Matlab Practical: Numerical Integration

Introduction

This practical concerns numerical integration, and allows you to get some practice with simple programming in Matlab. The function we will integrate numerically is

\[ y(x) = xe^{-x} \]

As \( y(x) \) can also be integrated analytically, we can check the accuracy of our results.

The practical consists of a series of exercises for you to do: these are boxed. Before each exercise, any necessary Matlab commands and concepts are briefly introduced; anything that should be typed into Matlab is denoted by courier. As you are actually typing the Matlab commands, think about why they behave as they do; if at any time you do not understand then please ask.

Vectors

A basic data type in Matlab is the vector; an ordered list of numbers. Open Matlab and enter the following commands at the command prompt

```matlab
a = [1 3];
b = [1:3];
c = [1:0.5:3];
```

This code creates three separate variables \( a \), \( b \) and \( c \). By examining the contents of the vectors \( a \), \( b \) and \( c \) (you can ask Matlab to print the value of any variable by typing its name without a terminating semi-colon), can you work out the difference between the syntaxes used in the construction of \( a \), \( b \) and \( c \)? Check you understand by making a new variable \( z \) which contains the elements 10, 12, 14, 16, 18, 20.

Accessing elements of vectors

Individual elements of a vector can be accessed using brackets. In particular

```matlab
c(3) % no semi colon
```

prints the 3\(^{rd}\) element of the variable \( c \) to the screen (note everything after the \% is interpreted by Matlab as a comment, and anything like this in the sample code does not have to be typed in). What happens if you attempt to look at \( c(10) \)? Contrast this behaviour with what happens if you assign \( c(10) \) to have a new value…

```matlab
c(10) = 1; % assigning c(10) to be 1
```

Exercise One

Use Matlab to create a vector \( x \) which contains the numbers 0, 0.1, 0.2, …, 5. What number is stored in \( x(10) \)?
Vector valued functions and simple plots

Since the basic data type in Matlab is the vector, the built in functions take vector inputs and return vector outputs without any complaint. Before entering the following code, think about what it might do

\[ d = [0:6\pi]; \]
\[ e = \sin(d); \]
\[ plot(d,e); \]

Run the code. Try the same code with \( d \) defined differently

\[ d = [0:0.01:6\pi]; \]
\[ e = \sin(d); \]
\[ plot(d,e); \]

Persuade yourself why it gives a more pleasing plot of \( y=\sin(x) \). By typing

```
help plot
```

examine the myriad of options that can be passed to \texttt{plot}.

**Exercise Two**

Use Matlab to plot \( z = \cos(x) \) between \( x=0 \) and \( x=2\pi \), passing an appropriate option to make the curve green.

Element-wise operations

Imagine you wanted to plot the function \( y=x^2 \). A natural thing to try might be as follows, which attempts to create a new vector \( y \) where each element of \( y \) is the corresponding element of the vector \( x \) multiplied by itself

\[ x = [0:0.1:5]; \]
\[ y = x*x; \]
\[ plot(x,y) \]

However Matlab doesn’t like this! (If you know about matrices you will note that the error message is consistent with the rules of matrix-matrix multiplication \( n \times 1 \) times \( n \times 1 \) doesn’t go). To create the variable \( y \), you actually need to use the element-wise multiplication operator \( .* \) (i.e. a full stop followed by an asterisk)...try using the following code and convince yourself why this works better.

\[ x = [0:0.1:5]; \]
\[ y = x.*x; \]
\[ plot(x,y) \]
Exercise Three
Use Matlab to plot the function \( y(x) = xe^{-x} \) between \( x=0 \) and \( x=5 \) (note that \( \exp \) is used by Matlab to mean the function “e to the power of”). By finding \( dy/dx \) by hand, confirm the \( x \) coordinate of the maximum of the curve.

Exercise Four
Using integration by parts, check that the exact value of \( A(a) = \int_0^a y(x)dx = \int_0^a xe^{-x}dx \), i.e. the area between the curve \( y(x) \), the \( x \)-axis and the lines \( x=0 \) and \( x=a \), is given by \( A(a) = 1 - e^{-a} - ae^{-a} \).

Slices, sums and lengths of vectors
By typing it in line-by-line, examine the output of the following code (note the lack of semi-colons, meaning that the return value is immediately printed to the screen)

```matlab
k = [0:2:10]
k(1)
k(1:2)
k(2:5)
sum(k)
h = k(1:3)
sum(h)
length(k)
length(h)
k(2:10)    % should give an error
k(5:2)     % should give an error
```

If this sample code has not shown you how to
- access a subset of a vector
- find the length of a vector (i.e. the number of elements it contains)
- find the sum of a vector (i.e. the sum of its elements)
then please ask.

Exercise Five
Find an approximation to \( A(a) \) in the case \( a = 2 \) by programming the formula for the Trapezium rule using 10 strips (i.e. 11 function evaluations at \( x=0, x=0.2, \ldots x=2.0 \)).

Recall from the previous practical that code that is often repeated can be put into a function and saved in a file (“script”) for repeated use, although you have to take some care to make sure the file is in the correct directory. In particular the exact area according to the answer to Exercise 4 can be implemented by creating a file (in the current working directory) called `exactArea.m` which contains the following code

```matlab
function retVal = exactArea(a)
    retVal = 1 - exp(-a) - a*exp(-a);
```
Exercise Six
Type in the function `exactArea(a)` as per the above to a new file, save it as `exactArea.m` and check it works (by calling it from the Matlab command line).

Exercise Seven
Write your own function `myTrap(a,N)` which returns the approximate value of $A(a)$ according to the trapezium rule with upper limit $x = a$, and which uses $N$ strips.

For loops

Of course, like all programming languages, Matlab provides a mechanism to automatically do the same thing more than once; i.e. to loop over certain commands. This is exactly what computers are good at. Consider

```matlab
for i=1:4
    a=5*i
    % again no semi-colon
end
```

This simple program loops over $i = 1, 2, 3, 4$ and does something inside the loop (i.e. something different for each value of $i$). Although this simple example has done little to illustrate it, looping is obviously very powerful, as it allows the same calculation to be repeated a number of times (with different inputs each time through the loop).

Exercise Eight
By writing an appropriate `for` loop, investigate how the error in the approximate value for $A(2)$ (i.e. the area up to $x=2$) according to your implementation of the trapezium rule changes with the number of strips (Hint: try changing $N$ in powers of 2, i.e. $N=2,4,8,16,32,...$). To see any difference between the results according to the trapezium rule for large values of $N$, you may need to issue the Matlab command `format long` (just type it at the command prompt) which makes Matlab print results to the screen using a higher accuracy.
Exercises for more advanced programmers

If you have got this far you are obviously good enough at Matlab that you don’t need my help any more (or more likely have used it or something similar already), and from here on in you are on your own in terms of notes in the handout, although if you are confused then please ask.

Exercise Nine: advanced
Repeat the above using the built in function trapz (the command help trapz shows some examples of its use)

Exercise Ten: advanced
In practice it would be unusual to use the Trapezium rule to perform numerical integration: repeat using the Matlab quad (which uses a more accurate method) and has a wealth of options that can be passed to control its behaviour. The simplest way to use quad is to create a file containing the function y(x).

Exercise Eleven: advanced
Monte Carlo integration can be useful in certain circumstances. This very powerful technique involves using random samples of a function to approximate its mean value, and so to approximate its integral. If \( \{x_i\} \) is a set of \( N \) uniform random numbers between 0 and \( a \), then an estimate for the mean value of \( y \) over this range, \( \langle y \rangle \), is given by

\[
\langle y \rangle \approx \frac{1}{N} \sum_{i=1}^{N} y(x_i)
\]

However, by definition the mean value of \( y \) over the range 0 to \( a \) is exactly

\[
\langle y \rangle = \frac{1}{a} \int_{0}^{a} y(x)dx = \frac{A(a)}{a}
\]

Comparing these two results allows us to estimate \( A(a) \).

Write a function to find a Monte-Carlo estimate of \( A(a) \) for different values of \( a \) and \( N \). To generate the random numbers, you will to use the built-in function \texttt{rand}. How does the error in the approximation behave with \( N \) for this method? In what situations might it be useful?

Nik Cunniffe