Definite Integrals

Leigh J. Little SUNY Brockport November 18, 2022

Contents

1 Now What? 1

2 Introduction 1
2.1 No Anti-derivative . . . 2
2.2 Function Not Known . . . 2

3 Notation for Tabular Data 2

4 The Trapezoidal Rule 3
4.1 Example . . . 5
4.2 Two Parts to the Story . . . 6

5 Improving the Trapezoidal Rule 6
5.1 The Trapezoidal Rule Function . . . 7
5.2 One More Function . . . 7
5.3 The New Main Driver . . . 7
5.4 Accuracy of the Trapezoidal Method . . . 8

6 Simpson's Rule 8

7 Summary of Trapezoidal and Simpson's Rules 9

8 Another Approach to Definite Integrals 9
8.1 MATLAB `integral` Function . . . 10
8.2 Example 1: $f(x) = x^2$ . . . 10
8.3 Example 2: $f(x) = x^2 e^{-x}$ . . . 11

1 Now What?

Now that a solid base of programming principles have been covered, it is a good time to ask what can we do with this? At this point, there is much that ... It is instructive to examine some applications of the programming topics that have been covered, but to also start incorporating some concepts that need to considered were these examples to be run in a real-world situation.

2 Introduction

Definite integrals of the form

$$I = \int_a^b f(x)\,dx$$

comprise an important component to mathematical and scientific problems. Integrals are used to determine net changes over a range of space or time. Areas, volumes, total mass, total amount of toxins removed, *etc.* can all be determined by evaluating a particular definite integral. If an anti-derivative of $f(x)$ exists, you can use the fundamental theorem of calculus to evaluate the integral

$$I = \int_a^b f(x)\,dx = F(b) - F(a)$$

where $F(x)$ is the anti-derivative of $f(x)$.

A problem is encountered when a computer has to evaluate an integral (note that this does not include symbolic manipulators like Maple and Mathematica). A computer can't do the algebraic steps necessary to compute the integral and thus must resort to some type of process to produce an approximation to the value

of the integral. However, there are two cases that arise where even symbolic manipulators need to resort to the numerical approximations we will discuss here.

2.1 No Anti-derivative

For the first case, consider the integral

$$I = \int_{\pi}^{2\pi} \frac{\sin x}{x}\,dx.$$

This integral is important in the physics of waves. However, there is no anti-derivative for $\frac{\sin x}{x}$, so the fundamental theorem of calculus can't be used to determine the value of the integral. A quick plot of $\frac{\sin x}{x}$ over the interval $[\pi, 2\pi]$ reveals that the value of this integral exists. How do we get an approximate value for this integral?

There are many common functions that don't have anti-derivatives. For example the indefinite integrals

$$\int e^{-x^2/2}\,dx, \quad \int \sin x^2\,dx, \quad \int \sqrt{1+x^4}\,dx$$

all arise in various physical problems, but have no anti-derivatives.

It turns out that in a probabilistic sense, if you were to pull a random function $g(x)$ out of a bag that held all possible functions, the probability that you would be able to find an anti-derivative of $g(x)$ is zero. A comprehensive table of derivatives contains about 40-45 formulas that can be used to take the derivative of any function in a straightforward way. However, tables of integrals can span several large volumes containing thousands of formulas.

2.2 Function Not Known

The second case where the fundamental theorem of calculus can't be used to compute a definite integral arises when the function $f(x)$ is not known in a functional form; instead, all that is known is a table of approximate values. This situation occurs frequently in scientific computing.

3 Notation for Tabular Data

Before beginning the discussion of the trapezoidal rule, it is instructive to discuss some mathematical notations related to tabular data.

The most common type of table is one where the x-coordinates are equally spaced. Suppose that we want to subdivide the interval $[a, b]$ into n equal parts. This can be done using the formulas

$$\begin{aligned} h &= \frac{b-a}{n} \\ x_i &= a + (i-1)h, \quad i = 1, 2, \ldots, n+1. \end{aligned}$$

These are the formulas that MATLAB uses when you use the `linspace` command. Note that we are using n in a different manner than we have previously. This usage is consistent with how other discussions of this topic define n.

There are a total of $n+1$ discrete points and n subdivisions. It helps to think of it this way; if you were to step from a to b in step sizes of h, you would take a total of n steps, but you would occupy a total of $n+1$ points in space during the process.

The hard part of tabulating a simple, known $y = y(x)$ type function is generating the x-coordinates. The corresponding y-coordinates simply go along for the ride. The resulting table can be written as (assuming $n = 4$) in Table 1. Note that x_1 is the same as a and x_5 is the same as b.

For computing purposes, it is helpful to work with a simpler notation for the y-coordinates. The $y(x_i)$ notation is bulky and can sometimes obscure the importance of the calculation being performed. Instead, we will use the shorthand notation $y_i = y(x_i)$. This simply means that the value of y_i is the value of $y(x)$ at the corresponding x_i. With this notation, the table of values can be written as in Table 2

x	$y(x)$
$x_1 = a$	$y(x_1)$
x_2	$y(x_2)$
x_3	$y(x_3)$
x_4	$y(x_4)$
$x_5 = b$	$y(x_5)$

Table 1: Table of values for some function $y(x)$ with $n = 4$. Notice that there are a total of 5 discrete points.

x	$y(x)$
$x_1 = a$	y_1
x_2	y_2
x_3	y_3
x_4	y_4
$x_5 = b$	y_5

Table 2: Simplified table of values.

4 The Trapezoidal Rule

One of the easiest methods for estimating the value of a definite integral is the trapezoidal rule. This method is often discussed in a first semester calculus course, but it also integrates (pun intended) well with the MATLAB material we have been discussing.

Consider the function $y(x)$ shown in Figure 1. We would like to estimate the value of $I = \int_a^b y(x)\,dx$.

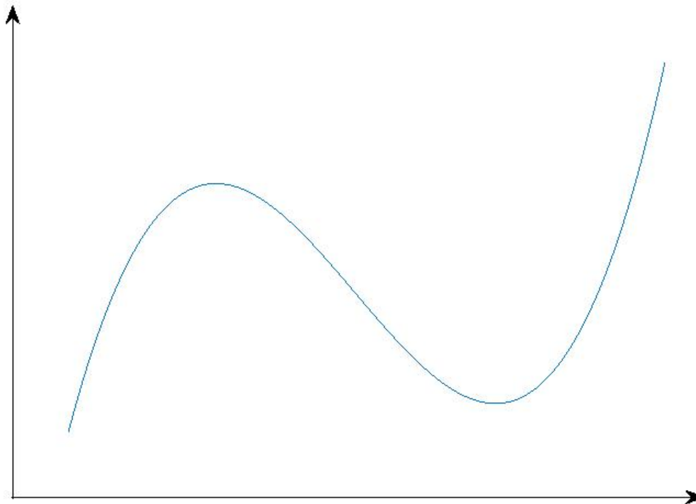

Figure 1: Some function $y(x)$ to integrate.

To estimate the value of the integral via the trapezoidal rule, the first step is to tabulate the function. This also means that we need to decide how many values in the table we would like. We will soon see that this plays a critical role. For now, assume we decide to have a total of 5 values in the table and the x-coordinates are equally spaced.

Now plot these points as shown in Figure 2 along with a graph of the function whose definite integral is desired. We can now draw 4 trapezoids that very coarsely approximate the area under the curve. Furthermore, we have enough information to compute the area of each of these trapezoids. For example, the first

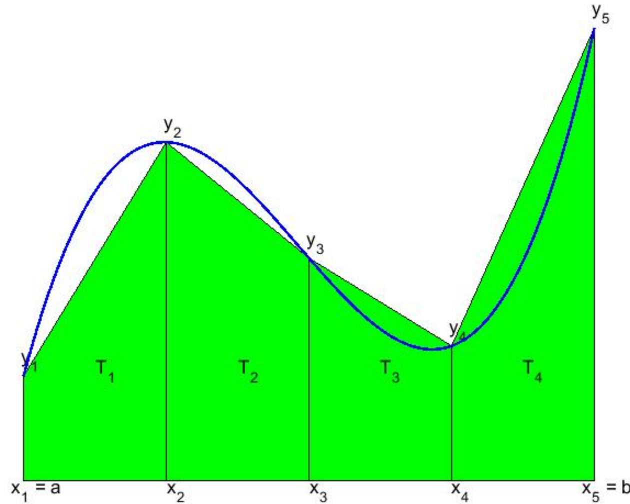

Figure 2: Trapezoidal method for $n = 4$.

trapezoid T_1 has an area of

$$\text{Area of } T_1 = \frac{1}{2}h(y_1 + y_2).$$

The height h of the trapezoid is the spacing between the x-coordinates and the two base lengths are the function values at the these x-coordinates.

The areas of the remaining trapezoids are

$$\text{Area of } T_2 = \frac{1}{2}h(y_2 + y_3),$$

$$\text{Area of } T_3 = \frac{1}{2}h(y_3 + y_4),$$

$$\text{Area of } T_4 = \frac{1}{2}h(y_4 + y_5).$$

This results in the crude approximation

$$I = \int_a^b y(x)\,dx \approx T_1 + T_2 + T_3 + T_4.$$

At first, this does not seem like a good technique, but remember that we can choose any number of trapezoids we wish. For example, if we decide to use 8 trapezoids instead of 4, we get the approximation shown in Figure 3. This seems to be a much better approximation.

Before beginning to write a MATLAB code for the trapezoidal rule, we should generalize the formula to any number of trapezoids and make some simplifications. Examine the formula for 4 trapezoids from before. Note that for the trapezoids in the middle, the right base of one trapezoid forms the left base of the next. This means that there are some common terms that can be added. We have

$$\begin{aligned} T_1 + T_2 + T_3 + T_3 &= \frac{1}{2}h(y_1 + y_2) + \frac{1}{2}h(y_2 + y_3) + \frac{1}{2}h(y_3 + y_4) + \frac{1}{2}h(y_4 + y_5) \\ &= \frac{1}{2}h(y_1 + 2y_2 + 2y_3 + 2y_4 + y_5). \end{aligned}$$

If we extend this formula for n trapezoids, we obtain the general trapezoidal rule

$$I_{\text{trap}} \approx \frac{h}{2}(y_1 + 2y_2 + 2y_3 + \cdots + 2y_{n-1} + 2y_n + y_{n+1}). \quad (1)$$

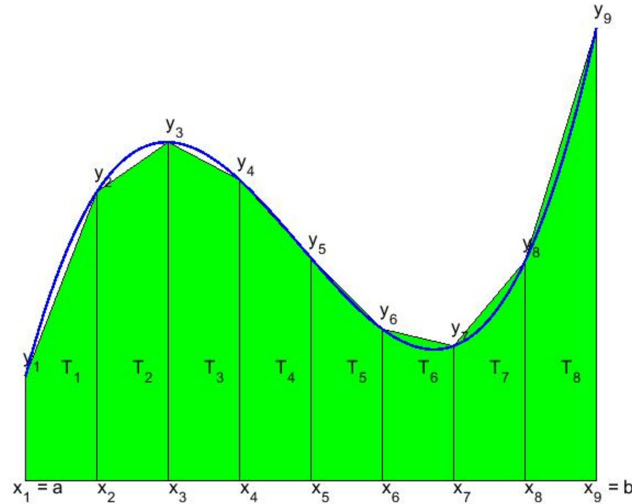

Figure 3: Trapezoidal method for $n = 8$.

Based on what we have done so far, you should see that we will need a loop somewhere in our code because this looks like we are adding up a series of values and multiplying the result by a scaling factor. In fact, this formula is an example of a *weighted average*. Rather than add the terms and divide by the number of terms, some terms are multiplied by different scaling factors (called *weights*) and the resulting sum is multiplied by a slightly different factor. For the trapezoidal rule, the weight of the first and last terms in the sum is one and the weight of all the other terms is two.

An outline of the code would be:

1) Assign the values of a and b.
2) Input a value for n
3) Compute the value of h
4) Generate the set of x-coordinates
5) Compute the corresponding y-coordinates
6) Execute a loop to accumulate the sum inside the parentheses of Equation (1)
7) Multiply this sum by $\frac{1}{2}h$

4.1 Example

For this example, suppose we want to estimate the value of

$$I = \int_0^2 x^4 e^{-x}\,dx = 24 - \frac{168}{e^2}.$$

This integral has a known value, so we can compare the approximation from the trapezoidal rule to this to gain insight into the accuracy of the method.

The MATLAB code to do this is given below:

```
clear
a = 0;
b = 2;
n = input('Input n ');
```

```
h = (b-a)/n;              % Could also do h = x(2)-x(1), but this is normally only
                          % done when one of a, b or n is not known.
x = linspace(a,b,n+1)';   % Note that we are generating n + 1 points.
y = x.^4.*exp(-x);

total = 0;                          % Set accumulation variable to 0
for i = 1:n+1
    if(i == 1 | i == n+1)
       total = total + y(i);        % The endpoints get a weight of 1
    else
       total = total + 2*y(i);      % The interior nodes get a weight of 2
    end
end
approx = h*total/2;                 % Compute integral approximation
exact = 24 - 168/(exp(1)^2);
relerr = (approx-exact)/exact
```

If this code is run for several values of n, the results below are obtained:

```
>> traprule
Input n 4
relerr =
    -3.924912724959407e-01
>> traprule
Input n 8
relerr =
    -2.052588080461482e-01
>> traprule
Input n 16
relerr =
    -1.048650320435156e-01
```

As implied by Figures 2 and 3, the error seems to improve as the number of trapezoids increases. However, the error does not seem to be decreasing very fast.

4.2 Two Parts to the Story

The trapezoidal rule comes from a branch of mathematics known as *numerical analysis*. The aim of numerical analysis is to obtain methods for solving various types of math problems (computing approximations to derivatives, integrals, differential equations, linear systems, etc.). There are always two parts to a method like the trapezoidal rule:

- The method itself. This is the sequence of calculations necessary to compute the approximation.
- The mathematical proof that the method works. In the case of the trapezoidal rule, this would be a proof the the absolute error goes to zero in the limit as n approaches infinity. We won't worry about the proofs in this class, but it is important to be aware that there is a great deal of rigorous theory that justifies the methods we will use.

5 Improving the Trapezoidal Rule

The trapezoidal rule code we wrote works, but can be improved by writing portions of it as functions. The rationale for this is that this is a frequently performed calculation. In a situation where it is needed, the person needing it might be using variable names other than x, y and h. Writing it as a function eliminates the need to go through the code and change variable names (and possibly change a working code into a non-working code in the process).

5.1 The Trapezoidal Rule Function

Prior to writing the function, the section of the code that performs the trapezoidal rule needs to be identified. In particular, we want to isolate just the section that does the trapezoidal rule. This corresponds to the lines

```
total = 0;
for i = 1:n+1
    if(i == 1 | i == n+1)
       total = total + y(i);
    else
       total = total + 2*y(i);
    end
end
approx = h*total/2;
```

The lines before this are tabulating the function values and the lines after this are computing the relative error. Also, note that this is the only section in the program that uses the variable `total`. Furthermore, the information that is needed are the step size `h` and the array of function values `y`. This means that we only need to send in `h` and `y` as inputs. The result we are looking for is `approx`, so this becomes the output variable. We obtain the following:

```
function [approx] = trap_fun(h,y)
n = length(y) - 1;                  %  Remember that n is the number of trapezoids
                                    %  The length of y is n + 1
total = 0;
for i = 1:n+1
    if(i == 1 | i == n+1)
       total = total + y(i);
    else
       total = total + 2*y(i);
    end
end
approx = h*total/2;
```

5.2 One More Function

One final change to make would be to evaluate the integrand (*i.e.* the function values) in a function rather than in the main code section. We could do something like

```
function [y] = yofx(x)
y = x.^4.*exp(-x);
```

5.3 The New Main Driver

With these changes, the main driver routine now looks like

```
clear
a = 0;
b = 2;
n = input('Input n ');
h = (b-a)/n;
x = linspace(a,b,n+1)';
y = yofx(x);

approx = trap_fun(h,y);
exact = 24 - 168/(exp(1)^2);
relerr = (approx-exact)/exact
```

This code will give the same results as the previous one.

5.4 Accuracy of the Trapezoidal Method

As mentioned previously, a second part of the trapezoidal method is a proof that the absolute error in the approximate value goes to 0 in the limit as n goes to infinity. This is beyond the scope of this course, but you can find the proof in any numerical analysis textbook. The main outcome of the proof is that the absolute error in the trapezoidal method can be expressed as

$$|\text{approximate value} - \text{exact value}| \leq Ch^2.$$

where C is a constant that depends on $f(x)$. Because h goes to 0 as n goes to infinity, this means that the absolute error goes to 0. We say that the trapezoidal rule is $O(h^2)$ accurate.

6 Simpson's Rule

Another rule for estimating the values of definite integrals that you may have heard of before is Simpson's Rule. There is an analogy between the development of integration algorithms and the development of interpolation algorithms we did previously.

The trapezoidal method fits a line to consecutive points in a table and then computes the area of the trapezoid that lies beneath the line. However, a line is not terrible accurate.

When we were examining interpolation, we found that if we take 3 consecutive points in a table and fit a quadratic to these points, we would obtain greater accuracy.

This is the idea behind Simpson's method. In this case, we take 3 points in the table, fit a quadratic to these points, then integrate this quadratic to determine the area underneath it (see Figure 4).

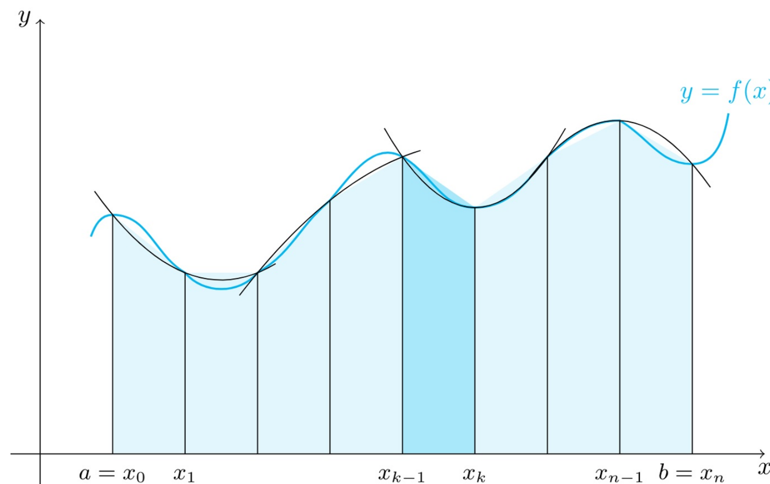

Figure 4: Simpson's method for approximating a definite integral (image from newbedev.com). The points in the table are taken in groups of three. The resulting quadratics are integrated to produce an approximate value. Note that the subscripts on the x-coordinates start at 0 instead of 1.

Deriving the formula for Simpson's rule is much more complicated than for the trapezoidal rule. The surprising outcome is that if the x-coordinates in the table are equally spaced, then the formula for the Simpson's approximation is very similar to the trapezoidal formula

$$I_{\text{Simpson}} \approx \frac{h}{3}\left(y_1 + 4y_2 + 2y_3 + \cdots + 2y_{n-1} + 4y_n + y_{n+1}\right).$$

Note that the weights of the y_i terms is different than for the trapezoidal rule. The first and last terms have a weight of 1. The inner terms with an even numbered subscript have a weight of 4 and the inner terms with an odd numbered subscript have a weight of 2. A function that will compute this is given by:

```
function [approx] = simp_fun(h,y)
n = length(y) - 1;
```

```
total = 0;
for i = 1:n+1
    if(i == 1 | i == n+1)
       total = total + y(i);
    elseif(mod(i,2) == 0)
       total = total + 4*y(i);
    else
       total = total + 2*y(i);
    end
end
approx = h*total/3;
```

There is one important drawback to Simpson's rule and that is that n has to be an even number (*i.e.*, the number of points in the table must be odd). The trapezoidal rule does not have this restriction.

A formal proof of Simpson's rule reveals that the absolute error can be expressed as

$$|\text{approximate value} - \text{exact value}| \leq Ch^4.$$

We say that Simpson's rule is $O(h^4)$ accurate. What this means in a practical sense is that Simpson's rule is much more accurate than the trapezoidal rule and requires about the same amount of computational effort. This is why Simpson's rule is the most common integration rule that is used (when it is possible to do so).

7 Summary of Trapezoidal and Simpson's Rules

Here is a summary of the trapezoidal and Simpson's Rules:

- The trapezoidal rule is $O(h^2)$ accurate.
- Simpson's Rule is $O(h^4)$ accurate.
- Because h usually satisfies $h < 1$ we have the inequalities

  $$1 > h > h^2 > h^3 > h^4 > \cdots.$$

  This means that a method that has a larger power of h in its order of accuracy is going to be more accurate than a method with a lower power (asymptotically in the limit as $h \to 0$).
- The trapezoidal method can easily be modified to handle the case when the x-coordinates in the table are not equally spaced. This is not true for Simpson's Rule.
- MATLAB has a `trapz` trapezoidal method function that will integrate a given table of values for either equally or non-equally spaced points. It does not have a basic Simpson's Rule function (for the reason in the last bullet).
- You should use Simpson's rule for simple approximations when possible to do so (equally spaced x-coordinates and an odd number of points in the table) because it is significantly more accurate than the trapezoidal method for about the same amount of computational effort.

8 Another Approach to Definite Integrals

We have examined the trapezoidal rule for approximating definite integrals. There are many other methods for estimating integrals. These include the midpoint and Simpson's Rules along with more sophisticated methods such as Gaussian quadrature. These methods all have two things in common:

- The calculation ultimately reduces to computing a weighted average of function values.
- The value of n is an input to the method.

The second item can be a problem. As you saw in the homework assignment, there is no 'formula' that will tell you how large n needs to be in order to obtain a desired level of accuracy. This value varies from problem to problem. Arbitrarily setting n to a large value like $n = 10,000,000$ is a bad idea because this is certainly going to result in more work than is needed. It will also require a large number of resources in terms of the memory needed to store the table of function values.

It would be convenient if a method could be devised that determined the value of n automatically. This would ensure that the integral had the necessary accuracy without any interaction from the user. Such methods exist and are called *adaptive* methods. Programming an adaptive method is beyond the scope of this course, but fortunately MATLAB provides a very good adaptive method.

8.1 MATLAB `integral` Function

In most cases MATLAB's `integral` function can be used to evaluate definite integrals. This function will produce an approximate integral value to any desired degree of accuracy and can even handle infinite limits. How is this function used? The basic calling syntax for the `integral` function is

```
I = integral(@function_name,lower,upper)
```

Here, `lower` is the lower limit of the integral and `upper` is the upper limit of the integral. `@function_name` is called a *function handle*. It is easier to explain what this is with an example.

8.2 Example 1: $f(x) = x^2$

As with any new function, the best way to get an idea of how to properly use the function is to test it on problems you already know the answer to. The definite integral

$$I = \int_a^b x^2 \, dx$$

can easily be shown to be

$$I = \frac{1}{3}(b^3 - a^3).$$

In order to use the `integral` function to compute this integral, we need to write our own MATLAB function that evaluates the integrand. In this case, the integrand is x^2.

We will name the function `xsq.m` Enter the following into the `xsq.m` file and save it.

```
function y = xsq(x)
y = x.^2;
```

Note that the function is set up in such a way that it will work if x is a vector or a scalar.

We can then use the `integral` function using the following syntax:

```
>> I = integral(@xsq,2,3)   % Note:  @xsq contains the integrand
ans =
    6.3333
```

This will evaluate

$$I = \int_2^3 x^2 \, dx = \frac{19}{3}.$$

The answer is exact to the number of digits displayed. Note how the information regarding the function we want to evaluate is provided. In order to evaluate a definite integral, we need to first write a MATLAB function that evaluates the integrand. This function *must* be written in such a way that it will work if x is either a vector or a scalar.

Recall that this is only a numerically calculated approximation to the integral. It is correct to the digits displayed, but how accurate is this approximation? To see this, calculate the relative error

```
>> I = integral(@xsq,2,3)
I =
  6.3333
>> relerror = (I-19/3)/19/3
relerror =
   3.1164e-17
```

This error indicates that we have 16 digits of accuracy (*i.e.*, it is as exact as we could hope for on an IEEE 754 computer system). Due to the method that the `integral` function uses to compute the approximation, it will always do well in evaluating integrals of polynomials.

8.3 Example 2: $f(x) = x^2 e^{-x}$

You can show that

$$I = \int_a^b x^2 e^{-x}\,dx = -e^{-x}(x^2+2x+2)\Big|_a^b$$

To use the integral function to compute this, first write a MATLAB function called `efun.m` containing the following:

```
function y = efun(x)
y = x.^2.*exp(-x);
```

Note the use of the dot operators to ensure that this will evaluate properly if `x` is either a scalar or a vector.

We can now compute the integral

$$I = \int_{-1}^{2} x^2 e^{-x}\,dx$$

```
>> I = integral(@efun,-1,2)
I =
   1.364928996092919e+00
>> a = -1;
>> ll = -exp(-a)*(a^2+2*a+2);
>> b = 2;
>> ul = -exp(-b)*(b^2+2*b+2);
>> exact = ul-ll
exact =
       1.364928996092919e+00
>> rel = abs(I-exact)/exact
rel =
      0
```

As in the first case, the integral gives an approximation that is exact to machine precision.