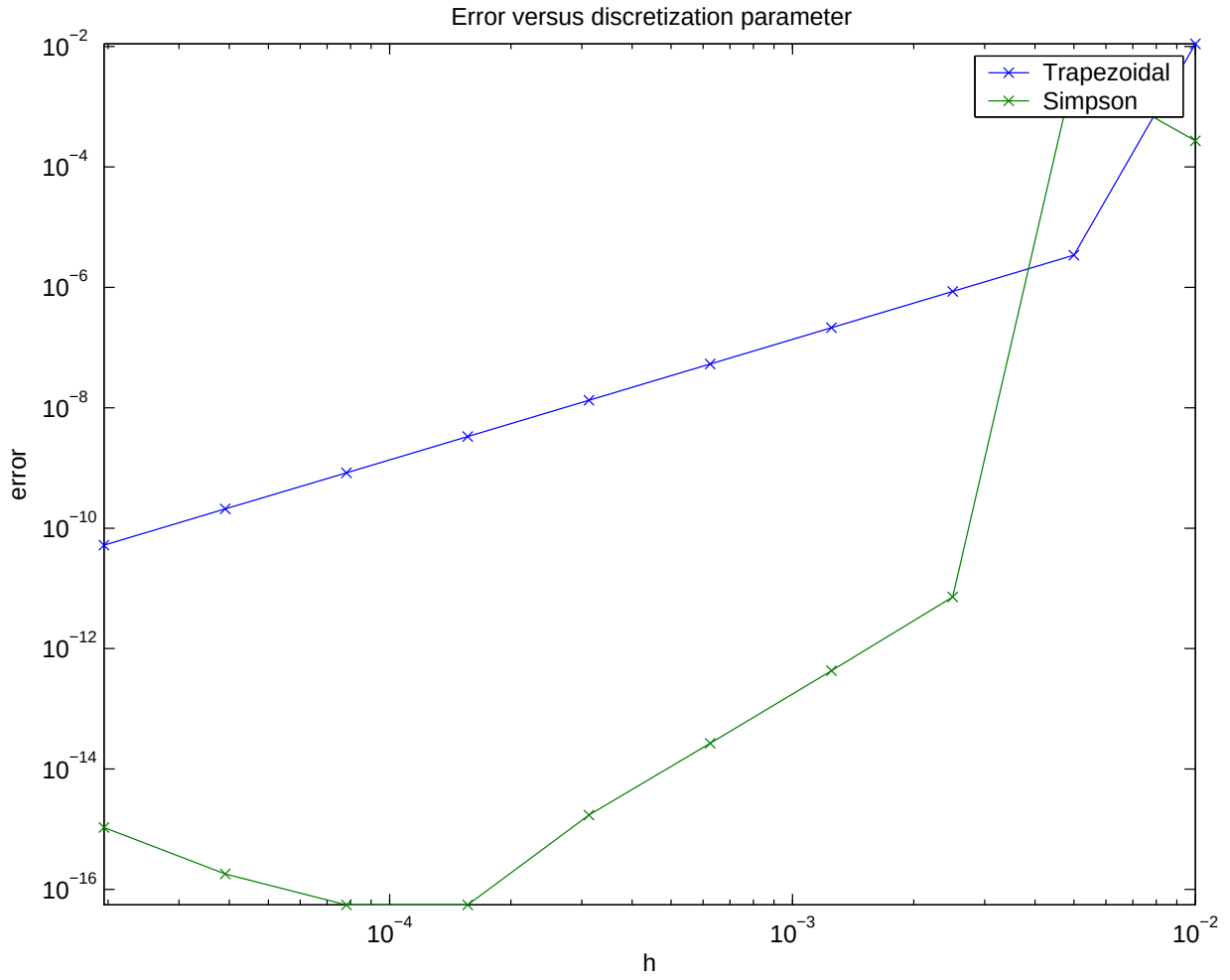
APPM4720/5720 — Solution to homework problem 1.3

The graphs below are generated by the file

`hw01p3.m`



The line corresponding to the trapezoidal rule clearly has slope 2 corresponding to the $O(h^2)$ expected convergence.

The line corresponding to Simpson's rule has a section where the slope is 4 corresponding to the $O(h^4)$ expected convergence. But we also see a couple of "random" entries before convergence is reached, and then once the error hits machine precision 10^{-15} there is only noise.

If we did not have a reference value, then we could plot the function

$$d(h) = I(h) - I(h/2),$$

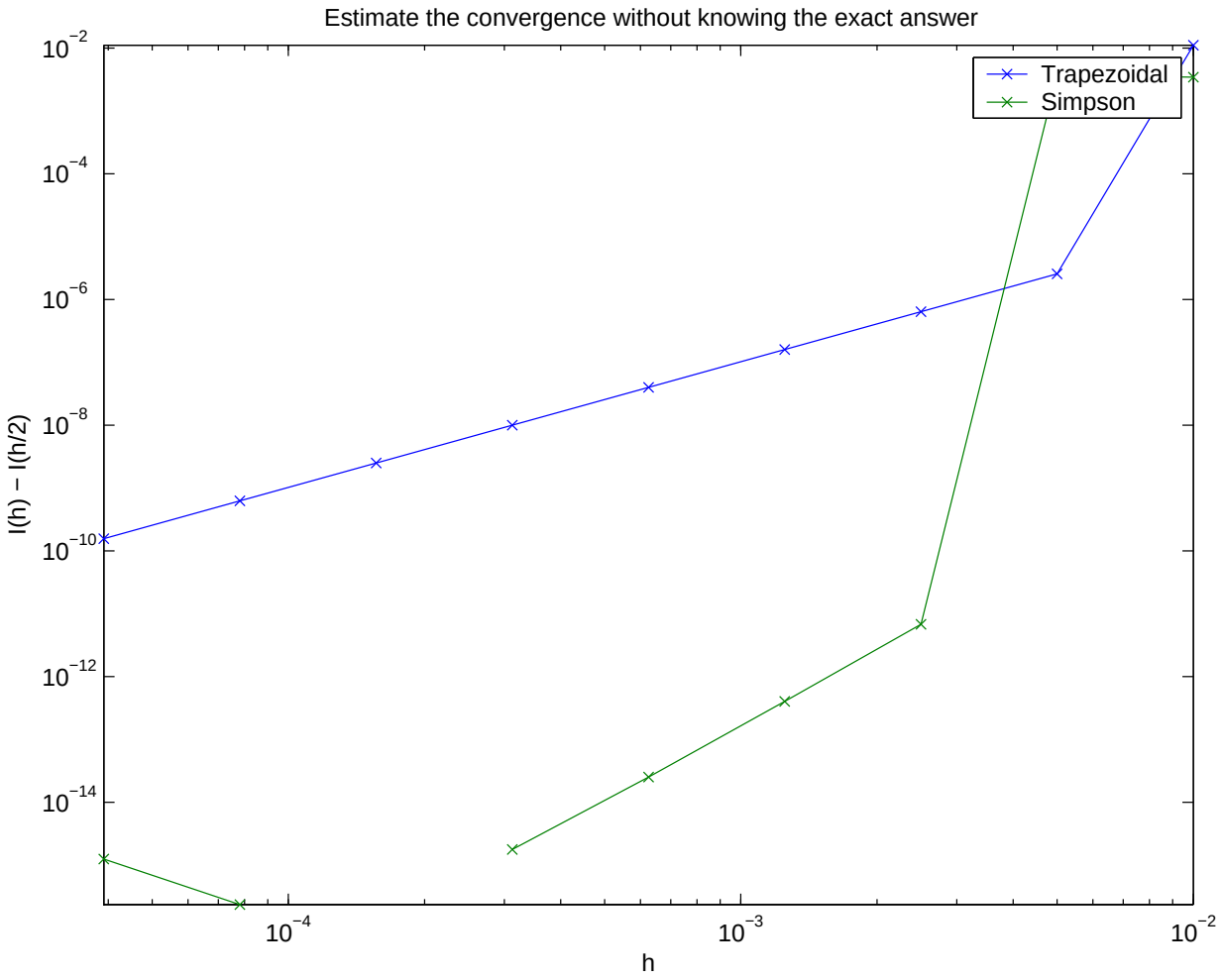
where $I(h)$ is the estimated value resulting from a step size h. Since we guess that

$$I(h) \sim I(0) + c\,h^p$$

(where $I(0)$, c, and p are all unknown) we find that

$$d(h) \sim \big(I(0) + c\,h^p\big) - \big(I(0) + c\,(h/2)^p\big) = c\,(1 - 2^{-p})\,h^p,$$

and we can read off p by plotting $\log(d)$ versus $\log(h)$. (Note: we actually set $d(j) = |I(h) - I(h/2)|$ since we may have $I(h) < I(h/2)$ and this would result in error messages when we try to take the logarithm.)



```
function hw01p3

a          = 0;
b          = 1;
n_quad_vec = 50.*(2.^(1:10));

%f = inline('exp(t.*t).*cos(37*t)','t');
%f = inline('exp(t)','t');
f = inline('cos(1./(0.01 + t.*t))','t');

[I_ref,nfunc] = quad(f,a,b,1e-13);
fprintf(1,'Matlab used %5d function evaluations and estimated I = %20.13e\n',...
        nfunc,I_ref)

figure(1)
xx = linspace(a,b,1000);
plot(xx,f(xx))
title('The function to be integrated')

I1_vec = zeros(size(n_quad_vec));
I2_vec = zeros(size(n_quad_vec));
h_vec  = (b-a)./n_quad_vec;

for icount = 1:length(n_quad_vec)
  n_quad = n_quad_vec(icount);
  I1_vec(icount) = quad_trap_1d(f,a,b,n_quad);
  I2_vec(icount) = quad_simp_1d(f,a,b,n_quad);
end
err1_vec = abs(I1_vec - I_ref);
err2_vec = abs(I2_vec - I_ref);

fprintf(1,'\n=== RESULTS FOR THE TRAPEZOIDAL RULE\n\n')
fprintf(1,'    h                       I_est              |I_est-I_ref|          |I_est-I_ref|/
for icount = 1:length(n_quad_vec)
  h     = h_vec(icount);
  I_est = I1_vec(icount);
  err   = err1_vec(icount);
  err_last = err1_vec(max(icount-1,1));
  fprintf(1,'%12.5e     %20.13e   %20.13e            %8.2e                %8.2e\n',...
          h,I_est,err,err/(h*h),err_last/err)
end
fprintf(1,'\n I_ref=          %20.13e\n',I_ref)

fprintf(1,'\n=== RESULTS FOR SIMPSONS RULE\n\n')
fprintf(1,'    h                       I_est              |I_est-I_ref|          |I_est-I_ref|/
for icount = 1:length(n_quad_vec)
  h     = h_vec(icount);
  I_est = I2_vec(icount);
  err   = err2_vec(icount);
  err_last = err2_vec(max(icount-1,1));
```

```
  fprintf(1,'%12.5e     %20.13e   %20.13e             %8.2e                     %8.2e\n',...
          h,I_est,err,err/(h*h*h*h),err_last/err)
end
fprintf(1,'\n I_ref=          %20.13e\n',I_ref)

figure(2)
loglog(h_vec,err1_vec,'x-',...
       h_vec,err2_vec,'x-')
legend('Trapezoidal','Simpson')
axis tight
title('Error versus discretization parameter')
xlabel('h')
ylabel('error')
%print -depsc hw01p3_fig1.eps

figure(3)
diff_err1_vec = abs(I1_vec(1:(end-1)) - I1_vec(2:end));
diff_err2_vec = abs(I2_vec(1:(end-1)) - I2_vec(2:end));
loglog(h_vec(1:(end-1)),diff_err1_vec,'x-',...
       h_vec(1:(end-1)),diff_err2_vec,'x-')
legend('Trapezoidal','Simpson')
axis tight
title('Estimate the convergence without knowing the exact answer')
xlabel('h')
ylabel('I(h) - I(h/2)')
%print -depsc hw01p3_fig2.eps

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function I1 = quad_trap_1d(f,a,b,n_quad)

h  = (b-a)/n_quad;
xx = linspace(a,b,n_quad+1);
ww = [0.5*h, h*ones(1,n_quad-1), 0.5*h];

I1 = sum(ww.*f(xx));

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function I2 = quad_simp_1d(f,a,b,n_quad)

if ~(mod(n_quad,2) == 0)
  fprintf(1,'Error, n_quad is not even.\n')
  keyboard
end
```

```
h     = (b-a)/n_quad;
xx    = linspace(a,b,n_quad+1);
ww    = zeros(1,n_quad+1);
ww(1)   = (h/3);
ww(end) = (h/3);
ww(2:2:(end-1)) = (4*h/3)*ones(1,n_quad/2);
ww(3:2:(end-2)) = (2*h/3)*ones(1,n_quad/2-1);

I2 = sum(ww.*f(xx));

return
```