):-1:1
      dcoefs(:, j) = i * coefs(:, j);
      j = j + 1;
end
dcoefs = dcoefs;
ppd = mkpp(pp.breaks, dcoefs);
d2coefs = ppd.coefs;
yyd = ppval(ppd, x);
fPrime = yyd;
end
```