

J. Terwilliger and J. Ott. *Handbook of Human Genetic Linkage*. The Johns Hopkins University Press, Baltimore, 2001.

Jing Hua Zhao
MRC Epidemiology Unit
jinghua.zhao@mrc-epid.ac.uk

Non-Standard Fonts in PostScript and PDF Graphics

by Paul Murrell and Brian Ripley

Introduction

By default, all of the text produced by R graphics devices uses a sans-serif font *family* (e.g., Helvetica), and in a PostScript or PDF plot only text in Western European languages has been handled until recently. This article describes several ways in which the range of font families has been extended for R 2.3.0.

It has been possible to change the overall font family that is used to draw text within a plot (Murrell, 2004), but the choice of font family for PDF and PostScript devices was largely restricted to the set of Adobe Core 14 fonts—Helvetica (4 faces), Times (4 faces), Courier (4 faces), Symbol, and ZapfDingbats (see Table 1). As from R 2.3.0 much more choice is available.

The examples in this article are written following the conventions needed for a modern Unix-alike operating system such as Linux, but will work with minor changes (for example to the locale names) on most other R platforms including Windows.

A little terminology will be helpful. A *character* is an abstract concept, and a *glyph* is a visual representation of a character. Characters can have more than one representation, for example crossed and uncrossed sevens, and the variants on Greek letters much loved by mathematicians (see `?plotmath`). This matters as East Asian languages may write the same character in different ways.¹

Minor conveniences

The default font family on a PDF or PostScript device can be controlled using the `family` argument. For example, the expression `pdf(family="Times")` opens a PDF device with the default font family set to Times (a serif font). This is the simplest way to select a font family, if the same font (possibly in different faces, e.g. bold) is to be used for all of the text in a plot.

Prior to R 2.3.0, only a fixed set of font families could be specified via the `family` argument (mostly

related to the Adobe Core 14 fonts), but it is now possible to specify any *valid* font family as the default. For example, the expression `pdf(family="mono")` opens a PDF device with the default font family set to be a mono-spaced font (by default Courier on PDF and PostScript devices).

The section on “Font databases” later in this article will clarify what constitutes a “valid” font family.

Changing font family in-line

While it has been possible for some time to be able to change the font family within a plot (e.g., have the title in Times and the remaining labels in Helvetica), in the traditional graphics system, this change could only be made via the `par()` function.

It is now also possible to specify the `family` argument in low-level traditional graphics functions. For example, if the default font family was sans-serif, adding a label to a plot with a serif font used to require code of the form ...

```
> op <- par(family="serif")
> text("A label")
> par(op) # restore the setting
```

... but now the same thing can be achieved by the following code.

```
> text("A label", family="serif")
```

Internationalization

As Martin Maechler likes to say, R has ‘for ever’ supported ISO Latin 1, the character set of the major² Western European languages. Further internationalization features were introduced into R in version 2.1.0 (Ripley, 2005), which allowed users in non-Western-European locales to use variable names and produce output in their native language.

There was good support for graphical devices on Windows, and as from R 2.3.0 on NT-based versions of Windows it is even possible to change to another language and output in that language.

The support for the `X11()` graphics device was as good as the X server could provide, often limited by

¹http://en.wikipedia.org/wiki/Han_unification

²The definition is rather circular, as those are the languages written in Latin-1. Icelandic is the most prominent exception.

Table 1: Predefined font families for the PDF and PostScript devices which cover most of the Adobe Core 14 fonts. The Adobe Core fonts are usually available with Adobe (Acrobat) Reader and in PostScript printers and viewers. The ZapfDingbats font is always included in PDF files produced by R, being used to draw small circles.

"Helvetica"	"Times"	"Courier"
Helvetica	Times	Courier
Helvetica-Bold	Times-Bold	Courier-Bold
Helvetica-Oblique	Times-Italic	Courier-Oblique
Helvetica-BoldOblique	Times-BoldItalic	Courier-BoldOblique
Symbol	Symbol	Symbol

the availability of glyphs unless extra fonts were installed and are selected.

These features did not at first include graphical output on PDF or PostScript devices, but this has been much improved in R 2.3.0, with support for many Eastern European and Asian locales.

European and Cyrillic fonts

R now automatically selects an appropriate encoding for PDF and PostScript output based on your locale. In multi-byte locales such as those using UTF-8, an appropriate single-byte encoding is chosen, so now the PDF and PostScript devices allow encodings other than ISO Latin 1.

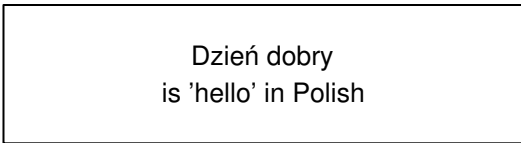
For example, consider the character `ń`, which is needed in a Polish locale to write 'hello'. This character is part of the ISO Latin 2 character set. For demonstration purposes, we will temporarily switch to a Polish locale using the following call:

```
> Sys.setlocale(category="LC_CTYPE",
  locale="pl_PL.utf8")
```

Now when we start a PDF or PostScript device, it automatically selects an ISO Latin 2 encoding so that the appropriate characters are available.

To show an example for users who do not have a Polish keyboard, we will use an explicit Unicode escape sequence to enter the character we want. This will only work in UTF-8 locales in R 2.3.0.³

```
> library(grid)
> pdf("polish.pdf", width=3, height=0.8)
> grid.text(" Dzień\xu0144 dobry
  is 'hello' in Polish ")
> dev.off()
```



Dzień dobry
is 'hello' in Polish

On systems without access to UTF-8 locales, we can achieve the same thing by switching to a Latin-2 locale: a Windows⁴ user could write

```
> Sys.setlocale(category="LC_CTYPE",
  locale="polish")
> grid.text(" Dzień\xf1 dobry
  is 'hello' in Polish ")
```

It is still possible to explicitly specify an encoding when starting the pdf device if the automated selection is not correct on your platform.

This process ought to work well whenever the language you want can be written in a single-byte locale: this includes all the Western and Eastern European, Cyrillic and Greek languages written in ISO 8859-1 (Western), -2 (Eastern), -5 (Cyrillic), -7 (Greek), -13 (Baltic rim), 15 (Western with Euro) and KOI8-R, KOI8-U and the equivalent Microsoft codepages.

Installed fonts

By specifying the correct encoding you can create a PDF or PostScript *file* that contains the correct *description* of the characters you want, but if you actually want to view or print that text, you also need to have an appropriate font installed on your system that contains the relevant characters.

In the above example we were fortunate because the default Helvetica font does include the glyph `ń`. In general, you may have to specify an appropriate font (and ensure that it is installed on your system!). The section “Font databases” will describe the issue of specifying new fonts in detail.

Even if the glyph is known to R, it may not be known to your viewer. For example, to view non-Western European languages with Adobe Reader you may need to install a Central European font pack.⁵

A word of warning: the Adobe Core fonts are often replaced by clones. This happens both in

³Unicode escape sequences will work in all locales in R 2.3.1 on platforms which support multi-byte character sets.

⁴NT4, 2000 or later, or (untested) a Polish-language version of 95/98/ME.

⁵<http://www.adobe.com/products/acrobat/acrrasianfontpack.html>

PostScript viewers (ghostscript uses the equivalent URW fonts) and PDF viewers (recent versions of Adobe Reader use multiple-master fonts) as well as in some printers. So it can be hard to tell if you have exceeded the set of glyphs that can reasonably be assumed to be always available. If you use anything beyond ISO Latin 1, it is safest to embed the fonts (see below). However, for local use where up-to-date URW fonts can be assumed the coverage is much wider including Eastern European and Cyrillic glyphs.

CJK fonts

Some support has also been added for locales with very large character sets, such as the Chinese, Japanese, and Korean (CJK) ideographic languages. On PDF and PostScript devices, it is now possible to select from a small set of font families covering many of the glyphs used in writing those languages.

There are tens of thousands of (originally) Chinese characters which are common to Chinese (where they are called *hanzi*), Japanese (*kanji*), and Korean (*hanja*). However, different glyphs may be used, for example in Simplified Chinese (used in most of PR China and in Singapore) and Traditional Chinese (used in Taiwan and some of PR China).

For example, the following code selects one of the CJK font families to display 'hello' in (Traditional) Chinese.

```
> pdf("chinese.pdf", width=3, height=1)
> grid.text("\u4F60\u597D", y=2/3,
            gp=gpar(fontfamily="CNS1"))
> grid.text(
  "is 'hello' in (Traditional) Chinese",
  y=1/3)
> dev.off()
```



Note that we select the CJK font family only to display Chinese: although these font families can display English, they do so poorly. You should switch back to a more standard font such as Helvetica for producing Latin characters.

The appropriate CJK fonts include non-Chinese glyphs, for example Hiragana and Katakana for use in Japanese and Hangeul for use in Korean.

Again, R does not check whether the fonts are properly installed on your system. For example, in order to view the file `chinese.pdf` with Adobe Reader, you will have to download the Traditional Chinese font pack.⁶

The CJK font families available in R have been chosen because they are widely (or freely) available for common print or viewing technology. Table 2 lists the font families and where they can be obtained.

The CJK fonts can be used in mathematical formulas (see `?plotmath`), but the metric information on the sizes of characters is not as good as for Type 1 fonts, so the result may not be aesthetically pleasing.

Note that in this example we did not need to select an appropriate UTF-8 locale. It will in fact work in any UTF-8 locale, and also in the older multi-byte locales used on Unix-alikes (such as EUC-JP) and in the double-byte locales used on Windows for the CJK languages.

Font databases

In the previous two sections we mentioned that, in addition to having the correct encoding, you may also have to specify an appropriate font in order to get all of the characters that you need. Here is an example where that becomes necessary: we want to write 'hello' in Russian and the Cyrillic characters are not included in Helvetica; so we need to specify a font that does contain Cyrillic characters. (A simpler workaround would be to use the URWHelvetica family which does since 2002 include Cyrillic glyphs, but the versions installed with ghostscript on Windows are from 2000. So we need to illustrate a more general procedure.)

Again, we will pretend to be Russian temporarily by setting the locale for the R session:

```
> Sys.setlocale(category="LC_CTYPE",
                locale="ru_RU.utf8")
```

Now the single-byte encoding is automatically set to ISO Latin 5, but we must also specify a font that contains Cyrillic characters. We will generate a new one using the function `Type1Font()` and specifying the font name and a path to metric (`.afm`) files for the font.

The font used in this example is from the PSCyr font pack⁷. For this demonstration, the files have only been downloaded to a local directory (not formally installed) to keep the path manageable⁸. The following command creates a Type 1 font object.

⁶<http://www.adobe.com/products/acrobat/acrrasianfontpack.html>

⁷<ftp://ftp.vsu.ru/pub/tex/font-packs/ps Cyr>

⁸The location of the installed `.afm` files will vary depending on your system. Most of the fonts used in this article are from TeX packages, so on a Linux system they might reside in `/usr/share/texmf/fonts/afm/public/<fontpackname>`.

Table 2: The CJK fonts available for use in R. These were collected from examples found on various systems, and suggestions of Ei-ji Nakama.

Font family name in the font database	PostScript font name	PDF font name
"Japan1"	HeiseiKakuGo-W5: a Linotype Japanese printer font.	KozMinPro-Regular-Acro: from the Adobe Reader 7.0 Japanese Font Pack
"Japan1HeiMin"	HeiseiMin-W3: a Linotype Japanese printer font.	HeiseiMin-W3-Acro: a version of the PostScript font, from the Adobe Reader 4.0 Japanese Font Pack.
"Japan1GothicBBB"	GothicBBB-Medium: a Japanese-market PostScript printer font.	GothicBBB-Medium
"Japan1Ryumin"	Ryumin-Light: a Japanese-market PostScript printer font.	Ryumin-Light
"Korea1"	Baekmuk-Batang: a TrueType font found on some Linux systems, and from ftp://ftp.mizi.com/pub/baekmuk/ .	HYSMyeongJoStd-Medium-Acro: from the Adobe Reader 7.0 Korean Font Pack
"Korea1deb"	Batang-Regular: a TrueType font found on some Linux systems, probably the same as Baekmuk-Batang.	HYGothic-Medium-Acro: from the Adobe Reader 4.0 Korean Font Pack.
"CNS1" (Traditional Chinese)	MOESung-Regular: from Ken Lunde's CJKV resources; can be installed for use with <code>ghostscript</code>	MSungStd-Light-Acro: from the Adobe Reader 7.0 Traditional Chinese Font Pack.
"GB1" (Simplified Chinese)	BousungEG-Light-GB: a TrueType font found on some Linux systems, and from ftp://ftp.gnu.org/pub/non-gnu/chinese-fonts-truetype/ .	STSong-Light-Acro: from the Adobe Reader 7.0 Simplified Chinese Font Pack.

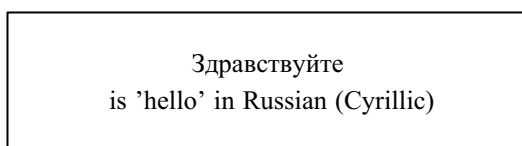
```
> RU <- Type1Font(
  "TimesNewRomanPSMT-Regular",
  c("fonts/afm/public/pscyr/times.afm",
    "fonts/afm/public/pscyr/timesbd.afm",
    "fonts/afm/public/pscyr/timesi.afm",
    "fonts/afm/public/pscyr/timesbi.afm"))
```

This next command registers the Type 1 font in the font database for PDF devices. The database records all valid font family names for use with PDF devices. After this, for the duration of the current R session, the font family "RU" can be used as a font family name in graphics functions.

```
> pdfFonts(RU=RU)
```

Now we can use this font family like any other and to produce Cyrillic characters (we do not assume a Russian keyboard so fake typing the Cyrillic keys by specifying the appropriate Unicode values⁹).

```
> pdf("russian.pdf", width=3, height=0.8)
> grid.text("\u0417\u0430\u0440\u0434\u043e
  \u0430\u0432\u0435\u0441\u0442\u0432\u0435
  \u0443\u0439\u0442\u0435
  is 'hello' in Russian (Cyrillic)",
  gp=gpar(fontfamily="RU"))
> dev.off()
```



A separate function, `postscriptFonts()`, is provided for querying and modifying the font database for PostScript devices, and there is a `CIDFont()` function for defining new CJK (CID-keyed) fonts¹⁰.

Embedding fonts

The Russian example above is not quite complete yet. If you try to view it with Adobe Reader, it will almost certainly not look as we intended¹¹. The problem is that Adobe Reader knows nothing about the PSCyr font so it substitutes a font it knows about, trying to make it look as close as possible to the font we've asked for.

By defining a Type 1 font object, we have given R enough information about the Cyrillic font to create

a PDF or PostScript file, but if we want to view that file or print it, we may also have to install the font for the viewing software or the printer. Furthermore, if we want to share the file with someone else (e.g., put it on the web), then that someone else may have to install the font.

The best solution to all of these problems is to embed (enough of) the font within the file itself. This means that the font becomes part of the document and viewing software or printers do not need any additional information to display or print the text. (Note that licence restrictions on the fonts may limit or prohibit this approach.)

The new R function `embedFonts()` performs this task (by calling `ghostscript`¹²).

Here is how to embed the PSCyr fonts for the `russian.pdf` plot; the file `russianembed.pdf` is the result. Because we have the fonts in a local directory, we must specify a font path so that `ghostscript` knows where to find the font.

```
> embedFonts("russian.pdf",
  outfile="russianembed.pdf",
  fontpaths="fonts/type1/public/pscyr")
```

For most of the examples in this article we have embedded the fonts using this function, which is why you can view all of these different character sets without having to download any extra fonts.

A Computer Modern fonts example

Computer Modern fonts are the fonts designed by Donald Knuth for the \TeX system (Knuth, 1984) and are the default fonts used in \LaTeX (Lamport, 1994).

For a long time, it has been possible to make use of Computer Modern fonts to produce PostScript plots specifically for inclusion in \LaTeX documents via a special "ComputerModern" font family (see `?postscript`). However, this mechanism does have some limitations: some characters are not available (e.g., less-than and greater-than signs, curly braces, and a number of mathematical symbols) and, more importantly, it does not work for PDF output. A simple way to avoid these restrictions is to use a different set of Computer Modern fonts that contain a more complete set of characters.

Gaining access to a wider set of Computer Modern characters can be achieved simply by using a version of the Computer Modern fonts that has

⁹This series of Unicode characters has been broken across lines for this article due to typesetting limitations; do *not* reproduce these line breaks if you want to try this code yourself. Also, as there are at least three incompatible single-byte encodings for Russian, we do not attempt to show how this might be done other than on a UTF-8 system.

¹⁰Defining a new CID-keyed font for use with PDF devices requires in-depth knowledge of Adobe font technology and the font itself, so is not recommended except perhaps for expert users.

¹¹What this actually looks like for you will depend on the fonts you have installed for Adobe Reader, however it is unlikely that you will have the PSCyr fonts so it should at least look different to the "correct" output; on a vanilla Linux system, the Cyrillic characters are all missing, and the Latin text is a serif font.

¹²`ghostscript` should be installed already on most Linux systems and is available as a standard download and install for Windows (<http://www.cs.wisc.edu/~ghost/>). You may need to tell R where `ghostscript` is installed *via* the environment variable `R_GSCMD`.

been reordered and regrouped for different encoding schemes. We will use two such fonts in this example.

The first set of fonts is the CM-LGC font pack¹³, which provides a Type 1 version of most of the Computer Modern fonts with an ISO Latin 1 encoding (e.g., a less-than sign will produce a less-than sign, *not* an upside-down exclamation mark as in \TeX). The second font is a special Computer Modern symbol font¹⁴ developed by Paul Murrell which covers almost all of the Adobe Symbol encoding (for producing mathematical formulas).

The following code demonstrates the use of the CM-LGC fonts and the special Computer Modern symbol font to produce a PDF format “plot” that includes mathematical symbols. The fonts have again been unpacked in a local directory.

First of all, we define a new Type 1 Font object for these fonts and register it with the PDF font database:

```
> CM <- Type1Font("CM",
  c(paste("cm-lgc/fonts/afm/public/cm-lgc/",
    c("fcmr8a.afm", "fcmr8a.afm",
      "fcmr18a.afm", "fcmr18a.afm"),
    sep=""),
    "cmsyase/cmsyase.afm"))
> pdfFonts(CM=CM)
```

Now we can use this font family in a PDF plot:

```
pdf("cm.pdf", width=3, height=3,
  family="CM")
# ... much drawing code omitted ...
dev.off()
```

Finally, we can embed these fonts in the document so that we can embed the plot in other documents (such as this article) for anyone to view or print.

```
> embedFonts("cm.pdf",
  outfile="cmembed.pdf",
  fontpaths=
  c("cm-lgc/fonts/type1/public/cm-lgc",
    "cmsyase"))
```

The final result is shown in Figure 1.

Summary

With R 2.3.0, it is now easier to produce text in PDF and PostScript plots using character sets other than English. There is built-in support for a wider range of encodings and fonts and there is improved support for making use of non-standard fonts in plots. There is also now a utility for embedding fonts so that plots can be more easily included in other documents and shared across systems.

We believe that it is now possible for users to produce high-quality graphical output in all human

languages with a sizeable number of R users. Anyone whose native language is not covered is invited to contribute suitable encoding files and locale mappings so our coverage can be extended.

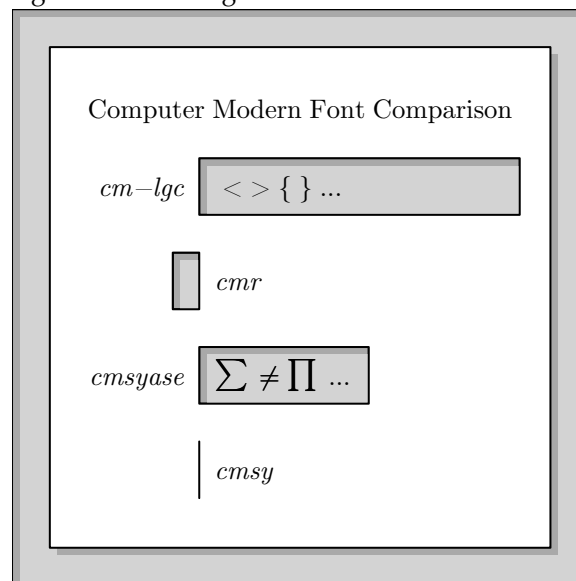


Figure 1: A “plot” that demonstrates the use of Computer Modern fonts in PDF output.

Acknowledgements

Ei-ji Nakama provided the initial patch of the C code for non-ISO Latin 1 encodings in multi-byte locales, and for CJK (CID-keyed) font support, and he and colleagues have patiently explained Japanese typography to us.

Bibliography

- D. Knuth. *The \TeX book*. Addison-Wesley, Reading, MA, 1984.
- L. Lamport. *\LaTeX : a document preparation system*. Addison-Wesley, Reading, MA, 1994.
- P. Murrell. Fonts, lines, and transparency in R graphics. *R News*, 4(2):5–9, September 2004. URL <http://CRAN.R-project.org/doc/Rnews/>.
- B. D. Ripley. Internationalization features of R 2.1.0. *R News*, 5(1):2–7, May 2005. URL <http://CRAN.R-project.org/doc/Rnews/>.

Paul Murrell
The University of Auckland, New Zealand
paul@stat.auckland.ac.nz

Brian D. Ripley
University of Oxford, UK
ripley@stats.ox.ac.uk

¹³<http://www.ctan.org/tex-archive/help/Catalogue/entries/cm-lgc.html>, and available as a package for some Linux systems.

¹⁴<http://www.stat.auckland.ac.nz/~paul/R/CM/CMR.html>

The doBy package

by Søren Højsgaard

This article is not about rocket science; in fact it is not about science at all. It is a description of yet another package with utility functions.

I have used R in connection with teaching generalized linear models and related topics to Ph.d. students within areas like agronomy, biology, and veterinary science at the Danish Institute of Agricultural Sciences.

These students, many of whom are familiar with the SAS system, have almost all come to appreciate R very quickly. However, they have also from time to time complained that certain standard tasks are hard to do in R – and certainly harder than in SAS. The **doBy** package is an attempt to make some of these standard tasks easier.

Airquality data

The presentation of the package is based on the `airquality` dataset which contains air quality measurements in New York, May to September 1973. (Note that months are coded as 5, ..., 9).

```
> head(airquality)
```

	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8.0	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
5	NA	NA	14.3	56	5	5
6	28	NA	14.9	66	5	6

The summaryBy function

With the summary procedure of SAS (PROC SUMMARY) one can easily calculate things like “the mean and variance of x for each combination of two factors A and B ”. To calculate mean and variance of Ozone and Wind for each combination of Month, do:

```
> summaryBy(Ozone + Wind ~ Month,
+ data = airquality, FUN = c(mean,
+ var), prefix = c("m",
+ "v"), na.rm = TRUE)
```

	Month	m.Ozone	m.Wind	v.Ozone	v.Wind
1	5	23.62	11.623	493.9	12.471
2	6	29.44	10.267	331.5	14.207
3	7	59.12	8.942	1000.8	9.217
4	8	59.96	8.794	1574.6	10.407
5	9	31.45	10.180	582.8	11.980

The result above can clearly be obtained in other ways. For example by using the `aggregate` function, the `summarize` function in the `Hmisc` package or by

```
> a <- by(airquality, airquality$Month,
+ function(d) {
+ c(mean(d[, c("Ozone",
+ "Wind")], na.rm = T),
+ diag(var(d[, c("Ozone",
+ "Wind")], na.rm = T)))
+ })
> do.call("rbind", a)
```

However, my students have found this somewhat cumbersome!

The orderBy function

Ordering (or sorting) a data frame is possible with the `orderBy` function. Suppose we want to order the dataframe by Temp and by Month (within Temp) and that the ordering should be decreasing. This can be achieved by:

```
> x <- orderBy(~Temp + Month,
+ data = airquality, decreasing = T)
```

The first lines of the result are:

	Ozone	Solar.R	Wind	Temp	Month	Day
120	76	203	9.7	97	8	28
122	84	237	6.3	96	8	30
121	118	225	2.3	94	8	29
123	85	188	6.3	94	8	31
126	73	183	2.8	93	9	3
127	91	189	4.6	93	9	4

Again, this can clearly be achieved in other ways, but presumably not with so few commands as above.

The splitBy function

Suppose we want to split data into a list of dataframes, e.g. one dataframe for each month. This can be achieved by:

```
> x <- splitBy(~Month, data = airquality)
```

Information about the grouping is stored as a dataframe in an attribute called `groupid`:

```
> attr(x, "groupid")
```

	Month
1	1
2	2
3	3
4	4
5	5