Lab 4: Numerical Integration

There are certain integrals, important integrals, that we cannot integrate analytically. For example, we cannot compute $\int_0^4 e^{-x^2}\,dx$ analytically. There are other integrals that we probably could compute analytically if we worked very hard or if our computer algebra system is good enough, but we only need a reasonably good numerical value. When you use your TI83 to integrate (and you will), the calculator does not integrate analytically and use the Fundamental Theorem of Calculus (it doesnt know how to integrate analytically). The TI83, 85 and 86 (and maybe more) use some sort of numerical scheme to find an approximate value of the integral.

For these reasons, we want to be able to compute an approximation of a definite integral. The first approach is reasonably obvious. We recall that the Riemann sums that we used to define the definite integral in M160. In the section on numerical integration in the textbook, we are introduced to the trapezoidal rule and Simpsons rule. In this lab we will show how to implement five numerical integration schemes in Matlab (right and left Riemann sums, midpoint rule, trapezoidal rule and Simpsons rule), compare the accuracy of these techniques and discuss the error of some of these schemes.

We begin by introducing you to Matlabs Riemann sum tool. Its not amazing. It may be somewhat helpful. The command gives a Riemann sum picture and value for integrals on $[0,1]$. For example, if you type

```
x=sym(x);
rsums x^2+sin(x)
```

A window appears that lets you play with the number of partitions used in a Riemann sum definition of $\int_0^1 \left(x^2+\sin x\right)dx$. Play with it for a while.

When writing codes for this lab, it is surely best to use M-files for each approximate integration scheme. We remind you how to use M-files by writing the code for computing left Riemann sums. We begin by clicking on `file` on the Matlab Command Window and `new`. You have the choice of `M-file` or `figure`. Choose `M-file`. You will get a new window called the `Matlab Editor/Debugger`. In the editor window type the following code.

```
delx=(b-a)/number;
lsum=0;
for j=1:number,
    lsum=lsum+vpa(subs(f,x,a+(j-1)*delx))*delx;
end
lsum
```

You then click `file`, `save` as and type `leftsum` when it asks for the `file name`. This will be saving the above code as what Matlab refers to as an M-file. The M-file `leftsum.m` can then be used to approximate $\int_0^4 e^{-x^2}\,dx$ as follows.

```
x=sym(x);
f=exp(-x\^{}2);
a=0;
b=4;
number=256;
leftsum
```

The first five commands above obviously set the function f as a symbolic function of the symbolic variable x, and sets a, b and *number*. Then the `leftsum` command calls the M-file given above. The code given in `leftsum` is really quite easy. We first compute *delx* (based on *number*, the number of subintervals we will have in the computation and the length of the interval), the width of the subdivisions in the left

Riemann sum. We then set $lsum = 0$ so that we are sure that the value that we are going to sum with does not have some previously set value. The `for-end loop` cycles through each of the subintervals and computes the area of the appropriate rectangle, adding that sum to *lsum*. This computation is done by the command

`lsum=lsum+vpa(subs(f,x,a+(j-1)*delx))*delx`

We should emphasize that `subs(f,x,a+(j-1)*delx)` is Matlabs way of computing

$f(a+(j-1)*delx)$ when f is a symbolic function. Then of course, the `vpa` command evaluates the symbolic value and then that value is multiplied by *delx*. You should verify that the above code does produce what is wanted for a left sum calculation. You should also realize that with only very minor changes in the M-file `leftsum.m`, you will get an M-file that will do right sums, call it `rightsum.m`, and an M-file that will that will apply the midpoint rule to approximate the integral, call it `middlesum.m`.

We have two more approximate integration schemes to implement. They will not be as easy as were the left sum, right sum and the midpoint rule, but they will be almost as easy. The way we alter the above code to gives us an M-file that will do approximate integration using the trapezoidal rule is to change the M-file `leftsum.m` as follows.

```
delx=(b-a)/number;
trapsum= vpa(subs(f,x,a))+ vpa(subs(f,x,b));
for j=1:number-1,
    trapsum=trapsum+2*vpa(subs(f,x,a+j*delx));
end
trapsum=0.5*delx*trapsum
```

It should be fairly obvious that the above code is the same as the trapezoidal rule formula given in the text. To extend one of these codes to a Simpsons Rule is a bit more difficult but not major. We must remember that when we apply Simpsons Rule, we assume that *number* is even. (When you are done with this lab, try the M-file `Simpson.m` with an odd *number*. See how badly it messes up.) The code in `Simpson.m` should look something like the following.

```
delx=(b-a)/number;
simpsum= vpa(subs(f,x,a))+4*vpa(subs(f,x,a+delx))+ vpa(subs(f,x,b));
for j=1:(number-2)/2,
    simpsum=simpsum+4*vpa(subs(f,x,a+(2*j+1)*delx));
    simpsum=simpsum+2*vpa(subs(f,x,a+2*j*delx));
end
simpsum=delx*simpsum/3
```

You should now have five M-files `leftsum.m`, `rightsum.m`, `middlesum.m`, `trapezoid.m` and `simpson.m` Use these M-files to approximate the integral $\int_0^4 e^{-x^2}dx$ using $number = 10, 20, 40, 100$ and 256. Compare and contrast your results. You should understand that if you ask Matlab to calculate the above integral (using `Int(f,0,4)`), Matlab gives you an exact answer in terms of the `erf` function (which you probably dont know too much about). However, if you write `vpa(Int(f,0,4))`, Matlab will evaluate that term for you to 32 decimal places of accuracy.

One thing that you should notice is that most of the arithmetic done in the five codes is floating point arithmetic (not symbolic). The truth of the matter is that integration schemes are best suited for floating point arithmetic. It is however, very convenient to have the symbolic definition of the function.

The last topic that we want to study is that of the error in our numerical integration schemes. Hopefully with the calculations performed above you saw that for a given scheme, the more subdivisions used give better results. Also that the trapezoidal rule and midpoint rule are generally better than the right and left hand sum rules, and Simpsons rule is better than the other four. In the text we were given bounds on the error for the trapezoidal rule and Simpsons rule. Here we include the error bound for the midpoint rule

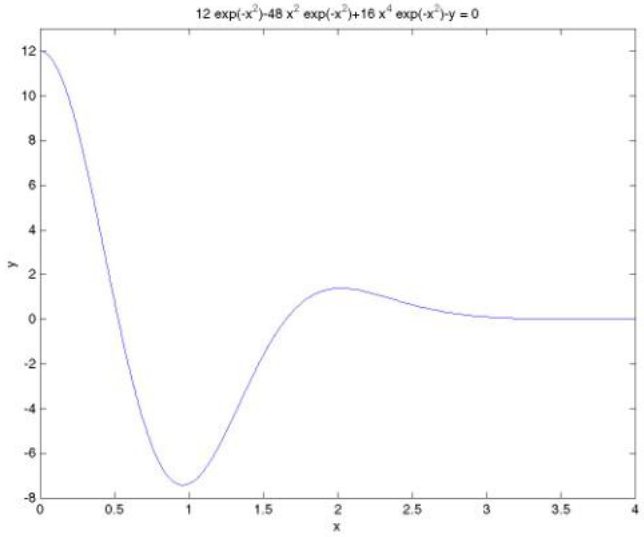
also.

$$|E_M| \le \frac{K_M(b-a)^3}{24n^2}, \qquad |E_T| \le \frac{K_T(b-a)^3}{12n^2}, \qquad |E_S| \le \frac{K_S(b-a)^5}{180n^4},$$

where K_M and K_T are bounds on the absolute value of the second derivative of f and K_S is a bound on the absolute value of the fourth derivative of f. You should be aware that the easiest way to get bounds on these derivatives is to have Matlab evaluate these derivatives and plot them. You can then choose a bound from the plot. For example, if we type

```
fp4=diff(f,4);
```

the function $fp4$ will be the fourth derivative of f (this assumes that f has already been defined as the symbolic version of our function). If you wanted to, you could omitted the semi-colon at the end of the command (but there is no real need to see how ugly the fourth derivative is). Then if you plot the function $fp4$ (Probably the best way to plot $fp4$ is to make y a symbolic variable and write `ezplot(fp4-y,[0,4,-8,13])`. The term `[0,4,-8,13]` indicates that we want the plot for $0 < x < 4, -8 < y < 13$. It should be clear that we want the plot of $fp4$ on the interval $[0,4]$. Some experimentation is generally needed to convince us that we want y in $[-8,13]$. It is easy to see that $\left|f^{(4)}\right| \le 12$ for $0 \le x \le 4$. Then it is easy to see that if we choose n=10, the error in using Simpsons rule should be less than or equal to $12 \cdot 4^5/(180 \cdot 10^4) = 0.00683$. Similar calculations can be done for the other values of n and for the other schemes. Use the results obtained by the error formulas and your numerical calculations using the trapezoidal rule, midpoint rule and Simpsons rule to determine bounds on the exact value of the integral.



Homework

1. Use your schemes for the five approximate integration schemes with $n = 10$, 20, 40, 100 and 256 to determine approximate values of $\int_0^4 e^{-x^2} dx$. Give your results in a table where you include the approximate value for each scheme and n and the error—based on the approximate value that the `int` command will give you.

2. Use the formulas for the error bounds for the midpoint, trapezoid and Simpsons rule to determine the bound given by these expressions. Include these values in your table—emphasizing the fact that the bound on the error that you get had better be greater than or equal to the actual error that you compute.