Technical note
Processing the Harmonized World Soil Database (Version 1.2) in R

D G Rossiter / 罗大伟
Visiting Scientist / 客座教授
Institute of Soil Science, Chinese Academy of Sciences / 中国科学院南京土壤研究所

December 7, 2012

Contents

This note is a first attempt at explaining how to access and query the Harmonized World Soil Database (HWSD) [3] using the open-source R project for statistical computing [7]. This allows integration of the HWSD with any other geographic coverage, as well as statistical summaries.

This note shows how to:

1. Access the HWSD at IIASA and import it to R;
2. Select a geographic window from the HWSD, either by a rectangular bounding box or a boundary polygon(s);
3. Project from the original Plate Carrée (non)projection to the UTM coordinate reference system;
4. Determine the area covered by each soil class;
5. Save the window in the original HWSD format and as a projected raster;
6. Link the attribute database to the raster and save the records for the window as either a CSV or Excel file;
7. Convert from the original raster format to polygons;
8. Create and display attribute raster and polygon maps.

There is certainly more that can be done in R with the HWSD[1], including integration with other freely-available geographic layers such as digital elevation models and satellite imagery. Readers are referred to the excellent textbook of Bivand et al. [1] from the UseR! Springer textbook series.

The only operation that can not be performed in R[2] is directly working with MS-Access databases (file extension `.mdb`), which is the format in which the HWSD attributes are supplied. However, R can work with SQL databases, as explained in §3. The author has exported the MS-Access database (44.6 Mb) to SQLite format (19.7 Mb) using the MDB Explorer program[3] on OS X. There are similar programs available for other platforms.

Several important R packages are used, including `sp` for spatial data [1, 6], `rgdal` for spatial data import, export and geometric transformation [4], `raster` for working with large raster (grid) image [2], and `RSQLite` for working with the SQLite format relational databases. These must loaded before their first use, as is show in the code.

> **Note:** The code in this document was tested with R version 2.15.0 (2012-03-30) and packages from that version or later running on Mac OS X 10.7.5. The text and graphical output you see here was written as a NoWeb file, including both R code and regular LaTeX source, and then run

[1] and maybe will be, in later versions
[2] as far as I know
[3] http://www.mdbexplorer.com/

through the excellent `knitr` package Version: 0.8 [9] on R to and automatically generated and incorporated into LaTeX. Then the LaTeX document was compiled into the PDF version you are now reading. The R code (file `R_HWSD.R`, supplied with this document) was also generated by `knitr` from the same source document. If you run this R code, or copy code from this document, your output may be slightly different on different versions and on different platforms.

1 Importing the HWSD into R

Task 1 : Download the HWSD database from IIASA. •

The HWSD is found at IIASA[4]. We do not use the Viewer, instead, we download the data for use in R. Three files are provided (Table 1).

Task 2 : Uncompress the compressed file `HWSD_RASTER.zip`. •

This will create a subdirectory `HWSD_RASTER` with three files: the band-interleaved image (`hwsd.bil`, 1.7 Gb), a small file giving the extent and resolution (`hwsd.blw`), and the header (`hwsd.hdr`). The latter two are automatically consulted on data import.

Task 3 : Import the world raster image to R. •

The `raster` package can work with very large images, such as this one, because it only reads the image into memory as necessary, otherwise keeping the image on disk. The `raster` function associates an R object name with the file on disk. The band-interleaved format is known to this command. The `raster` package depends on the `sp` package, which it automatically loads if needed. We load the `raster` package with the `require` function, which only loads a package if it's not already in the workspace:

[4] `webarchive.iiasa.ac.at/Research/LUC/External-World-soil-database/HTML/index.html`

file name	contents	format	size
`HWSD_raster.zip`	Raster soil unit map	band-interleaved image (`.bil`, `.blw`, `.hdr`)	19.7 Mb
`HWSD.mdb`	Soil attribute database	MS Access (`.mdb`)	44.6 Mb
`HWSD_META.mdb`	Soil attribute metadata	MS Access (`.mdb`)	0.8 Mb

Table 1: HWSD database files

```
> require(sp)

Loading required package:  sp

> require(raster)

Loading required package:  raster

> hwsd <- raster("./HWSD_RASTER/hwsd.bil")
```

Task 4 : Examine the raster image's properties. •

The `raster` package provides some useful commands for this, which are self-explanatory:

```
> ncol(hwsd)

[1] 43200

> nrow(hwsd)

[1] 21600

> res(hwsd)

[1] 0.008333 0.008333

> extent(hwsd)

class       : Extent
xmin        : -180
xmax        : 180
ymin        : -90
ymax        : 90

> projection(hwsd)

[1] "NA"
```

This raster is not provided with any projection information (reported as NA, "not available"). We know from the documentation [3] that this is a Plate Carrée[5] projection using the WGS84 datum; it just maps latitude and longitude directly to a grid cell, so that the figure is increasingly distorted towards the poles.

Task 5 : Provide the projection information for the raster database. •

This is a very simple "projection"; we use the `proj4string` function, which uses the syntax of the PROJ4 projection system [5]. We provide a "projection" (here, none, i.e., use the geographic coordinates), the datum, elipse, and translation to WGS84 (yes, all three are needed, and the datum name must be in upper case: WGS84):

```
> require(rgdal)
> (proj4string(hwsd) <- "+proj=longlat +datum=WGS84 +ellps=WGS84
+towgs84=0,0,0")

[1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

[5] French: "square plate"

2 Selecting a region

The entire database is very large; usually we want to work in some region.

2.1 Selecting by a bounding box

The `raster` package can crop an image to an "extent". This can be extracted from the bounding box of any `sp` object, or directly specified using the `extent` function. Here we will select a 2° by 2° tile centred near Nanjing, Jiangsu, China, and covering parts of Jiangsu and Anhui provinces. This same procedure can be used to select any tile of interest. We then crop to this extent with the `crop` function.

```
> hwsd.zhnj <- crop(hwsd, extent(c(117.5, 119.5, 31, 33)))
> nrow(hwsd.zhnj)

[1] 240

> ncol(hwsd.zhnj)

[1] 240

> bbox(hwsd.zhnj)

     min   max
s1 117.5 119.5
s2  31.0  33.0
```

The `unique` function shows the unique values in a raster:

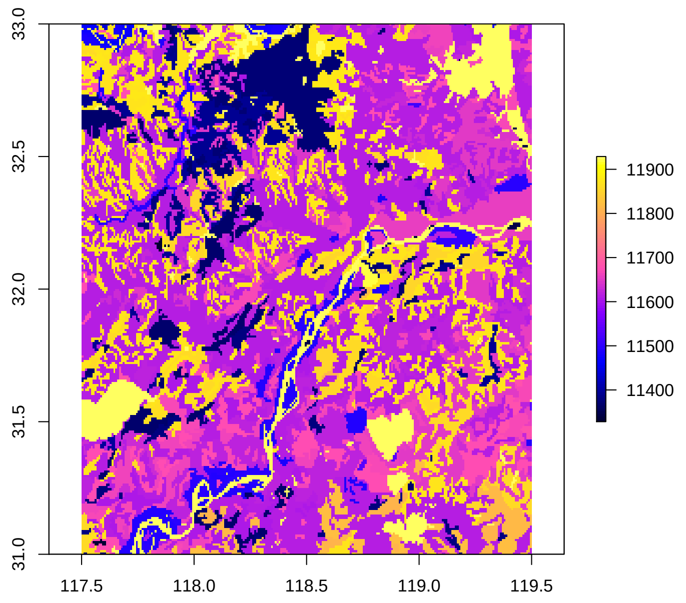
```
> unique(hwsd.zhnj)

 [1] 11328 11331 11341 11365 11367 11368 11372 11373 11375 11376 11377
[12] 11379 11381 11389 11390 11391 11392 11394 11434 11435 11460 11461
[23] 11466 11472 11474 11476 11481 11483 11485 11486 11488 11489 11490
[34] 11491 11492 11493 11495 11499 11501 11513 11535 11604 11605 11609
[45] 11613 11614 11615 11616 11617 11619 11620 11621 11623 11625 11627
[56] 11630 11634 11645 11649 11650 11651 11652 11655 11656 11657 11661
[67] 11663 11665 11667 11668 11671 11672 11673 11675 11677 11678 11679
[78] 11680 11814 11815 11817 11818 11823 11834 11857 11858 11859 11860
[89] 11863 11870 11875 11876 11877 11878 11925 11927 11928 11929
```

This is the only content of the raster database: each pixel has a code, which links to the attribute database, see below.

Task 6 : Display the tile with a suitable colour scheme. •

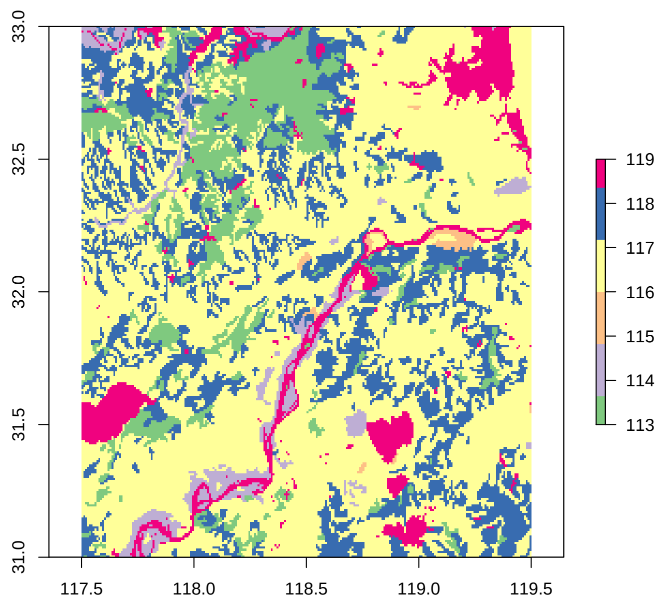
There are too many classes (98) to show with distinct colours. One way is to use a continuous colour ramp:

```
> plot(hwsd.zhnj, col = bpy.colors(length(unique(hwsd.zhnj))))
```



This looks good, since the codes appear to be ordered by similar soils. We can also use just the first three digits of the map unit codes, which presumably are also a meaningful grouping; to remove the 'hundreds' places we use the `%/%` "integer divide" operator . The `RColorBrewer` package provides colour palettes; here we select one that emphasizes differences between classes; we select it with the `brewer.pal` function:

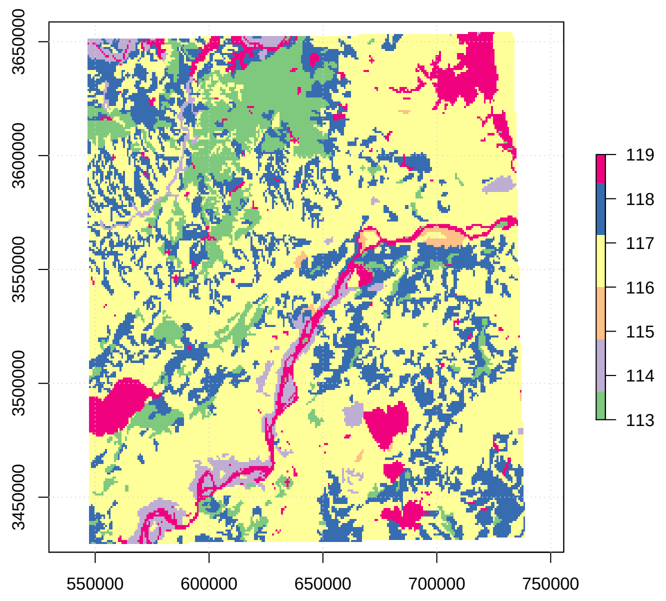
```
> hwsd.zhnj3 <- (hwsd.zhnj%/%100)
> freq(hwsd.zhnj3)

     value  count
[1,]   113   5919
[2,]   114   2152
[3,]   115    273
[4,]   116  32591
[5,]   118  12536
[6,]   119   4129

> require(RColorBrewer)

Loading required package:  RColorBrewer

> plot(hwsd.zhnj3, col = brewer.pal(length(unique(hwsd.zhnj3)), "Accent"))
```



This image is distorted from geographic reality, because it is not projected. We can see the effect of projection, using the `projectRaster` method and specifying a target coordinate reference system (CRS). Note that we use the nearest-neighbour resampling (`method="ngb"`) since this is a classified map.

We first determine the appropriate UTM zone for the centre of the window, recalling that UTM zone 30 is centred on 3° E.

```
> print(paste("UTM zone:", utm.zone <- floor((sum(bbox(hwsd.zhnj3)[1, ])/2 +
+     180)/6) + 1))

[1] "UTM zone: 50"

> proj4string.utm50 <- paste("+proj=utm +zone=", utm.zone, "+datum=WGS84
+units=m +no_defs +ellps=WGS84 +towgs84=0,0,0",
+     sep = "")
> hwsd.zhnj3.utm <- projectRaster(hwsd.zhnj3, crs = proj4string.utm50, method
= "ngb")
> unique(hwsd.zhnj3.utm)

[1] 113 114 115 116 118 119

> (cell.dim <- res(hwsd.zhnj3.utm))

[1] 787 924

> paste("Cell N dimension is ", round((((cell.dim[2]/cell.dim[1]) - 1) * 100,
+     1), "% larger than cell dimension E", sep = "")

[1] "Cell N dimension is 17.4% larger than cell dimension E"

> plot(hwsd.zhnj3.utm, col = brewer.pal(6, "Accent"), asp = 1)
> grid()
```



Notice how the region is now longer in the N–S direction (as shown by the results of the `res` function, above); at 32° N a degree of latitude is larger than a degree of longitude. Also, notice the region is slightly angled with respect to UTM north; this is because the region is not centred on the meridian of zone 50 (117° E = UTM 500 000 E) and the UTM projection is equal-angle but not equal area, becoming most distorted at the edge of the 6° zone.

Now that this is geometrically-correct, we can compute the area covered by each code, and the total area of the tile, here in km^2:

```
> (cell.area <- cell.dim[1] * cell.dim[2]/10^4)

[1] 72.72

> (tmp <- cbind(freq(hwsd.zhnj3.utm)[, 1], freq(hwsd.zhnj3.utm)[, 2] *
cell.area/10^2))

     [,1]    [,2]
[1,]  113  4280.2
[2,]  114  1570.7
[3,]  115   198.5
[4,]  116 23744.1
[5,]  118  9115.3
[6,]  119  2999.7
[7,]   NA  5003.8

> ix <- which(is.na(tmp[, 1]))
> sum(tmp[-ix, 2])

[1] 41909

> rm(cell.dim, cell.area, tmp, ix)
```

The area of a grid cell is about 72 ha; at the equator this would be about 100 ha (1 km^2). The tile covers almost 42 000 km^2. Notice also that there are some NA cells; these are the ones at the edges of the projected image, needed to keep the raster square.

We are done with the generalized map, so remove it:

```
> rm(hwsd.zhnj3.utm)
```

Back to the unprojected image, we can query at any location with the click function. When this is called, click with the mouse at a cell in the displayed image; this will return the coordinates of the point, and, if the optional argument click is set to TRUE, the code at the raster cell is returned. For example, clicking on the approximate peak of the Purple Mountain to the east of downtown Nanjing (118° 50' 30" E, 32° 04' 20" N according to Google Earth). The click function has optional arguments, which we use, to return the raster attribute value:

```
> plot(hwsd.zhnj, col = bpy.colors(length(unique(hwsd.zhnj))))
> xy <- click(hwsd.zhnj, n = 1, id = TRUE, xy = TRUE, type = "p")
```

```
> print(xy)

       x         y     hwsd
  118.83    32.09 11376.00

> (zjs.id <- xy["hwsd"])

 hwsd
11376

> rm(xy)
```

The coordinates are in decimal degrees. The result is the soil map unit code of the pixel.

2.2 Selecting by a bounding polygon

Another way to select a subset of the database is with the polygon boundary of a region, e.g., a country.

Task 7 : Make a SpatialPolygons object from the boundary of the Kingdom of Bhutan. •

We obtain the boundaries of Bhutan from the worldHires dataset of the mapdata package, which was created from what the authors call a "cleaned-up" version of the CIA World Data Bank II data of 2003[6]. We extracted the boundaries with the map function, and then converted them to a SpatialPolygons object with the map2SpatialPolygons function of the maptools package.

> **Note:** These appear to be the boundaries claimed by the country in question as of that date; in the case of Bhutan it appears to include some small border regions also claimed by the People's Republic of China. We do not resolve border disputes, just use this convenient data source to build a bounding polygon.

[6] http://www.evl.uic.edu/pape/data/WDB/

Note: The `fill` argument to the `map` function converts the boundary coordinates into a polygon by joining the last and first points.

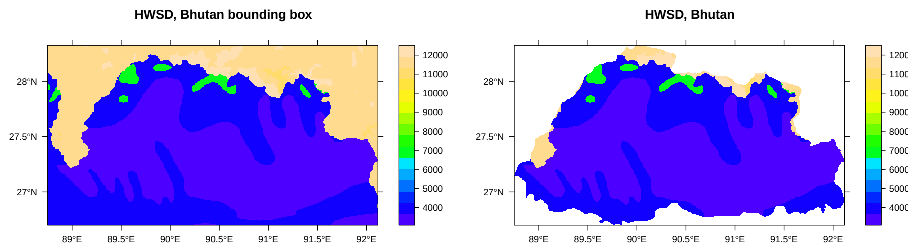
```
> require(maps)

Loading required package:  maps

> require(mapdata)

Loading required package:  mapdata

> str(tmp <- map("worldHires", "Bhutan", fill = TRUE, plot = FALSE))

List of 4
 $ x    : num [1:1666] 91.7 91.7 91.7 91.7 91.7 ...
 $ y    : num [1:1666] 27.8 27.8 27.8 27.8 27.8 ...
 $ range: num [1:4] 88.8 92.1 26.7 28.3
 $ names: chr "Bhutan"
 - attr(*, "class")= chr "map"

> require(maptools)

Loading required package:  maptools

> bhutan.boundary <- map2SpatialPolygons(tmp, IDs = tmp$names, proj4string =
CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"))
> bbox(bhutan.boundary)

    min   max
x 88.75 92.12
y 26.70 28.33

> class(bhutan.boundary)

[1] "SpatialPolygons"
attr(,"package")
[1] "sp"

> rm(tmp)
```

Task 8 : Extract the portion of the HWSD database within this polygon as a raster window. •

Working with a `RasterLayer` object we can only extract rectangular areas. So first we use the bounding box of Bhutan to extract Bhutan and some areas of neighbouring countries, using the `crop` function. We then convert the rectangular window to a `SpatialPixelsDataFrame` object as required by the `sp` package. At that point we can use `overlay` to find the pixels in the country. This command returns `NA` for pixels outside the polygon, and the (single) polygon ID for pixels inside. We then select the pixels that are not `NA`, i.e., have a code. We display maps with the `spplot` method; the `scales` argument here specifies that we want the axes to be drawn.

```
> hwsd.bhutan.box <- crop(hwsd, bbox(bhutan.boundary))
> hwsd.bhutan.box.sp <- as(hwsd.bhutan.box, "SpatialPixelsDataFrame")
> sort(unique(hwsd.bhutan.box.sp$hwsd))

 [1]  3650  3651  3662  3683  3717  3821  3849  3850  6998 11000 11004
[12] 11052 11103 11335 11378 11388 11404 11413 11423 11535 11540 11705
[23] 11710 11711 11718 11719 11721 11724 11727 11730 11732 11736 11740
[34] 11748 11750 11752 11754 11758 11759 11765 11775 11790 11814 11839
[45] 11864 11879 11909 11927 11930 11932
```

```
> spplot(hwsd.bhutan.box.sp, main = "HWSD, Bhutan bounding box", col.regions
= topo.colors(64),
+      scales = list(draw = TRUE))
> ix <- overlay(hwsd.bhutan.box.sp, bhutan.boundary)
> hwsd.bhutan.sp <- hwsd.bhutan.box.sp[!is.na(ix), ]
> sort(unique(hwsd.bhutan.sp$hwsd))

 [1]  3651  3662  3717  3821  3849  6998 11052 11103 11705 11710 11718
[12] 11719 11724 11727 11730 11740 11750 11765 11839 11864 11879 11909
[23] 11930

> spplot(hwsd.bhutan.sp, main = "HWSD, Bhutan", col.regions =
topo.colors(64),
+      scales = list(draw = TRUE))
```



There are only 23 different soil map units in Bhutan.

This can be converted back to a `RasterLayer` object:

```
> hwsd.bhutan <- as(hwsd.bhutan.sp, "RasterLayer")
> rm(hwsd.bhutan.box, hwsd.bhutan.box.sp, hwsd.bhutan.sp, ix)
```

3 Attribute database

There is no R package to read Access databases (file extension `.mdb`). However, R can work with SQL databases[7]; one option is the `RSQLite` package, which provides the interface to an SQL database via the `DBI` package. The author has exported the Access database to SQLite format[8], with file name `HWSD.sqlite`.

> **Note:** As an additional benefit, SQLite databases require much less disk storage than MS-Access databases with the same contents; in this case 12 Mb instead of 44.6 Mb, almost a four-fold reduction!

Task 9 : Connect to the SQLite version of the HWSD attribute databse and list the tables. •

We first load the `RSQLite` package with `require`; this automatically loads the `DBI` package if necessary. We then use the `dbDriver` function

[7] see [8, §4] for a discussion of R and relational databases
[8] using the MDB Explorer program on OS X

to specify the database driver to be used by DBI (in this case, SQLite), and then the dbConnect function with this driver and the name of the database on disk to set up a database *connection*; this variable in the R workspace then refers to the database and is used in every command which queries or manipulates it. The dbListTables function lists the relational tables in the database.

```
> require(RSQLite)

Loading required package:  RSQLite

Loading required package:  DBI

> m <- dbDriver("SQLite")
> con <- dbConnect(m, dbname = "HWSD.sqlite")
> dbListTables(con)

 [1] "D_ADD_PROP"       "D_AWC"            "D_COVERAGE"
 [4] "D_DRAINAGE"       "D_IL"             "D_ISSOIL"
 [7] "D_PHASE"          "D_ROOTS"          "D_SWR"
[10] "D_SYMBOL"         "D_SYMBOL74"       "D_SYMBOL85"
[13] "D_SYMBOL90"       "D_TEXTURE"        "D_USDA_TEX_CLASS"
[16] "HWSD_DATA"        "HWSD_SMU"
```

This database has 17 files, 16 of which are lookup tables of the attribute codes, while the remaining table HWSD_DATA is the list of map units.

Task 10 : Display the structure of the main table. •

SQL syntax used in SQLite is explained, with syntax diagrams, at the SQLite web page[9].

The dbGetQuery function requires a database connection and a query string in SQL format. SQL uses the PRAGMA command to display database structure; we include it in the query string.

> **Note:** Unlike R, SQL is not case-sensitive, so the command strings can be upper, lower, or mixed case. By convention I use upper-case for database names.

```
> dbGetQuery(con, "pragma table_info(HWSD_DATA)")$name

 [1] "ID"                 "MU_GLOBAL"           "MU_SOURCE1"
 [4] "MU_SOURCE2"         "ISSOIL"              "SHARE"
 [7] "SEQ"                "SU_SYM74"            "SU_CODE74"
[10] "SU_SYM85"           "SU_CODE85"           "SU_SYM90"
[13] "SU_CODE90"          "T_TEXTURE"           "DRAINAGE"
[16] "REF_DEPTH"          "AWC_CLASS"           "PHASE1"
[19] "PHASE2"             "ROOTS"               "IL"
[22] "SWR"                "ADD_PROP"            "T_GRAVEL"
[25] "T_SAND"             "T_SILT"              "T_CLAY"
[28] "T_USDA_TEX_CLASS"   "T_REF_BULK_DENSITY" "T_BULK_DENSITY"
[31] "T_OC"               "T_PH_H2O"            "T_CEC_CLAY"
[34] "T_CEC_SOIL"         "T_BS"                "T_TEB"
[37] "T_CACO3"            "T_CASO4"             "T_ESP"
[40] "T_ECE"              "S_GRAVEL"            "S_SAND"
[43] "S_SILT"             "S_CLAY"              "S_USDA_TEX_CLASS"
[46] "S_REF_BULK_DENSITY" "S_BULK_DENSITY"      "S_OC"
[49] "S_PH_H2O"           "S_CEC_CLAY"          "S_CEC_SOIL"
[52] "S_BS"               "S_TEB"               "S_CACO3"
[55] "S_CASO4"            "S_ESP"               "S_ECE"
```

[9] http://www.sqlite.org/lang.html

```
> dbGetQuery(con, "pragma table_info(HWSD_DATA)")$type

 [1] "INTEGER" "INTEGER" "TEXT"    "INTEGER" "INTEGER" "REAL"
 [7] "INTEGER" "TEXT"    "INTEGER" "TEXT"    "INTEGER" "TEXT"
[13] "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER"
[19] "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER"
[25] "INTEGER" "INTEGER" "INTEGER" "INTEGER" "REAL"    "REAL"
[31] "REAL"    "REAL"    "REAL"    "REAL"    "REAL"    "REAL"
[37] "REAL"    "REAL"    "REAL"    "REAL"    "INTEGER" "INTEGER"
[43] "INTEGER" "INTEGER" "INTEGER" "REAL"    "REAL"    "REAL"
[49] "REAL"    "REAL"    "REAL"    "REAL"    "REAL"    "REAL"
[55] "REAL"    "REAL"    "REAL"
```

The field names, data types, and units of measure and lookup tables are explained in detail in [3, §2].

Task 11 : Determine the number of records in the main table. •

We use the `count` SQL function to count selected records (in this case, all of them, as symbolized by the *), and name the result with the `as` SQL operator. We select all records (by omitting a `where` clause).

```
> dbGetQuery(con, "select count(*) as grid_total from HWSD_DATA")

  grid_total
1      48148
```

Task 12 : Display the ID, map unit code, whether it is a soil unit or not, the percent in map unit, the FAO 1990 class code, and the topsoil texture codes, for the first ten records of the main database. •

An SQLite database is not guaranteed to have any particular ordering, so "the first" may vary by implementation. We use the `limit` SQL operator to limit the number of records returned, and specify the fields to return.

> **Note:** The `paste` function with the `collapse` argument collapses a character vector into a single string, with the elements separated by the argument to `paste`.

```
> (display.fields <- c("ID", "MU_GLOBAL", "ISSOIL", "SHARE", "SU_CODE90",
"SU_SYM90",
+     "T_USDA_TEX_CLASS"))

[1] "ID"               "MU_GLOBAL"        "ISSOIL"
[4] "SHARE"            "SU_CODE90"        "SU_SYM90"
[7] "T_USDA_TEX_CLASS"

> tmp <- dbGetQuery(con, paste("select", paste(display.fields, collapse = ",
"),
+     "from HWSD_DATA limit 10"))
```

```
> dim(tmp)

[1] 10  7

> print(tmp[, display.fields])

   ID MU_GLOBAL ISSOIL SHARE SU_CODE90 SU_SYM90 T_USDA_TEX_CLASS
1   1      7001      0   100       201       UR               NA
2   2      7002      0   100       202       HD               NA
3   3      7003      0   100       198       WR               NA
```

```
4   4       7004      0   100         89       HSf               3
5   5       7005      0   100        199        GG              NA
6   6       7006      1    70         35       ANz              11
7   7       7006      1    20         32       ANh              11
8   8       7006      1    10         37       ANi               9
9   9       7007      1    80         35       ANz              11
10  10      7007      1    20         32       ANh              11

> rm(tmp)
```

We see that some map units (e.g., 7001) are non-soil. Some map units (e.g., 7004) have only one component, others (e.g., 7006) have several, with their proportions.

Task 13 : Display the structure of the lookup table for FAO 1990 soil classes. •

From the HWSD documentation we know that the lookup tables have names with pattern `D_*`; the table for FAO 1990 classes is `D_SYMBOL90`. Here we know the table is fairly small, so we read it into memory by selecting all rows; then we examine the structure.

```
> str(dbGetQuery(con, "select * from D_SYMBOL90"))

'data.frame': 193 obs. of  3 variables:
 $ CODE  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ VALUE : chr  "FLUVISOLS" "Eutric Fluvisols" "Calcaric Fluvisols" "Dystric Fluvisols" ...
 $ SYMBOL: chr  "FL" "FLe" "FLc" "FLd" ...
```

Task 14 : Show the map unit record for the pixel identified in the previous section. •

Again we use the `dbGetQuery` function, but now with a query string to find the map unit's record. Note the use of the `paste` function to build a query string with some fixed text (in quotes) and some text taken from a variable, here the soil map unit code saved as variable `zjs.id` during the interactive map query, above.

```
> (tuple <- dbGetQuery(con, paste("select * from HWSD_DATA where MU_GLOBAL =
",
+     zjs.id)))

     ID MU_GLOBAL MU_SOURCE1 MU_SOURCE2 ISSOIL SHARE SEQ SU_SYM74
1 12184     11376      34200         NA      1   100   1     <NA>
  SU_CODE74 SU_SYM85 SU_CODE85 SU_SYM90 SU_CODE90 T_TEXTURE DRAINAGE
1        NA     <NA>        NA      CMd        63         2        4
  REF_DEPTH AWC_CLASS PHASE1 PHASE2 ROOTS IL SWR ADD_PROP T_GRAVEL
1       100         1     NA     NA    NA NA  NA        0       10
  T_SAND T_SILT T_CLAY T_USDA_TEX_CLASS T_REF_BULK_DENSITY
1     42     38     20                9               1.41
  T_BULK_DENSITY T_OC T_PH_H2O T_CEC_CLAY T_CEC_SOIL T_BS T_TEB T_CACO3
1            1.3 1.45       5.1         32         12   38   4.3       0
  T_CASO4 T_ESP T_ECE S_GRAVEL S_SAND S_SILT S_CLAY S_USDA_TEX_CLASS
1       0     2   0.1       19     45     35     20                9
  S_REF_BULK_DENSITY S_BULK_DENSITY S_OC S_PH_H2O S_CEC_CLAY S_CEC_SOIL
1               1.42           1.36  0.5      5.2         35          9
  S_BS S_TEB S_CACO3 S_CASO4 S_ESP S_ECE
1   33   2.6       0       0     2   0.1

> tuple$SU_SYM90
```

```
[1] "CMd"
```

This is the code; we can find the corresponding name in the lookup table:

```
> dbGetQuery(con, paste("select * from D_SYMBOL90 where symbol='",
tuple$SU_SYM90,
+     "'", sep = ""))

  CODE             VALUE  SYMBOL
1   63 Dystric Cambisols    CMd
```

Indeed, the soils of the Purple Mountain area are in general shallow and with low base saturation, so Dystric Cambisols appears to be a correct classifiction.

Now we make a derived soil properties map in the raster window.

Task 15 : Extract a table of the map units in the raster window. •

One way to extract the appropriate records from the map unit database is to make a database table of the list of map units in the window, and then use this as a selection criterion with a JOIN. The dbWriteTable function creates a table; it requires an R data frame as the initial value. From this it infers the table structure.

```
> dbWriteTable(con, name = "WINDOW_ZHNJ", value = data.frame(smu_id =
unique(hwsd.zhnj)),
+     overwrite = TRUE)

[1] TRUE

> dbGetQuery(con, "pragma table_info(WINDOW_ZHNJ)")

  cid      name    type notnull dflt_value pk
1   0 row_names    TEXT       0       <NA>  0
2   1    smu_id INTEGER       0       <NA>  0
```

Now we join on the common field; the new table does not contribute any new fields. We also show how to sort the results, in this case by the FAO 1990 soil map unit symbol:

> **Note:** The select T.* clause selects the fields from the HWSD_DATA table; this is represented by T in the join clause. We do not need the fields from the table with the list of map units in the window, since the HWSD_DATA table has the same codes in field MU_GLOBAL.

```
> records <- dbGetQuery(con, "select T.* from HWSD_DATA as T join WINDOW_ZHNJ
as U on T.mu_global=u.smu_id  order by su_sym90")
> dim(records)

[1] 98 57

> head(records)[, display.fields]

     ID MU_GLOBAL ISSOIL SHARE SU_CODE90 SU_SYM90 T_USDA_TEX_CLASS
1 12469     11661      1   100        22      ACp                9
2 12479     11671      1   100        22      ACp                3
3 12622     11814      1   100        21      ACu               10
4 12623     11815      1   100        21      ACu               10
5 12625     11817      1   100        21      ACu               11
6 12626     11818      1   100        21      ACu                3
```

```
> sort(unique(records$SU_SYM90))

 [1] "ACp" "ACu" "ALf" "ALp" "ANh" "AT"  "ATc" "CMc" "CMd" "CMe" "CMo"
[12] "DS"  "FLc" "FLe" "GLe" "GLk" "GLm" "LP"  "LPd" "LPk" "LVh" "PLd"
[23] "PLe" "RGc" "RGd" "RGe" "UR"  "VRe" "WR"
```

Many of these fields are R **factors** although they were in the relational database as integers or characters; we have to inform R of this:

> **Note:** This is an example of building a valid R command string using `paste` to include both fixed and variable text (which changes each time through the loop), then parsing it with `parse` to build a valid R expression and finally evaluating it with `eval`.

```
> for (i in names(records)[c(2:5, 8:15, 17:19, 28, 45)]) {
+     eval(parse(text = paste("records$", i, " <- as.factor(records$", i,
")",
+         sep = "")))
+ }
```

We can assign the names for factor levels from the metadata lookup tables (not yet implemented here).

Task 16 : Remove fields with no data from the window's attribute table.
•

Some fields are completely undefined in this window. For example, the `MU_SOURCE2` field (second source of data) is not used in data from China; we check this with the `all` function applied to a logical vector created by the `is.na` function and the `!` ("not") logical operator:

```
> ix <- which(names(records) == "MU_SOURCE2")
> all(is.na(records[, ix]))

[1] TRUE
```

We find all these and remove them from the dataframe, thus simplifying the table:

```
> df <- records
> for (i in 1:length(names(records))) if (all(is.na(records[, i]))) df <-
df[-i]
> dim(records)

[1] 98 57

> dim(df)

[1] 98 48

> records <- df
> rm(df)
```

Now we have a table of just the units in our window, with just the defined fields.

This table is a flat file, and can be exported for use in spreadsheets or to be imported into a database program.

Task 17 : Export the map unit table as a comma-separated values (CSV) file. •

The `write.csv` function does just that:

```
> write.csv(records, file = "./HWSD_Nanjing.csv")
```

We can also write direct to Excel files with the `write.xls` function of the `dataframes2xls` package. This has the advantage that it correctly writes R factors as character variables, not as integers.

```
> require(dataframes2xls)
> write.xls(records, file = "./HWSD_Nanjing.xls")
```

```
Loading required package:  dataframes2xls
```

We can see the names of the map units with another table join. To do this, we repeat the previous query but save the results as a new table, which we name `tmp`. We can then use this for the next `join`, to return the map unit codes, symbols and names:

```
> dbGetQuery(con, "create table TMP as select * from HWSD_DATA as T join
WINDOW_ZHNJ as U on T.MU_GLOBAL=U.smu_id  order by su_sym90")

NULL

> head(window.fao90 <- dbGetQuery(con, "select CODE, VALUE, SYMBOL from
D_SYMBOL90 as U join TMP as T  on T.su_code90=U.code"))

  CODE             VALUE  SYMBOL
1   22 Plinthic Acrisols     ACp
2   22 Plinthic Acrisols     ACp
3   21    Humic Acrisols     ACu
4   21    Humic Acrisols     ACu
5   21    Humic Acrisols     ACu
6   21    Humic Acrisols     ACu
```

4 Raster attribute maps

The `raster` package is not suited to working with attribute databases linked to maps; instead the `sp` package is preferred.

Task 18 : Convert the HWSD window to a `SpatialPixelsDataFrame`, and add the attributes from the database. •

The `match` function finds the position of a given value in a lookup table. Here we match the SMU ID from the converted raster to the record in the attribute data frame. We then use that index to extract the proper record for each pixel, and add it to the dataframe. We then display two attribute maps: one categorical and one continuous.

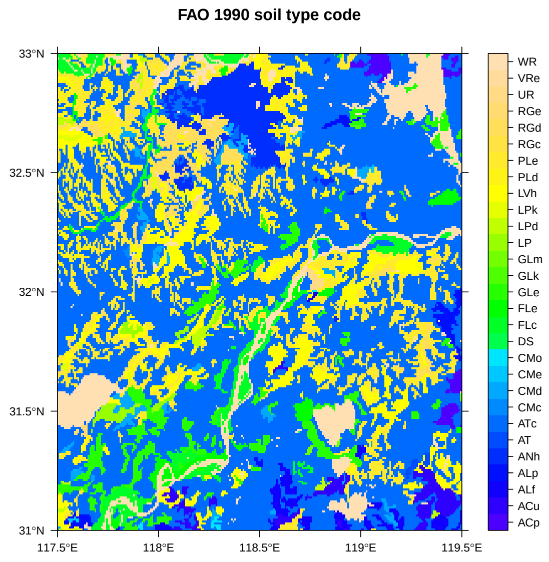
```
> hwsd.zhnj.sp <- as(hwsd.zhnj, "SpatialGridDataFrame")
> str(hwsd.zhnj.sp@data)

'data.frame': 57600 obs. of  1 variable:
 $ hwsd: int  11466 11466 11466 11466 11466 11466 11466 11466 11875 11875 ...

> m <- match(hwsd.zhnj.sp@data$hwsd, records$MU_GLOBAL)
> str(m)
```

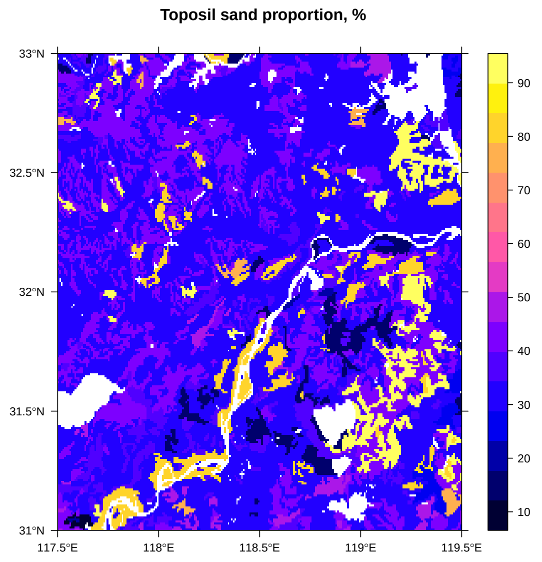
```
 int [1:57600] 54 54 54 54 54 54 54 54 87 87 ...

> hwsd.zhnj.sp@data <- records[m, ]
> rm(m)
```

Task 19 : Display a map of the FAO 1990 soil types, and a map of the topsoil sand proportion. •

We do this with the `spplot` method, specifying the variable to be displayed with the `zcol` argument:

```
> spplot(hwsd.zhnj.sp, zcol = "SU_SYM90", col.regions =
topo.colors(length(levels(hwsd.zhnj.sp$SU_SYM90))),
+     main = "FAO 1990 soil type code", scales = list(draw = TRUE))
> spplot(hwsd.zhnj.sp, zcol = "T_SAND", col.regions = bpy.colors(64), main = "Toposil sand
proportion, %",
+     scales = list(draw = TRUE))
```





5 Polygon maps

Although the HWSD is a raster dataset, it was created from a polygon map. These use much less storage and are generally more attractive. Modellers will want to use the raster but many others will prefer polygons.

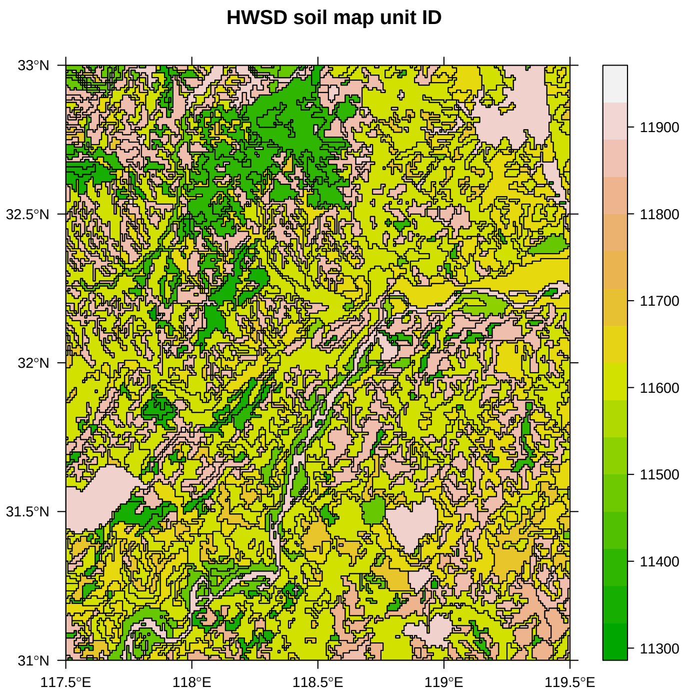
5.1 Raster to polygon

Task 20 : Convert the raster image to a polygon map; each polygon should be labelled with the code of the contiguous pixels that make up the polygon. •

The `raster` package has a function `rasterToPolygons` for this; it depends on yet another package, `rgeos`, to dissolve boundaries between polygons with the same code.

> **Note:** We time the complicated and slow raster-to-polygon operation with the `system.time` function. The conversion requires somewhat less than one minute on the author's system.

```
> require(rgeos)

Loading required package:  rgeos

> system.time(hwsd.zhnj.poly <- rasterToPolygons(hwsd.zhnj, n = 4, na.rm =
TRUE,
+     dissolve = TRUE))

   user  system elapsed
 52.791   0.124  52.929

> class(hwsd.zhnj.poly)

[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"

> str(hwsd.zhnj.poly@data)

'data.frame': 98 obs. of  1 variable:
 $ value: num  11328 11331 11341 11365 11367 ...

> spplot(hwsd.zhnj.poly, col.regions = terrain.colors(64), main = "HWSD soil
map unit ID",
+     scales = list(draw = TRUE))
```



There are only 98 map units (sets of polygons with the same code), as opposed to 57600 raster cells, a very large savings in memory and processing time.

Polygon maps with classes from the sp package are not projected in the same way as raster maps; there is no re-sampling necessary, just a re-projection of all the boundaries. This is accomplished by the spTransform function of the rgdal package. This requires a target Coordinate Reference System (CRS), which is stored in the proj4string "PROJ.4 format CRS specification string" in all sp objects. We defined the appropriate UTM CRS (including elipsoid, datum and offset from the WGS84 elipsoid) for the UTM version of the raster image in §2.1, so we can extract the required CRS from the reprojected image.

```
> proj4string.utm50

[1] "+proj=utm +zone=50+datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"

> hwsd.zhnj.poly.utm <- spTransform(hwsd.zhnj.poly, CRS(proj4string.utm50))
```

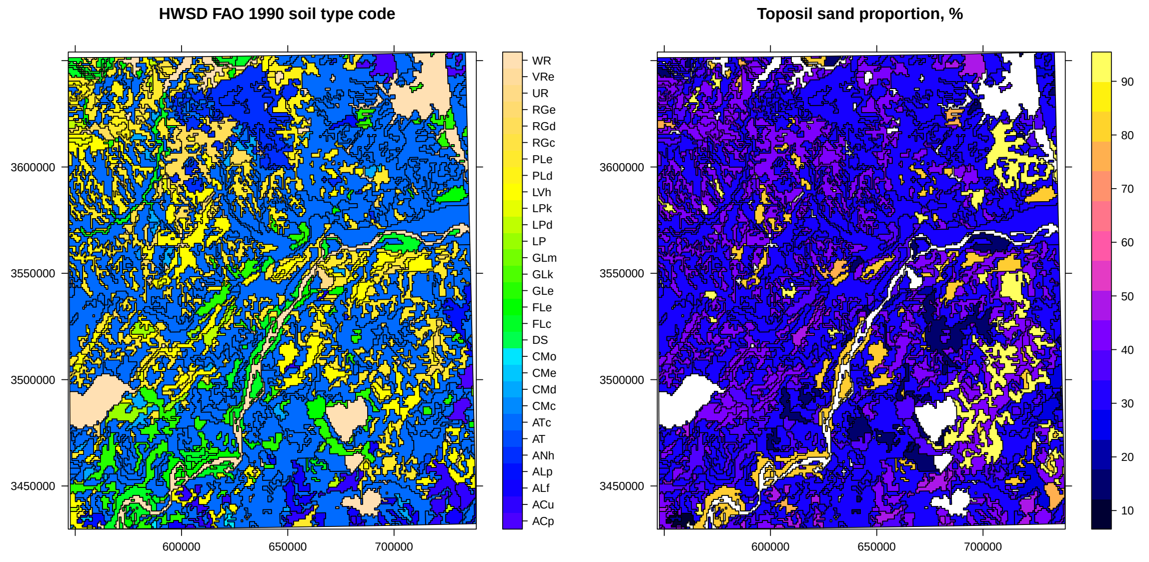
5.2 Polygon attribute maps

So far the polygons just have the soil map unit code.

Task 21 : Add the attribute database to the data frame, and display soil-type and topsoil sand proportion maps. •

The matching of attributes to codes is the same as in §4:

```
> m <- match(hwsd.zhnj.poly.utm$value, records$MU_GLOBAL)
> str(m)

 int [1:98] 42 46 50 76 79 80 11 12 43 44 ...

> hwsd.zhnj.poly.utm@data <- records[m, ]
> spplot(hwsd.zhnj.poly.utm, zcol = "SU_SYM90", col.regions =
topo.colors(length(levels(hwsd.zhnj.sp$SU_SYM90))),
+      main = "HWSD FAO 1990 soil type code", scales = list(draw = TRUE))
> spplot(hwsd.zhnj.poly.utm, zcol = "T_SAND", col.regions = bpy.colors(64),
+      main = "Toposil sand proportion, %", scales = list(draw = TRUE))
```



Some areas have no information for the attribute; these are water bodies, some urban areas, and other unsurveyed areas.

6 Cleanup

Task 22 : Remove temporary tables and disconnect the database. •

```
> dbRemoveTable(con, "WINDOW_ZHNJ")

[1] TRUE

> dbRemoveTable(con, "TMP")

[1] TRUE

> dbDisconnect(con)

[1] TRUE
```

A Troubleshooting

Sometimes an SQLite operation fails or is interrupted, and you are left with an error message that looks like `RS-DBI driver: (connection with pending rows, close resultSet before continuing)`. In that case, the following code will find any pending results and clear them:

```
> (ll <- dbListResults(con))
> if (length(ll) > 0) {
+     res <- ll[[1]]
+     if (!dbHasCompleted(res))
+         dbClearResult(res)
+ }
```

B Extracting a window

Here is a script that can be used to extract any rectangular (longitude and latitude) window from the HWSD, using the techniques presented in this note. The R code extracted from this note includes this as a code chunk. Files are written into a subdirectory `./window/`; this is created if necessary.

```
## R script to extract rectangular windows from the Harmonized World
## Soil Database
## Author: D G Rossiter
## Version: 07-Dec-2012
##### initialize
rm(list = ls())
while (length(dev.list()) > 0) dev.off()
##### function to find a UTM zone
long2UTM <- function(long) {
    return(floor((long + 180)/6) + 1)%%60
}
##### function to extract and format one rectangular window
## arguments:
## bbox: a `raster'-style extent argument, i.e., a vector of xmin,
## xmax, ymin, ymax
## name: a suffix for the file names (image, UTM image, csv, excel
## files, PDF of map unit codes) names start with 'HWSD_', in
## subdirectory 'window' and area name
## the image `hwsd' and the database must be already available in the
## environment
extract.one <- function(bbox, name = "window") {
    print(paste("Area name: ", name, "; bounding box: [", paste(bbox,
collapse = ", "),
        "]", sep = ""))
    # extract the window
    dir.create(paste("./window/", name, sep = ""), showWarnings = FALSE)
    setwd(paste("./window/", name, sep = ""))
    hwsd.win <- crop(hwsd, extent(bbox))
    # find the zone for the centre of the box
    print(paste("Central meridian:", centre <- (bbox[1] + bbox[2])/2))
    utm.zone <- long2UTM(centre)
    print(paste("UTM zone:", utm.zone))
    # make a UTM version of the window
    hwsd.win.utm <- projectRaster(hwsd.win, crs = (paste("+proj=utm +zone=",
        utm.zone, "+datum=WGS84 +units=m +no_defs +ellps=WGS84
+towgs84=0,0,0",
        sep = "")), method = "ngb")
    print(paste("Cell dimensions:", paste((cell.dim <- res(hwsd.win.utm)),
        collapse = ", ")))
    # write the unprojected and projected raster images to disk
    eval(parse(text = paste("writeRaster(hwsd.win, file='./HWSD_", name,
        "', format='EHdr', overwrite=TRUE)", sep = "")))
    eval(parse(text = paste("writeRaster(hwsd.win.utm, file='./HWSD_", name,
        "_utm', format='EHdr', overwrite=TRUE)", sep = ""))
    # extract attributes for just this window
    dbWriteTable(con, name = "WINDOW_TMP", value = data.frame(smu_id =
unique(hwsd.win)),
        overwrite = TRUE)
    records <- dbGetQuery(con, "select T.* from HWSD_DATA as T join
WINDOW_TMP as U on T.mu_global=u.smu_id  order by su_sym90")
    dbRemoveTable(con, "WINDOW_TMP")
    # convert to factors as appropriate
    for (i in names(records)[c(2:5, 8:15, 17:19, 28, 45)]) eval(parse(text =
paste("records$",
        i, " <- as.factor(records$", i, ")", sep = "")))
    # remove all-NA fields
    fields.to.delete <- NULL
    for (i in 1:length(names(records))) if (all(is.na(records[, i]))) {
        fields.to.delete <- c(fields.to.delete, i)
    }
    if (length(fields.to.delete > 1))
        records <- records[, -fields.to.delete]
    print(paste("Dimensions of attribute table: ", paste(dim(records),
collapse = ", "),
        " (records, fields with data)", sep = ""))
    # write attribute table in CSV formats
    eval(parse(text = paste("write.csv(records, file='./HWSD_", name,
".csv')",
        sep = "")))
    # make a spatial polygons dataframe, add attributes
    print(system.time(hwsd.win.poly <- rasterToPolygons(hwsd.win, n = 4,
        na.rm = TRUE, dissolve = TRUE)))
    # transform to UTM for correct geometry
    hwsd.win.poly.utm <- spTransform(hwsd.win.poly,
CRS(proj4string(hwsd.win.utm)))
    m <- match(hwsd.win.poly.utm$value, records$MU_GLOBAL)
    hwsd.win.poly.utm@data <- records[m, ]
    # plot the map unit ID
    print(paste("Number of legend categories in the map:", lvls <-
length(levels(hwsd.win.poly.utm$MU_GLOBAL))))
    p1 <- spplot(hwsd.win.poly.utm, zcol = "MU_GLOBAL", col.regions =
terrain.colors(lvls),
        main = paste("HWSD SMU code"), sub = paste("UTM zone", utm.zone),
        scales = list(draw = TRUE))
    eval(parse(text = paste("pdf(file='./HWSD_", name, "_SMU_CODE.pdf')",
        sep = "")))
    print(p1)
    dev.off()
    setwd("../..")
}  # end extract.one
##### main program
##### ####################################################
## read in HWSD raster database, assign CRS
require(sp)
require(raster)
hwsd <- raster("./HWSD_RASTER/hwsd.bil")
require(rgdal)
proj4string(hwsd) <- "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
## establish connection to attribute database
require(RSQLite)
m <- dbDriver("SQLite")
con <- dbConnect(m, dbname = "HWSD.sqlite")
## other packages to be used in the function
require(rgeos)
## call the function for each window we want to extract
extract.one(c(-77, -75, 41, 43), "Twin Tiers")
## clean up
dbDisconnect(con)
```

C Extracting a country

Here is a script that can be used to extract any rectangular window from the HWSD, using the techniques presented in this note. The R code extracted from this note includes this as a code chunk. The country name is as given in the CIA world database; this was explained in §2.2. Files are written into a subdirectory `./country/<country name>`; this is created if necessary.

```
## R script to extract a county from the Harmonized World Soil Database
## Author: D G Rossiter
## Version: 07-Dec-2012
##### initialize
##### ########################################################
rm(list = ls())
##### functions
##### ########################################################
## function to extract and format one country
## arguments:
## name: a country name, to extract the appropriate bounding polygon(s)
## this name must match the CIA database, see help(wordHires) will also
## be used a suffix for the file names (image, csv attributes) names
## start with 'HWSD_Country_', in subdirectory 'country' and area name
## the image `hwsd' and the SQLLite database must be already available
## in the environment
extract.one <- function(name = "") {
    print(paste("Country:", name))
    dir.create(paste("./country/", name, sep = ""), showWarnings = FALSE)
    setwd(paste("./country/", name, sep = ""))
    tmp <- map("worldHires", name, fill = TRUE, plot = FALSE)
    boundary <- map2SpatialPolygons(tmp, IDs = tmp$names, proj4string =
CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"))
    bbox <- bbox(boundary)
    print(paste("Bounding box: [", paste(t(bbox), collapse = ", "), "]",
        sep = ""))
    # extract the window
    hwsd.win <- crop(hwsd, extent(bbox))
    hwsd.win.sp <- as(hwsd.win, "SpatialGridDataFrame")  # overlay only works
for sp objects
    ix <- overlay(hwsd.win.sp, boundary)
    hwsd.win.sp <- hwsd.win.sp[!is.na(ix), ]
    hwsd.win <- as(hwsd.win.sp, "RasterLayer")  # convert back to raster
    # find the zone for the centre of the box
    print(paste("Central meridian:", centre <- (bbox[1] + bbox[2])/2))
    # write unprojected raster window image to disk
    eval(parse(text = paste("writeRaster(hwsd.win, file='./HWSD_raster_",
        name, "', format='EHdr', overwrite=TRUE)", sep = "")))
    # extract attributes for just this window
    dbWriteTable(con, name = "WINDOW_TMP", value = data.frame(smu_id =
unique(hwsd.win)),
        overwrite = TRUE)
    records <- dbGetQuery(con, "select T.* from HWSD_DATA as T join
WINDOW_TMP as U on T.mu_global=u.smu_id  order by su_sym90")
    dbRemoveTable(con, "WINDOW_TMP")
    # convert to factors as appropriate
    for (i in names(records)[c(2:5, 8:15, 17:19, 28, 45)]) eval(parse(text =
paste("records$",
        i, " <- as.factor(records$", i, ")", sep = "")))
    # include all fields
    print(paste("Dimensions of attribute table: ", paste(dim(records),
collapse = ", "),
        " (records, fields)", sep = ""))
    # write attribute table in CSV format
    eval(parse(text = paste("write.csv(records, file='./HWSD_attributes_",
        name, ".csv')", sep = "")))
    setwd("../..")
}  # end extract.one
```

```
##### main program
##### ##########################################################
## read in HWSD raster database, assign CRS
require(sp)
require(raster)
hwsd <- raster("./HWSD_RASTER/hwsd.bil")
require(rgdal)
proj4string(hwsd) <- "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
## establish connection to attribute database
require(RSQLite)
m <- dbDriver("SQLite")
con <- dbConnect(m, dbname = "HWSD.sqlite")
## packages for country boundaries
require(maps)
require(mapdata)
require(maptools)
## call the function for each window we want to extract
extract.one("Sri Lanka")
## clean up
dbDisconnect(con)
```

References

[1] Roger S. Bivand, Edzer J. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*. Springer, 2008. ISBN 978-0-387-78170-9. URL http://www.asdar-book.org/. 1

[2] Robert J. Hijmans and Jacob van Etten. *raster: Geographic analysis and modeling with raster data*, 2012. URL http://CRAN.R-project.org/package=raster. R package version 2.0-12. 1

[3] IIASA; FAO; ISRIC; ISS-CAS; JRC. *Harmonized World Soil Database (version 1.2)*. FAO and IIASA, Feb 2012. URL www.iiasa.ac.at/Research/LUC/External-World-soil-database/HTML/HWSD_Data.html. 1, 3, 12

[4] Timothy H. Keitt, Roger Bivand, Edzer Pebesma, and Barry Rowlingson. *rgdal: Bindings for the Geospatial Data Abstraction Library*, 2012. URL http://CRAN.R-project.org/package=rgdal. R package version 0.7-20. 1

[5] osgeo.org. PROJ.4. URL https://trac.osgeo.org/proj/. 3

[6] Edzer J. Pebesma and Roger S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2):9‘Ă‘Ş13, 2005. 1

[7] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0. 1

[8] R Development Core Team. *R Data Import/Export*. The R Foundation for Statistical Computing, version 2.15.2 (2012-10-26) edition, 2012. ISBN ISBN 3-900051-10-0. URL http://cran.r-project.org/doc/manuals/R-data.pdf. 10

[9] Yihui Xie. knitr: Elegant, flexible and fast dynamic report generation with R, 2011. URL http://yihui.name/knitr/. 2

Index of R Concepts