Maximally-Smooth Map Interpolation

Carl Schmertmann

7 May 2019

```
rm(list=ls())
graphics.off()
if (.Platform$OS.type == 'windows') windows(record=TRUE, height=10, width=10)

library(tidyverse)
library(rgdal)
library(sf)
library(sp)
library(spdep)
library(Matrix)
library(igraph)

rm(list=ls())
graphics.off()
```

Interpolation Problem

We often produce maps in which values (like population, life expectancy, pollution levels, ...) are associated with *large, irregular* polygons (like census tracts, zip codes, counties, ...). My objective here is to build a smooth map over a grid of *small, regular* grid that exactly matches the polygon values.

An example polygon map

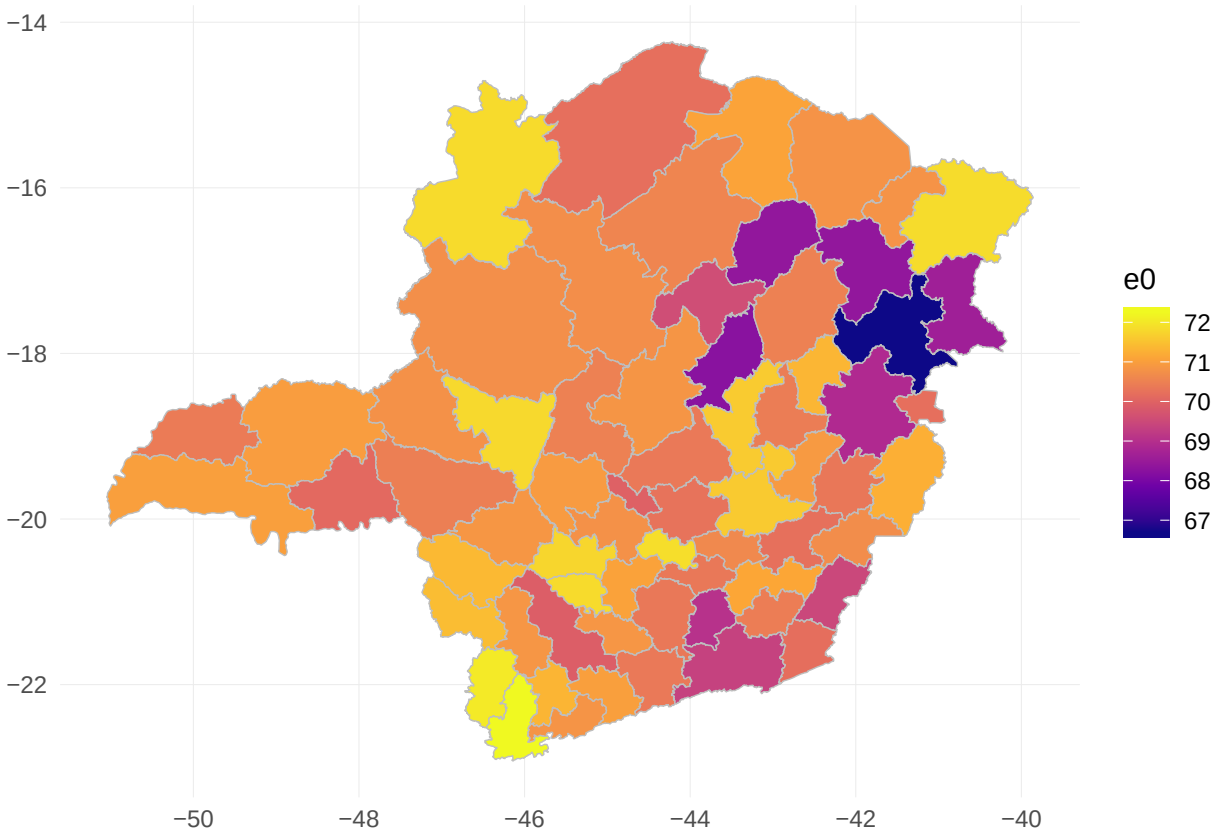
As an example, here is a map of male life expectancy estimates for 66 Brazilian census areas called *microregions*, in the state of Minas Gerais in 2010. To give a sense of scale, Minas Gerais has about the same land area as France.

```
############################################################
# get the geographical information and e0 estimates if necessary

if (!file.exists('muni.df.csv')) {
  file_url = 'http://topals-mortality.schmert.net/data/muni.df.csv'
  download.file(url = file_url, destfile = 'muni.df.csv', mode='wb')
}

if (!file.exists('e0_microregion_summary.csv')) {
  file_url = 'http://mortality-subregistration.schmert.net/data/e0_microregion_summary.csv'
  download.file(url = file_url, destfile = 'e0_microregion_summary.csv', mode='wb')
}

# filter out selected geography

geo = read.csv('muni.df.csv') %>%
          filter(ufabb == 'MG')  %>%
          select(ufabb,munid, municode, muniname, microcode, microname)
```

```
e = read.csv('e0_microregion_summary.csv') %>%
      filter(coverage_model=='partial',
             pctile==50,
             sex=='m',
             microcode %in% geo$microcode) %>%
      select(ufabb, microcode, sex, e0)

geo = left_join(geo, e)

############################################################
# get the shapefile data if necessary

prefix = 'http://mortality-subregistration.schmert.net/data/brasilmun2010_new'

for (suffix in c('.shp','.dbf','.shx')) {
  local_filename = paste0('brasilmun2010_new', suffix)
  if (!file.exists(local_filename)) {
    file_url       = paste0(prefix,suffix)
    download.file(url = file_url, destfile = local_filename, mode='wb')
  } # if
} # for

############################################################

## municipio map
municipality_map = st_read(dsn='.', layer='brasilmun2010_new') %>%
       filter(NAME2_ == 'MINAS GERAIS') %>%
       select('ID_', contains('NAME'), geometry) %>%
       mutate(municode = floor( as.numeric( as.character(ID_)) / 10)) %>%
       left_join(geo)
```

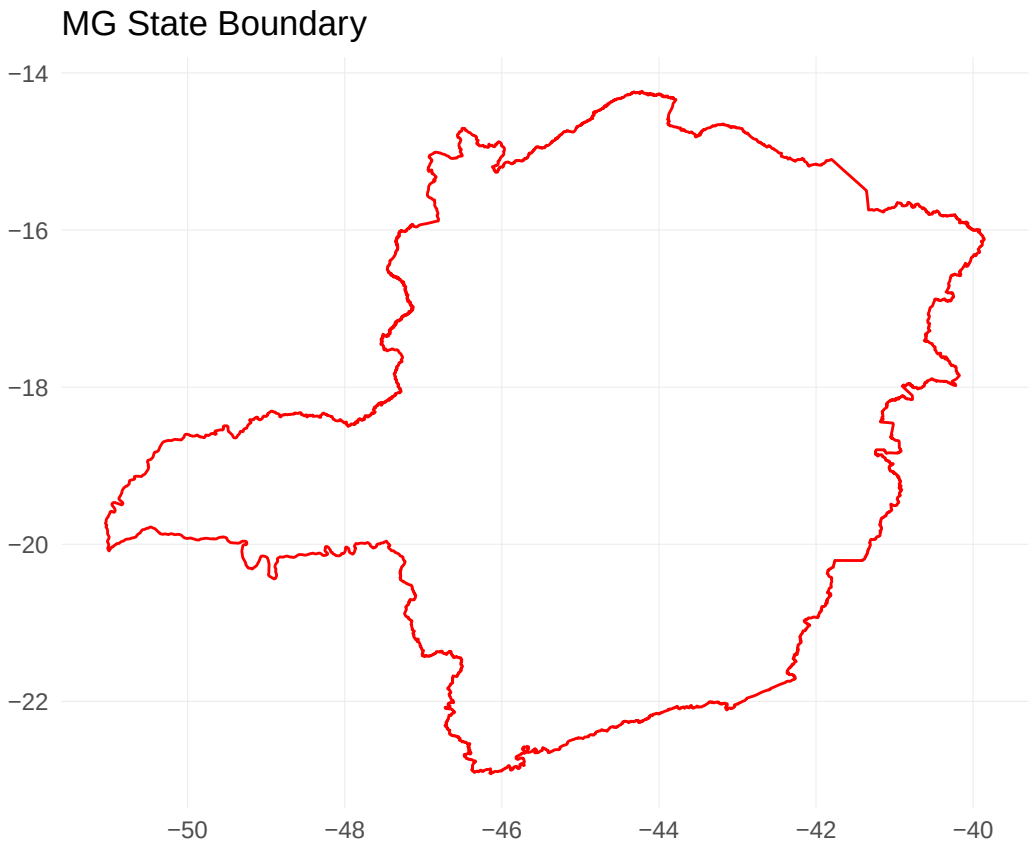
```
## Reading layer `brasilmun2010_new' from data source `C:\Users\User\Documents\GitHub\bonecave\poly-to-
## Simple feature collection with 5564 features and 79 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -73.99139 ymin: -33.75201 xmax: -34.7937 ymax: 5.27179
## epsg (SRID):    NA
## proj4string:    NA
```

```
## microregion map
microregion_map = municipality_map %>%
        group_by(microcode) %>%
        summarize(e0=mean(e0))

P = nrow(microregion_map)

ggplot(data=microregion_map, aes(fill=e0)) +
  geom_sf(color='grey',lwd=0.25) +
  scale_fill_viridis_c(option='C') +
  theme_minimal()
```

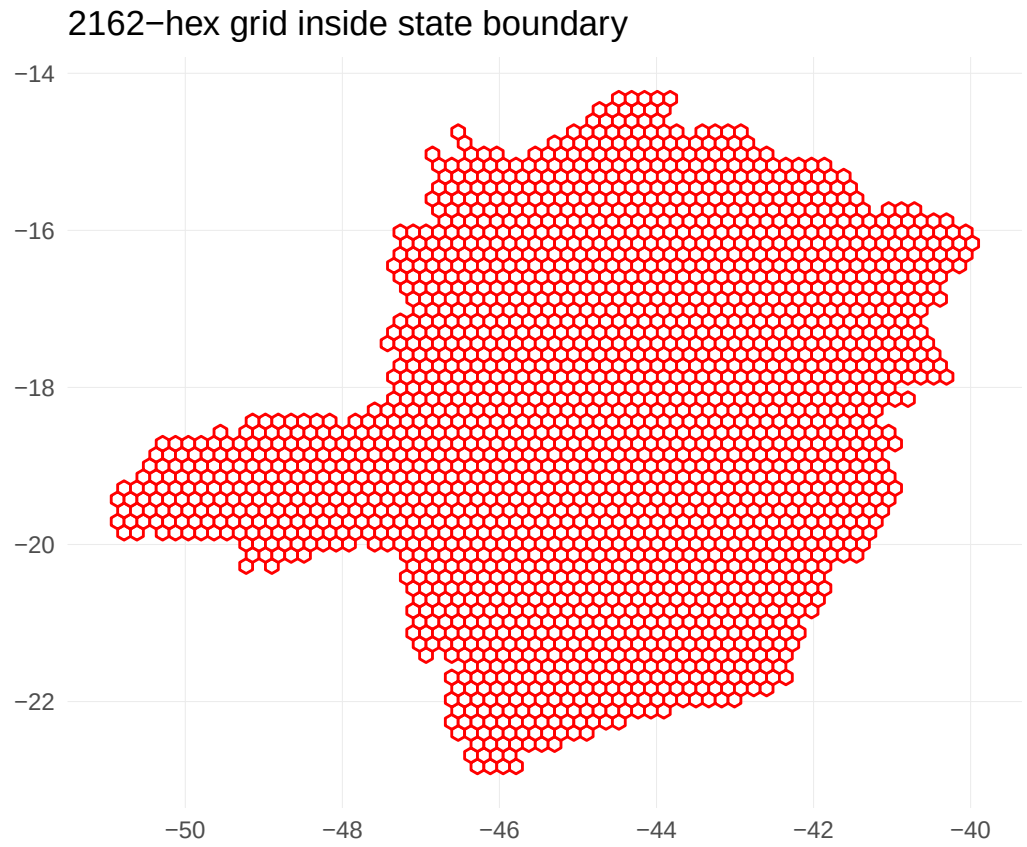


Create a finer grid of hexagons to match to the polygons

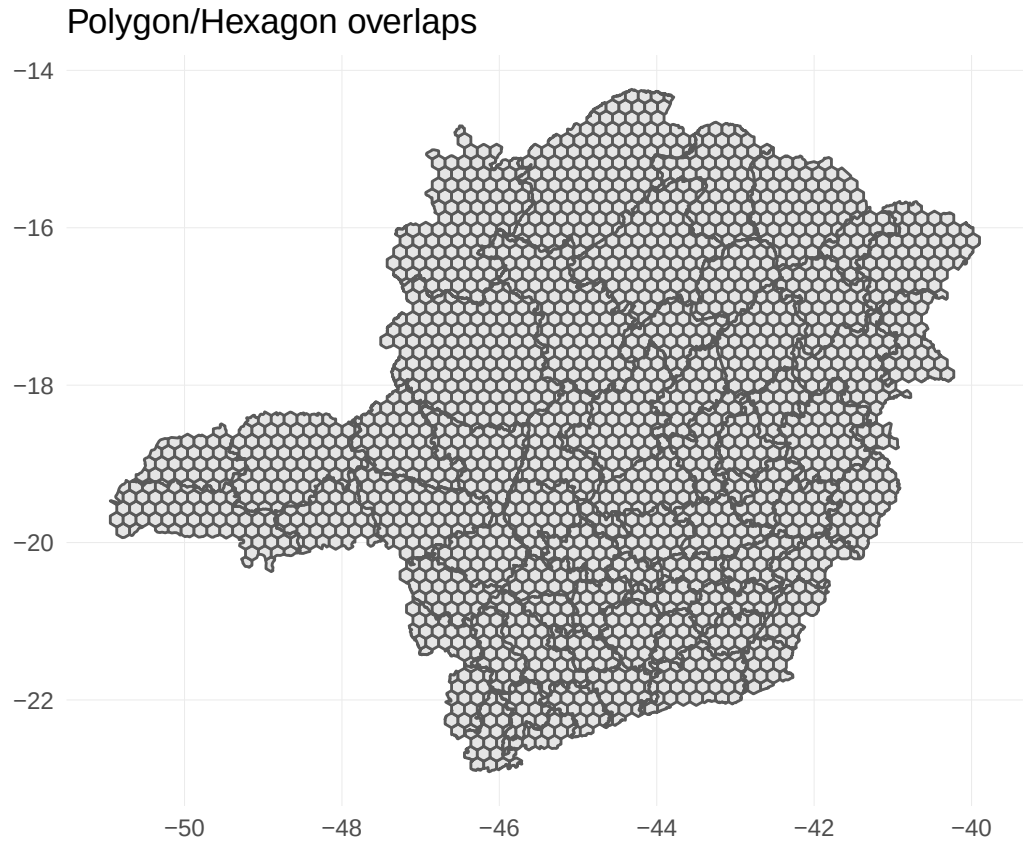
```
## make a single exterior boundary for the state

state_boundary = municipality_map %>%
                    group_by(NAME2_) %>%
                    summarize()

ggplot(data=state_boundary) +
  geom_sf(fill=NA, color='red') +
  labs(title='MG State Boundary') +
  theme_minimal()
```




```
nhex = 2500   # approx number of hexagons that would cover the state

# generate a grid of hexagons that covers the state
hex_points = spsample(as_Spatial(state_boundary),
                      type='hexagonal',
                      n=nhex)

hexagons = HexPoints2SpatialPolygons(hex_points) %>%
            st_as_sf()

H = nrow(hexagons)

ggplot(data=hexagons) +
   geom_sf(fill='white', color='red') +
   labs( title=paste0(H,'-hex grid inside state boundary' )) +
   theme_minimal()
```




```
## examine overlaps
overlap_map = st_intersection(hexagons, microregion_map)

ggplot(data=overlap_map) +
  geom_sf() +
  labs(title='Polygon/Hexagon overlaps') +
  theme_minimal()
```



Construct matrix descriptions of the maps

Hex-to-Polygon weight matrix

We find the areas of each of the polygon/hexagon intersections, and then fill in a $P \times H$ weight matrix W such that the cell on row p and column h contains the fraction of polygon p that lies in hexagon h. Most elements of W equal zero. Row sums of W equal one.

The condition to ensure that a set of values $\theta \in \mathbb{R}^H$ assigned to hexagons matches the polygon values $y \in \mathbb{R}^P$ is

$$y = W\theta$$

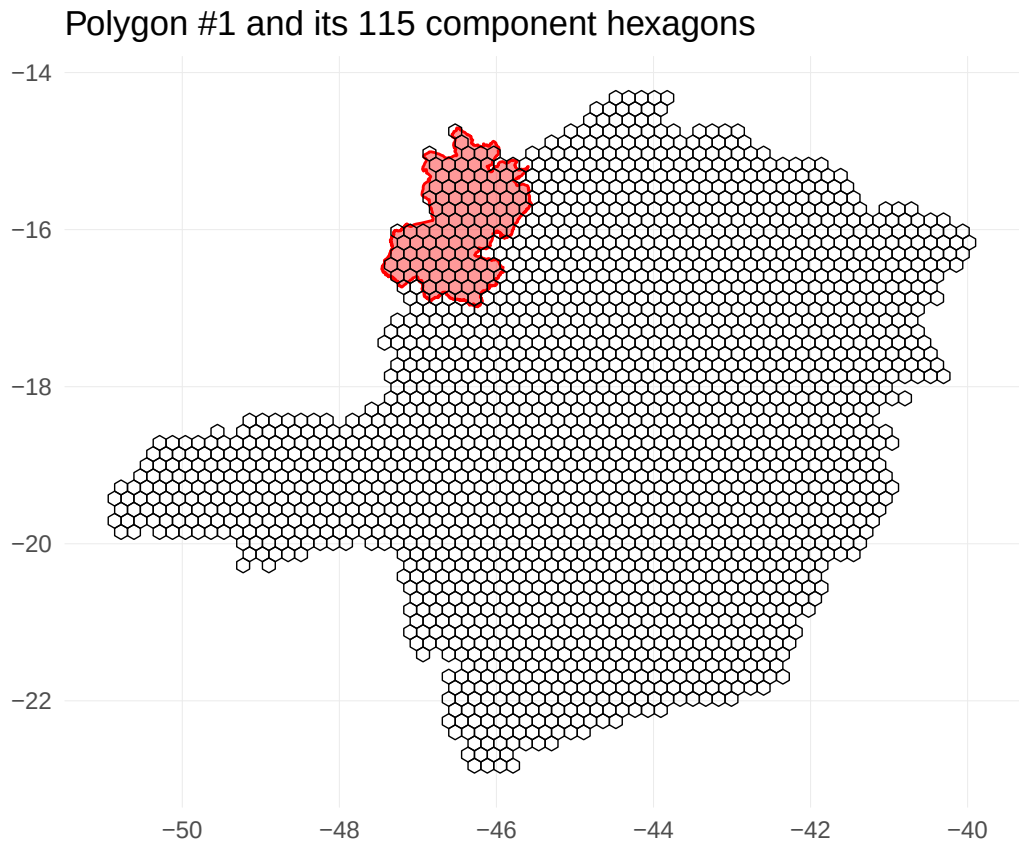
Note that this weights by *area*, which may be an odd criterion for some aggregation problems. (It's a little strange for life expectancy, for instance.)

```
A = st_area(overlap_map)  # areas of overlapping (polygon,hexagon) pairs

Imat = as.matrix(st_intersects(microregion_map, hexagons))
candidates = which(Imat, arr.ind = TRUE)
colnames(candidates) = c('microregion','hexagon')

W             = matrix(0, P, H)
W[candidates] = A
W             = prop.table(W,1)
```

```
ihex1 = which( W[1,] > 0 )
nhex1 = length(ihex1)
mina1 = min( W[1,ihex1])
maxa1 = max( W[1,ihex1])
```

As an example, the first polygon (shown below) comprises 115 whole and partial hexagons, out of the total set of 2162 hexagons. The first row of W therefore contains 2047 zero values and 115 positive values, where positive values give the fractions of polygon 1 that belong to the corresponding hexagons. Hexagon # 1542, for example, contains a fraction 0.01324 of the area of polygon #1, so $W[1, 1542] = 0.01324$. And so forth.

```
# show polygon 1 and its hexagons
ggplot(data=microregion_map[1,]) +
   geom_sf(lwd=.50, color='red',fill='red',alpha=.40) +
   geom_sf(data=hexagons, lwd=.25, fill=NA,color='black') +
   labs(title=paste('Polygon #1 and its',nhex1,'component hexagons')) +
   theme_minimal()
```



Quadratic differencing matrix

To describe the *smoothness* of a set of hexagon values $\theta \in \mathbb{R}^H$ we take the sum of the squares of differences between pairs of adjacent hexagons.

$$Q(\theta) = \sum_i \sum_j A_{ij}(\theta_i - \theta_j)^2$$

where A_{ij} is a (0,1) dummy equal to 1 if hexagons i and j are adjacent to one another on the map. The closer $Q(\theta)$ is to zero, the smoother the map.

A matrix expression for $Q(\theta)$ is

$$Q(\theta) = \theta' K \theta$$

where K is an $H \times H$ matrix such that

$$K_{ij} = \begin{cases} -1 & \text{if i and j} \neq \text{i are adjacent} \\ \#\text{ adjacent hexagons} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Now we construct K. This gets tricky because of the possibility that the hex map will have "islands" of one or more hexagons that are connected to one another, but not to any other part of the map. In these cases we add some extra adjacencies to make sure that all hexagons are connected. (i.e., we want ensure that we could get from any hexagon to any other in a finite # of steps that cross one adjacency per step).

```
##############################

## construct an adjacency list for the hexagons,
## SUCH THAT THERE ARE NO DISJOINT SUBGRAPHS (=unconnected
##   hexagons or sets of hexagons)

adj      = poly2nb(hexagons)
orig_adj = adj
agraph   = graph_from_adj_list(adj, mode='total')   # an igraph

disjoint = !is_connected(agraph)
if (disjoint) {
  cl  = clusters(agraph)
  ncl = cl$no
  icl = cl$membership

  # for each pair of subgraphs, find the pair of member hexagons
  # that are closest and add an artificial adjacency (think of this
  # as adding a ferry route between islands)

  print(paste('-----',ncl,'disjoint subgraphs of hexagons'))
  print(table( icl ))

  for (i in 1:ncl) {
      g1 = hexagons %>%
              mutate(hexnum=seq(icl)) %>%
              filter(icl == i)
      g2 = hexagons %>%
              mutate(hexnum=seq(icl)) %>%
              filter(icl != i)

      dist = st_distance(g1,g2)
      ix   = as.numeric( which( dist==min(dist), arr.ind=TRUE) )
      ii   = g1$hexnum[ix[1]]
      jj   = g2$hexnum[ix[2]]
      # (ii,jj) are the closest pair: add an adjacency

      print(paste('-- adding adjacency',ii,'<>',jj))
```

```
      tmp = union(adj[[ii]], jj)
      adj[[ii]] = tmp[tmp!=0]  # remove any "0"s from poly2nb
      tmp = union(adj[[jj]], ii)
      adj[[jj]] = tmp[tmp!=0]  # remove any "0"s from poly2nb
    }
} # if disjoint


## make quadratic penalty matrix for the hexagons,
## based on adjacencies

K       = matrix(0, H, H)

diag(K) = sapply(adj, length)   # the i=j elements

for (i in seq(adj)) { K[i, adj[[i]]] = -1 }
```

Solve for the smoothest map that replicates the polygon values

When there are more hexagons than polygons ($H > P$), then there are an infinite number of hexmaps $\theta \in \mathbb{R}^H$ that match the polygons' $y \in \mathbb{R}^P$ values. However, one map among these is uniquely the smoothest. This is the solution to a quadratic programming problem:

$$\begin{aligned} min_\theta \quad & \theta' K \theta \\ \text{subject to} \quad & W\theta = y \end{aligned}$$

which can be rewritten as an unconstrained Langrangian problem

$$min_\theta \, max_\lambda \quad L(\theta, \lambda) = \theta' K \theta + \lambda' (W\theta - y)$$

and solved as a linear system

$$\begin{bmatrix} 2K & W' \\ W & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix}$$

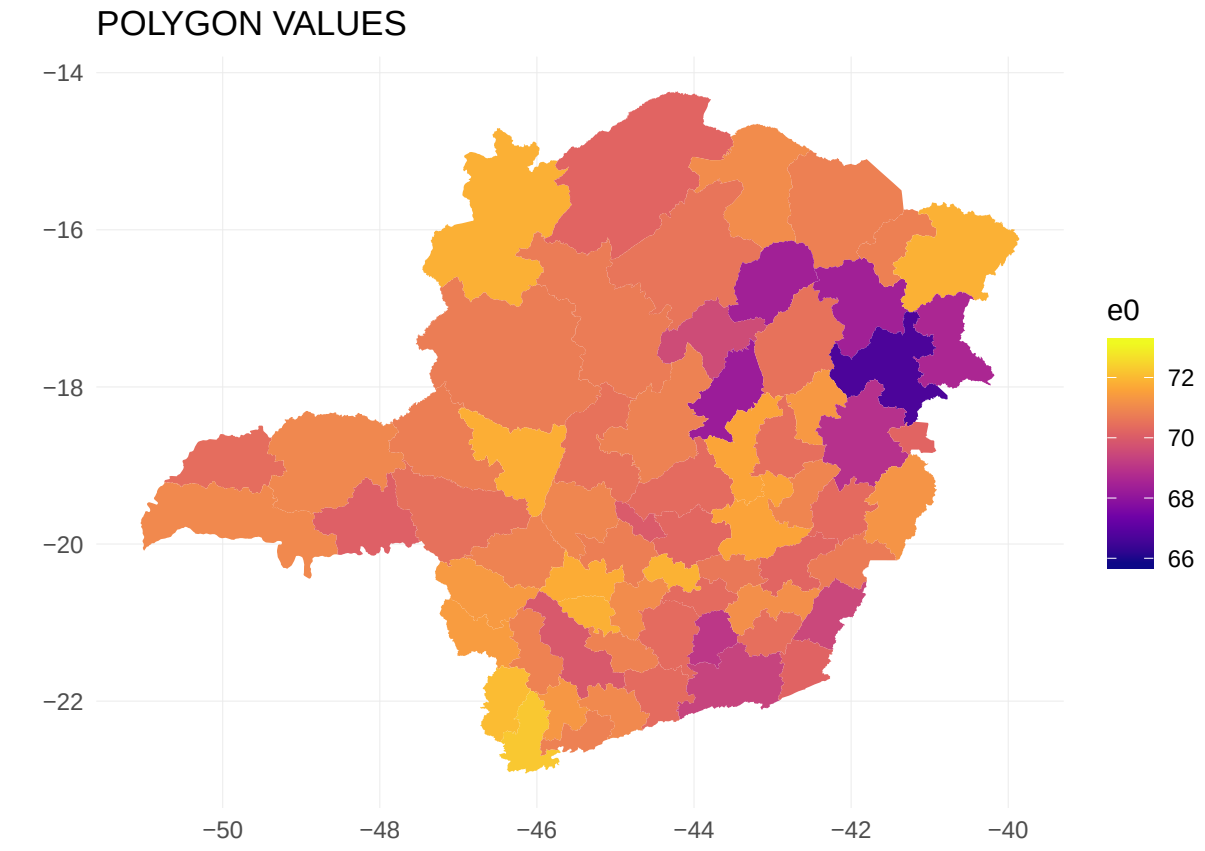
```
## use sparse matrices to solve the quadratic programming problem:
##    min z'Kz st Wz = polygon_values
## where z'Kz is the sum of sq diffs between adjacent hex values
##
## This will find the smoothest hexmap that exactly matches the
## polygon values when averaged by area

polygon_values = microregion_map$e0   # this is the vector of 'y' values

A = Matrix( rbind( cbind( 2*K, t(W)),
                   cbind(   W, diag(0,P)) ) )
b = Matrix( c(rep(0,H), polygon_values), ncol=1)


## experiment: trim the smoothest map slightly
tmp = (solve(A,b))[1:H]
q   = quantile(tmp,c(.01,.99))
optimal_hex_values = pmin( pmax(tmp,q[1]), q[2])

hexagons$e0 = optimal_hex_values
```

Now we have the smoothest map that exactly replicates the polygon values.

For example, the target value in our first polygon is y_1 =**71.82**. The values in the 115 hexagons that overlap that polygon are $[70.899, 70.81, 71.334 \ldots 72.086, 72.858, 73.011]$, and the weighted average of hex values for polygon 1 is [1st row of W]θ = **71.82**.

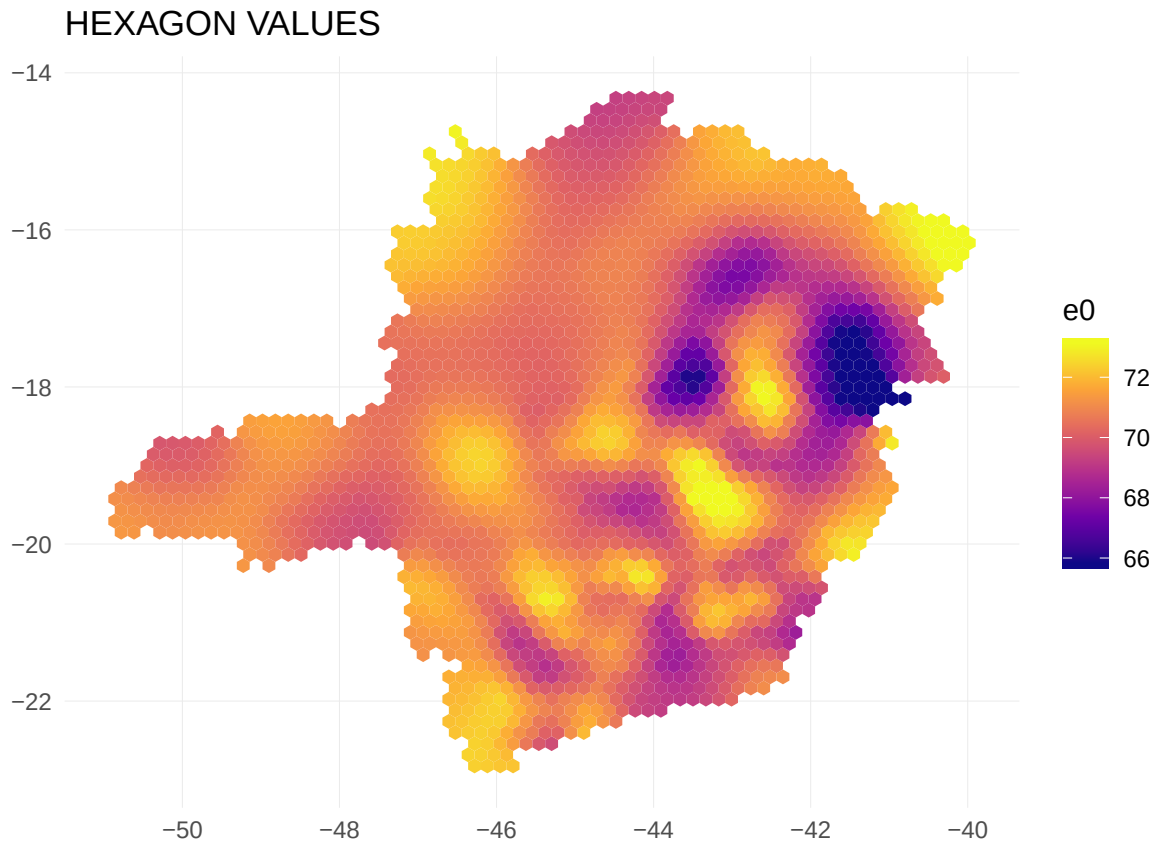
Original polygon and interpolating hexagon maps

```
## plot the smoothest hex map

polymap =
  ggplot(data=microregion_map, aes(fill=e0)) +
  geom_sf(color=NA) +
  scale_fill_viridis_c(limits=range(optimal_hex_values), option='C') +
  theme_minimal() +
  labs(title='POLYGON VALUES')

print(polymap)
```

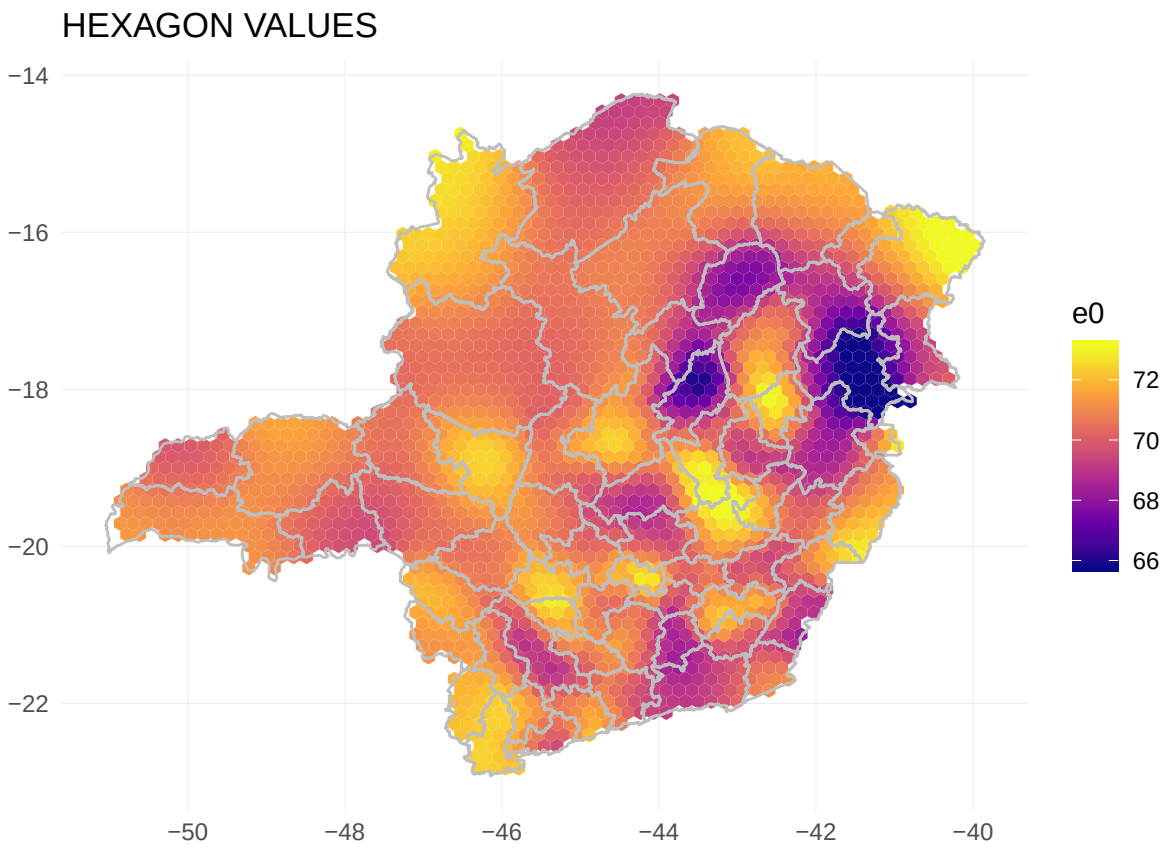



```
hexmap =
  ggplot( data=hexagons, aes(fill=e0)) +
  geom_sf(color=NA) +
  scale_fill_viridis_c(limits=range(optimal_hex_values), option='C') +
  theme_minimal() +
  labs(title='HEXAGON VALUES')
```

```
print(hexmap)
```




```
hexmap + geom_sf(data=microregion_map, color='grey',fill=NA)
```


HEXAGON VALUES
−14
−16
−18
−20
−22
−50
−48
−46
−44
−42
−40
e0
72
70
68
66