Solutions to Assignment #5

1. This is a good place to reveal that the Vandermonde matrix method is built into MATLAB/OCTAVE. Specifically, I will do all three parts using "`polyfit`" and "`polyval`". The former finds coefficients of the interpolating polynomial, given the interpolation points and the degree, and the latter evaluates the polynomial at a given point (or list of points).

a.

```
>> x = [1 1.25 1.6];
>> f = @(x) sin(pi * x);
>> P = polyfit(x,f(x),2);
>> f(1.4),  polyval(P,1.4),  abs(polyval(P,1.4) - f(1.4))
ans =   -0.951056516295154
ans =   -0.918228061740598
ans =   0.0328284545545553
```

b.

```
>> f = @(x) log10(3*x-1);
>> P = polyfit(x,f(x),2);
>> f(1.4),  polyval(P,1.4),  abs(polyval(P,1.4) - f(1.4))
ans =    0.505149978319906
ans =    0.507122062831104
ans =  0.00197208451119779
```

c.

```
>> f = @(x) (x-1).^(1/3);
>> P = polyfit(x,f(x),2);
>> f(1.4),  polyval(P,1.4),  abs(polyval(P,1.4) - f(1.4))
ans =    0.736806299728077
ans =    0.816944670038153
ans =   0.0801383703100754
```

So, how are `polyfit`/`polyval` *implemented? You may think of them as being these codes, which should be quite understandable.*

polyfitbyed.m

```
function P = polyfitbyed(x,y,n)
% POLYFITBYED  Re-implementation by Ed of the built-in polyfit.
%     This version is more limited than built-in polyfit because
%     only the  degree = (number of points - 1)  case is covered.
%
% Example of comparison:
%   >> x = [1 2 5 6]
%   >> polyfit(x,sin(x),3)
%   >> polyfitbyed(x,sin(x),3)
%
% See also: POLYVALBYED

if length(x) ~= (n+1), error('number of points must be n+1'), end
if length(y) ~= (n+1), error('number of points must be n+1'), end
y = y(:);      % makes y into a column vector
```

```
A = vander(x);
P = (A \ y)';
```

polyvalbyed.m

```
function y = polyvalbyed(P,x)
% POLYVALBYED  Re-implementation by Ed of the built-in polyval.
%     It is essentially the same as FASTEVAL, which is to say
%     it is an implementation of Horner's method, except for
%     reordering the coefficients to match POLYFIT/POLYVAL.
%
% Example:
%   >> x = [0.0 0.1 0.2 0.3 0.4];  P = polyfitbyed(x,sin(x),4)
%   >> polyvalbyed(P,0.25)
% You can compare this result to that from the POLYFIT/POLYVAL pair.
%
% See also: POLYFITBYED

n = length(P) - 1;
y = P(1);
for j=2:n+1
  y = P(j) + x * y;
end
```

2. **a.** Here $n = 2$ and $f'''(x) = -\pi^3 \cos(\pi x)$. Thus the Lagrange Remainder Theorem (LRT) says

$$|P(1.4) - f(1.4)| = \left| \frac{-\pi^3 \cos(\pi\,\xi(1.4))}{3!}(1.4-1)(1.4-1.25)(1.4-1.6) \right|$$
$$= \frac{\pi^3}{6}|\cos(\pi\,\xi(1.4))|(0.4)(0.15)(0.2) \le \frac{\pi^3}{6} \cdot 1 \cdot (0.4)(0.15)(0.2) = 0.062.$$

The actual absolute error computed in **1a.** is 0.033, so we are in good shape.

b. Again $n = 2$. Now $f'(x) = (3/\ln 10)(3x-1)^{-1}$ so $f'''(x) = (54/\ln 10)(3x-1)^{-3}$. The LRT says

$$|P(1.4) - f(1.4)| = \left| \frac{(54/\ln 10)(3\,\xi(1.4)-1)^{-3}}{3!}(1.4-1)(1.4-1.25)(1.4-1.6) \right|$$
$$\le \frac{54}{(\ln 10)(8)(6)}(0.4)(0.15)(0.2) = 0.0059.$$

All we know about $\xi(1.4)$ is that it is in the interval $[1, 1.6]$. Thus we have $|3\xi(1.4) - 1| \ge |3 \cdot 1 - 1| = 2$ so $|3\xi(1.4) - 1|^{-3} \le 2^{-3} = 1/8$.

The actual absolute error computed in **1a.** is 0.0020, so again we are in good shape.

c. Not graded. As described in class, the LRT does not apply because $f \notin C^3[1, 1.6]$. Indeed $f'(x)$ does not exist at $x = 1$, and $f'(x)$ is unbounded on $[1, 1.6]$.

3. (*This problem was merely included to give you a break from the LRT, and remind you to think of different "how-to" methods, perhaps.*)

It seems to me that what we know can be described this way:

$$\begin{bmatrix} 1 & 0 & 0^2 & 0^3 \\ 1 & 0.5 & 0.5^2 & 0.5^3 \\ 1 & 1 & 1^2 & 1^3 \\ 1 & 2 & 2^2 & 2^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ 6 \end{bmatrix} = \begin{bmatrix} 0 \\ y \\ 3 \\ 2 \end{bmatrix}$$

That is, we start with $P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ but we note $a_3 = 6$, and we "think Vandermonde".

But in fact the 1st, 2nd, and 4th equations form a system of three equations in three unknowns:

$$\begin{bmatrix} 1 & 0 & 0^2 \\ 1 & 1 & 1^2 \\ 1 & 2 & 2^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 3-6 \\ 2-48 \end{bmatrix} = \begin{bmatrix} 0 \\ -3 \\ -46 \end{bmatrix}$$

Solving this with MATLAB/OCTAVE in the obvious way gives $a_0 = 0$, $a_1 = 17$, and $a_2 = -20$. But then $P(x) = 17x - 20x^2 + 6x^3$ and so

$$y = P(0.5) = 17(0.5) - 20(0.5)^2 + 6(0.5)^3 = 4.25.$$

4. As mentioned in class, Neville's method was already online; here it is:

nev.m

```
function y = nev(xx,n,x,Q)
% NEV   Neville's algorithm as a function
%       y= nev(xx,n,x,Q)
% inputs:
%    n = order of interpolation (n+1 = # of points)
%    x(1),...,x(n+1)    x coords
%    Q(1),...,Q(n+1)    y coords
%    xx=evaluation point for interpolating polynomial p
% output:  p(xx)
%
% example:
%    >> x = [0 0.1 0.4 0.5];      % n=3 case with n+1=4 points
%    >> y = cos(x);
%    >> P = nev(0.187654321,3,x,y) % value of polynomial P(xx)
%    >> f = cos(0.187654321)       % value of function f(xx)

for i = n:-1:1
   for j = 1:i
      Q(j) = (xx-x(j))*Q(j+1) - (xx-x(j+n+1-i))*Q(j);
      Q(j) = Q(j)/(x(j+n+1-i)-x(j));
   end
end

y = Q(1);
```

a. Here we note that $2^{0.5} = \sqrt{2}$. Thus we approximate $f(0.5) = \sqrt{2}$ by computing $P(0.5)$ by using Neville's method:

```
>> x = -2:2;
>> nev(0.5,4,x,2.^x)
ans =  1.412109375
```

b. Here $\sqrt{2} = f(2)$. Thus we approximate $f(2) = \sqrt{2}$ by computing $P(2)$ by using Neville's method:

```
>> x = [0 (1/3) 1 3 4];
>> nev(2,4,x,sqrt(x))
ans =  1.17398999244855
```

c. The result in part **a.** is *a lot* more accurate. Why? Probably because, unlike $f(x) = \sqrt{x}$, there are no singularities in the derivatives of $f(x) = 2^x$. By contrast, $f(x) = \sqrt{x}$ is not in $C^{(n+1)}[a, b]$ on any interval $[a, b]$ including $x = 0$, for $n \geq 0$.

5. **a.** I wrote and then ran the obvious program using the Vandermonde matrix method:

erfintegrand.m

```
function v = erfintegrand(t);
% ERFINTEGRAND  Computes the coefficients of the 8th degree polynomial
%     interpolant of f(t) = e^{-t^2}.  Input t = (interpolation points).
% Example, to solve 5(a):
%   >> erfintegrand(0.0:0.1:0.8)

A = fliplr(vander(t));  % Vandermonde matrix with preferred order
y = exp(-t.^2);
v = (A \ y')';
```

```
>> erfintegrand(0.0:0.1:0.8)
ans =
 Columns 1 through 6:
   1.0000e+00   6.7672e-06  -1.0002e+00   1.9747e-03   4.8881e-01   3.6910e-02
 Columns 7 through 9:
  -2.3941e-01   8.2233e-02  -2.4125e-03
```

Thus

$$P(t) = 1 + 6.7672 \times 10^{-6}x - 1.0002x^2 + 0.0019747x^3 + 0.48881x^4 + 0.036910x^5 - 0.23941x^6 + 0.082233x^7 - 0.0024125x^8$$

b. We apply the L.R.T. with $n = 8$ and get:

$$|P(t) - f(t)| = \left| \frac{f^{(9)}(\xi(t))}{9!}(t - 0.0)(t - 0.1)\dots(t - 0.8) \right|.$$

But what is the 9th derivative of $f(t) = \exp(-t^2)$? This is a job for a computer program that differentiates *the same way you do*, but faster and with fewer errors. For example, Wolfram*Alpha*, which we can treat as an online-interface to Wolfram *Mathematica*.

Specifically, I went to

`http://www.wolframalpha.com`

and typed in, verbatim, "`9th derivative of f(t)=exp(-t^2)`". I got Figure 1. (Before the era of the internet 2.0, one could always look in a table or ask an expert; e.g. go to the "`Hermite polynomials`" Wikipedia page and you will get what has been in printed tables for about 150 years.)

O.k. *That* worked! Now we know

$$f^{(9)}(t) = -32\exp(-t^2)t(16t^8 - 288t^6 + 1512t^4 - 2520t^2 + 945).$$

Note that $\xi(t)$ is in $[0, 0.8]$. And we observe that

$$\begin{aligned}\left|f^{(9)}(t)\right| &= 32\exp(-t^2)|t|\left|16t^8 - 288t^6 + 1512t^4 - 2520t^2 + 945\right| \\ &\le 32\exp(-t^2)|t|\left(16|t|^8 + 288|t|^6 + 1512|t|^4 + 2520|t|^2 + 945\right) \\ &\le 32 \cdot 1 \cdot (0.8)\left(16\ 0.8^8 + 288\ 0.8^6 + 1512\ 0.8^4 + 2520\ 0.8^2 + 945\right) \\ &= 83335.6\end{aligned}$$

for any $0 \le t \le 0.8$. Returning to the bound, it follows that if $0 \le t \le 0.8$ then

$$\begin{aligned}|P(t) - f(t)| &= \frac{\left|f^{(9)}(\xi(t))\right|}{9!}|t - 0.0||t - 0.1|\dots|t - 0.8| \\ &\le \frac{83335.6}{362880}\,0.4^5\,0.8^4 = 0.00096 \le 10^{-3}.\end{aligned}$$

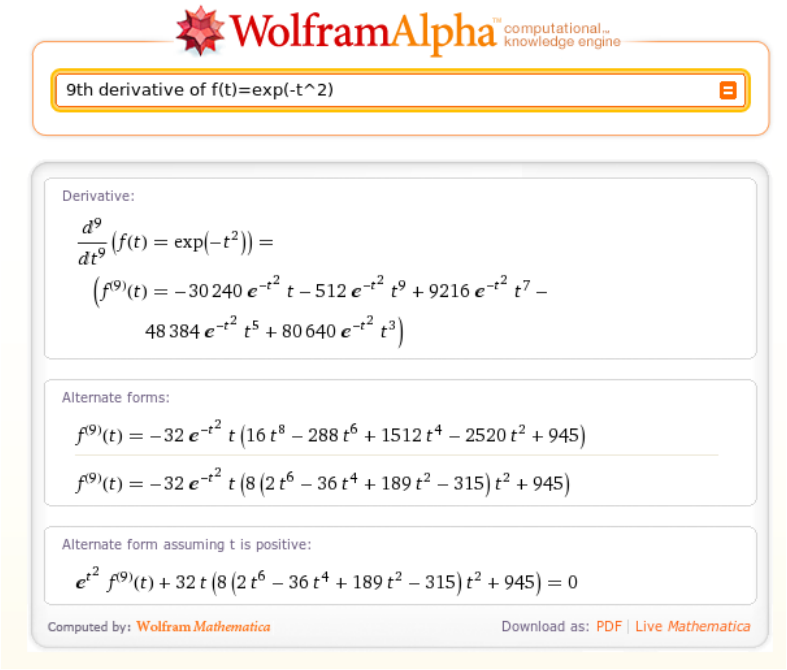

FIGURE 1. Screenshot. I asked Wolfram*Alpha* for the 9th derivative of $f(t)$.

At the last estimate step I have used the logic that if $t \in [0, 0.8]$ then the distance from t either to any one of the left half, or any one of the right half, of the interpolation points is at most half of the length of the interval, namely $|t - t_j| \le 0.4$ for $j = 0, 1, 2, 3, 4$ or for $j = 4, 5, 6, 7, 8$. The remaining points could be "far" from t, however: $|t - t_j| \le 0.8$ for all j.

Thus $\max_{0\le t\le 0.8} |P(t) - f(t)| \le 10^{-3}$. This turns out, not surprisingly, to be a conservative estimate.

c. and **d.** Yes, I do know how to integrate a polynomial:

$$\text{If} \quad P(t) = \sum_{j=0}^{8} a_j\, x^j \quad \text{then one antiderivative is} \quad Q(t) = \sum_{j=0}^{8} \frac{a_j}{j+1} x^{j+1}.$$

That is, you integrate polynomials by shifting the power by +1 and remembering to divide the coefficients by the new powers. So I wrote a program which uses the output of `erfintegrand.m` above:

erfbyed.m

```
function y = erfbyed(x)
% ERFBYED  Compute approximation to
%
%                2      /x
%   erf(x) = ---------  | exp(-t^2) dt
%            sqrt{pi}   /0
%
% using the antiderivative of the degree 8 polynomial P(t) found by
% equally-spaced polynomial interpolation of f(t) = exp(-t^2)
% on the interval [0.0,0.8].  Therefore this should only be used for
% inputs  0 <= x <= 0.8.
%
% Example.  Compare to the built-in "erf"; relative error:
%   >> erfbyed(0.8)
%   >> abs( erfbyed(0.8) - erf(0.8) )
```

```
P = erfintegrand(0:0.1:0.8);        % coefficients of  P(t)
Q = [0 P./(1:9)];                   % coefficients of  Q(t) = \int P(t) dt
y = (2 / sqrt(pi)) * fasteval(Q,x);
```

This program uses the `fasteval.m` code written in-class. That is, we evaluate the polynomial by Horner's method. It is online at

`http://www.dms.uaf.edu/~bueler/fasteval.m`

Running `erfbyed.m`, and doing part **d.** along the way,

```
>> erfbyed(0.8)
ans =    0.742100965073977
>> erf(0.8)
ans =    0.742100964707661
>> abs( erfbyed(0.8) - erf(0.8) )
ans = 3.66316199595929e-10
```

Thus we have about 9 digits correct, if we are to believe the built-in `erf`.

Somewhat incidentally, along the way we got

$$Q(t) = t + 3.38359 \times 10^{-6}\,t^2 - 0.33339\,t^3 + 0.00049367\,t^4 + 0.097762\,t^5 + 0.0061517\,t^6 - 0.034201\,t^7 + 0.010279\,t^8 - 0.00026806\,t^9.$$

Note that $Q(0) = 0$ because we have chosen the antiderivative with zero constant term. But who cares about the specific $Q(t)$? We already have the pattern by which to integrate polynomials, and we have correctly applied it inside the program. So the particular $Q(t)$ is not actually worth recording ...

e. From part **b.** we have

$$|P(t) - f(t)| \le 10^{-3}$$

for every $t \in [0, 0.8]$. But if $F(x) = (2/\sqrt{\pi})Q(x)$ then

$$|F(x) - \operatorname{erf}(x)| = \frac{2}{\sqrt{\pi}} \left| \int_0^x P(t) - f(t)\,dt \right| \le \frac{2}{\sqrt{\pi}} \int_0^x |P(t) - f(t)|\,dt$$
$$\le \frac{2}{\sqrt{\pi}} \int_0^x 10^{-3}\,dt = \frac{2 \times 10^{-3}}{\sqrt{\pi}} x$$

if $0 \le x \le 0.8$.

Note that the last inequality only applies for $x \in [0, 0.8]$ because we need the entire interval of integration, $t \in [0, x]$, to be inside *the interval on which we know $P(t)$ is close to $f(t)$.*

For our particular case, we have

$$|F(0.8) - \operatorname{erf}(0.8)| \le \frac{2 \times 10^{-3}}{\sqrt{\pi}} 0.8 = 0.00090.$$

From part **d.** we certainly believe this bound, but it is far too pessimistic. We have about 5 orders of magnitude more accuracy than the amount we can estimate in advance. But the "big deal" in numerical analysis is knowing *in advance* that a certain accuracy is achieved.