An Approach to Providing Mathematical Annotation in Plots

Paul MURRELL and Ross IHAKA

A simple method for providing mathematical annotation of plots produced with the R environment is described. Although the implementation is specific to R, a similar method could be used in any environment which uses an expression-based command interface and provides a basic quoting mechanism.

Key Words: Equations; Labels; Typesetting

1. INTRODUCTION

Providing mathematical annotation in plots can be important when those plots are to be used to present the results of a data analysis. In most statistical systems, there is no general mechanism for rendering mathematical annotation and it must be added using additional software. While this can be a useful approach, it makes it difficult to do things like include mathematical annotation in an axis label or to locate a mathematical annotation at a particular data location on a graph (e.g., when providing a label for a data symbol). Some systems do provide facilities within normal text annotation which can be useful for mathematical expressions (e.g. superscripts and subscripts), but do not provide sufficient support for general mathematical equations.

The graphics system GR-Z, which was used as a foundation for graphics capabilities in the S statistical system (Becker, Chambers, and Wilks 1988), did provide support for general mathematical annotation (Becker and Chambers 1976). However, these GR-Z capabilities were not made accessible via the normal S syntax and so were not available to most users.

This article describes an approach that has been implemented in the R data analysis environment (Ihaka and Gentleman 1996), which bears a close similarity to S. The approach partitions the problem into two subproblems. The first is that of describing the structure of the mathematical annotation; the second of rendering that description into graphics on the page.

Paul Murrell is Lecturer, and Ross Ihaka is Senior Lecturer, Department of Statistics, University of Auckland, Private Bag 92019, Auckland, New Zealand (E-mail: paul@stat.auckland.ac.nz; ihaka@stat.auckland.ac.nz).

2. DESCRIBING MATHEMATICAL ANNOTATION

Computations in R are carried out by typing *expressions* to an interpreter which parses and executes them. The expressions accepted by the interpreter are specified in infix notation. For example, the expression `f(a,b)` indicates that the computation should apply the procedure `f` to the arguments `a` and `b`. In addition, there is certain amount of syntactic sugar which permits binary operators to be typed in the form `a+b`, in addition to their infix form `"+"(a,b)`.

The expressions used to specify computations in R are similar to traditional mathematical notation. Because of this, they can also be used to describe the structure of mathematical expressions to be used in graphical annotation. The following table shows a number of simple mathematical expressions and how they can be represented with R expressions.

Mathematics	*R*
$(x+y)/2$	`(x + y)/2`
$\sqrt{x^2+y^2}$	`sqrt(x^2 + y^2)`
$\sum x_i$	`sum(x[i])`

In order to use R expressions to describe the layout of mathematical expressions, it is necessary to have a means of indicating that the expression is not to be evaluated, but rather treated as a symbolic description. This can be achieved in R by using the *quoting* mechanism provided by the `expression()` function. The R expression

`expression(`$expr_1, expr_2, \ldots$`)`

returns the unevaluated arguments $expr_1$, $expr_2$, ... in a *vector* of mode *expression.*

The R graphics code has been modified so that in any place where a vector of text annotation strings could be passed as an argument to a graphics function, it is also possible to pass a vector of expressions. When this happens, the expressions are processed by the underlying equation renderer rather than by the graphics code which draws text strings.

When the renderer receives an expression, it draws a representation of it on the graphics device using its built-in set of layout rules. Some of these rules specify special actions for mathematical operators such as summation and integration, and others the way in which accents and radicals should be handled. Additional rules provide simple text translations such as the translation of `alpha` into the corresponding Greek symbol α.

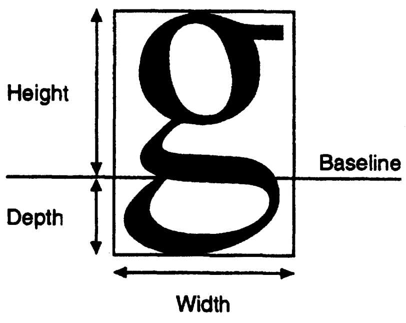

Figure 1. The Bounding Box of a Character.

3. THE EQUATION RENDERER

3.1 BASIC TYPESETTING

During typesetting, characters are placed on the page using the information contained in each character's bounding box. The bounding box for an unslanted character roughly corresponds to a rectangle which circumscribes the region in which the character will be drawn and is described by a width, height, depth, and baseline (see Figure 1). The bounding-box information is dependent upon the text's font.

It is not necessary that all of a character fits inside its bounding box (slanted or italic characters often protrude beyond the sides of their bounding boxes), but the dimensions of the box are designed so that characters may be positioned correctly relative to each other simply by aligning their bounding boxes. For example, Figure 2 shows how the text "hey there" is typeset by placing the bounding boxes of the characters side by side and vertically aligning the baselines of the boxes.

In normal typesetting, this is all that is required. In mathematical typesetting, however, the process is more complicated with characters potentially translated both vertically and horizontally. Despite this, the rules for laying out the elements of a mathematical expression are still based upon the positioning of the bounding boxes for the elements of the expression.

For example, the rules for typesetting a fraction (e.g., $\frac{1}{2}$) require placing the baseline of the numerator a sufficient distance above the horizontal line, placing the baseline of the denominator a sufficient distance below the horizontal line and aligning the bounding boxes of the numerator and denominator so that they are centered on the middle of the

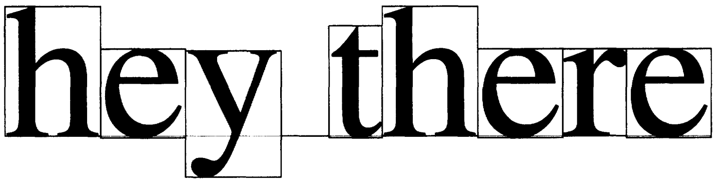

Figure 2. Typesetting for the Text "hey there".

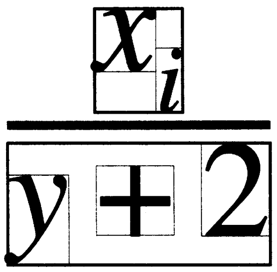

Figure 3. Typesetting for the Expression `over(x[i], y + 2)`.

horizontal line.

Because expression elements can themselves be expressions, equation layout rules must apply recursively. Since the rules are based on bounding boxes, this presents no difficulty so long as we can generate the bounding box for an arbitrary expression. This is achieved simply by generating a box which completely encapsulates the typeset expression. For example, Figure 3 shows how the expression `over(x[i], y + 2)` is typeset; the numerator and denominator are themselves mathematical expressions with their own bounding boxes.

The bounding box for this element is the smallest box which contains the whole unit.

3.2 PROCESSING MATHEMATICAL EXPRESSIONS

R expressions are represented internally as *lists* of the form:

$$(operator\ \ operand_1 operand_2 \ldots)$$

with the operands being either further lists (expressions) or *atoms*. For example, the expression `over(x[i], y + 2)` is stored internally as:

```
(over ([ x i) (+ y 2))
```

Here the the first operand, `([ x i)`, is a list, but its operands (`x` and `i`) are atoms.

Atoms can be converted into an individual characters, which have bounding boxes. For example, `x` converts to the character "x" and `2` converts to the character "2". Some special atoms are defined to allow for special sorts of characters. For example, `alpha` converts to the character "α". So the bounding box for an atom is just the bounding box for the character that corresponds to that atom.

Lists are more complicated. Consider a list which has just atoms for its operands; for example, `([ x i)` or `(+ y 2)`. These lists can be converted into a set of characters; the characters from the atoms plus possibly a character for the operator as well. For example, `([ x i)` converts to just "x" and "i", but `(+ y 2)` converts to "y", "+", and "2". The list of characters can be converted into a list of bounding boxes which are combined in the typesetting process according to the rules specified for the particular operator.

When an operand is a list, the processes of arranging bounding boxes and typesetting become intertwined. Consider the list `(over ([ x i) (+ y 2))` in which both operands are lists. To generate a bounding box for the operand `([ x i)`, the operand must be typeset (i.e., the list must be converted into bounding boxes and those bounding boxes must be arranged appropriately). Once the operand has been typeset, a bounding box for the operand can be generated by drawing a box around the typeset operand.

The complete process for this example looks something like:

```
    generate bounding box for "x"
    generate bounding box for "i"
    typeset ([ x i)
generate bounding box for ([ x i)
    generate bounding box for "y"
    generate bounding box for "+"
    generate bounding box for "2"
    typeset (+ y 2)
generate bounding box for (+ y 2)
typeset (over ([ x i) (+ y 2))
```

3.3 LAYOUT RULES

The layout rules used in our implementation are based upon those described in appendix G of *The TEXbook* (Knuth 1984). Our rules are a little simpler than those of TEX because R expressions contain much more structure than the stream of tokens that TEX rules are designed for.

TEX uses three font *sizes* in equations: text size, script size, and scriptscript size. It also uses a variety of *styles* in equations. The styles are

display style	(for formulas on a line by themselves)
text style	(for formulas embedded in text)
script style	(for subscripts and superscripts)
scriptscript style	(for second order subscripts and superscripts)

together with four "cramped" variants of these styles. Display and text styles are rendered in text size, script style in script size, and scriptscript style in scriptscript size.

The layout rule for a particular kind of element determines the layout and spacing between sub elements as well as the styles in which those sub elements are to be drawn; both of these being dependent on the current style. Although this sounds complicated, the rules are actually quite simple to implement (but beware that there is at least one typographical error in their statement in the *TEXbook*).

The layout rules also depend on a number of font-dependent parameters which must be tuned to achieve the best layout. Some guidance on suitable values was found in the METAFONT sources for the TEX computer modern fonts, but some experimentation was necessary, to obtain acceptable values.

3.4 DEVICE-SPECIFIC TYPESETTING

If a device is to be used to provide mathematical annotation it must possess a minimal level of functionality. Most importantly, it should provide access to a special symbol font and also have accurate font metric information available (i.e., information about the bounding boxes of the characters in the font). Ideally, the font information should include width, height, and depth measures for each individual character.

4. CAPABILITIES

The operators currently supported (for mathematical typesetting) by our implementation are shown in the following.

4.1 ARITHMETIC OPERATORS

The standard arithmetic operators for addition, subtraction, division, and multiplication are available. Note that the multiplication operator "*" provides the standard typesetting convention of indicating multiplication by juxtaposition.

Input expression	*Output*
`+ x`	$+x$
`- y`	$-y$
`x + y`	$x+y$
`x - y`	$x-y$
`x / y`	x/y
`x * y`	xy
`x %+-% y`	$x \pm y$
`x %/% y`	$x \div y$
`x %*% y`	$x \times y$

4.2 SUBSCRIPTS AND SUPERSCRIPTS

Subscripting is obtained by using the subsetting operator "[" and exponentiation by the exponentiation operator "^".

Input expression	*Output*
`x[i]`	x_i
`x^y`	x^y
`x[y]^z`	x_y^z

4.3 FRACTIONS

The `over` and `frac` command can be used to create fractions.

Input expression	*Output*
`over(x, y)`	$\frac{x}{y}$
`frac(x, y)`	$\frac{x}{y}$

The space above and below the rule separating the numerator and denominator is dependent on the style (display, text, script, or scriptscript) being used.

The `atop` command can be used to obtain one value above another, with no separating rule. This can be used, for example, to produce binomial coefficients.

Input expression	*Output*
`atop(x, y)`	${x \atop y}$

4.4 JUXTAPOSITION

There are several mechanisms which can be used to juxtapose symbols. As noted previously, the * operator can be used to juxtapose expressions. The `paste` function has the same effect, but can take an arbitrary number of arguments.

Input expression	*Output*
`paste(x, y)`	xy
`x * y`	xy

4.5 SPACING

The ~ operator can be used to separate expressions with one or more spaces.

Input expression	*Output*
`x ~ y`	$x\ y$
`x ~~ y`	$x\ \ y$
`x ~~~ y`	$x\ \ \ y$

Some care must be taken when using ~ in conjunction with other operators. Consider the expression `a~+~b`. This is parsed by R as `a~{+~b}`. This means that + is treated as a unary operator, leading to differing amounts of space to its left and right.

4.6 Comma Separated Lists

Comma separated lists can be obtained using the `list` function.

Input expression	*Output*
`list(x, y, z)`	x, y, z

4.7 Ellipsis

The symbol `...` can be used to generate ellipsis in a number of contexts.

Input expression	*Output*
`list(x[1], ... , x[n])`	$x_1, \ldots, x_n$
`x[1] + ... + x[n]`	$x_1 + \cdots + x_n$

To obtain explicit vertical positioning of the ellipsis, use `ldots` or `cdots`.

Input expression	*Output*
`list(x[1], cdots , x[n])`	$x_1, \cdots, x_n$
`x[1] + ldots + x[n]`	$x_1 + \ldots + x_n$

4.8 Radicals

R supports general radicals. These are produced with the `sqrt` function. Square roots are produced with a single argument, and general roots with two arguments.

Input expression	*Output*
`sqrt(x)`	$\sqrt{x}$
`sqrt(x, y)`	$\sqrt[y]{x}$

4.9 Relations

A number of relational operators are available.

Input expression	*Output*
`x == y`	$x = y$
`x != y`	$x \neq y$
`x < y`	$x < y$
`x <= y`	$x \leq y$
`x >= y`	$x \geq y$
`x > y`	$x > y$
`x %==% y`	$x \equiv y$
`x %~~% y`	$x \approx y$
`x %=~% y`	$x \cong y$
`x %prop% y`	$x \propto y$

4.10 SET RELATIONS

A number of relations from set theory are available.

Input expression	*Output*
`A %supset% B`	$A \supset B$
`A %supseteq% B`	$A \supseteq B$
`A %notsubset% B`	$A \not\subset B$
`A %subset% B`	$A \subset B$
`A %subseteq% B`	$A \subseteq B$
`a %in% A`	$a \in A$
`a %notin% B`	$a \notin B$

4.11 ARROWS

A wide collection of arrows are available.

Input expression	*Output*
`x %<->% y`	$x \leftrightarrow y$
`x %<-% y`	$x \leftarrow y$
`x %up% y`	$x \uparrow y$
`x %->% y`	$x \rightarrow y$
`x %down% y`	$x \downarrow y$
`x %<=>% y`	$x \Leftrightarrow y$
`x %<=% y`	$x \Leftarrow y$
`x %dblup% y`	$x \Uparrow y$
`x %=>% y`	$x \Rightarrow y$
`x %dbldown% y`	$x \Downarrow y$

4.12 Accents

Tilde, hat, and bar accents are available. There are are two forms of these accents—those drawn with accents taken from the fonts,

Input expression	*Output*
`tilde(x)`	$\tilde{x}$
`hat(x)`	$\hat{x}$

and those which are drawn with graphics commands

Input expression	*Output*
`widetilde(x)`	$\widetilde{xy}$
`widehat(x)`	$\widehat{xy}$
`bar(x)`	$\overline{xy}$

4.13 Symbolic Names

It is possible to obtain the letters of the Greek alphabet and a number of other useful symbols by spelling out their names.

Input expression	*Output*
`Alpha - Omega`	$A - \Omega$
`alpha - omega`	$\alpha - \omega$
`infinity`	∞
`32 * degree`	32°
`60 * minute`	$60'$
`30 * second`	$30''$

4.14 OPERATORS

Summation, product, and integration operators are available via functions that have a special meaning within mathematical expressions. Limits can be specified as optional arguments.

Input expression	*Output*
`sum(x[i])`	$\sum x_i$
`sum(x[i], i==1, n)`	$\sum_{i=1}^{n} x_i$
`product(plain(P)(X==x), x)`	$\prod_x \mathrm{P}(X = x)$
`integral(f(x)dx, a, b)`	$\int_a^b f(x)dx$

The functions `union` and `intersection` can be used to produce the corresponding set theoretic operators.

Input expression	*Output*
`union(A[i], i==1, n)`	$\bigcup_{i=1}^{n} A_i$
`intersection(A[i], i==1, n)`	$\bigcap_{i=1}^{n} A_i$

The functions `lim`, `liminf`, `limsup`, `inf`, `sup`, `min`, and `max` are available to produce limits, maxima, and minima.

Input expression	*Output*
`lim(f(x), x %->% 0)`	$\lim_{x \to 0} f(x)$
`min(g(x), x>=0)`	$\min_{x \geq 0} g(x)$

4.15 GROUPING

Visible grouping is available using parentheses and invisible grouping using braces. Visible grouping is useful for making explicit the order of evaluation of an expression. Invisible grouping is useful for clarifying which operands belong to an operator.

Input expression	*Output*
`(x + y)*z`	$(x+y)z$
`x^y + z`	x^y+z
`x^(y + z)`	$x^{(y+z)}$
`x^y + z`	x^{y+z}

More general grouping can be obtained with the `group` and `bgroup` functions. These functions have three arguments: a left delimiter, the body, and a right delimiter. The delimiters are chosen from `lfloor`, `rfloor`, `lceil`, `rceil`, "|", "||", "(", ")", "[", "]", "", "" and ".". The delimiters all create the obvious symbol as a delimiter, with the exception of ".", which produces nothing.

The function `group` produces fixed-size (small) delimiters and `bgroup` produces variable-size (large) delimiters.

Input expression	*Output*
`group("(",list(a,b),"]")`	$(a,b]$
`bgroup("(",atop(n,k),")")`	$\binom{n}{k}$

4.16 ABSOLUTE VALUE

The function `abs` provides a shorthand way of generating absolute values.

Input expression	*Output*
`abs(x)`	$\|x\|$

4.17 TYPEFACE

The typeface can be explicitly set using a set of functions which have special meaning within mathematical expressions.

Input expression	*Output*
`plain(x)`	x
`bold(x)`	**x**
`italic(x)`	*x*
`bolditalic(x)`	***x***

4.18 STYLE CHANGES

As mentioned previously, it is possible to obtain text in *display*, *text*, *script* and *scriptscript* styles.

Input expression	*Output*
`displaystyle(x)`	$\displaystyle x$
`textstyle(x)`	$\textstyle x$
`scriptstyle(x)`	$\scriptstyle x$
`scriptscript(x)`	$\scriptscriptstyle x$

In addition to font size, these styles also affect spacing as well as the positioning of superscripts and subscripts, and the placing of operator limits.

4.19 PHANTOMS

To help fine tune the positioning of annotation, we have introduced TeX's notion of "phantom" commands. These commands produce space, but place no ink on the page. `phantom` renders the space which would be occupied by its argument and `vphantom` renders just the vertical space occupied

Input expression	*Output*
`x + phantom(0) + y`	$x + + y$
`over(1, vphantom(0))`	$x + \frac{1}{\vphantom{0}}$

5. EXAMPLES

In this section we present a number of examples to show what is possible with the R graphics system, and also how the system can be used in a practical case.

5.1 EXAMPLE 1. GENERAL CAPABILITIES

This example shows the variety of effects which are possible with the annotation system. These are presented in Figure 4.

Elements (1) and (2) of figure 4 show typical uses of annotation in plot labels. The first of these was produced with the expression

```
    expression("Area"~(km^2))
```

showing the use of superscripting. The second was produced with the

```
    expression("Temperature"~(degree*K))
```

and shows how the special symbol denoting *degrees* can be produced (symbols for *minutes* and *seconds* can be produced in a similar fashion).

Elements (3) and (4) of Figure 4 show how to produce the kind of symbolism used in chemistry. They also illustrate the fine tuning of subscript placement. The expression

(1) Area (km^2)

(2) Temperature $(^\circ\mathrm{K})$

(3) $\mathrm{Fe}_2^{+2}\mathrm{Cr}_2\mathrm{O}_4$

(4) $\mathrm{Fe}_2^{+2}\mathrm{Cr}_2\mathrm{O}_4$

(5) $\frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$

(6) $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-x^2/2} dx$

(7) $\bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i$

(8) $s^2 = \frac{1}{N-1} \sum_{i=1}^{N} \left(X_i - \bar{X}\right)^2$

Figure 4. Examples Showing a Variety of Annotation Effects.

```
expression(Fe[2]^+2*Cr[2]*O[4])
```

was used to produce element (3). Close inspection of element (3) shows that the subscripts are not all placed at the same height (because of the presence of a superscript). If this unevenness is unacceptable, it is possible to introduce *phantom* superscripts which lower the other subscripts to the same level.

```
expression(Fe[2]^+2*Cr[2]^vphantom(0)*O[4]^vphantom(0))
```

Note that Knuth (1984, p. 179) describes this as a monstrosity, but it does have the virtue of simplicity.

The remaining elements of Figure 4 show a variety of equations which arise in statistics. The primary point of interest in these elements is the use of == to denote equality. Limited experience shows that this is a source of error (it is just too tempting to type = instead), but the fact that we are using R syntax makes it the best alternative.

```
expression(italic(over(1, sqrt(2*pi)*sigma) ~
           e^-{(x - mu)^2/2*sigma^2}))

expression(italic(Phi(x) == over(1, sqrt(2*pi))
           * integral(e^{-x^2/2}*dx, -infinity, x)))

expression(italic(bar(X) == over(1, N) ~
           sum(X[i], i == 1, N)))

expression(italic(s^2 == over(1, N-1) ~
           sum(bgroup("(", X[i] - bar(X), ")")^2,
               i == 1, N)))
```

We do not wish to indicate that this level of complexity *should* occur in graphical annotation, but rather to show that it is possible on those occasions that it is needed.

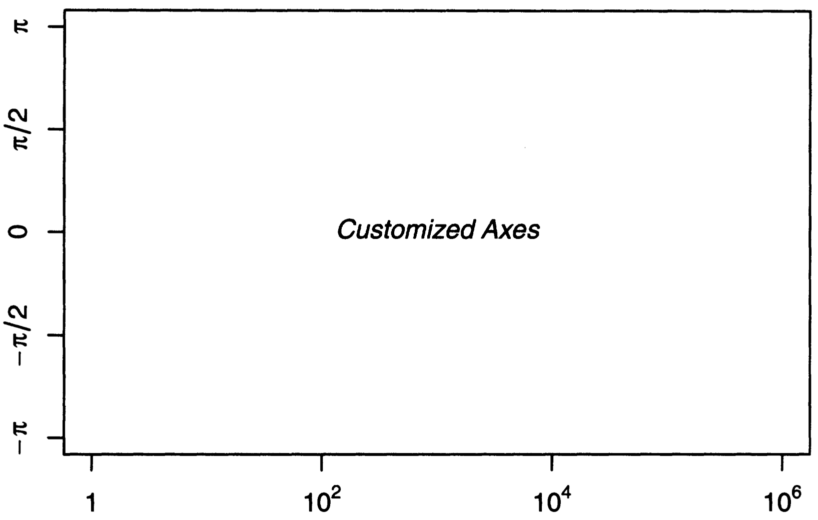

Figure 5. Creating Customized Axes With Suitable Annotation.

5.2 Example 2. Customized Axes

The standard axes produced by R are rendered with simple numerical labels at the plot tick marks. Often the results are quite unattractive and unsuitable for presentation purposes. This can be alleviated by having more general annotation at the tick marks.

The R function which draws axes has an argument which specifies a vector of (text) labels which are to appear at the tick marks. If instead a vector of expressions is used, it is possible to obtain more general annotation at the tick marks.

In this example (Figure 5), we show how to use the general facility to produce attractive annotation for logarithmic and circular axes.

```
plot(c(1, 1e6), c(-pi, pi), type="n",
     axes="f", log="x")
axis(1, at=c(1,1e2,1e4,1e6),
     labels=expression(1, 10^2, 10^4, 10^6))
axis(2, at=c(-pi, -pi/2, 0, pi/2, pi),
     labels=expression(-pi, -pi/2, 0, pi/2, pi))
text(1e3, 0, expression(italic("Customized Axes")))
box()
```

5.3 Example 3. A Real Example

Our next example shows a careful reproduction of a figure from Brillinger (1981). The figure shows an estimate of the normalized spectral measure for the series of mean monthly sunspot numbers for the period 1750–1965. The reproduction is shown in Figure 6. It contains mathematical elements in the labels on the x and y axes, and requires customization of the axis tick marks as well as nonstandard placement of the labeling relative to the axes.

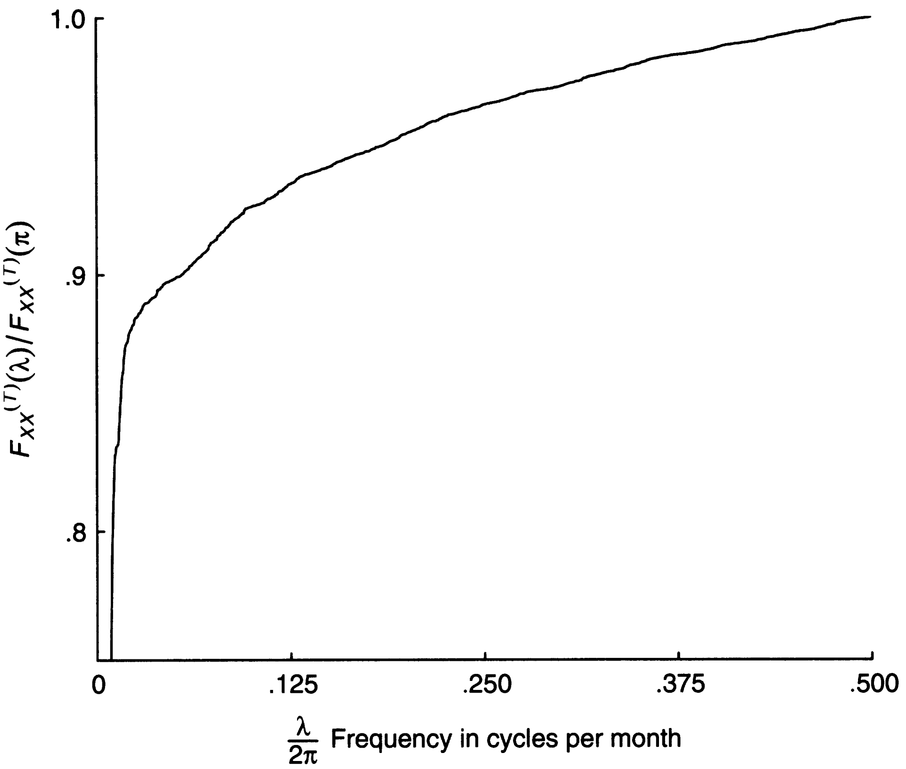

Figure 6. A Reproduction of Figure 5.10.1 From Brillinger (1981).

We will assume that the spectral measure can be estimated and stored in the variable `sm` by the statement,

```
sm <- spectral.measure(sunspots, normalize=TRUE)
```

The plot is then produced by customizing the graphics parameters and rendering it in steps.

```
par(tck=0.02, mgp=c(2, 0.25, 0), las=1)
plot(sm, type="l", ylim=c(0.75, 1.0),
    xaxs="i", yaxs="i", axes=F,
    xlab=expression(textstyle(over(displaystyle(lambda),
                                   displaystyle(2*pi)))
                    ~~"Frequency in cycles per month"),
    ylab=expression(italic({{F[XX]}^(T)}(lambda)/
                    {{F[XX]}^(T)}(pi)))
)
axis(1, at=c(0, .125, .25, .375, .5),
     lab=c("0", ".125", ".250", ".375", ".500"))
axis(2, at=c(.8, .9, 1),
     lab=c(".8", ".9", "1.0"))
box(bty="l")
```

The label on the x axis presents some difficulty because the ratio $\lambda/2\pi$ is rendered in a nonstandard way. In TEX terms, the ratio is typeset in *textstyle*, which means that the denominator and numerator are rendered quite close to the rule which appears between them. Normally this would mean that the numerator and denominator would be produced in *scriptstyle*, and would appear in a smaller font. To get the appearance of the original plot, this default choice of *scriptstyle* must be overridden. We have chosen to use *displaystyle*, but *textstyle* would produce the same results.

The label on the y axis is relatively straightforward to produce, but some care is required because the superscripts are not rendered directly above the subscripts. We have used `{...}` grouping to resolve these problems.

6. PROBLEMS

A number of problems surfaced during the development and testing of this system. Some of these are generic, and would apply to any system for rendering mathematics on graphics devices. Others are specific to the particular system we have developed.

A major difficulty is caused by differences between the fonts on different devices and the inconsistency of font metric information. This causes discrepancies in graphical output. This effect is particularly obvious in mathematical annotation. A more severe problem arises because some devices do not provide sufficient font metric information to allow high quality rendering. For example, Microsoft Windows only provides the height and depth for an entire font, not for each character in the font.

Another problem is that some devices have only a restricted range of font sizes. This has proved to be a problem with our device driver for the X Window System. Newer versions of X allow the use of scalable fonts, and these produce better results.

These problems are of a generic nature. A different kind of problem arises from the fact that we have used R command syntax to specify mathematical expressions. The R syntax provides convenient ways of expressing many mathematical operations; subscripts and superscripts are conveniently and efficiently specified using R's subsetting and exponentiation syntax. However, some very convenient options are not available because of their special meaning within normal R expressions. The most obvious of these is the necessity to specify an equality relation using `==` rather than `=` because of the latter's special meaning within R expressions. A similar problem is encountered when trying to specify a comma separated lists.

7. CONCLUSIONS

Using R language expressions to describe mathematical expressions together with a mathematical expression renderer, which can use such expressions as a description of what is to appear in a plot, provides a simple way to produce mathematical annotation in graphs. While the capabilities of the renderer are limited (compared to a system such as TEX) it provides most of the capabilities commonly required when producing statistical graphics.

The TEX layout rules provided good guidance when implementing this facility. Any-one interested in implementing something similar would do well to make a close study of Knuth's *The TEXbook.*

8. SOFTWARE

Versions of R containing the capabilities described in this paper are freely available and can be obtained from CRAN sites and in particular from StatLib. (The Comprehensive R Archive Network maintained by Kurt Hornik and Friedrich Leisch of the Technical University of Vienna in Austria.) The Cran master site URL is http://www.ci.tuwien.ac.at/R and the StatLib URL is http://lib.stat.cmu.edu/R/CRAN/.

[Received June 1998. Revised January 1999.]

REFERENCES

Becker, R. A., and Chambers, J. M. (1976), "GR-Z Display of Mathematical Expressions," unpublished technical memorandum, Bell Laboratories.

Becker, R. A., Chambers, J. M., and Wilks, A. R. (1988), *The New S Language*, Pacific Grove, CA: Wadsworth & Brooks/Cole.

Brillinger, D. R. (1981), *Time Series: Data Analysis and Theory* (2nd ed.), San Francisco: Holden-Day.

Ihaka, R., and Gentleman, R. (1996), "R: A Language for Data Analysis and Graphics," *Journal of Computational and Graphical Statistics*, 5, 299–314.

Knuth, D. E. (1984), *The TEXbook*, London: Addison-Wesley.