2 A Mathematica Primer

2.1 Basics

Mathematica is a very powerful package for performing symbolic and numerical mathematics. It has a fairly steep learning curve, and takes some patience and involves some frustration to become adept at using it. My experience with previous students is that once learned, Mathematica becomes an indispensable tool in their academic and research work.

The point of these notes is to present a basic introduction to the package. Perhaps the best way to start is with an example. The 1–D convective–diffusive problem discussed in class appeared as

\[
P \frac{dT}{d\bar{x}} = \frac{d^2T}{d\bar{x}^2} \tag{1}
\]

\[
T(0) = 0, \quad T(1) = 1 \tag{2}
\]

where

\[
T = \frac{T - T_0}{T_L - T_0}, \quad \bar{x} = \frac{x}{L}, \quad Pe = \frac{uL}{\alpha}
\]

The problem is "well posed" insofar as sufficient information is given (a second order ODE and 2 boundary conditions) to obtain a solution.

Start up Mathematica and type the following information in to the notebook window (or cut and paste – that should work):

\[
de = \text{pe}\ t'[x] - t''[x] == 0;
bc1 = t[0] == 0;
bcc2 = t[1] == 1;
soln = \text{DSolve} \{\{\text{de}, \text{bc1}, \text{bc2}\}, t[x], x\}[[1,1]]
\]

Now, with the cursor anywhere within the block of four lines, do "shift+enter". You should get something like this

\[
\text{Out}[4] = \frac{t[x]}{-1 + \text{E}^\text{pe}(\text{\textbar x})}/(-1 + \text{E}^\text{pe})
\]

The Out[4] will probably have a different number than 4; that is irrelevant. What is important is that the output returns the solution to the boundary value problem.

Now for some specific and some general points regarding Mathematica:

1. Mathematica works as a line interpreter as opposed to the batch process of Matlab. That is, it executes commands on a line–by–line basis, and keeps the results in memory. Several commands can be combined into a single block by using the semicolon (;) at the end of the line, per the example above. You can position your cursor at any vertical position in your notebook and re–execute an old command (via shift+enter), or edit an old command and re–execute it, or type in a new command. This is also distinct from Matlab.

2. Mathematica begins all intrinsic functions and mathematical constants with upper case letters, i.e., Sin[x] for sin(x), Pi for \(\pi\), and I for \(i = \sqrt{-1}\). It is strongly advised that you use lower case letters for all variables and parameters in your solution. By doing so you will avoid any conflict with a Mathematica–defined function or constant. For example, the temperature was denoted as \(t[x]\). 

3. The single equal sign ‘=' refers to an assignment in Mathematica, whereas the double equals sign ‘==' denotes the condition of equality. In the first line, the variable \(de\) is assigned to represent the differential equation, and likewise with the assignment of the boundary condition equations to \(bc1\) and \(bc2\). This assignment is not necessary; the equations could have been written out explicitly in the argument of the DSolve function.

4. By writing the dependent variable as \(t[x]\), it is implied that \(T\) is a function of \(x\).

5. The differential operator \(t'[x]\) is equivalent in Mathematica to \(D[t[x],x]\), and \(t''[x]\) would be \(D[t[x],x,2]\). The expression \(t'[0]\) (had this appeared in the problem) would imply the derivative of \(T\) with respect to \(x\) evaluated at \(x = 0\). This could also appear as \(D[t[x],x]/.x->0.\)
6. **soln** denotes the solution returned by the *Mathematica* function `DSolve`. The use of `DSolve` should be self-explanatory from the context. The solution appears as a list in the form of a replacement rule.

(a) A list is a group of one or more quantities, and the list construction and manipulation features of *Mathematica* enable one to perform vector and matrix mathematics. A 3 element vector would appear as `{a, b, c}`, whereas a 2×2 matrix would be `{{a, b}, {c, d}}`. The depth of the list is the dimensionality (or rank) of the list plus 1; a vector would have a depth of 2 and a matrix would have a depth of 3. A part, or element, of a list can be extracted using the double brace format as follows; `{a, b, c}`[[2]] gives b and `{{a, b}, {c, d}}`[[2, 1]] gives c. The solution to `DSolve` appears as a depth 3 list with one element – the reason the output appears as a list is because the DE can, in general, have more than one solution. To extract this one element from the list and assign it to `soln`, the part specification `[[1, 1]]` is included at the end of the assignment to `soln`. This may or may not be necessary – it is done here to avoid any subsequent problems with the list structure of the solution.

(b) A replacement rule is in the form `f[a]/. a -> b`. It is somewhat akin to an assignment, in that `a` is replaced by `b`, except that the replacement acts only within the command (or line) with which it is executed. The given line would compute `f[b]` (where `f` is some function) – yet the variable `a` will not be assigned the value `b` in subsequent calculations. This is different than `a=b` followed by `f[b];` for which `a` has now been assigned to `b` for all subsequent operations. As another example, `D[t[x],x]/.x->0` first computes `t'[x]` and then replaces `x` with zero. The operation `x=0`, followed by `D[t[x],x]`, would try to compute `D[t[0],0]` and would give an error because `x` has been assigned to the constant 0 and is not a valid variable. The solution `soln` to `DSolve` appears as a replacement; `t[x] -> f[x]`, where `f[x]` is shorthand for the actual solution. It does not assign `t[x]` to the solution; the operation `t[.5]`, for example, would simply return `t[.5]`. To compute the solution at `x = .5`, one would use `t[x]/.soln/.x->.5`. This command first replaces `t[x]` with the functional form of the solution, and then replaces `x` with 0.5. Any other parameters appearing in the solution (i.e., `Pe`) would also have to be given values (via a direct assignment or a replacement) to obtain a numerical answer.

7. *Mathematica* offers comprehensive online help – which includes a complete hypertext version of the *Mathematica* book. There you can find more information on the strategy used in the code and on other features (such as the plotting function).

### 2.2 Functions

The solution to the DE appears as a function of `x` with `Pe` as a parameter, and it will be useful to put this into a functional form which can be further manipulated and plotted. An arbitrary function, say `f[x] = x^2`, can be coded in *Mathematica* via the following:

```mathematica
f[x_] := x^2
```

Followed by `shift+enter`. This would give

```
In[1]:=f[a]
Out[1]:=a^2
In[2]:=f[2]
Out[2]:=4
In[3]:=f[f[a]]
Out[3]:=a^4
```

and so on. Some points:

1. It is **absolutely important** to only use square brackets `[...]` for the arguments of functions (both intrinsic to *Mathematica* and user-defined). Likewise, the curly brackets `{...}` are used **only** for list constructions (vectors, matrices, etc.). Only the parenthesis `(....)` are used for algebraic grouping of terms in an expression.

2. The colon-equal `:=` in the function definition is a delayed assignment. If only the `=` assignment was used, then `f[x]` would be assigned to `x^2`, yet `f[a]` would have no meaning: it would return simply `f[a]`. 

---

2
3. The underscore \( x_\) denotes that \( x \) appearing in the function definition is a dummy argument: it can be replaced by any other symbol, number, or function.

For our case, we can generate a function for the solution via the following command:

\[
tfunc[x_, pe_] := \text{Evaluate}[t[x] /. \text{soln}]
\]

The \text{Evaluate}[] command does what it implies: in this case, it evaluates the replacement of the symbolic solution into \( t[x] \). The result is then used to define the function \( tfunc[x, pe] \). A plot of the temperature profile, for different values of \( Pe \), could then be generated using

\[
\text{Plot}\{tfunc[x, .1], tfunc[x, 1], tfunc[x, 10], tfunc[x, 100]\}, \{x, 0, 1\}, \text{PlotRange} \to \text{All}\]

2.3 Modules

Often we are faced with a mathematical/numerical task which 1) we want to perform repeatedly and/or 2) is a sub–element in a larger overall project, yet which is 3) not something trivial which we can write into a single line, per the function definition in the previous section. A common example is solving numerically a system of linear equations; this task is, in fact, so common that most high–level packages such as \text{Mathematica} and Matlab have it built–in to their intrinsic functions. In low level programming (fortran, C++), a task of this sort would involve the coding of a \textit{subroutine}: this is a subunit of the program which is called from the main program to perform specified tasks and return the results. In \text{Mathematica} the equivalent concept is a \textit{module}.

As an example, say we want to manipulate the 1–D, convection/diffusion solution to do the following: plot a dimensionless heat flux at the outflow surface as a function of the dimensionless mean temperature. The heat flux at the outflow is

\[
q''_L = k \left( \frac{dT}{dx} \right)_{x=L} = \frac{k(T_1 - T_0)}{L} \left( \frac{dT}{dx} \right)_{x=1}
\]

in which \( T \) and \( \tau \) are the scaled (dimensionless) variables. So...

\[
\bar{q} = \frac{q''_L L}{k(T_1 - T_0)} = \left( \frac{dT}{dx} \right)_{x=1} = \text{dimensionless heat flux} = \text{func} \: Pe
\]

and the dimensionless mean temperature is obtained by

\[
T_m = \int_0^1 T \, d\tau = \text{another func} \: Pe
\]

The explicit formulas for these quantities can be determined in \text{Mathematica}
\[ q_{\text{bar}}[p_e] := \text{Evaluate}[D[t\text{func}[x, p_e], x] /. x \to 1] \]

\[ \text{In}[15] := q_{\text{bar}}[p_e] \]
\[ \text{Out}[15] = (E^{p_e} p_e) / (-1 + E^{p_e}) \]

\[ t\text{mean}[p_e] := \text{Evaluate}[\text{Simplify}[	ext{Integrate}[t\text{func}[x, p_e], \{x, 0, 1\}]]] \]

\[ \text{In}[19] := t\text{mean}[p_e] \]
\[ \text{Out}[19] = (1 - E^{p_e} + p_e) / (p_e - E^{p_e} p_e) \]

An explicit formula cannot be found for \( \bar{\eta} \) as a function of \( T_m \), because \( T_m(Pe) \) cannot be analytically inverted so that \( Pe = \text{func}(T_m) \). This problem needs to be done numerically. For a given value of \( T_m \), the steps are 1) numerically find \( Pe \) from the \( T_m \), and 2) use this \( Pe \) to obtain \( \bar{\eta} \). This is done in a module. The code is

\[ \text{qbartm}[t_m_] := \text{Module}[\{\text{petm}\}, \]
\[ \text{petm} = p_e /. \text{FindRoot}[t\text{mean}[p_e] - t_m == 0, \{p_e, 10\}][[1]]; \]
\[ q_{\text{bar}}[\text{petm}] \]
\[ \text{Plot}[q_{\text{bar}}[t_m], \{t_m, .0001, .49999\}, \text{PlotRange} \to \text{All}] \]

![Graph showing \( q_{\text{bar}} \) as a function of \( T_m \)]

The \texttt{Module} command, in the general form, has two arguments separated by a comma. The first, which is enclosed in \{\ldots\}, denotes the local variables for the procedure. In this case there is one local variable, \texttt{petm}. This quantity is used only within the module; it will remain undefined outside of the module. The second argument consists of a series of commands, each separated by a semicolon. The last command (without a trailing semicolon) is what is returned by the module. The example here has two commands: the first uses the \texttt{Mathematica} \texttt{FindRoot} intrinsic function to find the value of \( Pe \) corresponding to the given \( T_m \). See the online help for an explanation of the workings; it was necessary to experiment around with this command to arrive at the chosen starting value of \( Pe = 10 \). Note also that the minimum and maximum \( T_m \) values are 0 and 1/2, corresponding to \( Pe \to \infty \) and 0. The second command evaluates \( \bar{\eta} \) for the computed \( Pe \).

Why is \( \bar{\eta} \) an (almost) linear function of \( T_m \)? Is there a simple explanation?