Solutions to Assignment #8

1. As shown in the example in section 4.5, the first three results are the composite trapezoid rule with $n = 1, 2, 4$ subintervals:

$$R_{1,1} = \frac{1}{2}[f(0) + f(1)] = 0.183939720585721,$$
$$R_{2,1} = \frac{1}{4}[f(0) + 2f(0.5) + f(1)] = 0.16778619275694,$$
$$R_{3,1} = \frac{1}{8}[f(0) + 2f(0.25) + 2f(0.5) + 2f(0.75) + f(1)] = 0.162488405093166,$$

where $f(x) = x^2e^{-x}\,dx$ is the integrand. I am keeping *all* computed digits because I know the extrapolation in the Romberg technique will use these digits to get a much better answer.

To build the rest of the table requires the formula

$$R_{k,j} = R_{k,j-1} + \frac{1}{4^{j-1} - 1}\left(R_{k,j-1} - R_{k-1,j-1}\right).$$

In particular,

$$R_{2,2} = R_{2,1} - \frac{1}{3}\left(R_{2,1} - R_{1,1}\right) = 0.16778619275694,$$
$$R_{3,2} = R_{3,1} - \frac{1}{3}\left(R_{3,1} - R_{2,1}\right) = 0.160722475871908,$$
$$R_{3,3} = R_{3,2} - \frac{1}{15}\left(R_{3,2} - R_{2,2}\right) = 0.16061052869799.$$

The exact value comes from a standard integration-by-parts exercise, starting with this calculation of the antiderivative:

$$\int x^2e^{-x}\,dx = -x^2e^{-x} + \int 2xe^{-x}\,dx = -x^2e^{-x} + 2\left(-xe^{-x} + \int e^{-x}\,dx\right) = -(x^2+2x+2)e^{-x} + C$$

Thus the exact is:

$$I = \int_0^1 x^2e^{-x}\,dx = -(x^2 + 2x + 2)e^{-x}\Big]_0^1 = 0.160602794142788.$$

The absolute error is

$$|R_{3,3} - I| = 7.7 \times 10^{-6}.$$

This is *much* better than the quality of $R_{3,1}$, for which the absolute error is about 1.9×10^{-3}. But it required the same total number of function evaluations, namely 5 evaluations of $f(x)$.

By the way, what I entered at the MATLAB/OCTAVE *command line to do this "by hand" looked like:*

```
>> format long g
>> f = @(x) x.^2.*exp(-x);
>> R11 = 0.5*(f(0)+f(1))
R11 =    0.183939720585721
>> R21 = 0.25*(f(0)+2*f(0.5)+f(1))
R21 =     0.16778619275694
>> R31 = 0.125*(f(0)+2*f(0.25)+2*f(0.5)+2*f(0.75)+f(1))
```

```
R31 =     0.162488405093166
>> R22 = R21 + (1/3)*(R21-R11)
R22 =     0.162401683480679
>> R32 = R31 + (1/3)*(R31-R21)
R32 =     0.160722475871908
>> R33 = R32 + (1/15)*(R32-R22)
R33 =      0.16061052869799
```

2. See the code `http://www.dms.uaf.edu/~bueler/buelerromberg.m` online.

3. The first stage is to interpolate at equally-spaced points. Figure 1 shows the result. We see that the $N = 1, 2$ cases are very inaccurate, while the $N = 8, 16$ cases are fairly close. Thus it is not clear we should use the $N = 1, 2$ cases in the extrapolation, but that's what the instructions say so we will do it.

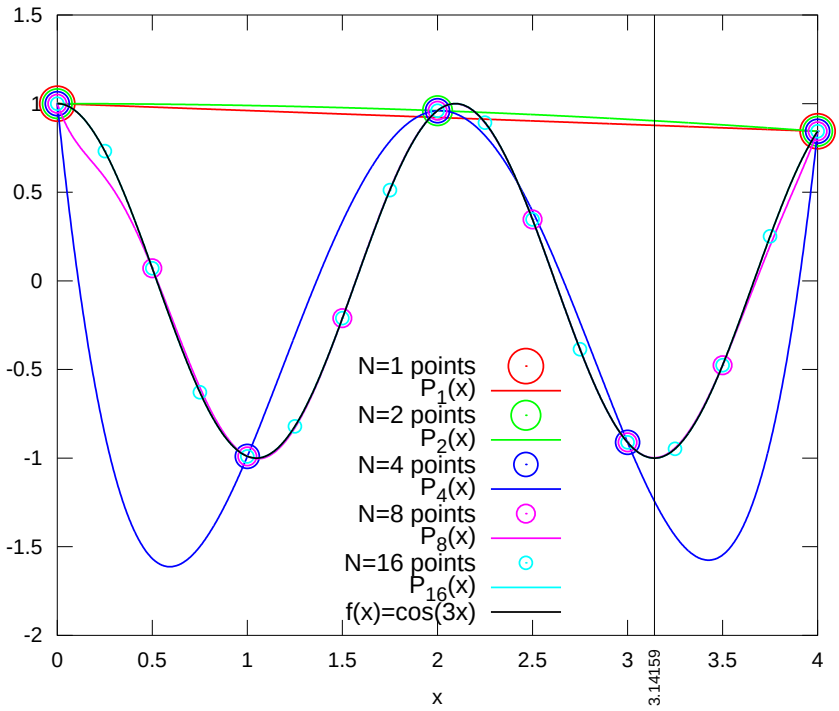

FIGURE 1. Equally-spaced polynomial interpolation of $f(x) = \cos 3x$ for $N = 1, 2, 4, 8, 16$. The goal is to approximate $P_N(\pi)$.

(a) & (b) I wrote the following code, which uses the fact that we do not want the *coefficients* of any of the polynomials, but rather just their values at points—in one case we only want $P_N(x)$ at $x\pi$, and in another we want to approximate $F(h)$ at $h = 0$. Thus the best implementation surely uses *Neville's method*, which never computes the coefficients. In fact I would claim this is a good implemention of the proposed interpolation/extrapolation method, *but* we see that the result is not very good. Note that $f(\pi) = -1$ exactly:

superinterp.m

```
% SUPERINTERP  Attempt to build a super-interpolator by doing successive
% polynomial interpolation with equal spacing  h, and then extrapolating
% to h=0.  Uses Neville's method (nev.m) for both stages.
```

```
xx = pi;
f = @(x) cos(3*x);                      % test it on a smooth fcn
a = 0;  b = 4;

h = zeros(1,5);  PNPI = zeros(1,5);  % will extrapolate this data to h=0
for k = 1:5
  N = 2^(k-1);                          % N = 1,2,4,8,16
  h(k) = (b - a) / N;
  x = a:h(k):b;
  y = f(x);
  PNPI(k) = nev(xx,N,x,y);              % P_N(pi)
end

result = nev(0,4,h,PNPI)                % result = F(0)  if  F(h) = P_N(pi)
err = abs(result - f(pi))
```

```
>> superinterp
result =    -1.39882302032546
err =    0.398823020325463
```

The resulting plot of the extrapolation problem, on h-versus-$P_N(\pi)$ axes, shows why the result is not so good: the data which is to be extrapolated is very irregular and does not "trend smoothly" toward our expected value.

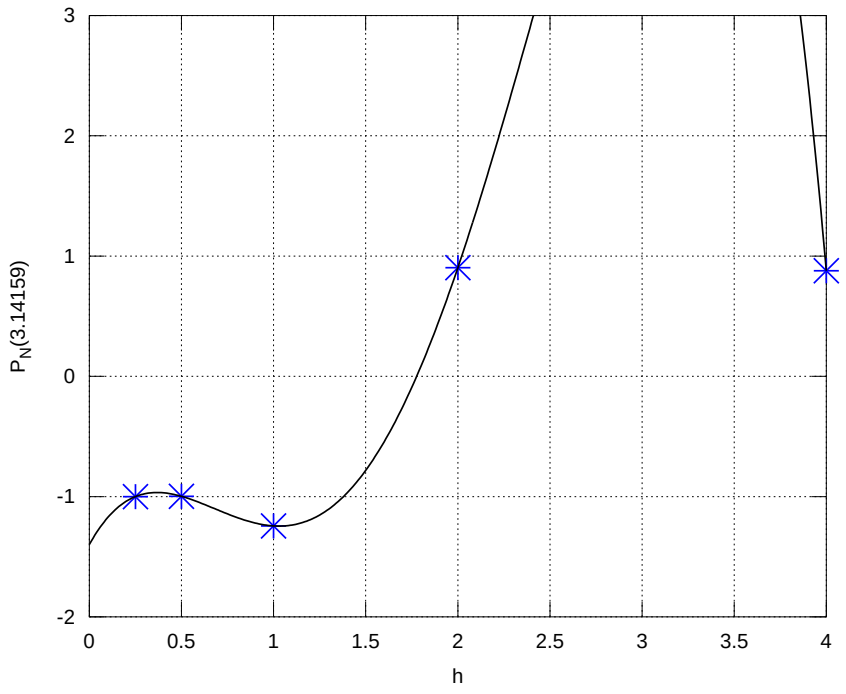

FIGURE 2. The blue stars are the results of the earlier polynomial interpolation stage, namely $P_N(\pi)$ for various N. These are considered as functions of $h = (b-a)/N$. The extrapolation to $h = 0$ is not great, though!

c. This method would need to be tested on other examples to see if this one is unusually bad. And we should write down Lagrange' remainder term and see if there is a pattern. That is, much more work is needed to give a numerical analysis of this method.

But so far it is not promising. The suggestion is that $F(h)$ is very irregular even for this smooth function, which causes the extrapolated value to be much worse than either the $N = 8$ or $N = 16$ values. The situation is much worse than the results we have seen for Romberg. Of course *integration* is a safer task than *extrapolation*; for example consider the error terms in Simpson's rule versus that for $n = 2$ degree polynomial interpolation.

Regarding efficiency: The use of Neville's method must mean there is not much waste. We could decrease the number of function evaluations by a factor of two by generating the Neville table line-by-line; this kind of thing is done in Romberg.

Also. You may be interested in how I made the figures:

showsuperinterp.m

```
% SHOWSUPERINTERP  Generates two figures from SUPERINTERP data.
% One figure shows the points used in interpolation.  Other figure
% shows the polynomial doing the extrapolation.

superinterp  % run it;  now "f", "a", "b", "h", "PNPI" are defined

set(0,'defaultlinelinewidth',2.0,'defaultlinemarkersize',14.0)
figure(1)
xp = a:(b-a)/1000:b;
for k = 1:5
  N = 2^(k-1);  x = a:h(k):b;   y = f(x);
  s = ['o' num2str(k) ';N=' num2str(N) ' points;'];  % nested, colored circles
  plot(x,y,s,'markersize',4.0+3*(5-k+1)),  hold on
  s = ['-' num2str(k) ';P_{' num2str(N) '}(x);'];  % curve with same color
  plot(xp,polyval(polyfit(x,y,N),xp),s)
end
yp = f(xp);  plot(xp,yp,'k;f(x)=cos(3x);')
legend('location','south')
plot([pi pi],[-2 1.5],'--k','linewidth',1.0)
text(pi,-2.4,'3.14159','fontsize',9.0,'rotation',90.0)
axis([0 4 -2 1.5]),  xlabel x,  hold off
print -dpdf allpoints.pdf

figure(2),  plot(h,PNPI,'*'),  xlabel h,  ylabel('P_N(3.14159)')
xp = 0:0.01:4;  yp = polyval(polyfit(h,PNPI,4),xp);
hold on,  plot(xp,yp,'k'),  hold off,  grid on
axis([0 4 -2 3]),  print -dpdf extrapolatethis.pdf
```

4 & 5. **a.** **n = 2 case.** First we convert the problem to an integral on $[-1,1]$ by the transformation $x = (1/2)[(b-a)t + a + b]$, using $a = 1$ and $b = 1.5$. (*See page 233 of the 9th edition of Burden&Faires, in section 4.7.* Then we apply the $n = 2$ rule, using the roots and coefficients found in the table in section 4.7 of Burden&Faires:

$$\int_1^{1.5} x^2 \ln x \, dx \quad \boxed{x = 0.5[0.5t + 2.5]} \quad \int_{-1}^{1} (0.5[0.5t + 2.5])^2 \ln(0.5[0.5t + 2.5]) \, (0.5)^2 \, dt$$

$$= (0.5)^4 \int_{-1}^{1} (0.5t + 2.5)^2 \ln(0.5[0.5t + 2.5]) \, dt$$

$$\approx (0.5)^4 (c_1 F(t_1) + c_2 F(t_2)) = 0.1922687.$$

In applying the rule, the "≈" step, we use $F(t) = (0.5t + 2.5)^2 \ln(0.5[0.5t + 2.5])$. Note $c_i = 1$ and $t_i = \pm\sqrt{3}/3$ here.

n = 3 case. Similarly,

$$\int_1^{1.5} x^2 \ln x \, dx = (0.5)^4 \int_{-1}^{1} (0.5t + 2.5)^2 \ln(0.5[0.5t + 2.5]) \, dt$$

$$\approx (0.5)^4 (c_1 F(t_1) + c_2 F(t_2) + c_3 F(t_3)) = 0.192259377.$$

exact value and comparison. The exact value is by integration by parts, for example:

$$\int_1^{1.5} x^2 \ln x \, dx = \frac{x^3}{3} \ln x \Big]_1^{1.5} - \int_1^{1.5} \frac{x^3}{3} \frac{1}{x} \, dx = C - \frac{1}{3} \int_1^{1.5} x^2 \, dx = C - \frac{1}{9} x^3 \Big]_1^{1.5} = 0.192259357,$$

where $C = 1.5^3 \ln(1.5)/3$. Thus the absolute error from the $n = 2$ rule is about 9×10^{-6} and from the $n = 3$ rule is about 2×10^{-8}. Both of these Gaussian rules are doing pretty well, given that they evaluate the integrand so few times!

b. **n = 2 case.** Similarly:

$$\int_1^{1.6} \frac{2x}{x^2 - 4} \, dx \quad \boxed{x = 0.5[0.6t + 2.6]} \quad 0.3 \int_{-1}^{1} \frac{0.6t + 2.6}{0.25(0.6t + 2.6)^2 - 4} \, dt \approx -0.730723036271920.$$

n = 3 case. Similarly,

$$\int_1^{1.6} \frac{2x}{x^2 - 4} \, dx \approx -0.733799022249413$$

exact value and comparison. The exact value is by integration using a substitution and then the rule $\int u^{-1} \, du = \ln|u| + c$:

$$\int_1^{1.6} \frac{2x}{x^2 - 4} \, dx \quad \boxed{u = x^2 - 4} \quad \int_{-3}^{-1.44} \frac{1}{u} \, du = \ln(1.44) - \ln(3) = -0.733969175080201.$$

Thus the absolute error from the $n = 2$ rule is about 3×10^{-3} and from the $n = 3$ rule is about 2×10^{-4}. Neither rule does quite as well as in part **(a)**. This can be attributed to the integrand being "less like" a polynomial, though making that precise would be pretty hard!

6. We consider $f(x) = 1, x, x^2, x^3$ in turn:

$$2 = \int_{-1}^{1} 1\,dx = a + b$$
$$0 = \int_{-1}^{1} x\,dx = -a + b + c + d$$
$$\frac{2}{3} = \int_{-1}^{1} x^2\,dx = a + b - 2c + 2d$$
$$0 = \int_{-1}^{1} x^3\,dx = -a + b + 3c + 3d$$

We have generated a linear system to solve, so here's the MATLAB/OCTAVE:

```
>> A = [1 1 0 0; -1 1 1 1; 1 1 -2 2; -1 1 3 3]
>> b = [2 0 2/3 0]'
>> v = A \ b
v =
                       1
                       1
      0.333333333333333
     -0.333333333333333
```

Thus the new rule is

$$\int_{-1}^{1} f(x)\,dx \approx f(-1) + f(1) + \frac{1}{3}f'(-1) - \frac{1}{3}f'(-1).$$

This is some kind of *correction* to the trapezoid rule, which is

$$\int_{-1}^{1} f(x)\,dx \approx f(-1) + f(1).$$

The trapezoid rule has degree of precision 1, but our new rule uses information about the derivative to "correct" for changing slope. (If the slope is constant then $f'(-1) = f'(1)$ and the correction is zero ... as it should be because the trapezoid rule already gets the linear case right.)