A Guide to Typesetting Mathematics using GNU eqn

Ted Harding

Preamble

Like the other **troff** preprocessors **pic** and **tbl**, which allow the user to format a complex structure (diagram, table) by typing in a specification of it expressed in a ***description language***—a reasonably close approximation to 'plain English' instructions, so **eqn** provides a mechanism for specifying how to lay out mathematical notation. The 'plain English' aspect of **eqn** in fact resembles what a lecturer might speak out loud while writing mathematics on a blackboard.

Encapsulating eqn *code*

In order for **eqn** to recognise a block of user input as something that **eqn** itself should interpret, rather than pass unchanged through to **troff**, the start and the end of the **eqn** block need to be flagged by marks that **eqn** will recognise. Then, when **eqn** recognises the start of a block, all that follows will be interpreted by **eqn**, and transformed into **troff** code which will result in suitably formatted output, until the end of the block is recognised. Then **eqn** will revert to ignoring the input until it recognises the next start of an **eqn** block.

There are two mechanisms for marking the start and end of an **eqn** block. The first is for ***displayed equations***, which are set apart from surrounding text by being printed on their own separate lines. In this case, the start of the block is flagged by **.EQ** at the beginning of an input line, and the end of the block is flagged by **.EN** at the beginning of an input line. These serve two purposes. Firstly, the **.EQ** and **.EN** are recognised by **eqn** as delimiting code that **eqn** is to interpret. Secondly, **eqn** will pass the '**.EQ**' and the '**.EN**' through to **troff**; then **troff** itself will evoke macros **EQ** and **EN** which will position the **troff**-formatted mathematics on the page. The various standard macro packages (**me**, **mm**, **ms**) embody implementations of the **EQ** and **EN** macros. The **mm** implementation behaves somewhat differently from the other two. Users may also write their own implementations.

The second is for ***in-line mathematics***, in which mathematical expressions occur within running text. Here the **eqn** blocks are flagged by ***delimiters*** placed at the start and end of the block. The default delimiter for both ends is **$**, placed at the start and at the end. The user may define other delimiters to be used instead of **$...$**.

The following section illustrates the most simple and basic usage of **eqn**, and shows the formatted mathematical outputs.

Basic Use

Simple algebraic expressions

These are entered much as one would write them on paper. For example:

```
.EQ
Z = X + Y
.EN
```

will produce:

$$Z = X + Y$$

which is (by default) centred. By following the **.EQ** by a parameter, it may be forced to be left-justified (**L**), or indented (**I**):

```
.EQ L
Z = X + Y
.EN
```

$Z = X + Y$

```
.EQ I
Z = X + Y
.EN
```

$Z = X + Y$

It is also possible to explicitly use **C** for a centred equation, but this is already the default.

The above is for the **me** and **ms** macro packages; the **mm** package behaves differently. With **mm**, whatever follows **.EQ** will be treated as an ***equation label*** (see below); the entire '**.EQ ... .EN**' block needs to be itself encapsulated between **.DS** and **.DE** tags, with justification (left, indented, centred, or right) controlled by a parameter to **.DS**, as in **.DS L**, **.DS I**, **.DS C**, **.DS CB**, **.DS R**, **.DS RB** (see **man groff_mm** for details). An example of how to add an ***equation number*** (or label):

```
.EQ (3)
Z = X + Y
.EN
```

$$Z = X + Y \qquad (3)$$

To illustrate placement of this equation ***in-line***:

```
The mathematical equation $Z = X + Y$
is understood by almost everyone.
```

The mathematical equation $Z = X + Y$ is understood by almost everyone.

The several progressive examples which follow will illustrate how to build more complicated equations and expressions.

Multiplication of terms, expressed by juxtaposition, can be constructed by again simply entering the expression as you would normally write it (but **note** the use of spaces to separate symbols and signs (***tokens***) in the expression):

```
.EQ
f ( x , y ) = a + b x + c y + p x y
+ ( 1 - x ) ( 2 - y )
.EN
```

$$f(x,y) = a + bx + cy + pxy + (1-x)(2-y)$$

Explicit indication of multiplication, using '×', is done with the ***keyword*** **times**:

```
.EQ
f ( x , y ) = a + b x + c y
+ p x y + ( 1 - x ) times ( 2 - y )
.EN
```

$$f(x,y) = a + bx + cy + pxy + (1-x)\times(2-y)$$

While **eqn** does quite a respectable job of laying out equations as one would normally expect to see them, often their appearance can be improved by inserting a little extra space here and there. The two tokens which are handy for this are '**~**' and '**^**': '**~**' inserts a space of about 2/7 of an ***em***, and '**^**' about 1/6 of an ***em***. The spacing in the above example can be tweaked using these, for example as follows:

```
.EQ
f ( x , y ) ~~=~~ a ~+~ b x ~+~ c y
~+~ p x y
~+~^ (^ 1 - x ) times ( 2 - y )
.EN
```

$$f(x,y) = a + bx + cy + pxy + (1-x)\times(2-y)$$

which does look a bit nicer!

Raising to a power can be expressed by using the ***keyword*** **sup**, and a subscript by using **sub**:

```
.EQ
a x sup 2 + b x + c = 0
.EN
```

$$ax^2 + bx + c = 0$$

```
.EQ
Y =
a sub 1 X sub 1 + a sub 2 X sub 2
.EN
```

$$Y = a_1 X_1 + a_2 X_2$$

Often, a group of terms must be kept together as an entity when being placed in an equation. This is done by using opening and closing ***braces***: **{...}**.

A common use of this is in composing ratios, which are entered as ***numerator*** **over** ***denominator***:

```
.EQ
Z ~=~~ U over V
 ~~=~~ {a + b X} over {c + d X}
.EN
```

$$Z = \frac{U}{V} = \frac{a+bX}{c+dX}$$

Without the **{...}**, this would have become:

```
.EQ
Z ~=~~ U over V
 ~~=~~ a + b X over c + d X
.EN
```

$$Z = \frac{U}{V} = a + b\frac{X}{c} + dX$$

showing how the **{...}** keep the numerator terms together, and the denominator terms together.

Another usage is where a group of terms is to be 'fed' to an operation which is to embrace the whole. For instance, the keyword **sqrt** puts a square root sign over the item that follows it, so that **$sqrt X$** generates $\sqrt{X}$, and **$sqrt X + Y$** generates $\sqrt{X} + Y$. However, **$sqrt {X + Y}$** generates $\sqrt{X+Y}$.

We can now start to get more complicated. The above quadratic equation $ax^2 + bx + c = 0$ has the pair of solutions

```
.EQ
x ~=~~ {
  - b +- sqrt {b sup 2 - 4 a c}
} over {2 a}
.EN
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Note the special ***token*** '**+-**', which generates the 'plus-or-minus' symbol. While **$+-$** generates '±', **$+ -$** would generate '+−', so it is important to pay attention to spaces which separate tokens. The two 'spacing' tokens **~** and **^** act like spaces in this respect, so do not need to be separated from their surroundings (see some of the examples of **eqn** code above).

As a general rule of good practice, when entering **eqn** code, it is wise to explicitly separate distinct entities, since **eqn** may (wisely) generate slightly different layout with spacing, in some cases, which is better suited to mathematical typography.

Note how what you type for **eqn** resembles how you might read the above equation out loud:

x equals: minus b plus-or-minus the square root of b-squared-minus-4-a-c, all over 2 a

Defining your own tokens as macros

Following on from the preceding example, if you would prefer writing **squared** to writing **sup 2**, then you could do so by defining **squared** as a keyword:

```
.EQ
define squared %sup 2%
b squared - 4 a c
.EN
```

$$b^2 - 4ac$$

This defines '**squared**' as a macro: when it is encountered, it is replaced by its definition '**sup 2**'.

The general format for such definitions is

define ***keyword*** **%** ***definition*** **%**

where ***definition*** is anything interpretable by **eqn**, and **%** can be replaced by anything that does not occur in ***definition***. Such a definition will persist throughout the document until it is removed by

undef ***keyword***

or else replaced by a new definition of ***keyword***.

If ***definition*** is a mathematical expression, it is wise to enclose it in **{...}** (see below) so that it does not interact undesirably with neighbouring **eqn** code, but care is needed to avoid isolating **eqn** keywords from elements they may need to refer to. For example, defining **squared** as

define squared %{sup 2}%

would not work, since **sup** needs an item to its left as the entity which will be given the superscript, but the '**{**' will isolate **sup** from the left, so that there is nothing for it to apply a superscript to. The result, in this case, would be a syntax error.

Macros with arguments

When such a definition may be used with elements which vary from case to case, the variable elements can be represented by ***macro arguments***, denoted by **$1**, **$2**,…,**$9** (maximum of 9) in the definition. Simple illustrative example:

```
.EQ
define thing %{$5 + $8}%
thing( A, B, C, D, E, F, G, H, I )
.EN
```

$$E + H$$

As a more complex example, suppose we would be repeatedly using the above formula for the solution of a quadratic equation, but with coefficients which vary from case to case. Then the effort of re-typing the full **eqn** representation of the solution every time can be avoided by defining the expression as a macro. This is illustrated below. Note that the entire definition is enclosed in **%{...}%** (see above).

```
.EQ
define quadsol %{
 $1 ~=~ {
 - $3 +- sqrt {$3 sup 2 - 4 $2 $4}
 } over {2 $2}
}%
.EN
```

```
For the quadratic equation
$a x sup 2 + b x + c = 0$,
the solution is:
.EQ
quadsol(x, a, b, c)
.EN
```

For the quadratic equation $ax^2 + bx + c = 0$, the solution is:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
For the quadratic equation
$P Y sup 2 + Q Y + R =~ 0$,
the solution is:
.EQ
quadsol(Y, P, Q, R)
.EN
```

For the quadratic equation $PY^2 + QY + R = 0$, the solution is:

$$Y = \frac{-Q \pm \sqrt{Q^2 - 4PR}}{2P}$$

Big brackets for tall expressions: **left** *and* **right**

```
.EQ
( {a + b} over {c + d} )
.EN
```

$$(\frac{a+b}{c+d})$$

certainly doesn't look right!

```
.EQ
left (
{a + b} over {c + d}
right )
.EN
```

$$\left(\frac{a+b}{c+d}\right)$$

looks much better. For certain "extensible" entities, including the various brackets **(**, **)**, **{**, **}**, **[**, **]**, you can use **left** and **right** so as to make them match the vertical extent of the object they embrace. This is done by constructing the bracket from three parts: top, centre, bottom; of which the centre is extensible. For parentheses (as above), the centre is a straight line, which may not look too good for very tall ones.

Piles and Matrices

These compose vertical stacks, and rectangular arrays, of items. The **pile** keyword is for stacks, and its usage is as follows:

```
pile {
thing1 above thing2 above ... thingk
}
```

Thus the following **eqn** code composes the stack that is shown to its right:

eqn

```
.EQ
pile {A above {c+d}
  above {u sup 2 + v sup 2}
  above {cos x}
}
.EN
```

Result

$$\begin{matrix} A \\ c+d \\ u^2+v^2 \\ \cos x \end{matrix}$$

With **pile**, each item is vertically centred as shown above. You can force each item to be left-justified by using **lpile**, or right-justified by using **rpile**:

eqn

```
.EQ
lpile {A above {c+d}
  above {u sup 2 + v sup 2}
  above {cos x}
}
.EN
```

Result

$$\begin{matrix} A \\ c+d \\ u^2+v^2 \\ \cos x \end{matrix}$$

Now, if appropriate, we could enclose the pile in extended parentheses using **left(** and **right)**:

eqn

```
.EQ
left ( pile {A above {c+d}
  above {u sup 2 + v sup 2}
  above {cos x}
} right )
.EN
```

Result

$$\left(\begin{matrix} A \\ c+d \\ u^2+v^2 \\ \cos x \end{matrix}\right)$$

Note that the sizes of the parentheses are not ideal; we shall come back to this sort of issue later.

To compose a rectangular array, the keyword is **matrix**. Its usage is

matrix { *list-of-columns* }

where each column is defined just like a **pile**, but using the keyword **ccol** (for centred), or **lcol** or **rcol** (for left- or right-justified). Thus:

eqn

```
.EQ
matrix{
  ccol{ A above {B+C} }
  ccol{ D above {E+F} }
}
.EN
```

Result

$$\begin{matrix} A & D \\ B+C & E+F \end{matrix}$$

One useful application in Mathematics for the ***pile*** or the ***matrix*** is in displaying how an expression may take different forms under different conditions. Example (also illustrating the extensible **{**):

```
.EQ
F ( X ) ~=~ left {
matrix{
  lcol{ {4 X} above
        { 1 } above
        {4 ( 1 - X )}
  }
  lcol{
    { roman{"if "}
      0 <= X <= 1 smallover 4
    } above
    { roman{"if "}
      1 smallover 4 <= X <=
      3 smallover 4
    } above
    { roman{"if "}
      3 smallover 4 <= X <= 1
    }
  }
}
.EN
```

$$F(X) = \begin{cases} 4X & \text{if } 0 \le X \le \frac{1}{4} \\ 1 & \text{if } \frac{1}{4} \le X \le \frac{3}{4} \\ 4(1-X) & \text{if } \frac{3}{4} \le X \le 1 \end{cases}$$

Note that the "if " (including the space) is intended as ***plain text*** to be interpolated in the mathematics. The **{roman{"if "}** causes it to be set in Roman type, rather than Italic type which is the default for mathematical symbols.

Mathematical names that **eqn** recognises, and will automatically print in Roman when encountered in input (unless 'protected' by quotes), are:

max	min	lim	arc	sin	cos	tan
exp	log	ln	sinh	cosh	tanh	det
Re	Im	and	if	for		

However, 'and', 'if' and 'for' will be printed with no following space, and their unquoted use (as opposed to the quoted use of "if " above) should be reserved for use within mathematical expressions. The user can define further names (e.g. 'Var' for "variance of") to be handled in the same way within mathematical expressions (see later).

Note also the use of **smallover**. This works just like **over**, but the numerator and denominator will be in smaller type, and put closer to the horizontal line in the fraction, as in $\frac{1}{4}$ and $\frac{3}{4}$ above.

Multiline equations

Often it arises that a long expression must be split over several lines; or that a sequence of successive steps in the derivation of a result should be presented over several lines. An example which illustrates both:

$$\begin{aligned} e^{i\theta} &= 1 + (i\theta) + \tfrac{1}{2!}(i\theta)^2 + \tfrac{1}{3!}(i\theta)^3 + \tfrac{1}{4!}(i\theta)^4 + \cdots \\ &= 1 - \tfrac{1}{2!}\theta^2 + \tfrac{1}{4!}\theta^4 + \cdots \\ &\quad + i\,(\,\theta - \tfrac{1}{3!}\theta^3 + \tfrac{1}{5!}\theta^5 + \cdots) \\ &= \cos\theta + i\sin\theta \end{aligned}$$

where the three steps in the derivation correspond to the three '=' signs, and the second step has itself been split into two lines—first, because of length; secondly, to highlight the two separate infinite series: one for $\cos\theta$, and one for $\sin\theta$. ***Note*** that the spacing between the two lines of the second step is narrower than the spacings between the steps, to emphasise that these two lines belong together.

The **eqn** code for the above is shown below. There are some new elements in it which will be explained in the following part of this page. Each of the four lines above has its own **.EQ I ... .EN** block, (see below); and the spacings are achieved using **.sp 0.5** and **.sp 0.25m**.

```
.EQ I
e sup {i theta} ~~mark =~~
1 ~+~ ( i theta ) ~+~
1 smallover {2^!}^( i theta ) sup 2
~+~1 smallover {3^!}^( i theta ) sup 3
~+~1 smallover {4^!}^( i theta ) sup 4
~+~...
.EN
.sp 0.5m
.EQ I
lineup =~~
1 ~-~ 1 smallover {2^!}^theta sup 2
  ~+~ 1 smallover {4^!}^theta sup 4
  ~+~...
.EN
.sp 0.25m
.EQ I
lineup {hphantom{=~~1}} ~+~i^ (^theta
~-~ 1 smallover {3^!}^ theta sup 3
~+~ 1 smallover {5^!}^ theta sup 5
~+~... ^)
.EN
.sp 0.5m
.EQ I
lineup =~~ cos theta ~+~ i^sin theta
.EN
```

Nearly everything in the above code has already been explained, but there are new elements **mark** and **lineup**, and **hphantom**, which will be explained next; **mark** and **lineup** are part of **eqn**, while **hphantom** is a specially-written add-on.

mark ***and*** **lineup**

These are used to achieve the vertical lineup between the three '=' signs and, along with **hphantom**, to align the '+' in line 3 with the '−' in line 2 above it.

When using **mark** and **lineup**, the **.EQ** should have an explicit positioning parameter following it (**L** or **I** or **C** or **R**, here **I**), else it is unlikely to work.

In the **eqn** block for line 1, where it would nomally be **e sup {i theta} ~~=~~** the token **mark** has been inserted before the '**=**' so as to give **e sup {i theta} ~~mark =~~**.

This has the effect that the horizontal position of the left-hand end of the item immediately following **mark**, here '**=**', is stored for future reference in subsequent **.EQ ... .EN** blocks.

When, in a later block, the token **lineup** occurs, the left-hand end of the item immediately following **lineup** is then vertically aligned to the position stored when **mark** was encountered.

Thus the **lineup**s in lines 2 and 4 ensure that the '=' signs in lines 2 and 4 are vertically aligned with the '=' sign in line 1.

hphantom

Now for **hphantom**. This is a 'special' which has the effect that **hphantom{** ***expression*** **}** generates an invisible **eqn** object which has height zero, and width equal to that of the output which would be created by **eqn** from the **eqn** code in ***expression***.

Thus the code **hphantom{=~~1}** in the third block creates an invisible object occupying exactly as much horizontal space as would be occupied by the result of '**=~~1**'. Subsequent formatted output continues from the right-hand end of this object.

Then, when **hphantom{** ***expression*** **}** is preceded by **lineup**, the left-hand end of this invisible object is vertically aligned to the horizontal position stored when **mark** was encountered. Thus

```
e sup {i theta} ~~mark =~~1 ~+~
lineup {hphantom{=~~1}} ~+~
```

in the first and third blocks result in the two '+' signs in lines 1 and 3 being vertically aligned, and hence in the '+' sign in line 3 being aligned with the '−' in line 2.

I have also written 'specials' **hphantom{...}** (an invisible object, width = 0 and height equal to **...**) and **hphantom{...}** (invisible object matching both the width and the height of **...**).

***Code for* `hphantom`**

`hphantom` is defined by:

(a) a **`groff`** macro **`hphntmsrc`**, which can be placed at the start of the document or in a macro file; and:

(b) a 'special' definition **`hphantom`** within **`eqn`** code which evokes **`hphntmsrc`**

and similarly for **`vphntmsrc`** & **`vphantom`**, and for **`phntmsrc`** & **`phantom`**:

```
.de hphntmsrc
.nr 0h 0
.nr 0d 0
.nr 0skew 0
.nr 0skern 0
.ds 0s \\&\h'\\n(0wu'
..
.de vphntmsrc
.ds 0s
.nr 0w 0
.nr 0skew 0
.nr 0skern 0
..
.de phntmsrc
.ds 0s
.ds 0s \\&\h'\\n(0wu'
..
.EQ
define phantom 'special phntmsrc'
define vphantom 'special vphntmsrc'
define hphantom 'vcenter special hphntmsrc'
.EN
```