GaRGoyLe tutorial (version 1.0)

31-5-2013

Contents

1 Introduction

GaRGoyLe is just a catchy name for a mapping system based on GRASS, R, GMT and LaTeX.

It is a set of R functions which main objective is to create "fancy" maps in PDF format from GRASS raster or vector layers. It has two main functions:

- `GMT_map`. This function uses GMT to generate a PDF file from a set of GRASS display commands. This file contains the main map image and a legend.
- `latex_map`. This function uses LaTeX to add other cartographic elements (title, a scale bar, text boxes with metadata, other images, etc.) to the map.

One of the objectives of these functions is to be easy to use. This aim is achieved by using default values for most of their parameters. In this sense, GaRGoyLe functions try to be like GRASS modules or R functions.

1.1 Requirements

GaRGoyLe uses R, GRASS, GMT and LaTeX, these programs should be installed in the system; however, only R and GMT are really critical. You can use any PNG file with a map instead of a set of GRASS commands to create the base map. LaTeXis needed by `latex_map` but you can obtain simple map just with `GMT_map`. The R package `spgrass6` is needed to access the GRASS geometric information (`GRASS region`) and the R package `classInt` is needed by the functions `create_palette` and `create_palette_v`. The next LaTeX packages are needed by the function `latex_map`:

- `a0poster`
- `sciposter`
- `textpos`
- `fancyhdr`
- `keystroke`
- `fancybox`
- `fancyvrb`
- `setspace`
- `tikz`

Some of the functions that interact with GRASS vector layers are conceived assuming that the PostgreSQL driver is being used to store tables. If it is not the case they may not work properly.

1.2 Notes

- R has to be invoked from within a GRASS session.
- The function `GMT_map` is a wrapper to GMT. It calculate map positions, like GMT, in centimeters from the paper bottom left corner and increasing X to the right and Y to the top.
- The function `map_latex` creates a .tex file and compiles it. So, it calculates map positions like LaTeX, in centimeters from the paper top left corner and increasing X to the right and Y to the bottom.
- Along this tutorial, a GRASS map is called a layer, the term map will refer to the PDF files generated by some of the functions in the package.

1.3 Available functions

1.3.1 Basic functions

There are just two basic functions which allow the user to create maps:

- `GMT_map`, generate a PDF file with a basic map created with GMT from GRASS layers including legends, grid and annotations (if so decided by the user).
- `latex_map`, generate and compile a LaTeX file that includes the PDF file generated by `GMT_map`. It allows the user to include auxiliary elements to the map, like scale bars, numerical scale, titles, logos, and several text blocks with auxiliary information.

1.3.2 Functions to create legends

The next three functions are used to create legends from the RGB coded color specifications of any GRASS layer:

- `rast2leg`, create a GMT legend file from information contained in the GRASS `cats` and `colr` files.
- `vect2leg`, create a GMT legend file from two colums of the table linked to a vector layer. One of the columns contains the labels and the other the RGB coded color specifications.
- `vectthematic2leg`, creates a GMT legend file using the GRASS module `d.vect.thematic`.

1.3.3 Functions to manage color palettes

The next four functions create or change the color palette of a GRASS layer.

- `create_palette`, generates a GRASS palette file for a raster layer with a style determined by the user. The available styles are `fixed`, `sd`, `equal`, `pretty`, `quantile`, `kmeans`, `hclust`, `bclust`, `fisher` and `jenks`. See the manual of the R package `classInt` (http://cran.r-project.org/web/packages/classInt/classInt.pdf) to obtain more information.
- `create_palette_v`, generates a color palette in the column `grassrgb` of the table linked to a GRASS vector layer. It works in a very similar way to `create_palette`. It can also create a legend file with the palette created.
- `put_palette`, generates a GRASS palette file for a raster layer introducing vectors of values and RGB codes.
- `randomgrassrgb`, fills the `grassrgb` column in the table linked to a vector layer using random colors.

1.3.4 A function to create text labels

- `grasspoints2GMTtext`, generates a GMT labels file from a GRASS points, or centroids, vector layer. These labels can be later included in a `GMT_map` call, using the parameter textfile, to be shown in the resulting map.

1.3.5 Internal functions that can be of use

The next functions are called by the previous ones but in some cases they can be useful.

- `colr_grass`, returns the file path of a GRASS `colr` file. It looks first at the user's mapset and secondly at the mapset where the layer is located. If no `colr` file is found it returns NULL.
- `catfile_grass`, returns the path to the `cats` file of a GRASS raster layer.
- `dbname`, returns the database which contains the table linked to a vector layer.
- `tablename`, returns the table linked to a vector layer.
- `query_grass`, queries the table linked to a vector map via `v.db.connect`.
- `gscale`, generates LaTeX code to draw a graphical scale on the map using the LaTeX package `tikz`.
- `distance`, calculates the distance between two points.
- `selmon`, returns the selected GRASS display monitor

1.3.6 Other utilities

- `papers()` simply returns a data frame with the international standard paper sizes (a0, a1, etc.) with their size in centimeters. It is helpful when trying to decide the best place to draw the cartographic elements in the map.

1.4 Location North Carolina

The next examples are based on the North Carolina dataset available form the GRASS web site:

http://grass.fbk.eu/download/data.php

1.5 Beginning

After entering in the GRASS environment, the best option is to open a new terminal to execute R. As the terminal is open inside the GRASS environment, it will share all the GRASS environment variables allowing R to read and write GRASS files.

```
gnome-terminal -e R
```

Now, you have a terminal to introduce grass commands and another to execute R code. So, bear in mind in which one you have to write the next examples. In order to distinguish them, GRASS commands will be prompted by `#`.

The next think to do is to load the required libraries and GaRGoyLe:

```
library(classInt)
library(spgrass6)
library(GaRGoyLe)
```

2 Function GMT_map

The next examples use the North Carolina State DEM (elev_state_500m). First of all we have to set an appropriate region to have a good zoom:

```
# g.region n=358000 s=-15500 w=90500 e=969000
# d.erase
```

The easiest way to generate a map is to execute GMT_map with a set of GRASS display orders in the display parameter:

```
GMT_map(display="d.rast elev_state_500m")
```

or just:

```
GMT_map("d.rast elev_state_500m")
```

The result appears on Figure 1, top left map.

The default behavior is to create a 15 centimeters wide map, in the low left corner of an a4 paper, and without legend. The paper orientation (portrait or landscape) is decided, by the function, taking into account the length/width ratio. The grid size is also automatically calculated (although it does not mean it is a sensible size).

The function returns a list with three elements, a vector with the GMT commands used to create the map, the denominator of the scale map and the width in centimeters of the map main box. This allow the user to save the commands and execute later with any small change needed, or to include the width as a variable in further calculations.

The function will also send to the standard output the width in centimeters and the scale of the created map. This is useful to establish an adjust numeric scale for the map if needed.

2.1 Options

This simple behaviour can be modified by adding several parameters:

- **Give a name to the output files**

  GMT_map generates two files, a PNG file containing the result of the GRASS display orders processed by the GRASS PNG driver and a PDF file containing the map. The default names of both files are "outmap.png" and "outmap.pdf". The parameter output allows the user to name those files.

  ```
  GMT_map("d.rast elev_state_500m",output="mymap")
  ```

  The result will be a PNG file called mymap.png and a PDF map called mymap.pdf.

- **Map creation without visualizing it**

  The option graph determines if the PDF map is to be displayed after being created. The default value is TRUE, but it can be changed:

  ```
  GMT_map("d.rast elev_state_500m",graph=F)
  ```

- **Modifying the grid**

  A compromise grid size is calculated by the function GMT_map but it may easily become too dense (as in the previous example) or too loose, the user can establish a given size (in meters) with the parameter grid.

  ```
  GMT_map("d.rast elev_state_500m",grid=200000)
  ```

  The result appears on Figure 1 top right map.

- **Eliminate the grid**

  If `grid`=0 it disappears:

  ```
  GMT_map("d.rast elev_state_500m",grid=0)
  ```

  The result appears on Figure 1 middle left map.

- **Modifying the annotations**

  The coordinates that appear at the bottom and left of the page is controlled by the parameter `anots`. By default, annotations are matched to the grid. The next option will make a denser grid but with less annotations:

  ```
  GMT_map("d.rast elev_state_500m",grid=100000,anots=200000)
  ```

  The result appears on figure 1 middle right map.

  If you want to just eliminate the annotations simply include the parameter `anots=0` in the function.

  ```
  GMT_map("d.rast elev_state_500m",grid=200000,anots=0)
  ```

- **Changing the map size**

  ```
  GMT_map("d.rast elev_state_500m",grid=200000,width=20)
  ```

  Instead of specifying the map size we can use the parameter `scale` to pass its denominator:

  ```
  GMT_map("d.rast elev_state_500m",grid=200000,scale=4000000)
  ```

  Regardless of the parameter you use, the text output of the function will include the map scale and width in centimeters.

- **Changing the map position on the paper**

  Parameters `dxcm` and `dycm` allow you to place the map in the output paper defining the position of the lower left corner of the map in relation to the lower left corner of the paper. The default values of parameters `dxcm` and `dycm` are both 2.5.

  In the next example, we are going to change the size and position of the map

  ```
  GMT_map("d.rast elev_state_500m",output="elevation",scale=4000000,
  grid=100000,dxcm=4,dycm=5)
  ```

  The result appears on Figure 1 bottom left map.

  If the map contains annotations, the lower left corner of the map is not the lower left corner of the box containing the image, but the position of the annotations located below and to the left of the box.

- **Forcing paper orientation**

  GMT_map calculates by default the proper orientation of the map from the height/width ratio. But we can force the orientation to portrait with:

  ```
  GMT_map("d.rast elev_state_500m",ori="p")
  ```

  similarly we can force the paper orientation to landscape:

  ```
  GMT_mapa("d.rast elev_state_500m",ori="l")
  ```

- **Adding more layers using GRASS visualization modules**

  You can include any display command in the map. In the next example we first define a set of commands and then use them to create the map.

  ```
  display="d.rast elev_state_500m;d.vect boundary_county type=boundary;
  d.vect nc_state type=boundary width=5"
  ```

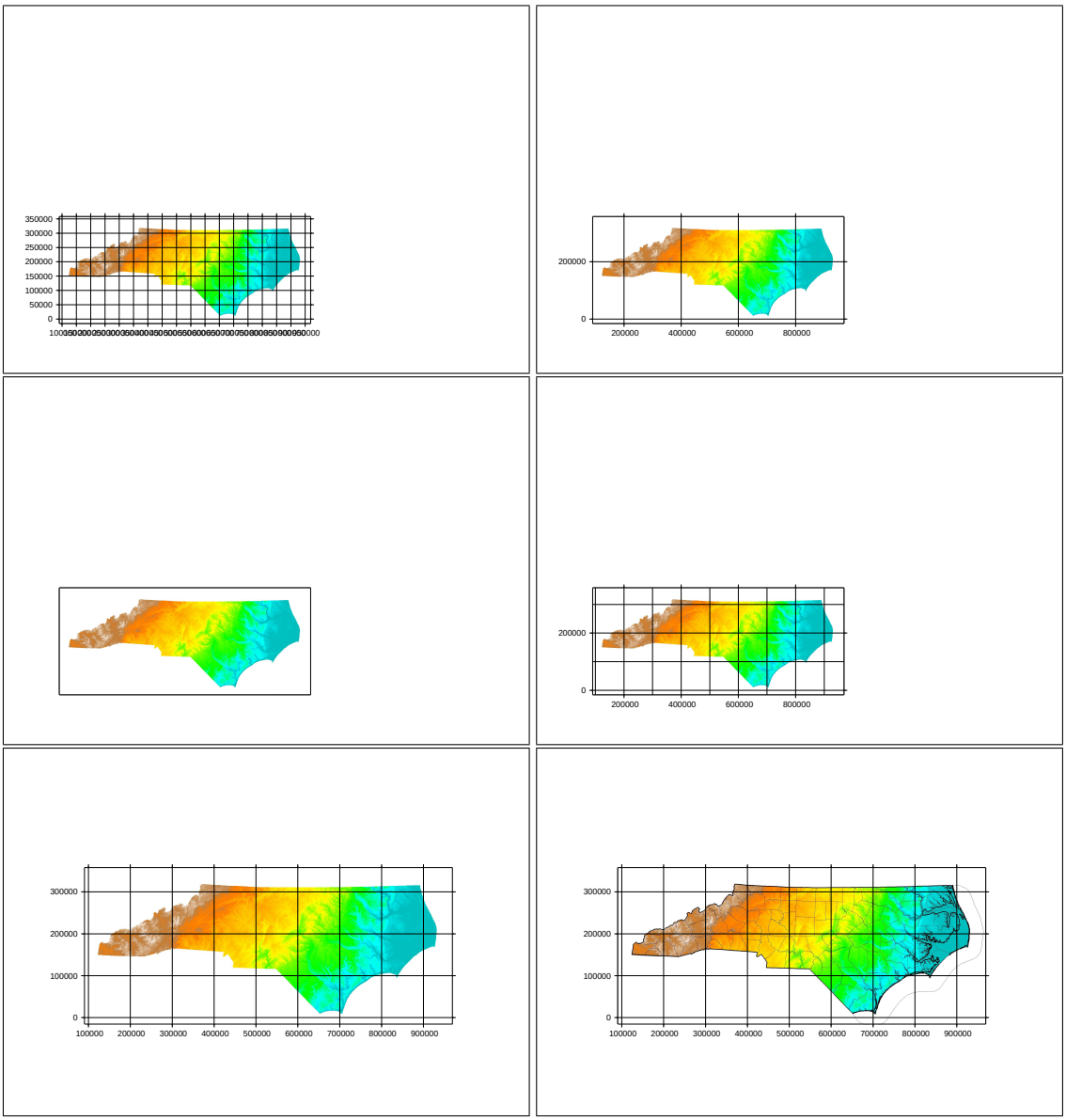

Figure 1: Basic usage of GMT_map. All maps are a4 size in origin and were reduced to a width of 7 cm

```
GMT_map(display,output="elevation",scale=4000000
,grid=100000,dxcm=4,dycm=5)
```

The result appears on figure 1 bottom right map.

- **Do not create the PNG file**

  The slower part of the generation of a map is to create the PNG file with the GRASS PNG driver. However, the final design is usually achieved by trial and error after several tests in which the PNG image is always the same. To prevent the function re-create the PNG file each time it runs we can use the `png=F` option to

skip this step. If so, the function assumes that there is a PNG file with the same name that we have passed with the parameter output or, if you have not used this option, a file called `outmap.png`

- **Paper sizes**

  If you want to change the size of the paper where the map is to be drawn you have to add the `paper` parameter:

  ```
  GMT_map("d.rast elev_state_500m",output="elevation",scale=2000000,
  grid=100000,dxcm=4,dycm=5,paper="A2")
  ```

  Of course, a larger paper allows us for a more detailed scale.

- **Mapping the display monitor**

  If, instead of a set of GRASS commands, you pass "mon" as `display`, `GMT_map` will map the contents of the selected GRASS display monitor.

  ```
  GMT_map(display="mon",dxcm=3,dycm=4,width=20)
  ```

  It is important to remember that `GMT_map` starts and selects the PNG display driver, *deselecting* the GRASS display monitor previously selected. You need to select it back after using `GMT_map`.

- **Working without GRASS**

  Altho `GaRGoyLe` is chiefly conceived to map GRASS layers, it can be used with other programs as long as you can save a display of our data in PNG format. If you pass the name of your file in the `display` parameter, and use the `coords` parameter to pass the coordinates of the map hedges in a four elements vector `c(west, east, sout, north)`, you will obtain the same result as with GRASS data.

  ```
  GMT_map("mymap.png",dxcm=3,dycm=4,width=20,
  coords=c(75000,1032000,-27500,375000))
  ```

- **Other options**

  Several GMT options can be passed to the function with the `gmtset` parameter. You can browse all the (about 100) GMT options in the manual of the `gmtdefaults` GMT module[1]. Some of them can be used with `GMT_map`. You can include them in the `gmtset` parameter of the function or save the GMT script returned by `GMT_map` and add them as `gmtset` calls.

  The most handy is probably `ANNOT_FONT_SIZE_PRIMARY`. The default option is 14 pt, a bit too large, to choose smaller annotations try adding `gmtset="ANNOT_FONT_SIZE_PRIMARY=10"` to the `GMT_map` function.

  Another useful option is `Y_AXIS_TYPE=ver_text` that will rotate the y axis annotations.

  ```
  GMT_map("mymap.png",dxcm=2,dycm=4,coords=c(75000,1032000,-27500,375000),
  width=20,gmtset="ANNOT_FONT_SIZE=8 Y_AXIS_TYPE=ver_text")
  ```

  GMT saves all its parameters in a file called `.gmtdefaults4` maintaining them active between different calls to GMT commands. One of the first things `GMT_map` does is to erase the `.gmtdefaults4` file to avoid the parameters passed to a function to affect the next one.

[1] http://gmt.soest.hawaii.edu/gmt/html/man/gmtdefaults.html

3 Legends

GMT_map will not create a legend unless it is said to. The procedure depends on whether the map contains raster or vector data and whether the variable to be shown in the map is qualitative or quantitative. The four cases can be summarized as follows:

- **Raster quantitative variable**, the function GMT_map includes a parameter (Zscale) to indicate the name of the GRASS raster layer that will be used to generate a color scale to be shown in the map.
- **Raster qualitative variable**, the function rast2leg creates a GMT legend file from the cats and colr files of a GRASS raster layer.
- **Vector qualitative variable**, the function vect2leg creates a GMT legend file from two columns in the table linked to the vector map. One of the columns contains the color palette and the other one the text labels.
- **Vector quantitative variable**, the function vectthematic2leg creates a GMT legend file from an execution of the GRASS module d.vect.thematic. This function cal also change the color palette.

3.1 Color scales for quantitative variables in raster layers (color ramps)

The legend for a quantitative variable layer consists of a GMT color scale. The parameter scaleZ allows the user to pass the name of the GRASS raster layer from which the scale is computed:

```
GMT_map("d.rast elev_state_500m",scale=4000000,grid=100000,
dxcm=4,dycm=5,scaleZ="elev_state_500m",scaleunit="m",output="elevation")
```

It will create a GMT scale file with extension .cpt from the GRASS colr file. If the later do not exists, the function will use r.colors to create it based on the rainbow GRASS palette. It is necessary because some grass modules (r.mapcalc for instance) do not create a colr file for the output map, in these cases GRASS uses the rainbow palette as default to display the layers.

The result of the previous order appears on figure 3 top left map.

The color scale and the value intervals are obtained from the GRASS color palette file, unless scaleinter parameter is used (see below).

```
\% -10 2000
-10:0:0:255 0:0:191:191                    -10 0 0 255 0 0 191 191
0:0:191:191 10:0:255:255                    0 0 191 191 10 0 255 255
10:0:255:255 30:0:255:0                     10 0 255 255 30 0 255 0
30:0:255:0 120:255:255:0                    30 0 255 0 120 255 255 0
120:255:255:0 350:255:127:0                 120 255 255 0 350 255 127 0
350:255:127:0 700:191:127:63                350 255 127 0 700 191 127 63
700:191:127:63 2000:255                     700 191 127 63 2000 255 255 255
```

Figure 2: GRASS color codification (colr file) and GMT color codification .cpt file)

Several parameters can control the appearance of this scale:

- decscale, determines the number of decimal places to display in the numbering of the scale (default is zero).

- `scaletype`, scale type: "d" = discrete intervals scale, "c" = continuous scale (default). If using a discrete interval scale, the number of intervals is equal to the number of intervals defined in the color palette file layer GRASS. Each interval between two values appear with the color corresponding to the lower range so that the color corresponding to the highest values be lost.

- `scaleoption`, it allows to pass other options to the GMT psscale module (see the module manual[2]). By default the length between annotations in the color scale is proportional to the difference between annotation values (see Figure 3 upper left map). If you want to disable this option you must use the "-L" option with the `scaleoption` parameter.

  ```
  GMT_map("d.rast elev_state_500m",scaleZ="elev_state_500m",
  scale=4000000,grid=100000,dx=4,dy=5,scaleunit="m",scaleoption="-L")
  ```

  The result of this order appears on figure 3 top right map.

- `scalepos`, determines the position and size of the scale, should be coded as a vector of float numbers `c(posX,posY,height,width)` all expressed in cm.

  ```
  GMT_map("d.rast elev_state_500m",scaleZ="elev_state_500m",
  scale=4000000,grid=100000,dx=4,dy=5,scaleunit="m",scaleoption="-L",
  scalepos=c(23,2,3,0.5))
  ```

  The result can be seen on figure 3 middle left map.

  By default the color scale is located 1 cm to the right of the map, vertically centered with it, its height is half of the map and one centimeter in width.

- `range`, minimum and maximum values on the scale are calculated from the range of values to be displayed and the color ramp defined in the `colr` file. You can specify a different range with the parameter `range`:

  ```
  GMT_map("d.rast elev_state_500m",scaleZ="elev_state_500m",
  scale=4000000,grid=100000,dx=4,dy=5,scaleunit="m",scaleoption="-L",
  range=c(100,500),decscale=1,scalepos=c(23,2,3,0.5))
  ```

  The result can be seen on figure 3 middle right map.

  The scale does not take exactly the values passed but a compromise between these and the limits of colr legend in the file.

- `scaleunit`, units in which the variable is measured, appears on the color scale.

- `scalevar` variable name, appears next to the color scale

- `scaleinter` intervals between annotations in the color scale. The default is to use intervals colr file the GRASS raster layer. This option can not be used with the `-L` option in the `scaleoption` parameter.

The next example shows the use of the last three parameters:

```
GMT_map("d.rast elev_state_500m",scaleZ="elev_state_500m",decscale=1,
scale=4000000,grid=100000,dx=3,dy=5,scaleunit="m",scalepos=c(22.5,4,5,0.5),
scalevar="Elevation",scaleinter=250)
```

You can see the results on figure 3 bottom left map.

The font size of the annotations and scale unit label can be modified using GMT global variable `ANNOT_FONT_SIZE`. The font size used to write the name of the scale variable can be modified using GMT global variable `LABEL_FONT_SIZE`

[2] http://gmt.soest.hawaii.edu/gmt/html/man/psscale.html

```
GMT_map("d.rast elev_state_500m",scaleZ="elev_state_500m",decscale=1,
scale=4000000,grid=100000,dx=3,dy=5,scaleunit="m",scalepos=c(22.5,4,5,0.5),
scalevar="Elevation",scaleinter=250,gmtset="ANNOT_FONT_SIZE=8 LABEL_FONT_SIZE=14")
```

You can see the results on figure 3 bottom right map.

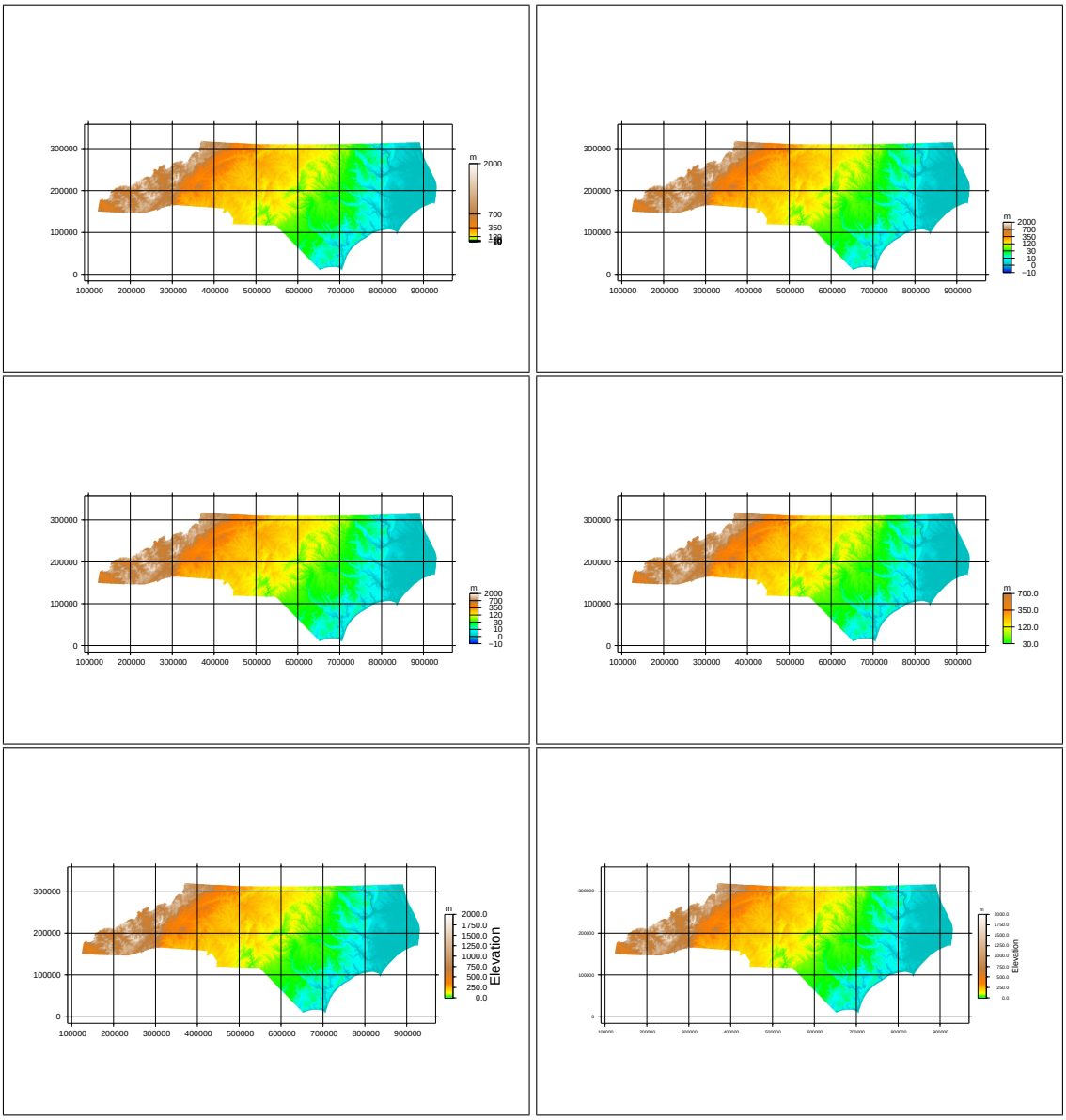

Figure 3: GMT_map with quantitative scales

3.2 Legends for qualitative variables in raster layers

Creating a legend for a qualitative variable with GMT is quite more complicated than to create a color scale, so it is also more complicated with GMT_map, but not that much.

The legend must be created with the function `rast2leg` or even typing directly the text file (see the GMT pslegend module manual).

We are going to check some of the possibilities with a land use map (`landuse96_28m`), so first of all, we have to change the work region:

```
# g.region save=myregion
# g.region rast=landuse96_28m
```

As `GMT_map`, `rast2leg` has several parameters but just a few are really necessary, for example:

```
rast2leg("landuse96_28m",file = "landuse")
```

creates a GMT legend from the color palette (`colr`) and category (`cats`) files of the raster layer. The legend is saved in the file `landuse.leg` (Figure 4).

```
G 0c
N 1
S 0.25c s 0.5c 255/0/0 0.5p 0.75c High Intensity Developed
G 0.25c
S 0.25c s 0.5c 255/51/76 0.5p 0.75c Low Intensity Developed
G 0.25c
S 0.25c s 0.5c 255/255/0 0.5p 0.75c Cultivated
G 0.25c
S 0.25c s 0.5c 229/229/204 0.5p 0.75c Managed Herbaceous Cover
G 0.25c
S 0.25c s 0.5c 127/204/127 0.5p 0.75c Evergreen Shrubland
G 0.25c
S 0.25c s 0.5c 127/25/101 0.5p 0.75c Deciduous Shrubland
G 0.25c
S 0.25c s 0.5c 228/124/26 0.5p 0.75c Mixed Hardwoods
G 0.25c
S 0.25c s 0.5c 153/127/127 0.5p 0.75c Bottomland Hardwoods/Hardwood Swamps
G 0.25c
S 0.25c s 0.5c 255/188/200 0.5p 0.75c Southern Yellow Pine
G 0.25c
S 0.25c s 0.5c 50/178/102 0.5p 0.75c Mixed Hardwoods/Conifers
G 0.25c
S 0.25c s 0.5c 0/0/255 0.5p 0.75c Water Bodies
G 0.25c
S 0.25c s 0.5c 255/255/255 0.5p 0.75c Unconsolidated Sediment
G 0.25c
```

Figure 4: GMT legend file of the map `landuse96_28m` created with `rast2leg`

Then you have just to include this file in a `GMT_map` call:

```
GMT_map("d.rast landuse96_28m",scale=100000,grid=2000,legend = "landuse")
```

The result (Figure 5 top left map) is not appropriate because we must change the size and position of the box containing the legend to allow all the labels to be shown. Parameter `legendpos` dos the trick, it is encoded as the

scalepos parameter. Default values are 3 cm to the right of the map, aligned with the upper limit of the map, 5 cm wide and 5 cm high. So to adjust the size of the legend:

```
GMT_map("d.rast landuse96_28m",scale=100000,grid=2000,legend="landuse",
legendpos=c(16,13,11,12.5))
```

The result appears on figure 5 top right map.

rast2leg includes several parameters to alter the look of the legend:

- title, to be incorporated into the legend box
- font, size and font for the title of the legend (default "14 Helvetica"), the available types appear in GMT documentation[3]
- gap, provides a jump in cm between the title and the first element of the legend (default 0)
- size indicates the size of the symbols in the legend (default 0.5)
- symbol indicates the type of symbols that appear in the legend (s = square is the default option). You can check the several available options in the manual of the GMT module psxy.
- space1, cm space between the left edge of the legend and the center of the symbols (default value 0.25)
- space2, cm space between the left edge of the legend and the beginning of the text (default value 0.75)
- space3, cm space between the legend items (default value 0.25)
- columns, number of legend columns into which the legend (default value1).
- file, name the file containing the legend, add the ". leg" (the default name is the same as that of the layer that is used to generate the legend.

Now we are going to check some of these options (title, symbol and space between lines):

```
rast2leg("landuse96_28m",file="landuse",title="Land Use",symbol="c",space3=0.1)
```

```
GMT_map("d.rast landuse96_28m",scale=100000,grid=2000,legend="landuse",
legendpos=c(16,13,11,11))
```

The result appears on figure 5 bottom left map.

The parameter gmtsetleg allows you to change global GMT that will just affect to the legend layout. For example gmtsetleg="ANOT_FONT_SIZE=6" change the font size the labels of the legend while gmtsetleg= "ANOT_FONT=4" modify the font.

In the next example we are changing the font and size of the legend, the size of the symbols to match them and the size of the legend.

```
rast2leg("landuse96_28m",file = "landuse",title="Land Use",symbol="c",
space3=0.2,size=0.3)
```

```
GMT_map("d.rast landuse96_28m",scale=100000,grid=2000,legend="landuse",
legendpos=c(16,13,8,10),gmtsetleg = "ANOT_FONT_SIZE = 10 ANOT_FONT = 4")
```

The result appears on figure 5 bottom right map.

[3] http://gmt.soest.hawaii.edu/gmt/pdf/GMT_Docs.pdf

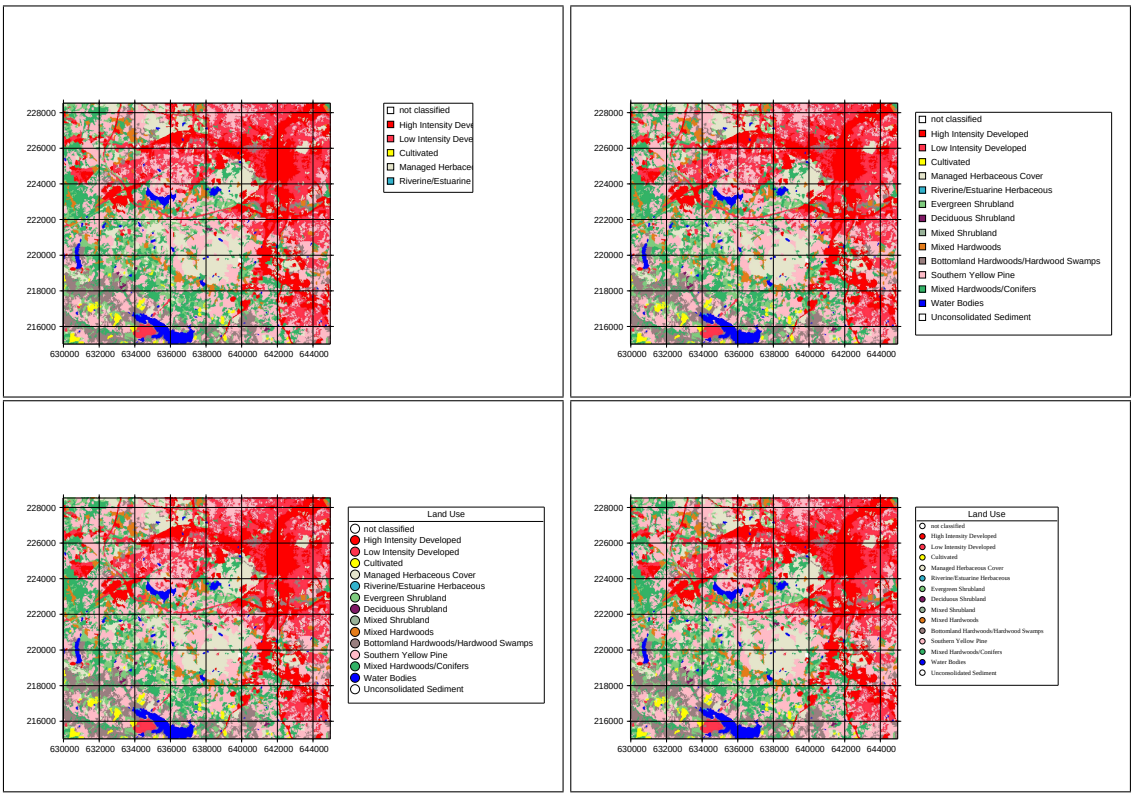

Figure 5: GMT_map with qualitative legends

3.3 Legends from vector maps

The function `vect2leg` creates a GMT legend file from a vector layer with the colors encoded in an RGB column. It complements the GRASS order `d.vect -a`.

It has the same parameters as `rast2leg` except that instead of using a raster layer as main parameter it uses a vector one. It also requires a parameter called `column` containing the text labels that will appear in the legend and a parameter `columnRGB` to state the name of the column storing the color palette using RGB code.

The function `randomgrassrgb` assigns a random RGB palette to a vector layer. The parameter `column` indicates which column will be used to group objects that will have the same color. The default is `cat`. The following example shows how to use these two functions and to modify the font size within the legend with the parameter `gmtsetleg`.

First of all, we have to come back to the GRASS region used in sections 2 and 3:

```
# g.region myregion
```

Next we have to fix a minor problem. Vector layers in the North Carolina location store their tables in DBF format. So, you need to create a PostgreSQL database (if you have permission to do that on your system) or use an available one to store the tables of the new maps:

```
# createdb nc_spm_08
# db.connect driver=pg database=nc_spm_08
```

Finally, you have to copy the layer, that is stored in the mapset PERMANENT, to your mapset and add a column to contain the color palette:

```
# g.copy vect=boundary_county,boundary_county
# v.db.addcol boundary_county column="grassrgb varchar(11)"
```

Now, from the R terminal we populate the palette column:

```
randomgrassrgb("boundary_county",column="name_locas")
```

create the legend:

```
vect2leg(map="boundary_county",file="boundary_county",title="Counties",
column="name_locas",columns=7)
```

and, finally, create the map:

```
display="d.erase;d.vect -a boundary_county type=area"
GMT_map(display,legend="boundary_county",width=20,grid=100000,dx=5,dy=2,
legendpos=c(-0.5,16.5,21,7),gmtsetleg="ANOT_FONT_SIZE=10",output="counties_map")
```

The result appears on Figure 7.

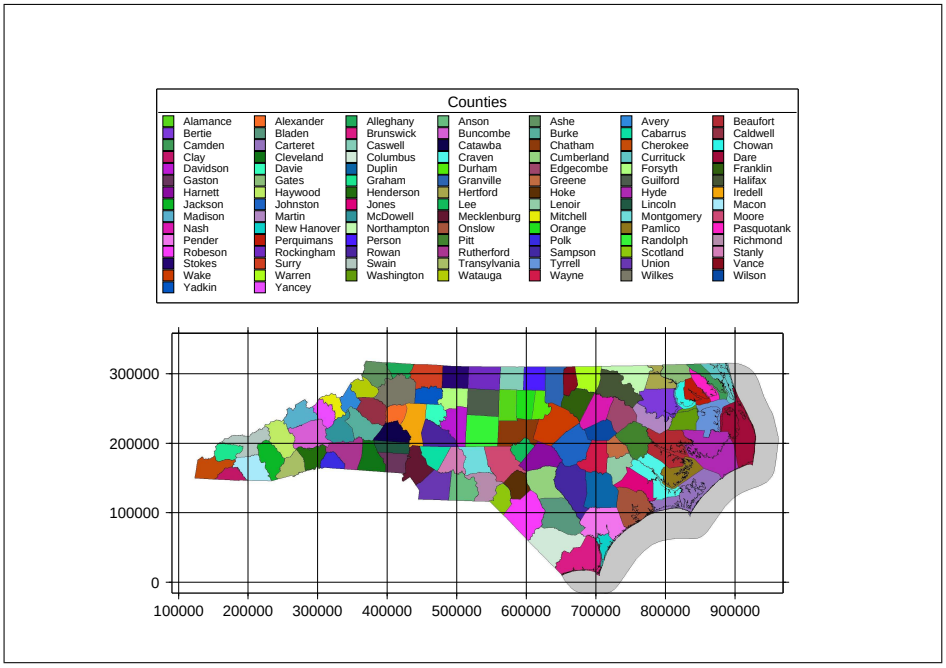

Figure 6: GMT_map with legends generated by randomgrassrgb and builded from d.vect.thematic

3.4 Legends from d.vect.thematic

The function `vectthematic2leg` creates a GMT legend file from an execution of the GRASS module `d.vect.thematic`. This function has the same parameters as `rast2leg` except that instead of using a raster map as the main parameter, it uses a complete call to the GRASS module `d.vect.thematic`.

We are going to check this function with the Wake County census blocks layer (`censusblk_swwake`). So, we have to change, in the GRASS terminal, the GRASS region.

```
# g.region n=231200 s=211800 w=628300 e=646500
```

Then we have to copy the vector layer to our mapset to store the new table in the database being able to modify it: We are going to create 3 new columns to store some statistics:

```
# g.copy vect=censusblk_swwake,censusblk_swwake
# v.db.addcol censusblk_swwake column="density float, white_index float,
pop_pr_house float"
```

Now we have to populate our columns with some indices from the older columns. Keep in mind the 0 values in some of the columns:

```
# v.db.update censusblk_swwake column=density value=10000.0*total_pop/area
# v.db.update censusblk_swwake column=white_index value=100.0*white_only/total_pop
where="total_pop>0"
# v.db.update censusblk_swwake column=white_index value=0 where="total_pop=0"
# v.db.update censusblk_swwake column=pop_pr_house value=total_pop/households
where="households>0"
# v.db.update censusblk_swwake column=pop_pr_house value=0 where="households=0"
```

and then we continue with R:

```
display="d.erase white;d.vect.thematic censusblk_swwake column=TOTAL_POP
themecalc=quartiles"
vectthematic2leg(display,title="Population",file="pob")
GMT_map(display=display,legend="pob",ori="l")
```

The inclusion of "`d.erase white`" prevents an inappropriate GMT background.

It is pretty easy to do a thematic atlas assuming our GRASS mapset storage the tables for the vector layers in PostgreSQL.

Now we create the population density map:

```
display="d.erase white;d.vect.thematic censusblk_swwake column=density
themecalc=quartiles"
vectthematic2leg(display,title="Population density",file="pob")
GMT_map(display=display,legend="pob",ori="l",legendpos=c(16,15.5,6.5,3.5),
output="pob2")
```

the percentage of white people map:

```
display="d.erase white;d.vect.thematic censusblk_swwake column=white_index
themecalc=quartiles"
vectthematic2leg(display,title="Percentage of white people",file="pob")
GMT_map(display=display,legend="pob",ori="l",legendpos=c(16,15.5,6.5,3.5),
output="pob3")
```

and the people per household map:

```
display="d.erase white;d.vect.thematic censusblk_swwake column=pop_pr_house
themecalc=quartiles"
```

```
vectthematic2leg(display,title="People per household",file="pob")
GMT_map(display=display,legend="pob",ori="l",legendpos=c(16,15.5,6.5,3.5),
output="pob4")
```

The resulting four maps appear on figure 7

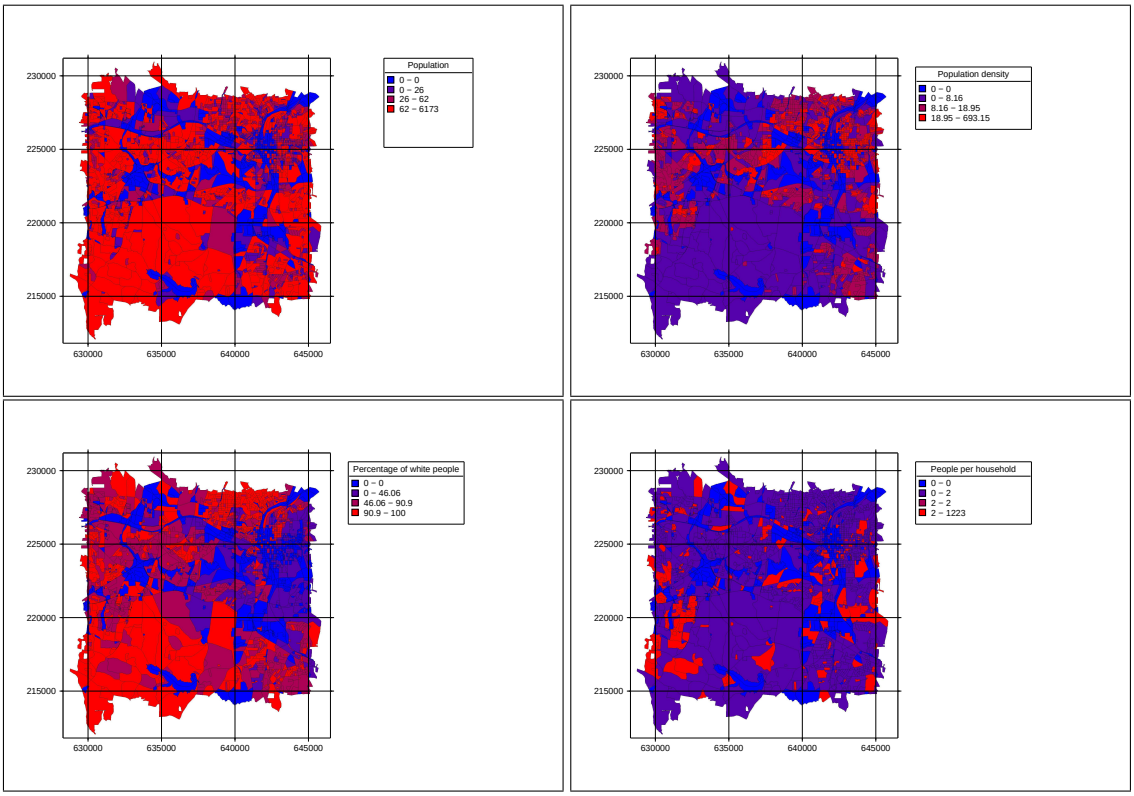

Figure 7: GMT_map with legends builded from d.vect.thematic

4 Function `latex_map`

The function `latex_map` adds, to a PDF map created with `GMT_map`, various cartographic elements such as titles, icons, text boxes and graphic or numeric scale.

Again, its use is very simply. The next command:

```
latex_map("counties_map",ori="l",output="map",scale=4000000)
```

create a new PDF file with just the original map (file `elev_state_500m.pdf`) and create an output file (`map.pdf`). However, we can use different parameters to introduce more elements. The simplest is to include a title.

```
latex_map("counties_map",ori="l",output="map",scale=4000000,
title="North Carolina State County Map")
```

The default positions of title and scale are `c(30,0,0)` and `c(30,0,19)`. That is a centered location on a landscape A4 paper with the title at the top of the map and the scale at the bottom. Those 3 numbers define the `textblock` (`textpos` package of LaTeX) that occupy each of the text fragments, indicating the width and the X/Y position in centimeters from the top left corner.

Title and scale positions can be modified (parameters `titlepos` and `scalepos`). Parameter `numscale` controls if the numerical scale will be displayed. Default value is `TRUE`.

Other important parameters to define the appearance of the final map are:

- `ori`: Map orientation can be "l" (landscape) or "p" (portrait). Landscape is the default value.
- `paper`: Paper size: a0, a1, a2, a3 or a4

4.1 Adding logos

It may be necessary to introduce other images in the final map along with the map. They can be logos, graphics or location maps. The parameters to add these images are:

- `logo`, a vector of image files to introduced into the map
- `logopos`, a matrix with 3 columns and as many rows as logos, the columns indicate the width of the logo, its position in centimeters from the upper left corner of the paper.

```
logos=c("grass_logo.png","R_logo.png","gmt_logo.png","LaTeX_logo.png")
logopos=rbind(c(1,1,19),c(1.15,2.5,19.2),c(1,1,20.5),c(1.25,2.5,20.5))
title="North Carolina State County Map"
latex_map("counties_map",ori="l",output="mapa",scale=4000000,
logos=logos, logopos=logopos, title=title)
```

4.2 Adding texts

If you want to add text boxes with metadata, you can use the parameters `latex` and `latexpos`.

- `latex`, a vector of files containig LaTeXcode.
- `latexpos`: matrix with 3 columns and many rows as elements in latex, the columns define the corresponding textblock (width, and position (X and Y) in centimeters.

```
latexpos=c(10,5.5,16.5)
latex_map("counties_map",ori="l",output="mapa",title=title,
scale=4000000, logo=logos,logopos=logopos,
latex=c("latex1.tex"),latexpos=latexpos)
```

The file `latex1.tex` includes the next code:

```
\colorbox{white}{
   \fbox{
      \begin{minipage}{6cm}
         {\small
```

```
          Counties in North Carolina State\\
          \vspace{-0.7cm} \\
          \hspace{1cm} projection: Lambert Conformal Conic\\
          \vspace{-0.7cm} \\
          \hspace{1cm} datum: nad83\\
          \vspace{-0.7cm} \\
          \hspace{1cm} layout: GaRGoyLe
           }
        \end{minipage}
    }
}
```

Note that logos and graphics could also be included inside latex files.

4.3 Adding a scale bar

The inclusion of a scale bar needs the latex tikz library. The scale is drawn only if the parameters `gscaleX` and `gscaley` have values greater than 0, the parameters required to control graphic scale in `latex_map` function are:

- `gscaleX` and `gscaley`, they define the position in centimeters of the rectangle around the scale from the top left corner of the paper.
- `ni`, number of intervals in the scale (default 5)
- `di`, width of each interval in meters, the default is to let the function to calculate it automatically.
- `gscaletype`, there are three types. First one (`gscaletype=1`) can be seen in most of the figures of this tutorial. If `gscaletype=2` we obtain the same scale bar but without a box. Finally, `gscaletype=3` is a quite simpler scale bar (see figure 11).

```
latex_map("counties_map",output="mapa",title=title, scale=4000000,
logo=logos,logopos=logopos,ori="l",latex=c("latex1.tex"),
latexpos=latexpos,gscaleX=5.5,gscaleY=10.9,ni=5,di=25000,numscale=F)
```

The result appears on figure 8.

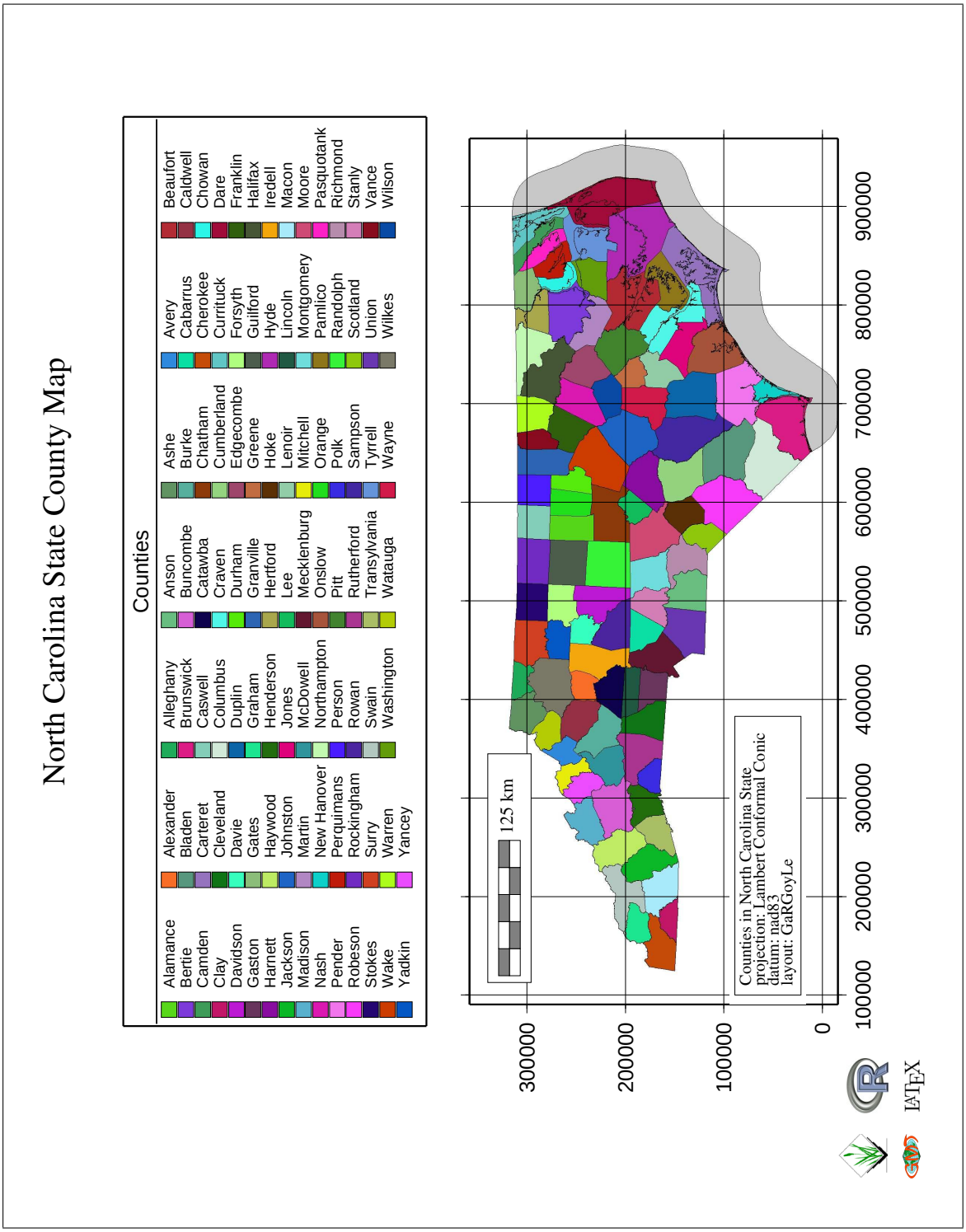

Figure 8: latex_map created from figure 7

5 Creating color palettes

Sometimes, it is useful to create color palettes to GRASS raster or vector layers. GaRGoyLe includes four functions to do that. The function `randomgrassrgb` has been already used to populate a column in a GRASS vector map with random values. The others are:

- `put_palette`, is the simplest one, you have to introduce 4 vectors: `val`, `red`, `green` and `blue`. This function creates a new palette for a raster layer.
- `create_palette`, in this one the user provide a vector of colors and choose a style to create the palette. This function also works on raster layers.
- `create_palette_v`. It modifies the grassrgb column of a table linked to a vector map. It works in a very similar way to `create_palette`.

The function `randomgrassrgb` is more adequate for qualitative variables and these three new ones for both cases. The first two for raster maps and the last one for vector maps.

5.1 put_palette

The function `put_palette` can be used for quantitative and qualitative raster layers. It creates the `colr` GRASS file from intervals and colors that are passed to the function. Its parameters are:

- `map`, which raster map is going to receive the palette
- `red`, `green` and `blue` are vector with the respective color intensities
- `val`, is a vector with value intervals.
- `type` can be "d" for discrete palettes or "c" for continuous palettes. The first case is more appropriate for qualitative variables and the second one for quantitative variables.

The vectors `val`, `red`, `green` and `blue` have to have the same length.

This example

```
red = c (0,0,0,255,255)
green = c (0,255,255,0,0)
blue = c (255,0,255,255,0)


put_palette("elev_state_500m", val=c(0,500,1000,1500,2000),
red=red,green=green,blue=blue,type="d")
```

produces the `colr` file in Table 1.

Table 3 shows the file that the `type="d"` option would have produced:

```
put_palette("elev_state_500m", val=c(0,500,1000,1500,2000),
red=red,green=green,blue=blue,type="c")
```

This is, of course, the best option for a quantitative variable like the map `elev_state_500m`. The next step is to create the map with the function `GMT_map`:

```
GMT_map(display="d.rast elev_state_500m",width=24,grid=100000)
```

Its result is shown if figure 9 top right map.

```
% 0 2000
0:0:0:255
500:0:255:0
1000:0:255:255
1500:255:0:255
2000:255:0:0
```

Table 1: colr file created by `put_palette` with `type="d"`

```
% 0 2000
0:0:0:255 500:0:255:0
500:0:255:0 1000:0:255:255
1000:0:255:255 1500:255:0:255
1500:255:0:255 2000:255:0:0
```

Table 2: colr file created by `put_palette` with `type="c"`

5.2 create_palette

This function allows to create the `colr` file for a GRASS raster layer that contains a quantitative variable. Instead of defining explicitly the intervals and colors, as in `put_palette`, a `style` and a `color` vector are introduced. The function creates the intervals and creates the `colr` file.

The parameters are:

- `map`, which raster map is going to receive the palette
- `color`, a vector of color names to build the palette
- `style`, one of "fixed" "sd" "equal" "pretty" "quantile" "kmeans" "hclust" "bclust" "fisher" or "jenks"
- `intervals`, number of intervals

```
create_palette("elev_state_500m",color=c("green","yellow","red"),
style="quantile",intervals=10)
```

Now you can draw the map:

```
GMT_map(display="d.rast elev_state_500m",width=24,grid=100000)
```

The result is the colr file in Table 3. Te map produces appears in Figure 9 top right map.

This function uses the R package `classInt`. The style parameter allows you to create different types of paletes. The styles supported by `classInt` are: `fixed`, `sd`, `equal`, `pretty`, `quantile`, `kmeans`, `hclust`, `bclust`, `fisher` and `jenks`. See the `classInt` manual for a comprehensive discussion of these styles.

Both `create_palette` and `put_palette` create the `colr` file in the directory `colr2` of the user's MAPSET. If the whole directory structure does not exist, the function will create it.

```
% 56.5197 151.072
56:0:255:0 81:56:255:0
81:56:255:0 90:113:255:0
90:113:255:0 97:170:255:0
97:170:255:0 104:226:255:0
104:226:255:0 111:255:226:0
111:255:226:0 118:255:170:0
118:255:170:0 125:255:113:0
125:255:113:0 135:255:56:0
135:255:56:0 151:255:0:0
```

Table 3: colr file created by `create_palette`

5.3 create_palette_v

If `create_palette` generates a color palette file for a raster layer, the function `create_palette_v` does the same for vector one. It assigns to each object in the layer a color combination that is stored in a column linked to the layer table (default value is grassrgb). It can also generate a GMT legend file that can be used with `GMT_map`.

The parameters used by this function are:

- `map`, which layer is used to generate the legend
- `column`, column containing the labels
- `columnRGB`, column containing the RGB combinations (default grassrgb)
- `color`, color scheme is used, the default is c ("green", "cyan", "blue", "magenta")
- `style`, color style (package classInt R), the default is "quantile".
- `intervals` into which the variable to create the palette (the default is 10)

On the other hand. it uses the parameters `gap`, `title`, `columns`, `symbol`, `size`, `file`, `space1`, `space2`, `space3` and `font` with the same purpose explained above.

In the following example, we first create a palette for the population variable layer municipalities. Since the `file` parameter is included, the function will also create a legend file which can then be used with `GMT_map`:

First of all we need to copy the `censusblk_swwake` vector layer to the user mapset, add the column for the color palette and change to a proper region for this layer:

```
# g.copy vect=censusblk_swwake,censusblk_swwake
# v.db.addcol censusblk_swwake column="grassgb varchar(11)
# g.region vect=censusblk_swwake
```

We can, then, create the palette and map the layer:

```
create_palette_v (map="censusblk_swwake",column="total_pop",
columnRGB="grassrgb",color=c("green","cyan","blue","magenta","red"),
style="quantile",interval=5,file="densityleg",title="Population density",gap=0.25)
GMT_map(display="d.erase white;d.vect -a censusblk_swwake type=area",
output="dens_map",legend="densityleg",ori="l")
```

The result appears on figure 9, lower right map.

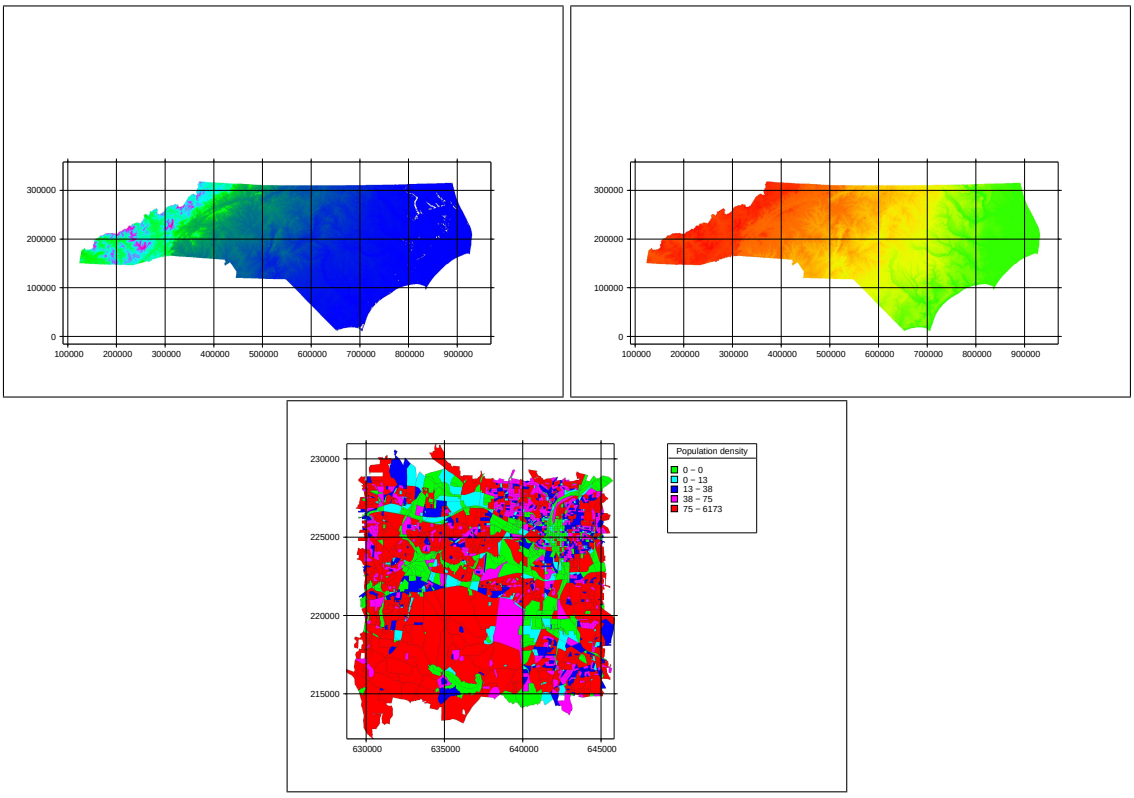

Figure 9: Results of the palette modifications

6 Text labels

It is difficult to find good text fonts for paper maps in GRASS. GMT includes several fonts which easily produce a nice effect on a paper map. The function`GMT_map` includes a parameter `textfile` that lets you enter a GMT file of text labels. The problem is reduced to generate this file from a layer with GRASS vector points or polygon centroids.

The function grasspoints2GMTtext is designed to do this task. It includes the following parameters:

- `map`, the name of the layer from which the labels are obtained.
- `variable`, variable that contains the labels.
- `size`, label size (default 12).
- `angle`, angle of the labels (default 0).
- `font`, text font for labels (default 1, see Annex).
- `just`, justification of the labels around its insertion point. It is a combination of two letters, "T" (top), "M" (middle) or "B" (bottom) and "L" (left), "C" (center) and "R" (right). By default the justification is "MC".

```
386387 299996 8 0 1 MC Ashe
418299 310320 8 0 1 MC Alleghany
893707 297594 8 0 1 MC Currituck
880252 126366 8 0 1 MC
864577 292272 8 0 1 MC Camden
458230 287076 8 0 1 MC Surry
820660 310202 8 0 1 MC Gates
796001 295541 8 0 1 MC Hertford
752057 306148 8 0 1 MC Northampton
498162 303764 8 0 1 MC Stokes
688881 300784 8 0 1 MC Warren
537498 285884 8 0 1 MC Rockingham
662657 285884 8 0 1 MC Vance
```

Table 4: GMT file with text labels

- `dd`, threshold distance between labels for being too close. In this case the function try to optimize the arrangement thereof (not guaranteed to get it). With a value of 0 (option default) is not optimized. This parameter is experimental and not working properly.
- `file`, files are saved in text labels.

If you do not like the arrangement of the labels, you can modify the text file, or even move the points with `v.digit`. Figure 10 shows two maps obtained with this function before and after moving the centroids. Some of the labels are still too close because some counties are composed of many small polygons, specially islands in the coast. A solution could be to previously filter the polygons by area.

```
# g.region myregion
grasspoints2GMTtext("boundary_county",column="name_locas",dd=0,
file="locas_label",size=8,distinct=T)
GMT_map(display="d.erase white;d.vect boundary_county color=red type=boundary",
textfile="locas_label",width=26.5,grid=0,anots=100000)
```

Table 4 shows the first lines of the file `locas_label` generated with the previous commands.

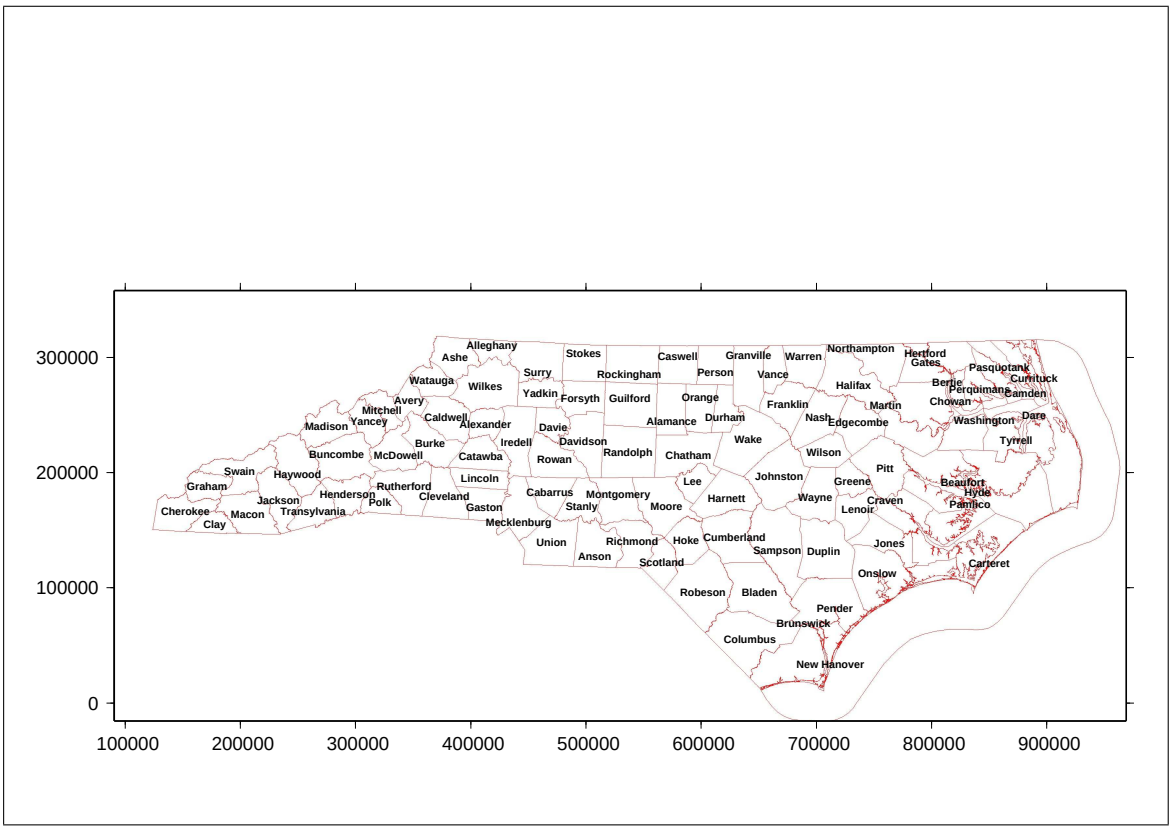

Figure 10: Text labels from centroids

7 Some examples from scratch

7.1 A small figure for a text document

The next example creates a PDF file large enough to contain the map but not larger. This way the PDF file can be included "as is" in a latex file without having to deal with wasted white space and keeping a correct numerical scale.

```
display="d.erase 220:240:240; d.rast -o elev_state_500m"

GMT_map(display,dxcm=0.75,dycm=-1,paper="a6",scale=10000000,grid=100000,
        anots=100000,scaleZ="elev_state_500m",scalepos=c(9.25,2.15,2,0.2),
        scaleoption="-L",gmtset="ANNOT_FONT_SIZE=6 Y_AXIS_TYPE=ver_text"
)

latex_map("outmap",output="smallmap",scale=10000000,
           bgcol=c(255,255,255),paper=c(11.5,4.75),ori="l",gscaletype=3,
           gscaleX=10,gscaleY=3,di=20000
)
```

You have to estimate a proper dimensions of your final output map for it to have room enough for all the cartographic elements. The result appears on figure 11.

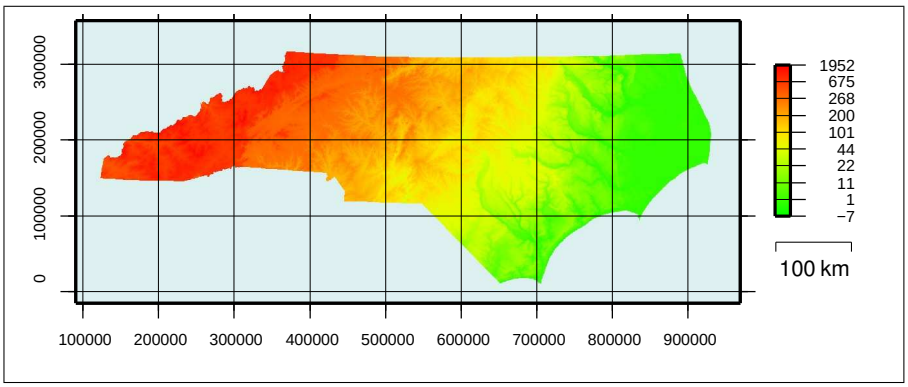

Figure 11: Small map to be included as a figure on a text document (scale=1:10,000,000)

The `dycm=-1` option in `GMT_map` anchors the map in the top hedge of the paper. It can be any size, but large enough to contain the whole map, legends, scale bars, etc.

The `paper=c(10,4.2515)` parameter in `latex_map` establish a user defined paper size. You can calculate it from the parameters passed to both functions or just try and correct.

7.2 A single map with multiple information

This map will contain two sets of text labels. Depending of the size of the urban area it will be used one or the other.

First of all, we need some GRASS commands to prepare the information:

```
# g.copy vect=urbanarea,urbanarea
# v.db.addcol urbanarea column="area float"
# v.to.db urbanarea option=area columns=area

# v.extract urbanarea output=urban1 type=centroid
  where="area>30000000 and name!='Danville'"

# v.extract urbanarea type=centroid output=urban2
  where="area<=30000000 and area>3000000 and name!='Danville'"

# v.extract urbanarea type=area  output=urbanarea2
  where="name!='Danville'"
```

Finally the R functions calls.

```
display="d.erase 200:240:240; d.rast -o elev_state_500m;
         d.vect boundary_county type=boundary color=grey width=3;
         d.vect railroads width=1 color=white;
         d.vect railroads width=3 color=black;d.vect urbanarea2 type=area"

grasspoints2GMTtext("urban1",column="name",file="Tlabel1",size=6,
                    distinct=T,just="BC"
)
grasspoints2GMTtext("urban2",column="name",file="Tlabel2",size=4,
                    distinct=T,just="BC"
)

textcolor=rbind(c(50,50,150),c(50,50,50))

GMT_map(display,dxcm=3,textfile=c("Tlabel1","Tlabel2"),textcolor=textcolor,
        scale=4000000,grid=0,anots=100000,
        gmtset="ANNOT_FONT_SIZE_PRIMARY=10",
        scaleZ="elev_state_500m",scalepos=c(2,2,3,0.5),scaleoption="-L"
)

title="A small map of North Carolina"
titlepos=c(30,0,5)

latex_map("outmap",output="mapa_counties",title=title,titlepos=titlepos,
           scale=4000000,bgcol=c(245,225,160),
           ori="l",gscaleX=19.5,gscaleY=17,ni=5,di=25000,numscale=F
)
```

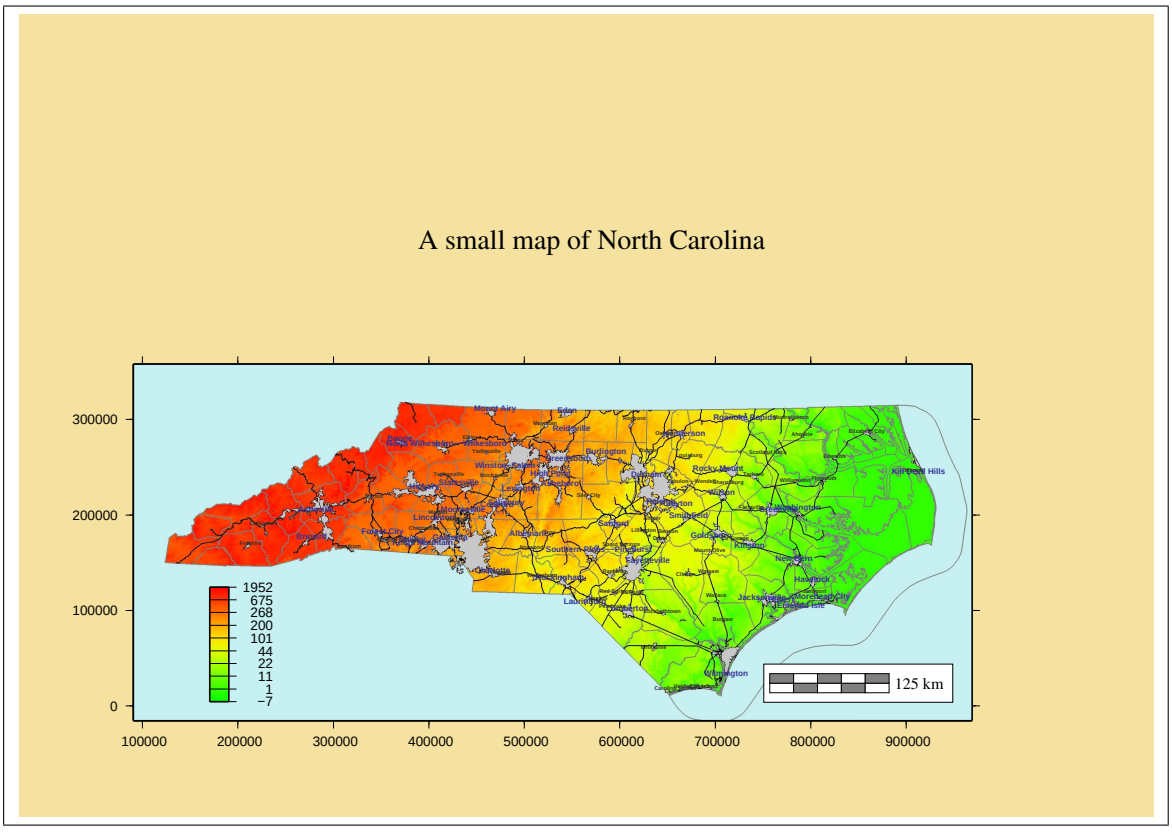

Figure 12: An a0 poster with several maps

7.3 Several maps on one sheet

In this example we will use `latex_map` to place four maps, obtained by `GMT_map`, in a single A3 paper.

```
# g.region n=231200 s=211800 w=628300 e=646500

create_palette_v (map="censusblk_swwake",column="total_pop",
                  color=c("green","cyan","blue","magenta","red"),
                  columnRGB="grassrgb",style="quantile",
                  interval=5,file="legend",title="Population",gap=0.25,
                  font="10 Helvetica"
)

GMT_map(display="d.erase white;d.vect -a censusblk_swwake type=area",
        dxcm=6,dycm=16,legendpos=c(10,5,3,3),output="atlas1",
        legend="legend",ori="l",scale=200000,
        gmtset="ANNOT_FONT_SIZE=9p PAPER_MEDIA=a3",graph=F
)

create_palette_v (map="censusblk_swwake",column="density",
                  columnRGB="grassrgb",style="quantile",
                  color=c("green","cyan","blue","magenta","red"),
                  interval=5,file="legend",title="Population density",
                  gap=0.25,font="10 Helvetica"
)

GMT_map(display="d.erase white;d.vect -a censusblk_swwake type=area",
        dxcm=25,dycm=16,legendpos=c(10,5,3.5,3),output="atlas2",
        legend="legend",ori="l",scale=200000,
        gmtset="ANNOT_FONT_SIZE=9p PAPER_MEDIA=a3",graph=F
)

create_palette_v (map="censusblk_swwake",column="white_index",
                  color=c("green","cyan","blue","magenta","red"),
                  columnRGB="grassrgb",style="quantile",interval=5,
                  file="legend",title="Percentage of white people",
                  gap=0.25,font="10 Helvetica"
)

GMT_map(display="d.erase white;d.vect -a censusblk_swwake type=area",
        dxcm=6,dycm=4,legendpos=c(10,5,5,3),output="atlas3",
        legend="legend",ori="l",scale=200000,
        gmtset="ANNOT_FONT_SIZE=9p PAPER_MEDIA=a3",graph=F
)
```

```
create_palette_v (map="censusblk_swwake",column="pop_pr_house",
                  columnRGB="grassrgb",style="quantile",interval=5,
                  color=c("green","cyan","blue","magenta","red"),
                  file="legend",title="People / Household",gap=0.25,
                  font="10 Helvetica"
)

GMT_map(display="d.erase white;d.vect -a censusblk_swwake type=area",
        dxcm=25,dycm=4,legendpos=c(10,5,3.5,3),output="atlas4",
        legend="legend",ori="l",scale=200000,
        gmtset="ANNOT_FONT_SIZE=9p PAPER_MEDIA=a3",graph=F
)

title="A Population Atlas of SW Wake County (North Carolina State)"
titlepos=c(42,0,1)
logos=c("North_Carolina.png","North_Carolina.png")
logopos=rbind(c(2,5,1),c(2,35,1))

latex_map(c("atlas1","atlas2","atlas3","atlas4"),output="four_maps",
logos=logos,logopos=logopos,title=title,titlepos=titlepos,ori="l",
scale=250000,scalepos=c(20,5,26.5),gscaleX=25,gscaleY=27,paper="a3")
```

You can see how `dxcm` and `dycm` vary from a `GMT_map` call to the next, in consequence, each of the 4 intermediate maps are located in a different position. When the four layers are grouped by `latex_map` they fit without problems.

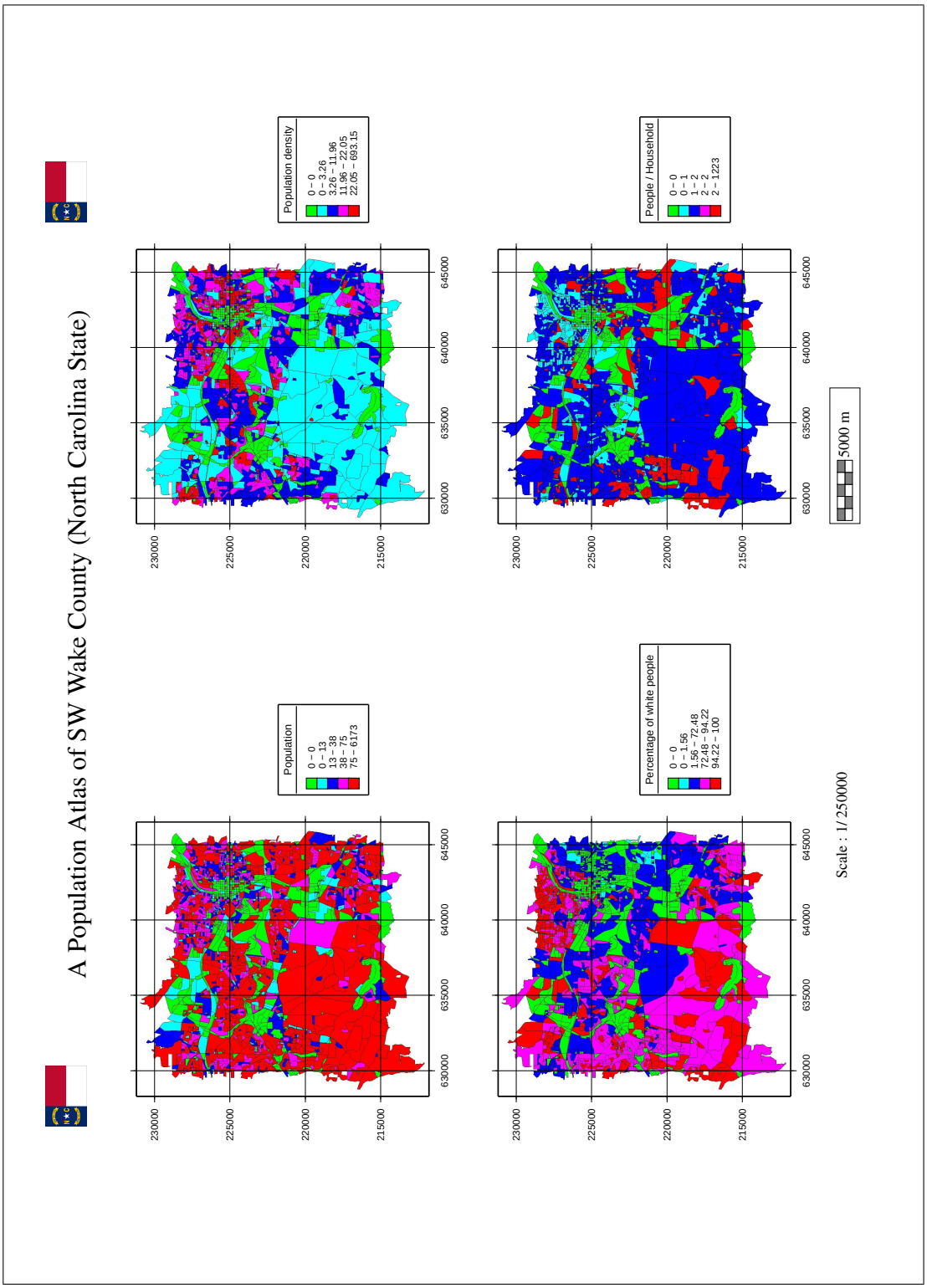

Figure 13: Several maps in the same A3 document

7.4 An A0 poster containing several maps

The next example will be an A0 poster showing maps of differente geomorphometric variables.

```
# g.region myregion

create_palette("slo_state_500m",style="fisher", intervals=10,sample=6000,
                color=c("green","yellow","orange","brown")

)

create_palette("pcurv_state_500m",style="fisher",intervals=10
               sample=6000,color=c("red","white","blue")
)

create_palette("tcurv_state_500m",style="fisher",intervals=10,
               sample=6000,color=c("red","white","blue")
)

GMT_map(display="d.erase white;d.rast elev_state_500m",ori="p",
        scale=1500000, paper="A0",dxcm=11,dycm=78,scaleZ="elev_state_500m",
        output="geom1", scalepos=c(62,15,15,1.5),scaleoption="-L",
        gmtset="ANNOT_FONT_SIZE=20p",graph=F
)

GMT_map(display="d.erase white;d.rast slo_state_500m",ori="p",
        scale=3000000,grid=100000,paper="A0",dxcm=5,dycm=59,
        scaleZ="slo_state_500m",output="geom2",scalepos=c(31,7,10,1),
        scaleoption="-L",gmtset="ANNOT_FONT_SIZE=20p",graph=F
)

GMT_map(display="d.erase white;d.rast asp_state_500m",ori="p",
        scale=3000000,grid=100000,paper="A0",dxcm=45,dycm=59,
        scaleZ="asp_state_500m",output="geom3",scalepos=c(31,7,10,1),
        scaleoption="-L",gmtset="ANNOT_FONT_SIZE=20p",graph=F
)

GMT_map(display="d.erase white;d.rast pcurv_state_500m",ori="p",
        scale=3000000,grid=100000,paper="A0",dxcm=5,dycm=41,
        scaleZ="pcurv_state_500m",output="geom4",scalepos=c(31,7,10,1),
        scaleoption="-L",gmtset="ANNOT_FONT_SIZE=20p",decscale=5,graph=F
)

GMT_map(display="d.erase white;d.rast tcurv_state_500m",ori="p",
        scale=3000000,grid=100000,paper="A0",dxcm=5,dycm=22,
        scaleZ="tcurv_state_500m",output="geom5", scalepos=c(31,7,10,1),
        scaleoption="-L",gmtset="ANNOT_FONT_SIZE=20p", decscale=5,graph=F
)
```

```
title="A Tiny Geomorphometric Atlas of North Carolina State\\\\
       Using GaRGoyLe"
titlepos=c(84,0,1)

logos=c("North_Carolina.png","grass.png","R.png","gmt.png","LaTeX.png")
logopos=rbind(c(4,5,110),c(4,70,110),c(4,77,110),c(4,70,115),c(4,77,115))

latex=c("latexelev","latexslo","latexasp","latexpcurv","latextcurv",
        "maintext","makingoff2"
)
latexpos=rbind(c(80,11.5,13),c(80,6.5,44),c(80,46.5,44),c(80,6.5,62),
               c(80,6.5,81),c(58,5,100),c(50,45,65)
)

latex_map(c("geom1","geom2","geom3","geom4","geom5"),output="four_maps",
          latex=latex,latexpos=latexpos,
          logos=logos,logopos=logopos,title=title,titlepos=titlepos,
          ori="p",scale=1500000,scalepos=c(20,5,155),
          paper="a0",bgcol=c(245,225,160)
)
```

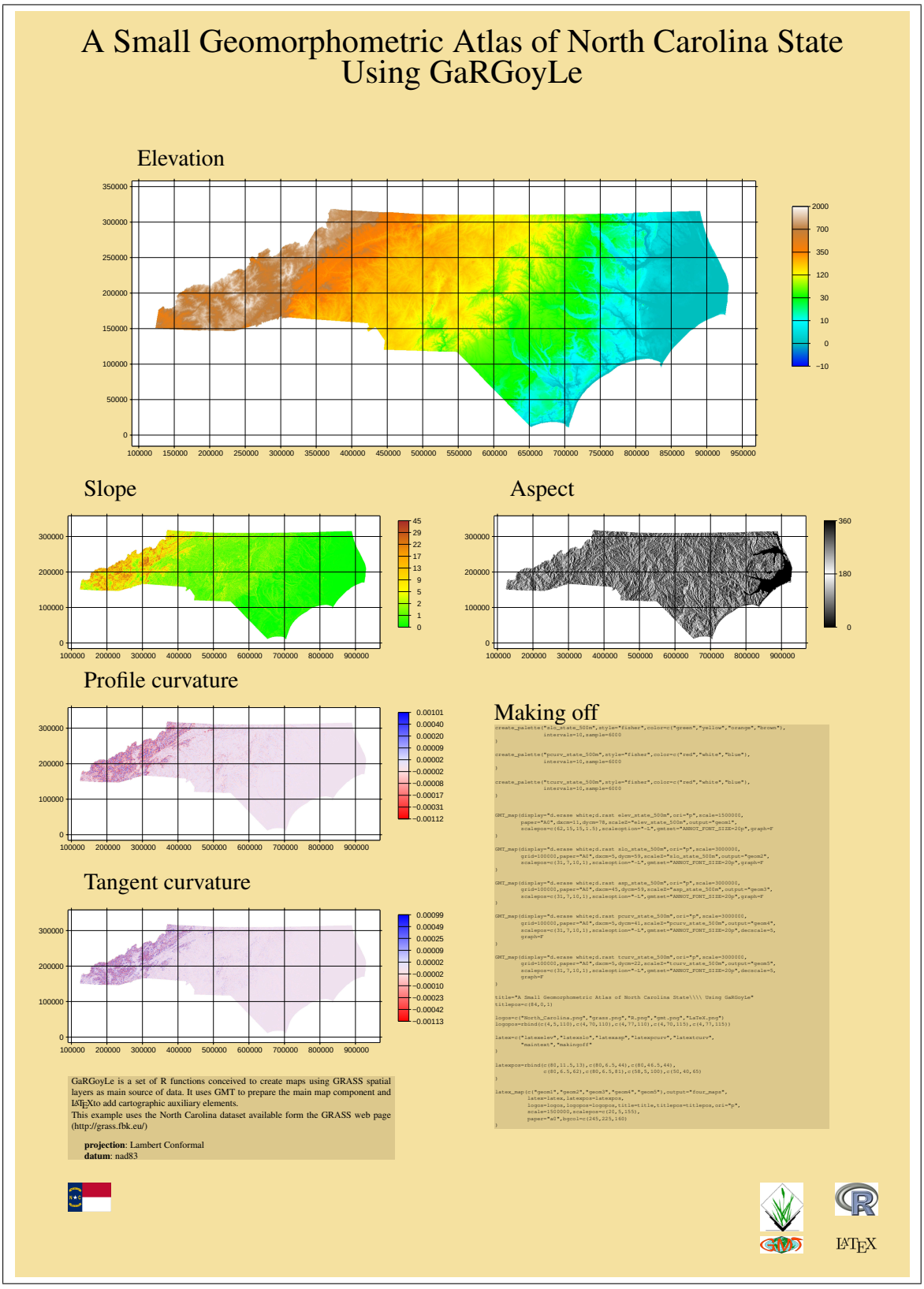

Figure 14: An a0 poster with several maps