More On Definite Integrals

Leigh J. Little SUNY Brockport November 1, 2021

Contents

1 Integrals with Parameters

Consider the definite integral

$$\int_a^b e^{rx}\,dx.$$

The variable r is a new type of variable relative to the previous definite integrals we have worked with. The r variable is called a *parameter*.

If you look at a table of integrals, you will see many formulas that have parameters. This is because the integration of a base form (like e^x) where the dummy variable of integration is multiplied by a constant is a very common alteration to the base form. It allows the reduction of many similar integrals to a single case. For the example above, we can use an integral substitution to obtain

$$\int_a^b e^{rx}\,dx = \frac{1}{r}e^{rx}\Big|_a^b.$$

Now we can use this formula to evaluate the integrals

$$\begin{aligned}\int_0^1 e^{2x}\,dx &= \frac{1}{2}e^{2x}\Big|_0^1 = \frac{1}{2}\left(e^2-1\right),\\ \int_{-1}^2 e^{-3x}\,dx &= \frac{1}{-3}e^{-3x}\Big|_{-1}^2 = \frac{1}{-3}\left(e^{-6}-e^3\right).\end{aligned}$$

Can we get MATLAB's `integral` function to evaluate integrals that have parameters? The answer is yes, but we will need a slightly different syntax to do it.

1.1 Example 1

Consider the base form examined above:

$$\int_a^b e^{rx}\,dx.$$

Write a MATLAB function that will evaluate the integrand. Note that we also need to send in the variable `r` to this function.

```
function [y] = integ1(x,r)
y = exp(r*x);
```

Now consider the specific case

$$\int_0^1 e^{2x}\,dx.$$

We can see that for this instance, $r = 2$. Based on what we did last time, it would appear that we need to do something like

```
clear
a = 0;
b = 1;
r = 2;
I = integral(@integ1,a,b);
```

However, there is now a problem. How do we let the `integral` function know about `r`? Programmatically the integrand has two variables, `x` and `r`. We need a way to tell MATLAB that `x` is the dummy variable of integration and `r` is the parameter. This problem can be solved using a slightly different form of the `integral` function. Instead of

```
I = integral(@integ1,a,b);
```

use the syntax

```
I = integral(@(x)integ1(x,r),a,b);
```

instead. The syntax `@(x)` tells MATLAB that the `x` variable in `integ1(x,r)` is the dummy variable of integration. The final version of the script would look like

```
clear
a = 0;
b = 1;
r = 2;
I = integral(@(x)integ1(x,r),a,b);
exact = (exp(2)-1)/2;
relerr = (I-exact)/exact
```

This gives the output

```
relerr =
  -1.3902e-16
```

which indicates that the approximate integral value `I` is more or less exact to machine precision.

1.2 Example 2

Now consider the integral

$$\int \cos(rx)\sin(sx)\,dx.$$

This integral can be evaluated using both integral substitution and integration by parts to obtain

$$\int \cos(rx)\sin(sx)\,dx = \begin{cases} \frac{1}{2r}\sin^2(rx), & r = s \\ \frac{1}{r^2-s^2}\left(r\sin(rx)\sin(sx) + s\cos(rx)\cos(sx)\right), & r \neq s \end{cases}$$

Note that we only need this to compute an exact value for comparison.

Write a MATLAB function to evaluate the integrand. This integrand has *two* parameters, r and s.

```
function [y] = integ2(x,r,s)
y = cos(r*x).*sin(s*x);
```

Suppose we want to evaluate

$$\int_0^{\pi} \cos(2x)\sin(3x)\,dx = \frac{6}{5}.$$

Here we can see that $r = 2$ and $s = 3$. The script that will approximate this integral is shown below:

```
clear
a = 0;
b = pi;
r = 2;
s = 3;
I = integral(@(x)integ2(x,r,s),a,b);
exact = 6/5
relerr = (I-exact)/exact
```

Again, we have used the `@(x)` syntax to tell the `integral` function that `x` is the dummy variable of integration and `r` and `s` are the parameters. This script gives the output

```
relerr =
     0
```

which again indicates that the approximate integral `I` is axact to machine precision.

2 Integration with a Table of Values

Suppose you want to compute the definite integral

$$\int_a^b f(x)\,dx.$$

but you don't know what the function $f(x)$ is. Instead, all you know is a table of values for $f(x)$.

The approach you would adopt in this situation depends on how much accuracy you require. One technique would be to use the trapezoidal method or Simpson's Rule like you did in Homework 22.

However a third option would be to use what we learned about splines. We have seen that the cubic spline allows you to determine unknown values from a table of values and that this process has very good accuracy. We build the spline from the table to values, then integrate the spline using the `integral` function.

2.1 Example 3

Suppose you have the table of values given in `integ3.dat`. We can load these into MATLAB using

```
load integ3.dat
x = integ3(:,1);
y = integ3(:,2);
```

then build the cubic spline using

```
T = spline(x,y);
```

Assuming we want to integrate over the entire range of `x`, we can get the range using

```
a = x(1);
p = length(x);
b = x(p);
```

Finally, we can compute the approximate integral using

```
I = integral(@(x)ppval(T,x),a,b);
```

Note that here we have used the built-in function `ppval`. Recall that this function evaluates the spline `T` at the given value of `x`. We have use the syntax `@(x)ppval(T,x)` to tell MATLAB that `x` is the variable of integration and `T` is the parameter.

The exact value of the function tabulated in `integ3.dat` is

$$I_{\text{exact}} = e^2 + 1.$$

This allows us to see how accurate this process is. The final version of the script in this example is

```
clear
load integ3.dat    % Load the data
x = integ3(:,1);
y = integ3(:,2);
T = spline (x,y);  % Create the cubic spline

a = x(1);          % Get the limits of integration
p = length(x);
b = x(p);
```

```
I = integral(@(x)ppval(T,x),a,b);  % Integrate the spline beween a and b
exact = 1 + exp(2);
relerr = (I-exact)/exact
```

This script gives the output

```
relerr =
    1.4138e-05
```

This error indicates that we have about 4 digits of accuracy in the approximate integral `I`. Note that this accuracy is much less than for the first two examples. This is because $f(x)$ is only available as a table of values.

For comparison, the approximate integral using the trapzoidal rule would be

$$I_{\text{trapezoidal error}} = 8.4016e-03.$$

and the relative error using Simpson's Rule is

$$I_{\text{Simpson error}} = 3.5733e-05.$$

Of the three approaches, the cubic spline technique gives the most accurate answer.

An important consideration is how much computational effort each of these methods require. If you look at the trapezoidal function and count the number of additions, subtractions, multiplications and divisions performed, you obtain a total of $3n + 6$ operations, where n is one less than the length of the input vector `y`. We say that the trapezoidal method has an $O(3n)$ work effort. As n gets large, the fixed operation count of 6 becomes less important, so it can be ignored relative to the $3n$ leading order term. Similarly, Simpson's Rule also has an operation count of $O(3n)$.

Performing an operation count for the `integral` function is less obvious because we can't see exactly what it is doing. Estimates for the computation of the spline are known. Creating the spline `T` requires approximately $O(14n)$ operations and the `ppval` function requires about 6 operations for each value of `x`. The effort for the `integral` function is not known, but clearly the cubic spline approach is going to require more effort than either the trapezoidal or Simpson's rules.

The discussion above illustrates an important aspect of scientific computing; higher accuracy requires more computational effort. In a real situation, the relative errors above seem to indicate that the cubic spline approach should not be used unless accuracy is critical. Simpson's Rule provides an accuracy that is nearly as good and requires much less computational effort and would be the better choice.