```
% compute Q-values for a barn owl auditory nerve fiber (courtesy of Christine
Koppl)

clear
% ----
A=load('ANF46.4.15.txt');   % load in data from text file

% ----
% plot tuning curve
semilogx(A(:,1),A(:,2),'bs-','LineWidth',2); xlim([1000 10000]); grid on;
ylabel('Threshold [dB SPL]','FontSize',15); xlabel('Frequency
[Hz]','FontSize',15);
title('Single-unit auditory nerve fiber threshold tuning curve for barn owl
(C. Koppl)')

% ----
% determine Qerb
[thresh, indx]= min(A(:,2));    % first find CF for fiber (i.e., freq. at
threshold) [dB SPL]
CF= A(indx,1);    % CF for fiber [kHz]
TC= 10.^(A(:,2)/20) * (20*10^-6); % convert tuning curve units to Pa (from dB
SPL)
F= min(TC)./TC;              % the equivalent "filter" (inverse of the TC),
normalized to unity at the peak
ERB= trapz(A(:,1),F.^2);    % integral computers the area under the square of
that filter (squared because we are concerned with the total power)
Qerb= CF/ERB;
disp(['estimated Qerb= ',num2str(Qerb),' (Note: Q10 ~ Qerb/1.8)']);
```

PHYS 2030 (Winter 2017) - HW 2

Solutions

Note: Solutions below use slightly different limits than what were prescribed in the assignment.

1. Consider the integral

$$\int_{-1.5}^{\pi} e^{-cx}\,dx$$

where c is a constant greater than zero.

(a) Make a sketch of the integrand and indicate the integral as an *area* to compute. Make sure to clearly label things!

(b) Can you find an explicit solution for this integral? If so, give it. If not, explain.

General solution is

$$\int e^{-cx}\,dx = -\frac{1}{c}e^{-cx} + k$$

where k is an arbitrary constant. Applying the limits of integration, we have

$$\int_{-1.5}^{\pi} e^{-cx}\,dx = \frac{1}{c}\left[e^{1.5c} - e^{-c\pi}\right]$$

(c) Assume $c = 2$. Solve the definite integral numerically (by hand) using the left-hand rule (LEFT), right-hand rule (RIGHT), the midpoint rule (MID), the trapezoid rule (TRAP) and Simpson's rule (SIMP). Do this for both $N = 2$ and $N = 3$ (where N is the number of divisions over the interval of integration). How does your estimate compare to the exact answer? How does it change with N?

See output of code at end (w/ correct limits).

(d) Which of the values you computed are underestimates? Overestimates? Explain.

Given that the function is decreasing over the interval of integration and concave up: RIGHT and MID will be underestimates, LEFT and TRAP will be overestimates. Given that MID is an underestimate, we might expect that SIMP is too (given that it is weighted more than TRAP). However, we see that SIMP is a slight overestimate (meaning that for our function here, the error associated with TRAP was more than double that of MID). As N increases, the estimates get better (i.e., less error).

(e) Write a Matlab script (i.e., a .m file) that does the computations for you. In your code, you should be able to readily change N and c (as well as your end points). There are many ways to do this. One suggested approach is to create an array of values of the integrand for $x \in [-1.5, \pi]$ (discretized into $N+1$ evenly spaced steps). Then you can use a *for* loop to go through each array element and compute the corresponding area. Summing those up throughout the loop will lead to the Riemann sum estimate.

Included at the end. Note that this is just one possible way to do it. There are a lot of different ways one could go about doing this (e.g., not even using a *for* loop at all!).

(f) Does your code return the same estimates as computed by hand for $N = 2$ and $N = 3$? If not, why? [Hint: it should] How does the estimate change for different values of N (e.g. 5,10, 100,10000)? As N increases, how does it compare to your exact answer?

Computational results should match what you computed by hand for $N = 2, 3$. Running your code, you should see that the estimates improve (i.e., less error) as N increases. It should be apparent that the rate of improvement (with regard to increasing N) differs greatly amongst the various estimates.

(g) What is the effect of varying c? Explain in the contexts of both your analytical answer and numerical simulations. Do both agree?

Varying c essentially has the effect of specifying how quickly the integrand blows up for negative values of x and approaches zero for positive x. The former contributes a significantly larger effect, thus the value of the integral increases with increasing values of c (provided $c > 0$).

(h) What happens when you change the end points (e.g., $\int_{-1.5}^{\pi} \rightarrow \int_{0}^{\pi/2}$)? Does this make sense? Explain.

Changing the limits of integration shifts around the bounds where the *area* is computed. By making the shift $\int_{-1.5}^{\pi} \rightarrow \int_{0}^{\pi/2}$, we would expect less area under the curve (particularly since we exclude $-x$ values where the integrand gets large). This is indeed what we obtain, either from our analytic (part 2) or numerical (part 5) solutions.

2. Consider the integral

$$\int_{-1.5}^{\pi} e^{-cx^2}\, dx$$

where c is a constant greater than zero.

(a) Make a sketch of the integrand and indicate the integral as an *area* to compute. How does this differ from that in Part I?

Unlike Part I, we are now dealing with an even function (i.e., it is symmetric about the vertical axis, such that $f(x) = f(-x)$. We can also see that once $|x|$ gets large (i.e., further from the y–axis), the contribution to the integral becomes increasingly small since the function quickly approaches the x–axis. This function is commonly referred to as the *bell-shaped* curve and has very important implications in practically all branches of science.

(b) Can you find an explicit solution for this integral? If so, give it. If not, explain.

There are means to solve this analytically, but we have not developed the tools to do such. Or put in other terms, there is no simple function that is the anti-derivative of the integrand.

(c) Modify your code from Part I to numerically estimate this integral (again assuming $c = 2$). How does your estimate vary with N?

The change you need to make should be almost trivial if you set your code up properly (see example code at the end). Similar to before, the numerical estimate improves with increasing N.

(d) What is the effect of varying c? Explain in the contexts of both your analytical answer and numerical simulations. Do both agree?

Changing c affects the *width* of the curve. Smaller c leads to a wider curve, while larger c makes it more narrow (as we might expect: this factor c controls how big the argument in the exponent gets and given the negative sign, how quickly the exponential approaches zero). Given that the integrand is constrained to pass through one when $x = 0$, we would expect that increasing c would decrease the value of the integral (which indeed we do see).

(e) Let $c = 1$. Suppose you wanted to know $\int_{-\infty}^{\infty} e^{-x^2}\,dx$. How might you go about computing this? Given sufficiently large N, what value do you find? Using google or wikipedia (or any other source of information at your disposal), search *normal distribution* as well as *error function*. Based upon those resources, can you determine explicitly what the value of estimate approaches? (Hint: it has π in it)

This question deals with some very important concepts in statistics. Briefly, one way to compute the integral is to use a limit, such that

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \lim_{a\to\infty}\int_{-a}^{a} e^{-x^2}\,dx$$

Practically, values as low as $a = 4$ (say N=1000) are plenty sufficient (i.e., contributions for $|x| > 4$ are somewhat negligible for our purposes here). What you should see is that

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}$$

Hence when you see things such as the error function (erf), a factor of $\frac{1}{\sqrt{\pi}}$ appears out front to effectively normalize the value of the integral to one (this has to do with using the integral in the context of a probability distribution: the probability of *something* happening being between 0 and 1, and that probability being 1 when you consider all possibilities occurring). Also note that given the symmetry of the integrand, errors associated with your estimates will cancel out when the interval of integration is symmetric about the y-axis (e.g., LEFT(10)$\approx$ LEFT(10000) for $\int_{-4}^{4} e^{-x^2}\,dx$).

Additional Concepts

- There will always be some degree of error associated with your numerical estimate. For the integral given in Part I, describe clearly how that error changes with N. Is that different for LEFT, RIGHT, TRAP, MID and SIMP? Explain why you think there is a difference.

  LEFT and RIGHT are commonly referred to as *first*–order estimate and converge slowly towards the exact value for increasing values of N. More specifically, the error associated with LEFT and RIGHT decreases (roughly) at the same rate as the step–size decreases. MID and TRAP are called *second*–order estimate and are not only more accurate, but also converge faster with increasing N. Specifically, the error associated with TRAP and MID estimates decreases as (roughly) the square of the step–size decreases. SIMP is even better! The error for this *fourth*–order estimate has its error decrease as (roughly) the fourth power of the step–size decreases. The difference in these rates of decrease stem from how we are estimating the height of the individual rectangles. You can see that some proper thought can really go a long way in terms of reducing computation time. For example (with regard to Part I), the error associated with SIMP for $N = 10$ is roughly 0.02%, less than the 0.05% error for LEFT when $N = 10000$!

- What happens when the limits of integration are flipped (i.e., $\int_{\pi}^{-1.5} e^{-cx}\,dx$)? How does your estimate change? Explain.

  The sign should change, consistent with the notion that $\int_a^b f(x)\,dx = -\int_b^a f(x)\,dx$. This might seem a bit confusing, given the intuition that the integral represents the area underneath the curve and that area doesn't change per se. The key is that the term dx in the integral (i.e., the width of the rectangle in the Riemann sum) has a sign to it: it matters if you go from left to right or from right to left. Remember that we define in the Riemann sum

  $$\Delta x = (b-a)/N$$

  thus the width has a sign by definition.

- Modify your code so to compute the following integral:

  $$\int_{0.17}^{1.32\pi} \cos\left(\frac{\sin\left(x^2\right)}{e^{-x^3}}\right) dx$$

  This problem is a bit more tricky than it seems. Thus you need to do more than just modify your code to take the given function as the integrand. As x gets even remotely large (i.e., once you get above $x > 1$), the exponential term in the denominator of the $\cos$ argument will rapidly approach zero. Thus the amplitude of the argument of the cos will quickly blow up. Given that cos will always vary between -1 and 1, the integrand as a whole is a function that oscillates (in addition to other spectacular things). These oscillations tend to speed up quickly though and even with large N, you will not get a very good representation of the function. Thus while numerically you compute some values that seem to make sense for the integral (and they even converge for increasing N), you are likely not getting a good estimate for the specified integral since you will likely always be inherently undersampling. Your best bet here is to likely restrict your upper bound to something more reasonable, such as 2.

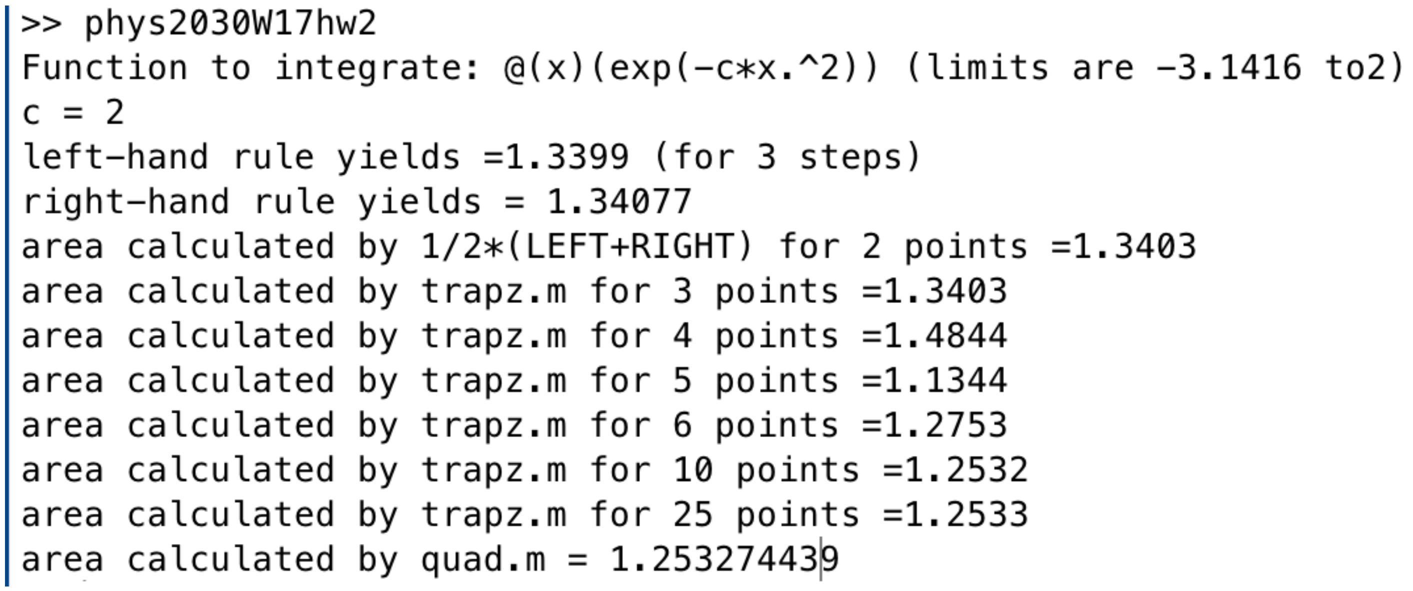
Possible Code

```
% ### riemann.m ### 8.25.09
% code to set up a script to numerically integrate a specified
% function; note user has ability to readily specify the analytic form of
% the function to integrate (as well as the # of divisions and end points)

% main points for students to get here:
% - notion of a for loop
% - specifying user parameters
% - setting up the code such that those parameters can readily be changed
% - simple computations using Matlab

clear

% —————
c= 2; % const.

s1= -1.5; % starting point
e1= pi; % ending point

N= 100; % # of points

% define form of the function
z= @(x) exp(-c*x)
%z= @(x) exp(-c*x.2̂)
%z= @(x) cos(sin(x.2̂)./exp(-x.3̂));

% —————-

x= linspace(s1,e1,N+1); % add one since N is # of 'boxes' and is really N-1

% define the function here
f= z(x);

% plot the function to see what it looks like over interval of integration
plot(x,f,'o-')
xlabel('x'); ylabel('f(x)');
grid on;

% step-size (width of a given rectangle)
delX= (e1-s1)/N;

% ++++++++
% use loop to evaluate LEFT, RIGHT and TRAP
sumL= 0; sumR=0; sumT=0;
for nn=1:N
sumL= sumL + f(nn)*delX;
sumR= sumR + f(nn+1)*delX;
sumT= sumT + 0.5*(f(nn)+f(nn+1))*delX;
```

```
end

% print answers to screen
disp(sprintf('left-hand rule yields = %g', sumL));
disp(sprintf('right-hand rule yields = %g', sumR));
disp(sprintf('trapezoid rule yields = %g', sumT));

% ++++++++
% use midpoint rule (requires resampling)
xM= linspace(s1,e1,2*N+1);
fM= z(xM);
sumM= 0;
for nn=1:N
sumM= sumM + fM(2*nn)*delX;
end
disp(sprintf('midpoint rule yields = %g', sumM));

% ++++++++
% use Simpson's rule
sumS= (1/3)*(sumT + 2*sumM);
disp(sprintf('Simpsons rule yields = %g', sumS));

% ++++++++
% use Matlab's trapz function
sumMAT= trapz(x,f);
disp(sprintf('Matlabs trapz function yields = %g', sumMAT));
% note that this yields essentially the same result (within numerical
% rounding error) of the trapezoid routine coded in above
% ### phys2030W17hw2.m ###  2016.01.17 CB

clear;
% ----------------------
% User parameters
c= 2;    % const. in argument of integral
F = @(x)(exp(-c*x)); % function to integrate
%F = @(x)(exp(-x.^2/2)); % function to integrate
xL= [-pi 2]; % integration limits

N= 2;    % Method A - # of "rectangles" for LEFT and RIGHT
pts= [3 4 5 6 10 25]; % Method B - # of points to consider integrating (via
trapz function)
dur= 1;     % Method B - pause duration [s] for trapz loop
% ----------------------

% ***************
% Show the curve
figure(1);
fplot(F,[xL(1),xL(2)]) % a quick way to plot a function
xlabel('x'); ylabel('F(x)');

% ***************
% Method A
% Approximate the integral via brute force LEFT and RIGHT Riemann sums
sumL= 0; sumR=0;
delX= (xL(2)-xL(1))/N;    % step-size
x= linspace(xL(1),xL(2),N+1);  % add one since N is # of 'boxes' and is
really N-1
for nn=1:N
    sumL= sumL + F(x(nn))*delX;
    sumR= sumR + F(x(nn+1))*delX;
end
Fs= func2str(F);
disp(['Function to integrate: ',Fs,' (limits are ',num2str(xL(1)),'
to',num2str(xL(2)),')']);
disp(['c = ',num2str(c)]);
disp(['left-hand rule yields =',num2str(sumL),' (for ',num2str(N+1),'
steps)']);
disp(sprintf('right-hand rule yields = %g', sumR));
disp(['area calculated by 1/2*(LEFT+RIGHT) for ',num2str(N),' points
=',num2str(0.5*(sumL+sumR))]);

% ***************
% Method B
% Approximate the integral via trapz for different numbers of points
for np=pts
    figure(2); clf % clear the current figure
    hold on % allow stuff to be added to this plot
    x = linspace(xL(1),xL(2),np); % generate x values
    y = F(x); % generate y values
    a2 = trapz(x,y); % use trapz to integrate
    % Generate and display the trapezoids used by trapz
    for ii=1:length(x)-1
        px=[x(ii) x(ii+1) x(ii+1) x(ii)];   py=[0 0 y(ii+1) y(ii)];
        fill(px,py,ii)
    end
    fplot(F,[xL(1),xL(2)]); xlabel('x'); ylabel('F(x)');
    disp(['area calculated by trapz.m for ',num2str(np),' points
=',num2str(a2)]);
    title(['area calculated by trapz.m for ',num2str(np),' points
=',num2str(a2)]);
    pause(dur);   % wait a bit
end

% ***************
% Method C
a1 = quad(F,xL(1),xL(2)); % use quad to integrate
msg = ['area calculated by quad.m = ' num2str(a1,10)]; disp(msg);
```

```
>> phys2030W17hw2
Function to integrate: @(x)(exp(-c*x)) (limits are -3.1416 to2)
c =2
left-hand rule yields =1384.6911 (for 3 steps)
right-hand rule yields = 8.09818
area calculated by 1/2*(LEFT+RIGHT) for 2 points =696.3946
area calculated by trapz.m for 3 points =696.3946
area calculated by trapz.m for 4 points =489.6538
area calculated by trapz.m for 5 points =401.1442
area calculated by trapz.m for 6 points =356.0616
area calculated by trapz.m for 10 points =296.2491
area calculated by trapz.m for 25 points =271.8202
area calculated by quad.m = 267.73667
```

```
% ### phys2030W17hw2.m ###  2016.01.17 CB

clear;
% ----------------------
% User parameters
c= 2;    % const. in argument of integral
F = @(x)(exp(-c*x.^2)); % function to integrate
%F = @(x)(exp(-x.^2/2)); % function to integrate
xL= [-pi 2]; % integration limits

N= 2;    % Method A - # of "rectangles" for LEFT and RIGHT
pts= [3 4 5 6 10 25]; % Method B - # of points to consider integrating (via
trapz function)
dur= 1;     % Method B - pause duration [s] for trapz loop
% ----------------------

% ***************
% Show the curve
figure(1);
fplot(F,[xL(1),xL(2)]) % a quick way to plot a function
xlabel('x'); ylabel('F(x)');

% ***************
% Method A
% Approximate the integral via brute force LEFT and RIGHT Riemann sums
sumL= 0; sumR=0;
delX= (xL(2)-xL(1))/N;    % step-size
x= linspace(xL(1),xL(2),N+1);  % add one since N is # of 'boxes' and is
really N-1
for nn=1:N
    sumL= sumL + F(x(nn))*delX;
    sumR= sumR + F(x(nn+1))*delX;
end
Fs= func2str(F);
disp(['Function to integrate: ',Fs,' (limits are ',num2str(xL(1)),'
to',num2str(xL(2)),')']);
disp(['c = ',num2str(c)]);
disp(['left-hand rule yields =',num2str(sumL),' (for ',num2str(N+1),'
steps)']);
disp(sprintf('right-hand rule yields = %g', sumR));
disp(['area calculated by 1/2*(LEFT+RIGHT) for ',num2str(N),' points
=',num2str(0.5*(sumL+sumR))]);

% ***************
% Method B
% Approximate the integral via trapz for different numbers of points
for np=pts
    figure(2); clf % clear the current figure
    hold on % allow stuff to be added to this plot
    x = linspace(xL(1),xL(2),np); % generate x values
    y = F(x); % generate y values
    a2 = trapz(x,y); % use trapz to integrate
    % Generate and display the trapezoids used by trapz
    for ii=1:length(x)-1
        px=[x(ii) x(ii+1) x(ii+1) x(ii)];   py=[0 0 y(ii+1) y(ii)];
        fill(px,py,ii)
    end
    fplot(F,[xL(1),xL(2)]); xlabel('x'); ylabel('F(x)');
    disp(['area calculated by trapz.m for ',num2str(np),' points
=',num2str(a2)]);
    title(['area calculated by trapz.m for ',num2str(np),' points
=',num2str(a2)]);
    pause(dur);   % wait a bit
end

% ***************
% Method C
a1 = quad(F,xL(1),xL(2)); % use quad to integrate
msg = ['area calculated by quad.m = ' num2str(a1,10)]; disp(msg);
```



```
>> phys2030W17hw2
Function to integrate: @(x)(exp(-c*x.^2)) (limits are -3.1416 to2)
c = 2
left-hand rule yields =1.3399 (for 3 steps)
right-hand rule yields = 1.34077
area calculated by 1/2*(LEFT+RIGHT) for 2 points =1.3403
area calculated by trapz.m for 3 points =1.3403
area calculated by trapz.m for 4 points =1.4844
area calculated by trapz.m for 5 points =1.1344
area calculated by trapz.m for 6 points =1.2753
area calculated by trapz.m for 10 points =1.2532
area calculated by trapz.m for 25 points =1.2533
area calculated by quad.m = 1.253274439
```