mathml: Translate R Expressions to MathML and LaTeX

by Matthias Gondan and Irene Alfarone

Abstract This R package translates R objects to suitable elements in MathML or LaTeX, thereby allowing for a pretty mathematical representation of R objects and functions in data analyses, scientific reports and interactive web content. In the R Markdown document rendering language, R code and mathematical content already exist side-by-side. The present package enables use of the same R objects for both data analysis and typesetting in documents or web content. This tightens the link between the statistical analysis and its verbal description or symbolic representation, which is another step towards reproducible science. User-defined hooks enable extension of the package by mapping specific variables or functions to new MathML and LaTeX entities. Throughout the paper, examples are given for the functions of the package, and a case study illustrates its use in a scientific report.

1 Introduction

The R extension of the markdown language (Xie, Allaire, and Grolemund 2018; Allaire et al. 2023) enables reproducible statistical reports with nice typesetting in HTML, Microsoft Word, and LaTeX. Moreover, since recently (R Core Team 2022, version 4.2), Rs manual pages include support for mathematical expressions (Sarkar and Hornik 2022; Viechtbauer 2022), which is already a big improvement. However, except for special cases such as regression models (Anderson, Heiss, and Sumners 2023) and R's own plotmath annotation, rules for the mapping of built-in language elements to their mathematical representation are still lacking. So far, R expressions such as `pbinom(k, N, p)` are printed as they are and pretty mathematical formulae such as $P_{\mathrm{Bi}}(X \le k; N, p)$ require explicit LaTeX commands like `P_{\mathrm{Bi}}\left(X \le k; N, p\right)`. Except for very basic use cases, these commands are tedious to type and their source code is hard to read.

The present R package defines a set of rules for the automatic translation of R expressions to mathematical output in R Markdown documents (Xie, Dervieux, and Riederer 2020) and Shiny Apps (Chang et al. 2022). The translation is done by an embedded Prolog interpreter that maps nested expressions recursively to MathML and LaTeX/MathJax, respectively. User-defined hooks enable extension of the set of rules, for example, to represent specific R elements by custom mathematical signs.

The main feature of the package is that the same R expressions and equations can be used for both mathematical typesetting and calculations. This saves time and potentially reduces mistakes, as will be illustrated below. Readers should have basic knowledge of knitr and R Markdown to be able to follow this article (Xie 2023; Allaire et al. 2023), while to extend and customize the package, some basic knowledge of Prolog is needed.

The paper is organized as follows. We start with a description of the technical background of the package, including the two main classes of rules for translating R objects to mathematical expressions. The next section illustrates the main features of the mathml package, potential issues and their workarounds using examples from the day-to-day perspective of a user. A case study follows with a scientific report written with the help of the package. The last section concludes with a discussion and ideas for further development.

2 Background

Similar to other high-level programming languages, R is homoiconic, that is, R commands (i.e., R "calls") are, themselves, symbolic data structures that can be created, parsed and modified. Because the default response of the R interpreter is to evaluate a call and return its result, this property is not transparent to the general user. There exists, however, a number of built-in R functions (e.g., `quote()`, `call()` etc.) that allow the user to create R calls which can be stored in regular variables and then, for example, evaluated at a later stage or in a specific environment (Wickham 2019). The present package includes a set of rules that translate such calls to a mathematical representation in MathML and LaTeX. For a first illustration of the mathml package, we consider the binomial probability.

```
term <- quote(pbinom(k, N, p))
```

The term is quoted to avoid its immediate evaluation (which would raise an error anyway since

the variables k, N, p have not yet been defined). Experienced R users will recognize that the expression is a short form for

```
term <- call("pbinom", as.name("k"), as.name("N"), as.name("p"))
term

#> pbinom(k, N, p)
```

As can be seen from the output, to the variable term is not assigned the result of the calculation, but instead an R call (see, e.g., Wickham 2019, for details on "non-standard evaluation"), which can eventually be evaluated with eval(),

```
k <- 10
N <- 22
p <- 0.4
eval(term)

#> [1] 0.77195
```

The R package mathml can now be used to render the call in MathML or in MathJax/LaTeX. MathML is the dialect for mathematical elements on HTML webpages, whereas LaTeX is typically used for typesetting printed documents, as shown below.

```
library(mathml)
substr(mathml(term), 1, 70)

#> [1] "<math><mrow><msub><mi>P</mi><mtext>Bi</mtext></msub><mo>&af;</mo><mrow"

mathjax(term)

#> [1] "${P}_{\\mathrm{Bi}}{\\left({{X}{\\le}{k}}{;}{{N}{,}{p}}}\\right)}$"
```

Some of the curly braces are not really needed in the LaTeX output, but are necessary in edge cases. The package also includes a function mathout() that wraps a call to mathml() for HTML output and mathjax() for LaTeX output. Moreover, the function math(x) adds the class "math" to its argument, such that a special knitr printing function is invoked (see the vignette on custom print methods in Xie 2023). An R Markdown code chunk with mathout(term) thus produces:

$P_{\mathrm{Bi}}(X \le k; N, p)$

Similarly, inline() produces inline output, `r inline(term)` yields $P_{\mathrm{Bi}}(X \le k; N, p)$.

3 Package mathml in practice

The currently supported R objects are listed below, roughly following the order proposed by Murrell and Ihaka (2000).

3.1 Basic elements

mathml handles the basic elements of everyday mathematical expressions, such as integers, floating-point numbers, Latin and Greek letters, multi-letter identifiers, accents, subscripts, and superscripts.

```
term <- quote(1 + -2L + a + abc + "a" + phi + Phi + varphi + roof(b)[i, j]^2L)
math(term)
```

$1.00 + -2 + a + abc + \mathrm{a} + \phi + \Phi + \varphi + \hat{b}^2_{ij}$

```
term <- quote(round(3.1415, 3L) + NaN + NA + TRUE + FALSE + Inf + (-Inf))
math(term)
```

$3.142 + nan + na + T + F + \infty + (-\infty)$

An expression such as `1 + -2` may be considered aesthetically unsatisfactory. It is correct R syntax, though, and is reproduced accordingly, without the parentheses. Parentheses around negative numbers or symbols can be added as shown above for `+ (-Inf)`.

To avoid name clashes with package `stats`, `roof()` is used to put a hat on a symbol (see next section for further decorations). Note that an R function `roof()` does not exist in base R, it is provided by the package for convenience and points to the identity function.

3.2 Decorations

The package offers some support for different fonts as well as accents and boxes etc. Internally, these decorations are implemented as identity functions, so that they can be introduced into R expressions without side-effects.

```
term <- quote(bold(b[x, 5L]) + bold(b[italic(x)]) + italic(ab) + italic(42L))
math(term)
```

$\mathbf{b}_{\mathbf{x5}}+\mathbf{b}_{x}+ab+42$

```
term <- quote(tilde(a) + mean(X) + box(c) + cancel(d) + phantom(e) + prime(f))
math(term)
```

$\tilde{a}+\overline{X}+\boxed{c}+\cancel{d}+\ +f'$

Note that the font styles only affect the display of identifiers, whereas numbers, character strings etc. are left untouched.

3.3 Operators and parentheses

Arithmetic operators and parentheses are translated as they are, as illustrated below.

```
term <- quote(a - ((b + c)) - d*e + f*(g + h) + i/j + k^(l + m) + (n*o)^{p + q})
math(term)
```

$a-[(b+c)]-de+f\cdot(g+h)+i/j+k^{(l+m)}+(no)^{p+q}$

```
term <- quote(dot(a, b) + frac(1L, nodot(c, d + e)) + dfrac(1L, times(g, h)))
math(term)
```

$a\cdot b+\frac{1}{c(d+e)}+\dfrac{1}{g\times h}$

For multiplications involving only numbers and symbols, the multiplication sign is omitted. This heuristic does not always produce the desired result; therefore, mathml defines alternative R functions `dot()`, `nodot()`, and `times()`. These functions calculate a product and produce the respective multiplication signs. Similarly, `frac()` and `dfrac()` can be used for small and large fractions.

For standard operators with known precedence, mathml is generally able to detect if parentheses are needed; for example, parentheses are automatically placed around `d + e` in the `nodot`-example. However, we note unnecessary parentheses around `l + m` above. These parentheses are a consequence of `quote(a^(b + c))` actually producing a nested R call of the form `^(a, (b + c))` instead of `^(a, b + c)`:

```
term <- quote(a^(b + c))
paste(term)
```

```
#> [1] "^"       "a"       "(b + c)"
```

For the present purpose, this R feature is unfortunate because the extra parentheses around `b + c` are not needed. The preferred result is obtained by the functional form `quote(^(k, l + m))` of the power, or curly braces as a workaround (see `p + q` above).

3.4 Custom operators

Whereas in standard infix operators, the parentheses typically follow the rules for precedence, undesirable results may be obtained in custom operators.

```
term <- quote(mean(X) %+-% 1.96 * s / sqrt(N))
math(term)
```

$$\left(\overline{X}\pm 1.96\right)\cdot s/\sqrt{N}$$

```
term <- quote('%+-%'(mean(X), 1.96 * s / sqrt(N))) # functional form of '%+-%'
term <- quote(mean(X) %+-% {1.96 * s / sqrt(N)})   # the same
math(term)
```

$$\overline{X}\pm 1.96 s/\sqrt{N}$$

The example is a reminder that it is not possible to define the precedence of custom operators in R, and that expressions with such operators are evaluated strictly from left to right. Again, the problem can be worked around by the functional form of the operator or a curly brace to hide the parenthesis, and, at the same time, enforce the correct operator precedence.

More operators are shown in Table 1, including the suggestions by Murrell and Ihaka (2000) for graphical annotations and arrows in R figures.

Table 1: Custom operators in mathml

Operator	Output	Operator	Output	Operator	Arrow
A %*% B	$A \times B$	A != B	$A \neq B$	A %% B	$A \leftrightarrow B$
A %.% B	$A \cdot B$	A ~ B	$A \sim B$	A %->% B	$A \rightarrow B$
A %x% B	$A \otimes B$	A %~~% B	$A \approx B$	A %<-% B	$A \leftarrow B$
A %/% B	$\lfloor A/B \rfloor$	A %==% B	$A \equiv B$	A %up% B	$A \uparrow B$
A %% B	$mod(A,B)$	A %=~% B	$A \cong B$	A %down% B	$A \downarrow B$
A & B	$A \wedge B$	A %prop% B	$A \propto B$	A %<=>% B	$A \iff B$
A | B	$A \vee B$	A %in% B	$A \in B$	A %=>% B	$A \Rightarrow B$
xor(A, B)	$A \veebar B$	intersect(A, B)	$A \cap B$	A %<=% B	$A \Leftarrow B$
!A	$\neg A$	union(A, B)	$A \cup B$	A %dblup% B	$A \Uparrow B$
A == B	$A = B$	crossprod(A, B)	$A^{\mathrm{T}} \times B$	A %dbldown% B	$A \Downarrow B$
A <- B	$A = B$	is.null(A)	$A = \emptyset$		

3.5 Builtin functions

There is support for most functions from package `base`, with adequate use and omission of parentheses.

```
term <- quote(sin(x) + sin(x)^2L + cos(pi/2L) + tan(2L*pi) * expm1(x))
math(term)
```

$$\sin x + (\sin x)^2 + \cos(\pi/2) + \tan(2\pi)\cdot(\exp x - 1)$$

```
term <- quote(choose(N, k) + abs(x) + sqrt(x) + floor(x) + exp(frac(x, y)))
math(term)
```

$$\binom{N}{k} + |x| + \sqrt{x} + \lfloor x \rfloor + \exp\left(\frac{x}{y}\right)$$

A few more examples are shown in Table 2, including functions from `stats`.

Table 2: R functions from base and stats

Function	Output	Function	Output
sin(x)	$\sin x$	dbinom(k, N, pi)	$P_{\mathrm{Bi}}(X=k;N,\pi)$
cosh(x)	$\cosh x$	pbinom(k, N, pi)	$P_{\mathrm{Bi}}(X\leq k;N,\pi)$
tanpi(alpha)	$\tan(\alpha\pi)$	qbinom(p, N, pi)	$\arg\min_k[P_{\mathrm{Bi}}(X\leq k;N,\pi)>p]$
asinh(x)	$\sinh^{-1} x$	dpois(k, lambda)	$P_{\mathrm{Po}}(X=k;\lambda)$

Function	Output	Function	Output
log(p)	$\log p$	ppois(k, lambda)	$P_{\mathrm{Po}}(X \leq k;\lambda)$
log1p(x)	$\log(1+x)$	qpois(p, lambda)	$\arg\max_k[P_{\mathrm{Po}}(X \leq k;\lambda) > p]$
logb(x, e)	$\log_e x$	dexp(x, lambda)	$f_{\mathrm{Exp}}(x;\lambda)$
exp(x)	$\exp x$	pexp(x, lambda)	$F_{\mathrm{Exp}}(x;\lambda)$
expm1(x)	$\exp x - 1$	qexp(p, lambda)	$F^{-1}_{\mathrm{Exp}}(p;\lambda)$
choose(n, k)	$\binom{n}{k}$	dnorm(x, mu, sigma)	$\phi(x;\mu,\sigma)$
lchoose(n, k)	$\log\binom{n}{k}$	pnorm(x, mu, sigma)	$\Phi(x;\mu,\sigma)$
factorial(n)	$n!$	qnorm(alpha/2L)	$\Phi^{-1}(\alpha/2)$
lfactorial(n)	$\log n!$	1L - pchisq(x, 1L)	$1 - F_{\chi^2(1\,\mathrm{df})}(x)$
sqrt(x)	$\sqrt{x}$	qchisq(1L - alpha, 1L)	$F^{-1}_{\chi^2(1\,\mathrm{df})}(1-\alpha)$
mean(X)	$\overline{X}$	pt(t, N - 1L)	$P(T \leq t; N-1\,\mathrm{df})$
abs(x)	$\lvert x \rvert$	qt(alpha/2L, N - 1L)	$T_{\alpha/2}(N-1\,\mathrm{df})$

3.6 Custom functions

For self-written functions, the matter is somewhat more complicated. For a function such as `g <- function(...) ...`, the name g is not transparent to R, because only the function body is represented. We can still display functions in the form `head(x) = body` if we embed the object to be shown into a call `"<-"(head, body)`.

```
sgn <- function(x)
{
  if(x == 0L) return(0L)
  if(x < 0L) return(-1L)
  if(x > 0L) return(1L)
}

math(sgn)
```

$$\begin{cases} 0, \text{ if } x=0 \\ -1, \text{ if } x<0 \\ 1, \text{ if } x>0 \end{cases}$$

```
math(call("<-", quote(sgn(x)), sgn))
```

$$\operatorname{sgn} x = \begin{cases} 0, \text{ if } x=0 \\ -1, \text{ if } x<0 \\ 1, \text{ if } x>0 \end{cases}$$

The function body is generally a nested R call of the form `{(L)`, with `L` being a list of commands (the semicolon, not necessary in R, is translated to a newline). The example also illustrates that mathml provides limited support for control structures such as `if` that is internally represented as `if(condition, action)`.

3.7 Indices and powers

Indices in square brackets are rendered as subscripts, powers are rendered as superscript. Moreover, mathml defines the functions `sum_over(x, from, to)`, and `prod_over(x, from, to)` that simply return their first argument. The other two arguments serve as decorations (*to* is optional), for example, for summation and product signs.

```
term <- quote(S[Y]^2L <- frac(1L, N) * sum(Y[i] - mean(Y))^2L)
math(term)
```

$$S^2_Y = \frac{1}{N} \cdot \sum \left(Y_i - \overline{Y}\right)^2$$

```
term <- quote(log(prod_over(L[i], i==1L, N)) <- sum_over(log(L[i]), i==1L, N))
math(term)
```

$$\log \prod_{i=1}^{N} L_i = \sum_{i=1}^{N} \log L_i$$

3.8 Ringing back to R

Rs `integrate` function takes a number of arguments, the most important ones being the function to integrate, and the lower and the upper bound of the integration.

```
term <- quote(integrate(sin, 0L, 2L*pi))
math(term)
```

$$\int_0^{2\pi} \sin x \, dx$$

```
eval(term)

#> 2.221482e-16 with absolute error < 4.4e-14
```

For mathematical typesetting in the form of $\int f(x)\,dx$, mathml needs to find out the name of the integration variable. For that purpose, the underlying Prolog bridge provides a predicate `r_eval/2` that calls R from Prolog. This predicate is used to evaluate `formalArgs(args(sin))` and returns the names of the arguments of `sin()`, namely, x.

Note that in the example above, the quoted term is an abbreviation for `call("integrate", quote(sin), ...)`, with `sin` being an R symbol, not a function. While the R function `integrate()` can handle both symbols and functions, mathml needs the symbol because it is unable to determine the function name of custom functions.

3.9 Names and order of arguments

One of R's great features is the possibility to refer to function arguments by their names, not only by their position in the list of arguments. At the other end, the Prolog handlers for R calls are rather rigid, for example, `integrate/3` accepts exactly three arguments in a particular order and without names, that is, `integrate(lower=0L, upper=2L*pi, sin)`, would not print the desired result.

To "canonicalize" function calls with named arguments and arguments in unusual order, mathml provides an auxiliary R function `canonical(f, drop)` that reorders the argument list of calls to known R functions and, if `drop=TRUE` (which is the default), also removes the names of the arguments.

```
term <- quote(integrate(lower=0L, upper=2L*pi, sin))
canonical(term)

#> integrate(sin, 0L, 2L * pi)

math(canonical(term))
```

$$\int_0^{2\pi} \sin x \, dx$$

This function can be used to feed mixtures of partially named and positional arguments into the renderer. For details, see the R function `match.call()`.

3.10 Matrices and Vectors

Of course, mathml also supports matrices and vectors.

```
v <- 1:3
math(call("t", v))
```

$$(1\,2\,3)^{\mathrm{T}}$$

```
A <- matrix(data=11:16, nrow=2, ncol=3)
B <- matrix(data=21:26, nrow=2, ncol=3)
term <- call("+", A, B)
math(term)
```

$$\begin{pmatrix} 11 & 13 & 15 \\ 12 & 14 & 16 \end{pmatrix} + \begin{pmatrix} 21 & 23 & 25 \\ 22 & 24 & 26 \end{pmatrix}$$

Note that the seemingly more convenient `term <- quote(A + B)` yields $A + B$ in the output—instead of the desired matrix representation. This behavior is expected because quotation of R calls also quote the components of the call (here, A and B).

3.11 Short mathematical names for R symbols

In typical R functions, variable names are typically longer than just single letters, which may yield unsatisfactory results in the mathematical output.

```
term <- quote(pbinom(successes, Ntotal, prob))
math(term)
```

$P_{\mathrm{Bi}}(X \leq successes; Ntotal, prob)$

```
hook(successes, k)
hook(quote(Ntotal), quote(N), quote=FALSE)
hook(prob, pi)
math(term)
```

$P_{\mathrm{Bi}}(X \leq k; N, \pi)$

To improve the situation, mathml provides a simple hook that can be used to replace elements (e.g., verbose variable names) of the code by concise mathematical symbols, as illustrated in the example. To simplify notation, `hook()` uses non-standard evaluation of its arguments. If the `quote` flag of `hook()` is set to `FALSE`, the user has to provide the quoted expressions. Care should be taken to avoid recursive hooks such as `hook(s, s["A"])` that endlessly replace the s from s_{A} as in $s_{\mathrm{A}_{\mathrm{A}_{\ldots}}}$.

The hooks can also be used for more complex elements such as R calls, with dotted symbols representing Prolog variables.

```
hook(pbinom(.K, .N, .P), sum_over(dbinom(i, .N, .P), i=0L, .K))
math(term)
```

$\sum_{i=0}^{k} P_{\mathrm{Bi}}(X = i; N, \pi)$

Further customization requires the assertion of new Prolog rules `math/2`, `ml/3`, `jax/3`, as shown in the Appendix.

3.12 Abbreviations

We consider the t-statistic for independent samples with equal variance. To avoid clutter in the equation, the pooled variance s^2_{pool} is abbreviated, and a comment is given with the expression for s^2_{pool}. For this purpose, mathml provides a function `denote(abbr, expr, info)`, with `expr` actually being evaluated, `abbr` being rendered, plus a comment of the form "with `expr` denoting `info`".

```
hook(m_A, mean(X)["A"]) ; hook(s2_A, s["A"]^2L) ;
hook(n_A, n["A"])
hook(m_B, mean(X)["B"]) ; hook(s2_B, s["B"]^2L)
hook(n_B, n["B"]) ; hook(s2_p, s["pool"]^2L)

term <- quote(t <- dfrac(m_A - m_B,
    sqrt(denote(s2_p, frac((n_A - 1L)*s2_A + (n_B - 1L)*s2_B, n_A + n_B - 2L),
                "the pooled variance.") * (frac(1L, n_A) + frac(1L, n_B)))))
math(term)
```

$t = \dfrac{\overline{X}_{\mathrm{A}} - \overline{X}_{\mathrm{B}}}{\sqrt{s^2_{\mathrm{pool}} \cdot \left(\frac{1}{n_{\mathrm{A}}} + \frac{1}{n_{\mathrm{B}}}\right)}}$, with $s^2_{\mathrm{pool}} = \frac{(n_{\mathrm{A}}-1)\cdot s^2_{\mathrm{A}} + (n_{\mathrm{B}}-1)\cdot s^2_{\mathrm{B}}}{n_{\mathrm{A}}+n_{\mathrm{B}}-2}$ denoting the pooled variance.

The term is evaluated below. `print()` is needed because the return value of an assignment of the form `t <- dfrac(...)` is not visible in R.

```
m_A <- 1.5; s2_A <- 2.4^2; n_A <- 27; m_B <- 3.9; s2_B <- 2.8^2; n_B <- 20
print(eval(term))

#> [1] -3.157427
```

3.13 Context-dependent rendering

Consider an educational scenario in which we want to highlight a certain element of a term, for example, that a student has forgotten to subtract the null hypothesis in a t-ratio:

```
t <- quote(dfrac(omit_right(mean(D) - mu[0L]), s / sqrt(N)))
math(t, flags=list(error="highlight"))
```

$$\frac{\overline{D} - \cancel{\mu_0}}{s/\sqrt{N}}$$

```
math(t, flags=list(error="fix"))
```

$$\frac{\overline{D}\boxed{-\mu_0}}{s/\sqrt{N}}$$

The R function `omit_right(a + b)` uses non-standard evaluation techniques (e.g., Wickham 2019) to return only the left part an operation, and cancels the right part. This may not always be desired, for example, when illustrating how to fix the mistake.

For this purpose, the functions `mathml()`, `mathjax()`, `mathout()` and `math()` have an optional argument `flags` which is a list with named elements. In this example, we use this argument to tell mathml how to render such erroneous expressions using the flag `error` which can be "asis", "highlight", "fix", or "ignore". For more examples, see Table 3.

Table 3: Highlighting elements of a term

Operation	error = asis	highlight	fix	ignore
omit_left(a + b)	b	$\cancel{a+}b$	$\boxed{a+}b$	$a+b$
omit_right(a + b)	a	$a\cancel{+b}$	$a\boxed{+b}$	$a+b$
list(quote(a), quote(omit(b)))	a	$a\not{b}$	$a\boxed{b}$	$a\,b$
add_left(a + b)	$a+b$	$\boxed{a+}b$	$\cancel{a+}b$	b
add_right(a + b)	$a+b$	$a\boxed{+b}$	$a\cancel{+b}$	a
list(quote(a), quote(add(b)))	$a\,b$	$a\boxed{b}$	$a\not{b}$	a
instead(a, b) + c	$a+c$	$\underbrace{a}_{\text{instead of } b}+c$	$\boxed{b}+c$	$b+c$

4 A case study

This case study describes a model by Schwarz (1994) from mathematical psychology using the features of package mathml. Schwarz (1994) presents a new explanation of redundancy gains that occur when observers respond to stimuli of different sources, and the same information is presented on two or more channels. In Schwarz's (1994) model, decision-making builds on a process of noisy accumulation of information over time (e.g., Ratcliff et al. 2016). In redundant stimuli, the model assumes a superposition of channel-specific diffusion processes that eventually reach an absorbing barrier to elicit the response. For a detailed description the reader may refer to the original article.

Schwarz's (1994) model refers to two stimuli A and B, presented either alone or in combination (AB, redundant stimuli), with the redundant stimuli being presented either simultaneously or with onset asynchrony τ. The channel activation is described as a two-dimensional Wiener process with drifts μ_i, variances σ_i^2, and initial conditions $X_i(t=0) = 0, i = \mathrm{A}, \mathrm{B}$. The buildup of channel-specific activation may be correlated with ρ_{AB}, but we assume $\rho_{\mathrm{AB}} = 0$ for simplicity.

A response is elicited when the process reaches an absorbing barrier $c > 0$ for the first time. In single-target trials, the first passages of c are expected at

```
ED_single <- function(c, mu)
  dfrac(c, mu)

# display as E(D; mu), c is a scaling parameter
hook(ED_single(.C, .Mu), E(`;`(D, .Mu)))
math(call("=", quote(ED_single(c, mu)), ED_single))
```

$$E(D;\mu)=\frac{c}{\mu}$$

One would typically use chunk option `echo=FALSE` to suppress the R code.

In redundant stimuli, the activation from the channel-specific diffusion processes adds up, $X_{AB}(t) = X_A(t) + X_B(t)$, hence the name, superposition. $X_{AB}(t)$ is again a Wiener process with drift $\mu_A + \mu_B$ and variance $\sigma_A^2 + \sigma_B^2$. For the expected first-passage time, we have

```
hook(mu_A, mu["A"])
hook(mu_B, mu["B"])
hook(sigma_A, sigma["A"])
hook(sigma_B, sigma["B"])
hook(mu_M, mu["M"])
hook(M, overline(X))

math(call("=", quote(E(D["AB"])), quote(ED_single(c, mu_A + mu_B))))
```

$$E(D_{AB})=E(D;\mu_A+\mu_B)$$

For asynchronous stimuli, Schwarz (1994) derived the expected first-passage time as a function of the stimulus onset asynchrony τ,

```
ED_async <- function(tau, c, mu_A, sigma_A, mu_B)
{ dfrac(c, mu_A) + (dfrac(1L, mu_A) - dfrac(1L, mu_A + mu_B)) *
    ((mu_A*tau - c) * pnorm(dfrac(c - mu_A*tau, sqrt(sigma_A^2L*tau)))
      - (mu_A*tau + c) * exp(dfrac(2L*c*mu_A, sigma_A^2L))
        * pnorm(dfrac(-c - mu_A*tau, sqrt(sigma_A^2L*tau))))
}

hook(ED_async(.Tau, .C, .MA, .SA, .MB), E(`;`(D[.Tau], `,`(.MA, .SA, .MB))))
math(call("=", quote(E(D[tau])), ED_async))
```

$$E(D_\tau)=\frac{c}{\mu_A}+\left(\frac{1}{\mu_A}-\frac{1}{\mu_A+\mu_B}\right)\cdot\left[(\mu_A\tau-c)\cdot\Phi\left(\frac{c-\mu_A\tau}{\sqrt{\sigma_A^2\tau}}\right)-(\mu_A\tau+c)\cdot\exp\left(\frac{2c\mu_A}{\sigma_A^2}\right)\cdot\Phi\left(\frac{-c-\mu_A\tau}{\sqrt{\sigma_A^2\tau}}\right)\right]$$

For negative onset asynchrony (i.e., B before A), the parameters are simply switched.

```
ED <- function(tau, c, mu_A, sigma_A, mu_B, sigma_B)
{
  if(tau == Inf) return(ED_single(c, mu_A))
  if(tau == -Inf) return(ED_single(c, mu_B))
  if(tau == 0L) return(ED_single(c, mu_A + mu_B))
  if(tau > 0L) return(ED_async(tau, c, mu_A, sigma_A, mu_B))
  if(tau < 0L) return(ED_async(abs(tau), c, mu_B, sigma_B, mu_A))
}

hook(ED(.Tau, .C, .MA, .SA, .MB, .SB), E(`;`(D[.Tau], `,`(.MA, .SA, .MB, .SB))))
math(call("=", quote(ED(tau, c, mu_A, sigma_A, mu_B, sigma_B)), ED))
```

$$E(D_\tau;\mu_A,\sigma_A,\mu_B,\sigma_B)=\begin{cases} E(D;\mu_A), \text{ if } \tau=\infty \\ E(D;\mu_B), \text{ if } \tau=-\infty \\ E(D;\mu_A+\mu_B), \text{ if } \tau=0 \\ E(D_\tau;\mu_A,\sigma_A,\mu_B), \text{ if } \tau>0 \\ E\left(D_{|\tau|};\mu_B,\sigma_B,\mu_A\right), \text{ if } \tau<0 \end{cases}$$

The observable response time is assumed to be the sum of D, the time employed to reach the threshold for the decision, and a residual M denoting other processes such as motor preparation and execution. Correspondingly, the expected response time amounts to

```
ET <- function(tau, c, mu_A, sigma_A, mu_B, sigma_B, mu_M)
  ED(tau, c, mu_A, sigma_A, mu_B, sigma_B) + mu_M

hook(ET(.Tau, .C, .MA, .SA, .MB, .SB, .MM),
     E(`;`(T[.Tau], `,`(.MA, .SA, .MB, .SB, .MM))))
math(call("=", quote(E(T[tau])), ET))
```

$E(T_\tau)$=$E(D_\tau;\mu_A,\sigma_A,\mu_B,\sigma_B)+\mu_M$

Schwarz (1994) applied the model to data from a redundant signals task (Miller 1986) with 13 onset asynchronies 0, ±33, ±67, ±100, ±133, ±167, ±∞ ms, where $\tau = 0$ refers to the synchronous condition, and ±∞ to the single-target presentations. Each condition was replicated 400 times. The observed mean response times and their standard deviations are given in Table 4.

Table 4: Miller (1986) data

τ	m	s	n
$-\infty$	231	56	400
-167	234	58	400
-133	230	40	400
-100	227	40	400
-67	228	32	400
-33	221	28	400
0	217	28	400
33	238	28	400
67	263	26	400
100	277	30	400
133	298	32	400
167	316	34	400
∞	348	92	400

Assuming that the model is correct, the observable mean reaction times follow an approximate Normal distribution around the model prediction $E(T_\tau)$ for each condition. We can, therefore, use a standard goodness-of-fit measure by z-standardization.

```
z <- function(m, s, n, tau, c, mu_A, sigma_A, mu_B, sigma_B, mu_M)
  dfrac(m - denote(mu[tau], ET(tau, c, mu_A, sigma_A, mu_B, sigma_B, mu_M),
              "the expected mean response time"),
    s / sqrt(n))

math(call("=", quote(z[tau]), z))
```

$z_\tau=\frac{m-\mu_\tau}{s/\sqrt{n}}$, with $\mu_\tau=E(T_\tau;\mu_A,\sigma_A,\mu_B,\sigma_B,\mu_M)$ denoting the expected mean response time

The overall goodness-of-fit is the sum of the squared z-statistics for each onset asynchrony. Assuming again that the architecture of the model is correct, but the parameters are adjusted to the data, it follows a $\chi^2(8\text{ df})$-distribution.

```
zv <- Vectorize(z, vectorize.args = c('m', 's', 'n', 'tau'))
hook(zv(.M, .S, .N, .Tau, .C, .MA, .SA, .MB, .SB, .MM), z[.Tau])

gof <- function(par, tau, m, s, n)
  sum(zv(m, s, n, tau, c=100L, mu_A=par["mu_A"], sigma_A=par["sigma_A"],
         mu_B=par["mu_B"], sigma_B=par["sigma_B"], mu_M=par["mu_M"])^2L)

math(call("=", quote(X["8 df"]^2L), gof))
```

$X^2_{8\text{df}}=\sum z^2_\tau$

with the degrees of freedom given by the difference between the number of observations (13) and the number of free model parameters $\theta = \langle \mu_A, \sigma_A, \mu_B, \sigma_B, \mu_M \rangle$; the barrier c is only a scaling parameter.

$\hat{\theta}$=$\arg\min gof(\theta)$

The best fitting parameter values and their confidence intervals are given in Table 5.

Table 5: Model fit

Parameter	Estimate	CI
μ_A	0.53	(0.51,0.55)
σ_A	4.55	(3.95,5.16)

Parameter	Estimate	CI
μ_B	1.36	(1.23,1.49)
σ_B	13.46	(7.80,19.11)
μ_M	161.09	(156.91,165.28)

The goodness-of-fit statistic indicates some lack of fit, $X^2(8 \text{ df}) = 28.34$, $p = 0.0004$. Given the large trial numbers in the original study, this is not an unexpected result. For more detail, especially on fitting the observed standard deviations, the reader is referred to the original paper (Schwarz 1994).

5 Conclusion

This package allows R to render its terms in pretty mathematical equations. It extends the current features of R and existing packages for displaying mathematical formulas in R (Murrell and Ihaka 2000), but most importantly, mathml bridges the gap between computational needs, presentation of results, and their reproducibility. The package supports both MathML and LaTeX/MathJax for use in R Markdown documents, presentations and Shiny App webpages.

Researchers or teachers can already use R Markdown to conduct analyses and show results, and mathml smoothes this process and allows for integrated calculations and output. As shown in the case study of the previous section, mathml can help to improve data analyses and statistical reports from an aesthetical perspective, as well as regarding reproducibility of research.

Furthermore, the package may also allow for a better detection of possible mistakes in R programs. Similar to most programming languages (Green 1977), R code is notoriously hard to read, and the poor legibility of the language is one of the main sources of mistakes. For illustration, we consider again Equation 10 in Schwarz (1994).

```
f1 <- function(tau)
{ dfrac(c, mu_A) + (dfrac(1L, mu_A) - dfrac(1L, mu_A + mu_B) *
    ((mu_A*tau - c) * pnorm(dfrac(c - mu_A*tau, sqrt(sigma_A^2L*tau)))
      - (mu_A*tau + c) * exp(dfrac(2L*mu_A*tau, sigma_A^2L))
        * pnorm(dfrac(-c - mu_A*tau, sqrt(sigma_A^2L*tau)))))
}

math(f1)
```

$$\frac{c}{\mu_A}+\left\{\frac{1}{\mu_A}-\frac{1}{\mu_A+\mu_B}\cdot\left[(\mu_A\tau-c)\cdot\Phi\left(\frac{c-\mu_A\tau}{\sqrt{\sigma_A^2\tau}}\right)-(\mu_A\tau+c)\cdot\exp\left(\frac{2\mu_A\tau}{\sigma_A^2}\right)\cdot\Phi\left(\frac{-c-\mu_A\tau}{\sqrt{\sigma_A^2\tau}}\right)\right]\right\}$$

The first version has a wrong parenthesis, which is barely visible in the code, whereas in the mathematical representation, the wrong curly brace is immediately obvious (the correct version is shown below for comparison).

```
f2 <- function(tau)
{ dfrac(c, mu_A) + (dfrac(1L, mu_A) - dfrac(1L, mu_A + mu_B)) *
    ((mu_A*tau - c) * pnorm(dfrac(c - mu_A*tau, sqrt(sigma_A^2L*tau)))
      - (mu_A*tau + c) * exp(dfrac(2L*mu_A*tau, sigma_A^2L))
        * pnorm(dfrac(-c - mu_A*tau, sqrt(sigma_A^2L*tau))))
}

math(f2)
```

$$\frac{c}{\mu_A}+\left(\frac{1}{\mu_A}-\frac{1}{\mu_A+\mu_B}\right)\cdot\left[(\mu_A\tau-c)\cdot\Phi\left(\frac{c-\mu_A\tau}{\sqrt{\sigma_A^2\tau}}\right)-(\mu_A\tau+c)\cdot\exp\left(\frac{2\mu_A\tau}{\sigma_A^2}\right)\cdot\Phi\left(\frac{-c-\mu_A\tau}{\sqrt{\sigma_A^2\tau}}\right)\right]$$

As the reader may know from their own experience, missed parentheses are frequent causes of wrong results and errors that are hard to locate in programming code. This particular example shows that mathematical rendering can help to substantially reduce the amount of careless errors in programming.

One limitation of the package is the lack of a convenient way to insert line breaks. This is mostly due to lacking support by MathML and LaTeX renderers. For example, in its current stage, the LaTeX package breqn (Robertson et al. 2021) is mostly a proof of concept. Moreover, mathml only works in

one direction, that is, it is not possible to translate from LaTeX or HTML back to R (see Capretto 2023, for an example).

The package mathml is available for R version 4.2 and later, and can be easily installed using the usual `install.packages("mathml")`. At its present stage, it supports output in HTML, LaTeX, and Microsoft Word (via pandoc, MacFarlane 2022). The source code of the package is found at https://github.com/mgondan/mathml.

6 Appendix: Customizing the package

6.1 Implementation details

For convenience, the translation of the R expressions is achieved through a Prolog interpreter provided by another R package rolog (Gondan 2022). If a version of SWI-Prolog (Wielemaker et al. 2012) is found on the system, rolog connects to it. Alternatively, the SWI-Prolog runtime libraries can be conveniently accessed by installing the R package rswipl (Gondan 2023). Prolog is a classical logic programming language with many applications in expert systems, computer linguistics and symbolic artificial intelligence. The strength of Prolog lies in its concise representation of facts and rules for knowledge and grammar, as well as its efficient built-in search engine for closed world domains. Whereas Prolog is weak in statistical computation, but strong in symbolic manipulation, the converse may be said for the R language. rolog bridges this gap by providing an interface to a SWI-Prolog distribution (Wielemaker et al. 2012) in R. The communication between the two systems is mainly in the form of queries from R to Prolog, but two Prolog functions allow ring back and evaluation of terms in R.

The proper term for a Prolog "function" is predicate, and it is typically written with name and arity (i.e., number of arguments), separated by a forward slash. Thus, at the Prolog end, the predicate `math/2` translates the representation of the R call `pbinom(K, N, Pi)` into a more general representation of an R function `fn/2` with the name `P_Bi`, one argument `X =< K`, and the two parameters `N` and `Pi`, as shown below.

```
math(pbinom(K, N, Pi), M)
 => M = fn(subscript('P', "Bi"), (['X' =< K] ; [N, Pi])).
```

`math/2` operates like a macro that translates one mathematical element (here, `pbinom(K, N, Pi)`) to another mathematical element, namely `fn(Name, (Args ; Pars))`. The low-level predicate `ml/3` is used to convert these basic elements to MathML.

```
ml(fn(Name, (Args ; Pars)), M, Flags)
 => ml(Name, N, Flags),
    ml(paren(list(op(;), [list(op(','), Args), list(op(','), Pars)])), X, Flags),
    M = mrow([N, mo(&(af)), X]).
```

The relevant rule for `ml/3` builds the MathML entity `mrow([N, mo(&(af)), X])`, with `N` representing the name of the function and `X` its arguments and parameters enclosed in parentheses. A corresponding rule `jax/3` does the same for MathJax/LaTeX. A list of flags can be used for context-sensitive translation (see, e.g., the section on errors above).

Several ways exist for translating new R terms to their mathematical representation. We have already seen above how to use "hooks" to translate long variable names from R to compact mathematical signs, as well as functions such as cumulative probabilities $P(X \leq k)$ to different representations like $\sum_{i=0}^{k} P(X = i)$. Obviously, the hooks require that there already exists a rule to translate the target representation into MathML and MathJax.

In this appendix we describe a few more ways to extend the set of translations according to a user's needs. As stated in the background section, the Prolog end provides two classes of rules for translation, macros `math/2,3,4` mirroring the R hooks mentioned above, and the low-level predicates `ml/3` and `jax/3` that create proper MathML and LaTeX terms.

6.2 Linear models

To render the model equation of a linear model such as `lm(EOT ~ T0 + Therapy, data=d)` in mathematical form (see also Anderson, Heiss, and Sumners 2023), it is sufficient to map the `Formula` in `lm(Formula, Data)` to its respective equation. This can be done in two ways, using either the hooks described above, or a new `math/2` macro at the Prolog end.

```
hook(lm(.Formula, .Data), .Formula)
```

The hook is simple, but is a bit limited because only R's tilde-form of linear models is shown, and it only works for a call with exactly two arguments.

Below is an example of how to build a linear equation of the form $Y = b_0 + b_1 X_1 + ...$ using the Prolog macros from mathml.

```
math_hook(LM, M) :-
    compound(LM),
    LM =.. [lm, ~(Y, Sum) | _Tail],
    summands(Sum, Predictors),
    findall(subscript(b, X) * X, member(X, Predictors), Terms),
    summands(Model, Terms),
    M = (Y == subscript(b, 0) + Model + epsilon).
```

The predicate `summands/2` unpacks an expression `A + B + C` to a list `[C, B, A]` and vice-versa (see the file `lm.pl` for details).

```
rolog::consult(system.file(file.path("pl", "lm.pl"), package="mathml"))

term <- quote(lm(EOT ~ T0 + Therapy, data=d, na.action=na.fail))
math(term)
```

$$EOT = b_0 + b_{T0} T0 + b_{Therapy} Therapy + \epsilon$$

6.3 n-th root

Base R does not provide a function like `cuberoot(x)` or `nthroot(x, n)`, and the present package does not support the respective representation. To obtain a cube root, a programmer would typically type `x^(1/3)` or better `x^{1/3}` (see the practice section why the curly brace is preferred in an exponent), resulting in $x^{1/3}$ which may still not match everyone's taste. Here we describe the steps needed to represent the n-th root as $\sqrt[n]{x}$.

We assume that `nthroot(x, n)` is available in the current namespace (manually defined, or from R package pracma, Borchers 2022), so that the names of the arguments and their order are accessible to `canonical()` if needed. As we can see below, mathml uses a default representation `name(arguments)` for such unknown functions.

```
nthroot <- function(x, n)
  x^{1L/n}

term <- canonical(quote(nthroot(n=3L, 2L)))
math(term)
```

$$nthroot(2,3)$$

A proper MathML term is obtained by `mlx/3` (the x in mlx indicates that it is an extension and is prioritized over the default ml/3 rules). `mlx/3` recursively invokes `ml/3` for translating the function arguments X and N, and then constructs the correct MathML entity `<mroot>...</mroot>`.

```
mlx(nthroot(X, N), M, Flags) :-
    ml(X, X1, Flags),
    ml(N, N1, Flags),
    M = mroot([X1, N1]).
```

The explicit unification `M = ...` in the last line serves to avoid clutter in the head of `mlx/3`. The Prolog file `nthroot.pl` also includes the respective rule for LaTeX and can be consulted from the package folder via the underlying package rolog.

```
rolog::consult(system.file(file.path("pl", "nthroot.pl"), package="mathml"))

term <- quote(nthroot(a * (b + c), 3L)^2L)
math(term)
```

$$\left[\sqrt[3]{a \cdot (b+c)}\right]^2$$

```
term <- quote(a^(1L/3L) + a^{1L/3L} + a^(1.0/3L))
math(term)
```

$$\sqrt[3]{a} + a^{1/3} + a^{(1.00/3)}$$

The file `nthroot.pl` includes three more statements `precx/3` and `parenx/3`, as well as a `math_hook/2` macro. The first sets the operator precedence of the cubic root above the power, thereby putting a parentheses around nthroot in $(\sqrt[3]{\ldots})^2$. The second tells the system to increase the counter of the parentheses below the root, such that the outer parenthesis becomes a square bracket.

The last rule maps powers like `a^(1L/3L)` to `nthroot/3`, as shown in the first summand. Of course, mathml is not a proper computer algebra system. As is illustrated by the other terms in the sum, such macros are limited to purely syntactical matching, and terms like `a^{1L/3L}` with the curly brace or `a^(1.0/3L)` with a floating point number in the numerator are not detected.

7 Acknowledgment

Supported by the Erasmus+ program of the European Commission (2019-1-EE01-KA203-051708).

References

Allaire, J. J., Y. Xie, C. Dervieux, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, et al. 2023. *Rmarkdown: Dynamic Documents for R*. https://github.com/rstudio/rmarkdown.

Anderson, D., A. Heiss, and J. Sumners. 2023. *Equatiomatic: Transform Models into 'LaTeX' Equations*.

Borchers, H. W. 2022. *pracma: Practical Numerical Math Functions*. https://CRAN.R-project.org/package=pracma.

Capretto, T. 2023. *latex2r: Translate Latex Formulas to R Code*. https://github.com/tomicapretto/latex2r.

Chang, W., J. Cheng, J. J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. 2022. *Shiny: Web Application Framework for R*. https://CRAN.R-project.org/package=shiny.

Gondan, M. 2022. *rolog: Query SWI-Prolog from R*. https://github.com/mgondan/rolog.

———. 2023. *rswipl: Embed SWI-Prolog*. https://CRAN.R-project.org/package=rswipl.

Green, T. R. G. 1977. "Conditional Program Statements and Their Comprehensibility to Professional Programmers." *Journal of Occupational Psychology* 50: 93–109.

MacFarlane, J. 2022. *Pandoc: A Universal Document Converter*.

Miller, J. O. 1986. "Timecourse of Coactivation in Bimodal Divided Attention." *Perception & Psychophysics* 40: 331–43.

Murrell, P., and R. Ihaka. 2000. "An Approach to Providing Mathematical Annotation in Plots." *Journal of Computational and Graphical Statistics* 9: 582–99.

R Core Team. 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Ratcliff, R., P. L. Smith, S. D. Brown, and G. McKoon. 2016. "Diffusion Decision Model: Current Issues and History." *Trends in Cognitive Sciences* 20: 260–81.

Robertson, W., J. Wright, F. Mittelbach, and U. Fischer. 2021. *Breqn: Automatic Line Breaking of Displayed Equations*. https://www.ctan.org/pkg/breqn.

Sarkar, D., and K. Hornik. 2022. *Enhancements to HTML Documentation*. https://blog.r-project.org/2022/04/08/enhancements-to-html-documentation/index.html.

Schwarz, W. 1994. "Diffusion, Superposition, and the Redundant-Targets Effect." *Journal of Mathematical Psychology* 38: 504–20.

Viechtbauer, W. 2022. *mathjaxr: Using 'Mathjax' in Rd Files*. https://CRAN.R-project.org/package=mathjaxr.

Wickham, H. 2019. *Advanced R*. Cambridge: Chapman; Hall/CRC.

Wielemaker, J., T. Schrijvers, M. Triska, and T. Lager. 2012. "SWI-Prolog." *Theory and Practice of Logic Programming* 12 (1-2): 67–96.

Xie, Y. 2023. *knitr: A General-Purpose Package for Dynamic Report Generation in R*. https://yihui.org/knitr/.

Xie, Y., J. J. Allaire, and G. Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown.

Xie, Y., C. Dervieux, and E. Riederer. 2020. *R Markdown Cookbook*. Cambridge: Chapman; Hall/CRC.

8 References

Matthias Gondan
Universität Innsbruck
Department of Psychology
Innsbruck, Austria
https://www.uibk.ac.at/psychologie/mitarbeiter/gondan-rochon/index.html.en
ORCiD: 0000-0001-9974-0057
Matthias.Gondan-Rochon@uibk.ac.at

Irene Alfarone
Universität Innsbruck
Department of Psychology
Innsbruck, Austria
https://www.uibk.ac.at/psychologie/mitarbeiter/alfarone/index.html.en
ORCiD: 0000-0002-8409-8900
Irene.Alfarone@uibk.ac.at