Improvements to the 'dvir' package

Alexander van der Voorn

Introduction

R has the ability to display mathematical equations in graphics using the "plotmath" feature. This provides us with most of the symbols and formatting, such as brackets and fractions, used for equations but is limited in its fonts. If we want these equations to be in a TeX or Tex-style font, there are several solutions (see Paul's "Introduction to 'dvir' package" report - can I reference that report?), depending on the context.

The 'dvir' package has been developed in response, allowing TeX equations and text to be overlaid on normal R graphics.

'dvir', in a simplified form, works as follows:

- `grid.latex()` is called with the TeX expression for the equation or text to be displayed on the R graphic
- `dvir` sends the expression to TeX (TeX Live in our case) which creates a corresponding DVI file for the expression
- `dvir` converts the DVI file from a binary file to a "readable" form
- Three sweeps of the converted DVI file are completed to extract necessary information about what to display:
  - Font sweep: The names of all fonts used are stored in the `fontconfig` file, to search for the necessary fonts using `kpsewhich`. This requires mapping the DVI fontname to an actual font file on the computer
  - Metric sweep: To determine the overall bounding box (size) of the expression to display. THis bounding box is used to create a grid viewport which can encompass all of the expression using the native DVI coordinates
  - Grid sweep: Convert all characters and symbols into (grid) "grobs" (grid graphical objects)
- These "grobs" are then displayed in the R graphics device

Code speed

One of the first things investigated in the package was the speed of running the code. Anecdotally, generating any R graphic with non-trivial LaTeXtext took a long time and as such improving the time efficiency of the package was desirous.

To tackle this, the first task was to profile the existing code to see where the time was being spent when the code ran. We used our two examples: the simple $x - \mu$ and our motivating example.

Using `profvis::profvis()` we were able to visually explore how the time was spent in these examples. In ten runs of the motivating example, the total average execution time was about 6269ms. Of this, about 3393ms on average, or 54%, was spent in calls to the `dvir` function `definePDFFont()`.

The purpose of `definePDFFont()` is to do a sweep of the DVI file from `grid.latex()` looking for all fonts required, before recording the font names in the font config file, searching the relevant directories for the font files and encoding the fonts. A variable `fonts` is saved with all this information. The further sweeps over the DVI file to determine the bounding box of each character and thus entire image, and to create grid grobs and viewports each redundantly called `definePDFFont()` rather than referring to the already existing variable `fonts` from the first sweep.

This was therefore an easy and quick win - simply changing the subsequent sweeps to ignore the font-defining op code and instead referring to the `fonts` variable created from the font sweep.

This resulted in fantastic savings. In 10 runs of the example after this change was made, the average total execution time was 3712ms. This is a reduction of 2557[ms, or 41%. The average time spent in `definePDFFont()` was just 1292ms, 62% less than before. A saving of nearly two thirds in the function is consistent with removing two of the three font sweeps.

Font support

An issue we discovered, somewhat unintentionally, was our examples were not rendering correctly on Ubuntu 20.04. Previously `dvir` had been created and tested only on Ubuntu 18.04 (is this correct?)

`dvir` on Ubuntu 20.04 did not reproduce our examples as expected - sometimes characters would display as a square with numbers inside (indicating there is no glyph for that given character in the incorrectly selected font), or the symbol was correct, but was the wrong font. It was discovered that the Computer Modern fonts weren't being used as the latest vesion of Pango (version 1.44 is used in Ubuntu 20.04) does not support True Type fonts. The `.pfb` Computer Modern fonts are on the computer, so Pango is specifically refusing to use them. I think the Cairo graphics device gave us the right characters, but wrong font, but pdf device gave us just the squares.

Potential solutions to this:

- Get an OpenType CM font (Like BaKoMa font) - at least CMSY font This sort of worked, but the encodings of the font were wrong. Our mu came up as theta, for example So we need to find CM font with correct encodings, or use something like fontforge (fonttools/ttx?) to manually edit encodings. Paul found some other potential fonts that had the right encodings If the above worked, then we'd need a way to install or edit the fonts on a user's computer.
- Create 18.04 Docker Image, which dvir would call when it needs to do its TeX processing Requires Docker on user's computer, for them to create a container using the image (which may be run automatically through dvir) Have created a docker image, but docker file needs tidying (installs some things that are needed for Paul's report, not dvir itself)

Paul's work at the end of 2020 has solved this problem in the the latest version of `dvir`: https://www.stat.auckland.ac.nz/~paul/Reports/dvir/type1/dvir-type1.html