Solutions to Assignment #9

Exercise 10.2. The individual functions e^x and $\cos(\pi x/2)$ are a basis for the set of all functions of the given form. We want the rule to be exact for e^x and $\cos(\pi x/2)$ so that the linearity of the rule will imply that it is exact for all of the functions. This is two equations in unknowns A_0, A_1:

$$f(x) = e^x: \qquad e - 1 = \int_0^1 e^x\,dx = A_0 e^0 + A_1 e^1 = A_0 + eA_1,$$

$$f(x) = \cos(\pi x/2): \qquad \frac{2}{\pi} = \int_0^1 \cos(\pi x/2)\,dx = A_0\cos(0) + A_1\cos(\pi/2) = A_0.$$

Thus $A_0 = 2/\pi \approx 0.6366198$ and $A_1 = e^{-1}\left(e - 1 - (2/\pi)\right) \approx 0.3979212$. The resulting rule is good for functions of the given form, but it is easy to check that it is *worse* for polynomials than the trapezoid rule. It does not get all linear functions $f(x) = ax + b$ correct.

Exercise 10.4 (Extra Credit). (*Sorry! This one was harder than I thought so I've made it Extra Credit worth 3 points.*) Let $f(x) = 1, x, x^2, x^3$ in turn. Write down this system of equations which we want to be exactly true:

(1) $$f(x) = 1: \qquad \tfrac{1}{2} = \int_0^1 x\,1\,dx = A_0 + A_1$$

(2) $$f(x) = x: \qquad \tfrac{1}{3} = \int_0^1 x\,x\,dx = A_0x_0 + A_1x_1$$

(3) $$f(x) = x^2: \qquad \tfrac{1}{4} = \int_0^1 x\,x^2\,dx = A_0x_0^2 + A_1x_1^2$$

(4) $$f(x) = x^3: \qquad \tfrac{1}{5} = \int_0^1 x\,x^3\,dx = A_0x_0^3 + A_1x_1^3$$

To solve this system of four nonlinear equations in four unknowns requires care. (Gauss' insightful indirect method—via orthogonal polynomials—guarantees there is a solution.) My technique was to combine the above equations like this: $x_0 * (1) - (2)$, $x_0^2 * (1) - (3)$, $x_0^3 * (1) - (4)$. Simplify to get a new system without A_0:

$$\begin{aligned}
\tfrac{1}{2}x_0 - \tfrac{1}{3} &= A_1(x_0 - x_1)\\
\tfrac{1}{2}x_0^2 - \tfrac{1}{4} &= A_1(x_0^2 - x_1^2)\\
\tfrac{1}{2}x_0^3 - \tfrac{1}{5} &= A_1(x_0^3 - x_1^3)
\end{aligned}$$

Now factor on the right sides: $x_0^2 - x_1^2 = (x_0 - x_1)(x_0 + x_1)$ and $x_0^3 - x_1^3 = (x_0 - x_1)(x_0^2 + x_0x_1 + x_1^2)$. Then we can eliminate the expression "$A_1(x_0 - x_1)$" to get two equations in x_0, x_1:

$$\begin{aligned}
\tfrac{1}{2}x_0^2 - \tfrac{1}{4} &= \left(\tfrac{1}{2}x_0 - \tfrac{1}{3}\right)(x_0 + x_1)\\
\tfrac{1}{2}x_0^3 - \tfrac{1}{5} &= \left(\tfrac{1}{2}x_0 - \tfrac{1}{3}\right)(x_0^2 + x_0x_1 + x_1^2)
\end{aligned}$$

The first of these can be solved for x_1. Then substitute into the second. The simplified result is a quadratic equation for x_0:

$$\left(\tfrac{1}{8} - \tfrac{1}{9}\right)x_0^2 + \left(\tfrac{1}{12} - \tfrac{1}{10}\right)x_0 + \left(\tfrac{1}{15} - \tfrac{1}{16}\right) = 0.$$

MATLAB can be used to find the roots of this polynomial:

```
>> roots([(1/8 - 1/9), (1/12 - 1/10), (1/15 - 1/16)])
ans =
  0.84495
  0.35505
```

Let $x_0 = 0.35505$. Then we see by symmetry or calculation that $x_1 = 0.84495$. Any of several equations above gives A_1 and then we get $A0$:

```
>> x0 = ans(2); x1 = ans(1);
>> A1 = ((1/2)*x0 - (1/3)) / (x0-x1)
A1 =  0.31804
>> A0 = (1/2) - A1
A0 =  0.18196
```

For an alternative method, I have posted a code which also shows how to use MATLAB's "qr" to find the orthogonal polynomial whose roots give x_0, x_1:

http://www.dms.uaf.edu/~bueler/problem104.m

Exercise 10.7. First I wrote this code that does the composite trapezoid and Simpson's rules:

trapsimp.m

```
function [ztrap zsimp] = trapsimp(n)
% TRAPSIMP  Compute both the composite trapezoid and Simpson's
% rule approximations of this integral -->     /1
% using n subintervals.  For example:          |  cos(x^2) dx
% >> [ztrap zsimp] = trapsimp(10)              /0

h = 1 / n;
x = 0 : h : 1;
ftrap = cos(x.^2);
ctrap = [1 2*ones(1,n-1) 1];  % coeffs for trapezoid rule
ztrap = (h/2) * (ctrap * ftrap');

xsimp = 0 : h/2 : 1;
fsimp = cos(xsimp.^2);
csimp = [1 repmat([4 2],1,n-1) 4 1];  % coeffs for Simpson's rule
zsimp = (h/6) * (csimp * fsimp');
```

Note that I captured the $1\,4\,2\,4\,2\,4\,2\,\ldots\,4\,1$ pattern of coefficients in Simpson's rule using the "repeat matrix" command. (Type `help repmat`.) Also I have interpreted the sums in these integration rules as dot products between coefficients and function values. A test is like this:

```
>> [ztrap zsimp] = trapsimp(10)
ztrap =    0.903121757123408
zsimp =    0.904524244850800
>> quad(@(x) cos(x.^2),0,1,[1e-12 1e-12])
ans =      0.904524237900272
```

It looks like both methods are working, and that the Simpson's rule estimate is much better, as expected. (This assumes the `quad` result is exact. As instructed.)

Then I wrote this code to do the convergence study:

testconvergence.m

```
% TESTCONVERGENCE  tests the convergence of the estimates from TRAPSIMP
% as we make h smaller

zexact = quad(@(x) cos(x.^2),0,1,[1e-12 1e-12]);

fprintf('(trap err)/h^2   (simp err)/h^4\n')
fprintf('------------------------------\n')
for h = [0.5 0.2 0.1 0.05 0.02 0.01 0.005]
  n = round(1/h);
  [ztrap zsimp] = trapsimp(n);
  fprintf('  %12.10f     %12.10f\n',...
          abs(ztrap-z)/(h^2),abs(zsimp-z)/(h^4))
end
```

Running it looks like this:

```
>> testconvergence
(trap err)/h^2   (simp err)/h^4
------------------------------
  0.1399698023     0.0003675544
  0.1402503248     0.0000187263
  0.1402480777     0.0000695053
  0.1402459925     0.0000820090
  0.1402453011     0.0000854962
  0.1402451985     0.0000860090
  0.1402451727     0.0000861533
```

As a result we see that if T_n is the n-subinterval composite trapezoid rule estimate then

$$\left| T_n - \int_0^1 \cos(x^2)\,dx \right| \approx 0.14\ h^2$$

while if S_n is the n-subinterval composite Simpson's rule estimate then

$$\left| S_n - \int_0^1 \cos(x^2)\,dx \right| \approx 8.6 \times 10^{-5}\, h^4.$$

That is, the composite trapezoid rule is second-order and the composite Simpson's rule is fourth-order on this nice (i.e. smooth) integral. Because of this difference in powers of h, Simpson's rule is a lot more effective.

Problem I. (a) (*I should have clarified that the "check" in this problem can be done as a* MATLAB *program that calls the downloaded program* `hermquad.m`.)

checkhq.m

```
% CHECKHQ  check that the n=2 case of Gauss-Hermite quadrature is exact
% as expected on f(x) = x^0, x^1, x^2, x^3 in the integrals
%      / +inf
%      |      f(x) exp(-x^2) dx
%      / -inf
```

```
[X W] = hermquad(2);
x = X';  w = W';
exact = [sqrt(pi) 0 sqrt(pi)/2 0];
for p = 0:3
  s(p+1) = sum(w .* (x.^p));
end
norm(s - exact)
```

Running it looks like this:

```
>> checkhq
ans = 4.00296604248672e-16
```

Which shows that the numerical method is getting the same results as the exact values (i.e. $\sqrt{\pi}, 0, \sqrt{\pi}/2, 0$) to at least 15 digits.

(b) Doing the integral by our Gauss-Hermite method and then by `quad` gives

```
>> [x, w] = hermquad(6);
>> x'    % a look at these points (abscissae)
ans =
  2.35060  1.33585  0.43608  -0.43608  -1.33585  -2.35060
>> format long
>> w' * cos(x)
ans =     1.38038841005073
>> quad(@(x) cos(x) .* exp(-x.^2),-20,20)
ans =     1.38038844704314
```

Assuming the second answer is exact, the Gauss-Hermite method with only $n = 6$ points is giving 8 digit accuracy.

The point here is that *on the interval where e^{-x^2} has significant size*, the factor $\cos(x)$ is well-approximated by a polynomial. Figure 1 shows the integrand on the short interval $[-3, 3]$. The "bell curve" function e^{-x^2} decays very rapidly; note $e^{-3^2} = e^{-9} = 0.000123$ is already very small. Thus $\cos(x)$ only needs to be well-approximated by a polynomial on (say) the interval $[-3, 3]$, and the six abscissae above (i.e. $-2.35060, -1.33585, \ldots, 2.35060$) are adequate for that job.

Problem II. (*The statement of the problem should have said "$N+1$-point Clenshaw-Curtis" quadrature. Sorry about that.*)

MATLAB is very well suited for this job. To do it we do need to know how to integrate polynomials. Specifically, if $p(x) = c_1 + c_2 x + c_3 x^2 + \cdots + c_N x^{N-1} + c_{N+1} x^N$ then

$$\int_{-1}^{1} p(x)\,dx = \sum_{j=1}^{N+1} \frac{1-(-1)^j}{j} c_j.$$

Using this form requires we reverse the coefficients produced by `polyfit`. I wrote this code:

clencurt.m

```
function z = clencurt(f,N)
% CLENCURT Use polynomial interpolation at N+1 Chebyshev points
% x_j = cos(pi j/N) to do the integral  -->       /1
% That is, do Clenshaw-Curtis integration.        |  f(x) dx
```

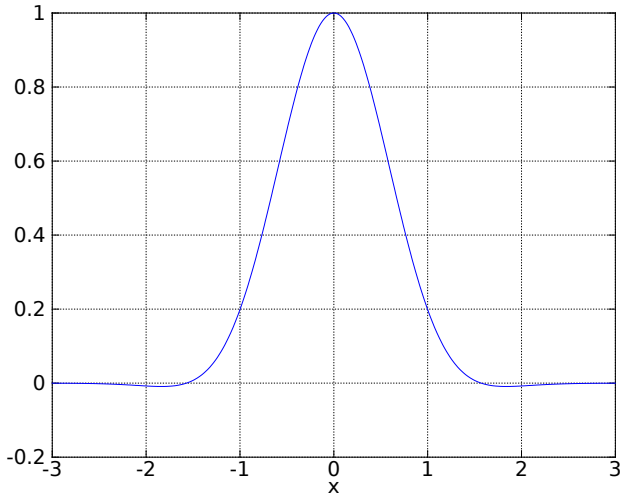

FIGURE 1. The integrand $\cos(x)e^{-x^2}$ is very much like e^{-x^2} times a polynomial because e^{-x^2} is so small outside a brief interval around zero.

```
% For example:                                          /-1
%   >> clencurt(@(x) cos(x),6)
% for which the exact answer is
%   >> 2 * sin(1)

x = cos(pi * (0:N) / N);
p = polyfit(x,f(x),N);
p = fliplr(p);
c = (1 - (-1).^(1:N+1)) ./ (1:N+1);
z = c * p';
```

That's the whole thing! To test it as suggested I needed to do this exact integral, which is by-parts:

$$\int_{-1}^{1} x\sin(x)\,dx = -x\cos(x)\Big|_{-1}^{1} + \int_{-1}^{1}\cos(x)\,dx = 2(\sin(1)-\cos(1)).$$

Now I test:

```
>> clencurt(@(x) x.*sin(x),4)
ans =    0.602182995236023
>> clencurt(@(x) x.*sin(x),6)
ans =    0.602337506553671
>> clencurt(@(x) x.*sin(x),10)
ans =    0.602337357879725
>> 2 * (sin(1) - cos(1))
ans =    0.602337357879513
```

That's 12 digits correct, and pretty good for 11 point polynomial interpolation!

Exercise 11.1. **(a)** This one is pure integration. From "$y' = t^3$" get the general solution

$$y(t) = \frac{t^4}{4} + C$$

Now use the initial condition to determine C:

$$0 = y(0) = \frac{0}{4} + C \qquad \Longrightarrow \qquad C = 0.$$

Thus $y(t) = t^4/4$.

(b) All functions $y(t)$ for which $y' = 2y$ are of the form

$$y(t) = Ae^{2t}.$$

Then $3 = y(1) = Ae^2$ so $A = 3e^{-2}$ so $y(t) = 3e^{-2}e^{2t} = 3e^{2(t-1)}$.

(c) Following the given advice you can start with

$$e^{-at}y'(t) = e^{-at}(ay(t) + b) \qquad \text{or} \qquad e^{-at}y'(t) - ae^{-at}y(t) = be^{-at}.$$

Integrating factors help undo the product rule, thus:

$$\left(e^{-at}y(t)\right)' = be^{-at}$$

Now both sides can be integrated:

$$\int_0^T \left(e^{-at}y(t)\right)' \, dt = \int_0^T be^{-at} \, dt$$

$$e^{-aT}y(T) - e^0 y(0) = -\frac{b}{a}e^{-at}\Big]_{t=0}^{t=T}$$

$$e^{-aT}y(T) = y_0 - \frac{b}{a}\left(e^{-aT} - 1\right)$$

$$y(T) = e^{aT}\left[y_0 + \frac{b}{a}\left(1 - e^{-aT}\right)\right]$$

Finally, recognizing that T is arbitrary,

$$y(t) = \left(y_0 + \frac{b}{a}\right)e^{at} - \frac{b}{a}.$$

It is worth checking that $y' = ay + b$ and $y(0) = y_0$.

Exercise 11.3. *Though the problem does not give this hint at the beginning, note that* $y(t) = 0$ *is* also *a solution.* Substitution shows that $y(t) = t^{3/2}$ is a solution:

$$\frac{3}{2}t^{1/2} = y'(t) = \frac{3}{2}\left(t^{3/2}\right)^{1/3} = \frac{3}{2}t^{1/2},$$

while also $y(0) = 0^{3/2} = 0$. Applying Euler's method requires noting $f(t, y) = \frac{3}{2}y^{1/3}$:

$$y_{k+1} = y_k + hf(t_k, y_k) = y_k + h\tfrac{3}{2}y_k^{1/3}.$$

If $y_0 = 0$ then for any $h > 0$:

$$y_1 = 0 + h\tfrac{3}{2}0^{1/3} = 0, \qquad y_2 = 0 + h\tfrac{3}{2}0^{1/3} = 0, \qquad , \ldots$$

and so on. The Euler solution is $y_k = 0$ for $k = 0, 1, 2, \ldots$.

How is this situation possible? Notice that the existence and uniqueness theorem 11.1.2 does not apply. If it did then it would say that this ODEIVP would have exactly one solution. But two different solutions have been found. The numerical solution follows $y(t) = 0$ and not $y(t) = t^{3/2}$.

The reason the theorem does not apply is that $\partial f/\partial y = \frac{1}{2}y^{-2/3}$ is not continuous or even bounded at $y = 0$ (i.e. in any rectangle around $(t_0, y_0) = (0, 0)$).

Exercise 11.4. Straightforward. Note $f(t,y) = (t+1)e^{-y}$, $t_0 = 0$, $y_0 = 0$, and $h = 0.1$. For Euler:

$$y_1 = y_0 + hf(t_0, y_0) = 0 + 0.1\left((0+1)e^{-0}\right) = 0.1.$$

For midpoint method:

$$y_{1/2} = y_0 + (h/2)f(t_0, y_0) = 0 + 0.05\left((0+1)e^{-0}\right) = 0.05,$$

$$y_1 = y_0 + hf(t_{1/2}, y_{1/2}) = 0 + 0.1f(0.05, 0.05) = 0.1\left((0.05+1)e^{-0.05}\right) = 0.099879.$$

For Heun's:

$$\tilde{y}_1 = y_0 + hf(t_0, y_0) = 0.1,$$

$$y_1 = y_0 + h\,\frac{f(t_0,y_0) + f(t_1,\tilde{y}_1)}{2} = 0 + 0.1\frac{1 + (0.1+1)e^{-0.1})}{2} = 0.099766.$$

The curious might ask what is the correct value? This ODE is separable so we can get exactly

$$y(t) = \ln\left(1 + t + \frac{t^2}{2}\right) \qquad \Longrightarrow \qquad y(0.1) = \ln(1.105) = 0.0998453349697161.$$

Alternatively I can ask `ode45` to take as many smaller steps as it needs to get high accuracy:

```
>> opts = odeset('RelTol',1e-14,'AbsTol',1e-14);
>> [tt,yy] = ode45(@(t,y) (t+1).*exp(-y),[0 0.1],0,opts);
>> length(yy)
ans =
       14
>> yy(end)
ans =   0.0998453349697158
```

That is, `ode45` took 14 steps and estimated $y(0.1) \approx 0.099845$. We conclude that both midpoint and Heun's are great improvements over Euler, as is `ode45` of course.

Exercise 11.6. See page 266 for the formulas of the fourth-order Runge-Kutta method. Here $f(t,y) = \lambda y$ so we just start applying the formulas in this case:

$$q_1 = f(t_k, y_k) = \lambda y_k,$$

$$q_2 = f(t_k + \tfrac{h}{2}, y_k + \tfrac{h}{2}q_1) = \lambda\left(y_k + \frac{h}{2}(\lambda y_k)\right) = \lambda y_k\left(1 + \frac{h}{2}\lambda\right),$$

$$q_3 = f(t_k + \tfrac{h}{2}, y_k + \tfrac{h}{2}q_2) = \lambda\left(y_k + \frac{h}{2}\,\lambda y_k\left(1 + \frac{h}{2}\lambda\right)\right) = \lambda y_k\left(1 + \frac{h}{2}\lambda + \frac{h^2}{4}\lambda^2\right),$$

$$q_4 = f(t_k, y_k + hq_3) = \lambda\left(y_k + h\,\lambda y_k\left(1 + \frac{h}{2}\lambda + \frac{h^2}{4}\lambda^2\right)\right) = \lambda y_k\left(1 + h\lambda + \frac{h^2}{2}\lambda^2 + \frac{h^3}{4}\lambda^3\right).$$

Then

$$\begin{aligned}
y_{k+1} &= y_k + \frac{h}{6}\left[q_1 + 2q_2 + 2q_3 + q_4\right] \\
&= y_k + \frac{h}{6}\,\lambda y_k\left[1 + 2\left(1 + \frac{h}{2}\lambda\right) + 2\left(1 + \frac{h}{2}\lambda + \frac{h^2}{4}\lambda^2\right) + \left(1 + h\lambda + \frac{h^2}{2}\lambda^2 + \frac{h^3}{4}\lambda^3\right)\right] \\
&= y_k + \frac{h}{6}\,\lambda y_k\left[6 + 3h\lambda + h^2\lambda^2 + \frac{h^3}{4}\lambda^3\right] \\
&= y_k\left[1 + h\lambda + \frac{1}{2}h^2\lambda^2 + \frac{1}{6}h^3\lambda^3 + \frac{1}{24}h^4\lambda^4\right],
\end{aligned}$$

as claimed.

On the other hand we know exactly what y_{k+1} should be! It should be the solution $y(t_{k+1})$ of the initial value problem

$$y(t)' = \lambda y(t), \qquad y(t_k) = y_k.$$

Thus, because $t_{k+1} = t_k + h$, we have the exact value

$$y(t_{k+1}) = y_k e^{\lambda h}.$$

Taylor series expansion of this exact value gives

$$y(t_{k+1}) = y_k \left[1 + h\lambda + \frac{1}{2}h^2\lambda^2 + \frac{1}{6}h^3\lambda^3 + \frac{1}{24}h^4\lambda^4 + \frac{1}{5!}h^5\lambda^5 + \frac{1}{6!}h^6\lambda^6 + \dots \right],$$

Finally we can compute the error, and we see that the first power that does not cancel is the fifth:

$$|y_{k+1} - y(t_{k+1})| = |y_k| \left| \frac{1}{5!}h^5\lambda^5 + \frac{1}{6!}h^6\lambda^6 + \dots \right|.$$

As the book says on page 259, the local truncation error is $1/h$ times the first neglected term in a Taylor series expansion for $y(t_{k+1})$. Thus the local truncation error is

$$\frac{1}{h}\ |y_{k+1} - y(t_{k+1})| \approx \frac{|y_k|\lambda^5}{5!}h^4$$

which is $O(h^4)$ as claimed.