A Primer on Mathematica

Jan M. Baetens
KERMIT, Department of Mathematical Modelling,
Statistics and Bioinformatics

March 23, 2015

Mathematica is a versatile, powerful application package for performing mathematics and publishing mathematical results. It runs on most popular workstation operating systems, including Microsoft Windows, Apple Macintosh OS, Linux, and other Unix-based systems. Mathematica is used by scientists and engineers in disciplines ranging from astronomy to zoology; typical applications include computational number theory, ecosystem modeling, financial derivatives pricing, quantum computation, statistical analysis, and hundreds more.

For sure, the best way to understand Mathematica is to see it in action. In the remainder of this computer laboratory, we briefly elaborate on its two main usages that are of importance for the forthcoming laboratories on cellular automata:

- As an end user tool: Mathematica can be used to perform computations, either numeric or symbolic, and the results can be visualized easily in two or three dimensions.
- As a programming tool: Mathematica provides a rich set of programming extensions to its end-user language. Programming can be done in procedural, functional, or logic (rule-based) style, or a mixture of all three.

1 The fundamentals

By means of so-called Mathematica notebooks, the Mathematica kernel may be used interactively, just as a calculator, which is similar to using the Matlab command window. Commands to be evaluated, such as 2+2, are entered directly into the notebook and can be evaluated by pressing simultaneously the shift and enter keys, while a new line is started by pressing the enter key. After executing this command `In[X]:=` appears in front of the original instruction, as well as the result of the computation, which is preceded by `Out[X]=`, such that the notebook should now look like

```
In[1]:=2 + 2
Out[1]=4
```

where the number between square brackets indicates how many times instructions were passed formerly to the Mathematica kernel. Of course, also longer expressions that span several lines and which include comments, enclosed between (* *), can be entered. For instance, entering

```
In[1]:=(5∗7−32/6)/4
          (∗A comment∗)
```

yields, contrary to MATLAB, an exact outcome expressed in terms of the fraction 89/12. Its capability of performing exact computations, rather than resorting to decimal approximations, is one of Mathematica's strong points. Nonetheless, numerical approximations can always be obtained by wrapping the `N` function around the outcome of the previous instruction, which can be referred to by using `%`, as follows:

```
In[1]:=N[%]
```

To exemplify Mathematica's ability to perform exact calculations, try to evaluate 2^{10000}.

Although it must be acknowledged that MATLAB has numerous built-in functions, the number of implemented functions in MATLAB pales before the number of functions with which Mathematica has been supplied. Functions are always capitalized, while their arguments are enclosed in square brackets `[]`. For instance, $\sin(\ln(6))$ can be calculated as

```
In[1]:=N[Sin[Log[6]]]
Out[1]=0.975687
```

Besides square brackets that are used to enclose a function's arguments, also parentheses `()` and braces `{}` have been assigned an important meaning in Mathematica. More specifically, parentheses are used in mathematical expressions, while braces are used to form lists, which play a very important role in Mathematica since they constitute its fundamental data structure. Because of that, many functions automatically act on the individual elements that are contained in a list, for instance

```
In[1]:=Sin[{Pi, Pi/2}]
Out[1]={0,1}
```

whereas many other functions, among which `Table` and Range are probably the most noteworthy within the framework of these notes, return lists. For instance, the `Table` command has the syntax `Table[expr,{x,xMin,xMax,step}]`, which evaluates `expr` at the values `xMin`, `xMin+step`,. . . , `xMin+n step`, where `n` is the largest integer so that `xMin+n step≤xMax`, and generates a list accordingly. This function can thus be used to create a list that contains the third power of the first ten natural numbers

```
In[1]:=Table[k^3,{k,0,9,1}]
Out[1]={0, 1, 8, 27, 64, 125, 216, 343, 512, 729}
```

The Range command with syntax Range`[min,max,step]` produces a list of numbers

$$\{\mathtt{min}, \mathtt{min+step}, \ldots, \mathtt{min} + n\,\mathtt{step}\}\,,$$

where `n` is the largest integer so that `xMin+n step≤xMax`.

Similar to Matlab, a name can be assigned to scalars, vectors, and matrices but also to lists, plots, etc. For instance, the previously created list `{0, 1, 8, 27, 64, 125, 216, 343, 512, 729}` could be assigned to a variable `list`, after which this variable can be used in subsequent instructions,

```
In[1]:=list=Table[k^3,{k,0,9,1}];
        list^2
Out[1]={0, 1, 64, 729, 4096, 15625, 46656, 117649, 262144, 531441}
```

where a semi-colon is used to suppress the result of the `Table` command.

Analogously to the declaration of variables, Mathematica allows to define functions, such as $f(x) = x^3 - 1$, that can be used throughout the remainder of the notebook or until it is cleared using the Clear function:

```
In[1]:= f[x_]:=x^3−1;
         f[3]
Out[1]=26

In[1]:= f[z]
Out[1]=z^3−1
In[1]:= Clear[f]
```

As an alternative to this intuitive manner of defining functions, also so-called pure functions can be used. A pure function has the syntax `expr` &, where `expr` is a Mathematica expression involving the # symbol, whereas the & operator tells Mathematica to treat `expr` as a function with argument #. Hence, the function $f(x) = x^3 - 1$ can be defined alternatively as

```
In[1]:=#^3−1&[3]
Out[1]=26
```

where the function was evaluated for $x = 3$.

Mathematica has several commands for generating a random number or a list of random numbers; `RandomInteger` and `RandomReal` are, perhaps, the most fundamental. `RandomInteger [{iMin,iMax},{n`$_1$`, n`$_2$`, ...}]`, respectively `RandomReal[{iMin,iMax},{n`$_1$`, n`$_2$`, ...}]`, generates a list of `{n`$_1$`, n`$_2$`, ...}` random integer, respectively real, between `iMin` and `iMax`.

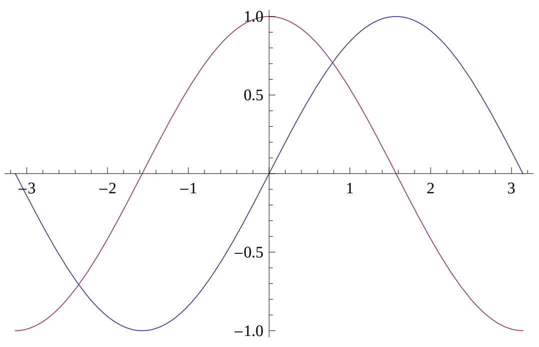
```
In[1]:=RandomInteger[{0, 5}, 5]
Out[1]={3, 0, 4, 0, 4}

In[1]:=RandomReal[{0, 5}, 5]
Out[1]={1.93451, 3.91802, 0.443231, 3.63488, 4.94378}
```

2 Visualization

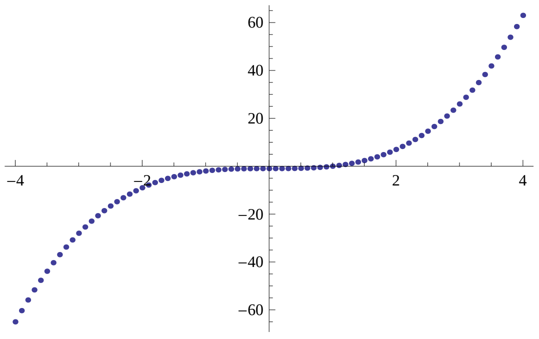
Mathematica is a very powerful tool to create high-quality 2D/3D plots of both functions and data. The most basic command to plot the former is the `Plot` command if functions of one independent variables are at stake, or the `Plot3D` command if the considered function is based upon two independent variables. For instance, a plot of both $\sin(x)$ and $\cos(x)$ can be obtained as follows:

```
In[1]:= Plot[{Sin[x], Cos[x]}, {x, −Pi, Pi}]
```



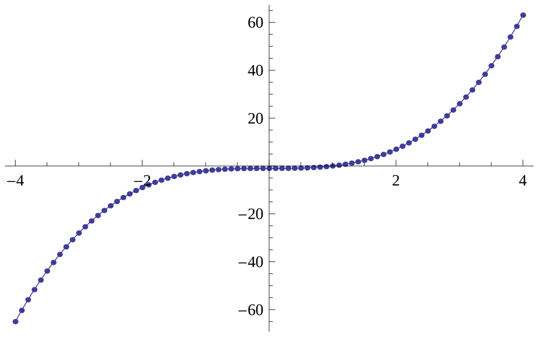
The counterpart of the `Plot`, respectively `Plot3D`, command to visualize data is the instruction `ListPlot`, respectively `ListPlot3D`. A plot of some points lying on the curve described by the function $f(x) = x^3 - 1$ can be generated as follows:

```
In[1]:=data = Table[{x, x^3−1}, {x, −4, 4, 0.1}];
        ListPlot[data]
```



Often, plots have to be combined in one figure. In Mathematica this can be accomplished by using the Show function, which takes any number of plots and combines them in one figure. For instance, the previous plot can be combined easily with a plot showing a line connecting the dots as follows

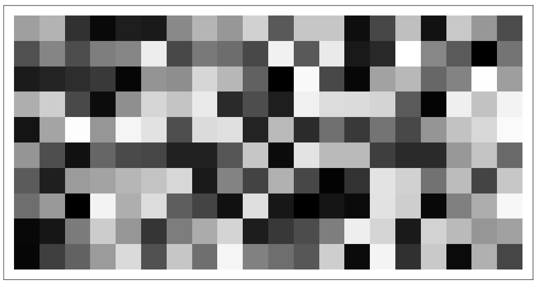
```
In[1]:=data = Table[{x, x^3−1}, {x, −4, 4, 0.1}];
        f1=ListPlot[data];
        f2=ListLinePlot[data];
        Show[f1,f2]
```



In order to customize the created plots, numereous options are available within any of the aforementioned plot commands, of which `Axeslabel`, `Frame`, `PlotLabel`, `PlotRange` and `PlotStyle` are probably of most interest within the framework of the forthcoming laboratories. Their meaning can be checked in Mathematica's outstanding documentation center.

In addition to the already discussed commands for visualizing data and functions, Mathematica has been supplied with a function that automatically generates a plot in which the values in an array are shown in a discrete array of squares, and which uses, by default, a grayscale output, in which zero values are shown white, and the maximum positive or negative value is shown black:

```
In[1]:=ArrayPlot[RandomReal[{0,1}, {10, 20}]]
```



Assignment 2.1

1 Write a function that allows to calculate the i-th Fibonacci number if you know that $F_i = F_{i-1} + F_{i-2}$ with $F_1 = F_2 = 1$. Then, use this function to create a list containing the first thirty Fibonacci numbers, and plot this sequence. **Hint**: recursion.

2 Given the parameter representation of a circle with radius R,

$$\begin{cases} x = R\cos\theta\,, \\ y = R\sin\theta\,, \end{cases}$$

where $0 \le \theta \le 2\pi$, formulate a function that generates a plot of a circle with a given radius. Don't forget to set appropriate axes and plot labels.

3 List manipulation

Since Mathematica's fundamental data structure is a (nested) list, numerous functions are available for manipulating lists. Suppose that a list containing ten randomly generated integers between zero and nine, for instance $\{7, 6, 1, 0, 6, 8, 9, 7, 0, 0\}$, is assigned to a variable `list`, then, several operations can be performed on this list:

1. the list may be sorted

```
In[1]:=Sort[list]
Out[1]={0, 0, 0, 1, 6, 6, 7, 7, 8, 9}
```

2. the order of the list may be reversed

```
In[1]:=Reverse[list]
Out[1]={0, 0, 7, 9, 8, 6, 0, 1, 6, 7}
```

3. elements may be dropped

```
In[1]:=Drop[list,2]
Out[1]={1, 0, 6, 8, 9, 7, 0, 0}
```

4. the list may be shifted

```
In[1]:=RotateLeft[list,2]
Out[1]={1, 0, 6, 8, 9, 7, 0, 0, 7, 6}
```

5. elements of the list may be accessed

```
In[1]:=list[[2]]
Out[1]=6
In[1]:=list[[2 ;; 4]]
Out[1]={6, 1, 0}
In[1]:=list[[{2, 4}]]
Out[1]={6,0}
In[1]:=Last[list]
Out[1]=0
```

6. the list may be partitioned in sublists

```
In[1]:=Partition[list,2]
Out[1]={{7, 6}, {1, 0}, {6, 8}, {9, 7}, {0, 0}}
```

7. nested lists may be flattened

```
In[1]:=Flatten[%]
Out[1]={7, 6, 1, 0, 6, 8, 9, 7, 0, 0}
```

8. elements that fulfill particular criteria may be selected from the list

```
In[1]:=Select[list,#>3&]
Out[1]={7, 6, 6, 8, 9, 7}
```

   where #>3& is a pure functioning returning True if its argument is larger than 3, and False otherwise.

9. elements that fulfill particular criteria can be replaced using the replacement operator `/.`

```
In[1]:=list /. {7 -> 100}
Out[1]={100, 6, 1, 0, 6, 8, 9, 100, 0, 0}

In[1]:=list /. {p_ /; p > 3 -> 100}
Out[1]={100, 100, 1, 0, 100, 100, 100, 100, 0, 0}
```

   where p_ is a so-called pattern matcher instruction that is commonly used in Mathematica. Extensive documentation on Mathematica's pattern matching abilities is available in the program's documentation center.

As mentioned earlier, many functions automatically act on a list's elements if applied to a list. In Mathematica, such functions are referred to as listable functions. However, arbitrary functions are not listable, *i.e.*

```
In[1]:=g[list]
Out[1]=g[{7, 6, 1, 0, 6, 8, 9, 7, 0, 0}]
```

any function can be mapped on a list using the Map function

```
In[1]:=Map[g,list]
Out[1]={g[7], g[6], g[1], g[0], g[6], g[8], g[9], g[7], g[0], g[0]}
```

Analogously, also pure functions can be mapped onto lists

```
In[1]:=Map[{#, #^3} &, list]
Out[1]={{7, 343}, {6, 216}, {1, 1}, {0, 0}, {6, 216}, {8, 512}, {9, 729},
 {7, 343}, {0, 0}, {0, 0}}
```

In Mathematica, all non-atomic expressions, that is all expressions other than numbers or symbols, are represented by head[args], which becomes obvious if we request the internal representation of a list using the function FullForm:

```
In[1]:=list // FullForm
Out[1]=List[7, 6, 1, 0, 6, 8, 9, 7, 0, 0]
```

showing that the head of a list is List, while the arguments are the list's elements. The head of an expression can be changed by using the Apply command. For example,

```
In[1]:=Apply[Plus, list]
Out[1]=44
```

Numerous other functions manipulating lists can be retrieved in the documentation center.

Assignment 3.1

1 The flight of a propeller airplane that travels between Brussels and Saint-Petersburg can be split up into four parts. Table 1 lists their duration, as well as the speed that is maintained during every part.

Table 1. Duration of and speed during every flight part

Part	1	2	3	4
Speed (km h^{-1})	200	250	400	300
Duration (hours)	2	3	2	1

(a) Determine the distance traveled during each part of the flight.

(b) Calculate the total flight distance between Brussels and Saint-Petersburg.

2 Construct a function that takes as input an arbitrary list consisting of sublists of length 2, for example {{1, 0}, {1, 9}}, and returns those sublists for which the second element is greater than five.

4 Iteration

Given a particular initial value x_0, it is sometimes needed to calculate the list

$$\{x_0, f(x_0), f(f(x_0)), \dots, f(\dots f(x_0))\} \; .$$

This can be accomplished easily using the `NestList` function

```
In[1]:=NestList[f,x0,4]
Out[1]={x0, f[x0], f[f[x0]], f[f[f[x0]]], f[f[f[f[x0]]]]}
```

Alternatively, the function `Nest[f,x0,n]` can be used if one is merely interested in the n-th application of f

```
In[1]:=Nest[f,x0,5]
Out[1]=f[f[f[f[f[x0]]]]]
```

Further, `FoldList` and `Fold` can be used to implement iteration.

5 Programming

Mathematica has a full-fledged, high-level programming language including loop constructs such as `For`, `Do` and `While`, flow control devices such as `If` and `Switch`, and scoping constructs which allow you to declare local variables. Programming with Mathematica is a deep subject and entire books are devoted to it. A word of warning is in order to experienced programmers. The procedural programming paradigm used for many compiled languages, such as C, C++, or Java, is emphatically not appropriate for a high-level language such as Mathematica, since code written using the procedural approach generally executes much slower than a task written following the functional programming approach. Yet, a detailed elaboration of the functional programming paradigm is beyond the scope of these notes, and procedural programming may still be adhered to.

For that purpose, the `For` function, syntax `For[start,test,incr,body]`, which executes `start`, then repeatedly evaluates body and `incr` until `test` fails to give True, as well as the `Do` function, syntax `Do[expr,i,imin, imax,step]`, which evaluates `expr` with the variable `i` successively taking on the values `imin` through `imax` (in steps of size `step`), are of major importance. Besides, conditional statements can be employed by using the `If` function, syntax `If[condition,t,f]`, which gives `t` if `condition` evaluates to `True`, and `f` if it evaluates to `False`.

In order to exemplify the strengths of the functional programming paradigm, consider the following piece of code for calculating the square of the first 20000 natural numbers that should look familiar to people experienced with procedural programming in MATLAB:

```
In[1]:=squareList = {};
In[2]:=Timing[For[i=0, i<=20000, i++,
        squareList = AppendTo[squareList, i^2]]]
Out[2]={1.343, Null}
```

where 1.343 indicates that it Mathematica 1.343 seconds do perform the calculation.

The functional programmed counterpart of this task looks like

```
In[1]:=Timing[Range[20000]^2;]
Out[1]={8.34836*10^-17,  Null}
```

Two major conclusions can be drawn upon the comparison of both programming paradigms. First, the functional counterpart is considerably faster, and, second, compared to the procedural approach, the functional approach is much shorter and more readable. In fact, it turns out that these conclusions are valid for most computational tasks.

6 Importing and exporting

Typically, technical computing applications are well suited for performing computations, but have major difficulties in importing or exporting data, figures, or animations, that were created by or must be used in other applications. In contrast, Mathematica has been supplied with extensive and easy-to-use importing and exporting capabilities, which can be called using the functions `Import` and `Export`. For instance, a .txt-file containing a list of data can be imported as follows:

```
In[1]:=data=ToExpression[Import["file.txt","List"]];
```

where the function `ToExpression` is wrapped around the `Import` function in order to convert the strings that are contained in the text file to numbers, which can then be used in subsequent Mathematica instructions. Analogously, our exemplary list $\{7, 6, 1, 0, 6, 8, 9, 7, 0, 0\}$ can be exported to a .txt-file:

```
In[1]:=data=Export["file.txt",list];
```

The list of data formats that Mathematica can handle is almost endless, and contains, among others, audio, binary, database, document, multimedia, raster, vector and XML formats.

Assignment 6.1

1 Write a function that determines the largest $n \in \mathbb{N}$ for which it holds that

$$\sum_{i=0}^{n} i \leq Q\,,$$

where $Q \in \mathbb{N}$. Test this function for $Q = 750$.

2 Write a function that evaluates the trace of a square matrix, defined as the sum of the diagonal elements. Your function should check the input structure and print an error message if the input is not a square matrix. Hint: the function `Dimensions` could be of use.

3 This assignment aims at constructing and visualizing the Mandelbrot set. Mathematically, the Mandelbrot set can be defined as the set of complex values c for which the iteration of the complex quadratic polynomial $z_{n+1} = z_n^2 + c$ with initial condition $z_0 = 0$ produces a sequence for which $|z_n|$ does not approach infinity as n grows infinitely large. It can be shown that this sequence will escape to infinity as soon as there is a z_ϵ for which $|z_\epsilon| > 2$, where ϵ is the escape rate, *i.e.* the number of iterations needed to obtain $|z_\epsilon| > 2$.

(a) Write a function that calculates $\ln(\epsilon + 1)$ for a given c. Hence, this function should have two input arguments and it should yield the number of iterations needed to obtain $|z_n| > 2$ or the maximum number of iterations that are allowed computed for points which either escape very slowly or simply orbit the origin. In the framework of this assignment, the maximum number of iterations is 50.

(b) Plot $\ln(\epsilon + 1)$, for $Re(c) \in [-2, 1.2]$ and $Im(c) \in [-1.4, 1.4]$, by means of the function `DensityPlot`, for which you provide the options `PlotPoints->100, Mesh->False, AspectRatio->Automatic, ColorFunction->Hue`. Be aware that it takes a while to render the plot.

(c) Explore an area with an increasing resolution and see what happens.