Rcaline: Modeling traffic-related pollution with R and the CALINE3 dispersion model

David Holstius

May 18, 2011

1 Introduction

`Rcaline` provides an interface to the CALINE family of line-source atmospheric dispersion models [1, 2]. These steady-state, Gaussian dispersion models are used to predict aerosol concentrations downwind from mobile emission source(s) such as highway traffic.

1.1 Features

At the heart of `Rcaline` is a Fortran library, `libcaline`, that wraps original code from the CALINE3 implementation created by the California Department of Transportation (CALTRANS).[1] Given the same inputs, `libcaline` has been tested to produce identical outputs. However, `libcaline` removes significant limitations found in previous implementations of CALINE: for example, `libcaline` can be used to model an unlimited number of roadway links and an unlimited number of receptors, bound only by available memory and CPU resources.

By providing access to `libcaline` within the R environment, `Rcaline` also makes it much easier to run CALINE using contemporary data sources, such as ESRI shapefiles, as input. `Rcaline` also provides full machine-precision access to CALINE model results in a convenient format. Thus, it's easy to use basic R commands—or third-party R packages—to visualize, compare, and export model results. This vignette illustrates the use of several other packages that complement `Rcaline`, including: `sp` and `maptools`, for handling spatial data; `rgeos`, for buffering; `ggplot2`, for visualization; and `automap`, for interpolation.

Finally, the R environment also provides useful scripting capabilities for automating large batches of model runs. For advanced users, it is possible to combine `Rcaline` with parallel computing tools, like the `multicore` package, to achieve significant speed gains in large model runs (e.g., 4x on a modern 4-core machine, or 8x on an 8-core machine) in pure R. Within a shell environment, `Rcaline` can also be scripted, with the use of GNU `make`, `qsub`, or other distributed computing tools.

[1] Support is planned for CALINE4 in a future release.

1.2 Scope and limitations

The CALINE3 model is most appropriately used for modeling dispersion of carbon monoxide (CO) attributable to free-flow traffic with wind speeds greater than 1.0 m/s. As with any model, care should be exercised to ensure that the practical application is theoretically well founded. For more on the theoretical scope and limitations of the CALINE model family, including terrain and other considerations, see [2].

2 Example Usage

In this section, we illustrate the use of `Rcaline` by applying it to highway data sourced from the OpenStreetMaps project [5] (Figure 1). There are three main steps: (1) preparing the input; (2) running the model; and (3) visualizing and/or exporting the output. Most of the work consists in preparing the input, and we'll illustrate how the construction of a receptor grid is a crucial step in that stage. The CALINE User's Guide [1] gives additional guidance beyond what we'll cover, so look to that for more information.

2.1 Setting up the model

2.1.1 Reading highway data from an ESRI shapefile

Contemporary GIS data is often exchanged in the form of shapefiles, and the `Rcaline` package includes an example shapefile containing highway data for West Oakland, CA. We can import it using the `maptools` package:

```
> library(maptools)
> highways.shp <- system.file("extdata", "WestOakland",
      "highways.shp", package = "Rcaline")
> highways.proj4 <- CRS(" +proj=utm +zone=10 +ellps=WGS84 +units=m")
> highways <- readShapeSpatial(highways.shp, proj4string = highways.proj4)
> stopifnot(is.projected(highways))
> coordnames(highways) <- c("x", "y")
> highway.segments <- Rcaline::as.segments(highways)
```

Note that, when working with Rcaline, all x, y, and z values should be in a *projected* coordinate system, with units in meters. Here, our coordinates are in UTM-10, so we're OK. If the coordinates in your shapefile are geographic (longitude and latitude), you'll need to use the `rgdal` package, a desktop GIS, or another tool to transform them first.

After reading in the shapefile, we converted it to a collection of individual segments using Rcaline's `as.segments` function. When we run the CALINE model, it will compute predicted contributions from each segment to each receptor, then sum these together to arrive at the predicted concentrations.

Figure 1 shows the roadway geometry that we just imported. We'll be using the `ggplot2` package for mapping and visualizing from here on, although there are also basic plotting functions available in R and the `sp` package.

```
> library(ggplot2)
> kilo <- function(x) x/1000
> easting <- function(...) scale_x_continuous("Easting (km)",
    formatter = kilo, ...)
> northing <- function(...) scale_y_continuous("Northing (km)",
    formatter = kilo, ...)
> map <- ggplot() + coord_equal() + easting() +
    northing()
> highway.centerlines <- geom_segment(aes(x = x0,
    y = y0, xend = x1, yend = y1), highway.segments)
> show(map + highway.centerlines)
```

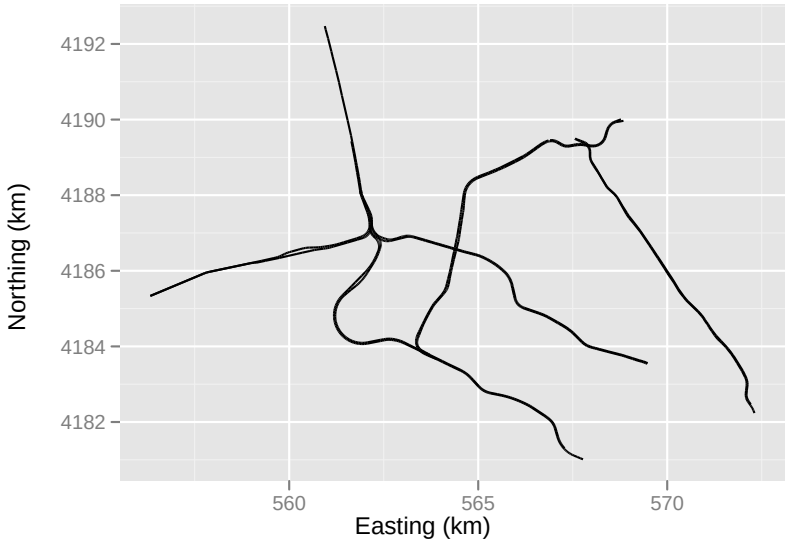

Figure 1: Example highway geometry.

2.1.2 Assigning and imputing roadway attributes

Although our shapefile contains an estimate of traffic volume (AADT), we'll need to construct values for a few other CALINE inputs.

First, we'll need to estimate the widths of the roadways. Using the number of lanes as a guide, we can estimate the width by assuming 3.5 m per lane plus a shoulder width of 3 m on each side. But some of the NLANES values are missing, so first we need to replace them with an imputed value. For that, we'll just use the median of the non-missing data.

```
> highway.segments <- within(highway.segments, {
    LANES <- replace(LANES, which(is.na(LANES)),
        median(LANES, na.rm = TRUE))
 })
> highway.segments <- transform(highway.segments,
```

```
    width = 3 + 3.5 * LANES + 3)
```

Two more required inputs are the height of the roadway above ground level, and its classification as either At Grade (AG); Fill (FL); Bridge (BR); or Depressed (DP). [2] For the moment, we'll assume all of the roadways are At Grade and 0.0 m above ground level.

```
> highway.segments <- transform(highway.segments,
    height = 0, classification = "AG")
```

The remaining required inputs are the *flow*, or traffic volume (vehicles per hour), and an *emissions factor*, given in grams per mile [per vehicle].[3] These can vary along the highway, but again, for now, we'll just assume they're constant. [4]

```
> highway.segments <- transform(highway.segments,
    flow = AADT/24, emissions = 10)
```

Because we're interested in hourly estimates (our meteorological records are hourly too), we have to remember to divide the average daily traffic by 24.

2.1.3 Importing meteorological records

CALINE requires four inputs corresponding to the prevailing meteorology:

- wind bearing;
- wind speed;
- Pasquill stability class; and
- mixing height.

Hourly values for these are usually available in the form of an "ISC-ready" input file, often with a .MET or .ASC file extension. `Rcaline` provides a convenience function, `read.ISC`, for parsing these.

```
> library(Rcaline)
> metfile <- system.file("extdata", "WestOakland",
    "OaklandSTP-2000.ASC", package = "Rcaline")
> meteorology <- read.ISC(metfile)$records
> with(meteorology, quantile(wind.speed))
     0%     25%     50%     75%    100%
 0.0000  2.0564  3.0846  4.4704 13.1430
```

[2] This classification system allows for a correction to be made depending on whether air can flow below as well as above the roadway. It also allows for a correction for links built on raised ground or in a trench—in which case air will also flow differently.

[3] Precise estimates for emission factors can be obtained with the use of a sophisticated model, such as EMFAC [4], that takes into account prevailing weather conditions (temperature and humidity) as well as the composition of the local vehicle fleet. Fleet composition and emissions inventories are sometimes available on a statewide or regional basis; for example, the California Air Resources Board provides an emissions inventory at this level. [3]

[4] Conveniently, these values just act as scalar multipliers of the predicted concentrations, so you can divide or multiply the model results to re-scale these values, as long as their spatiotemporal distribution doesn't change.

A wind rose depicting the distribution of wind speed and wind bearing is shown in Figure 2. The wind bearing is the direction that the wind is coming *from*, so in this case the winds are mainly from the west. They're generally over 1.0 m/s, too, which is good for the accuracy of the model's predictions.

```
> speed.breaks <- c(0, 1, 2, 3, 5, 10, floor(1 +
      max(meteorology$wind.speed)))
> wind.rose <- ggplot(data = meteorology) + coord_polar() +
      geom_bar(aes(wind.bearing, ..count../8760,
          fill = cut(wind.speed, speed.breaks, right = FALSE)),
          binwidth = 30)
> show(wind.rose + scale_x_continuous("", limits = c(0,
      360), breaks = seq(0, 360, by = 45)) + scale_y_continuous("",
      formatter = "percent") + scale_fill_brewer("Speed (m/s)",
      pal = "Set3"))
```

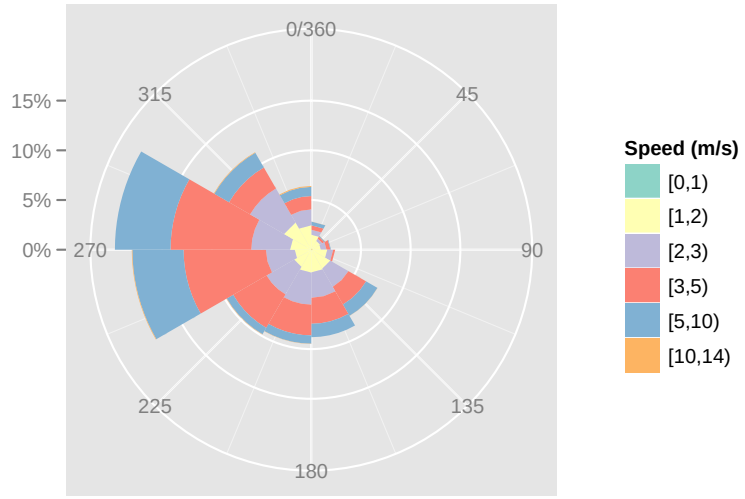

Figure 2: Wind rose for West Oakland meteorology.

2.1.4 Constructing a receptor grid

Now that we have the traffic-related input and the meteorology, we need to define a *receptor grid*. These are the locations at which we'll use the model to explicitly predict a pollutant concentration.

A receptor grid can be constructed out of regularly spaced points, but it doesn't have to be. For example, receptor locations might be defined by census block points, or by a selection of coordinates stored in a CSV file, or by a population-weighted sample.

But for starters, we'll construct a rather coarse, regularly spaced grid. We do this by constructing a buffer around the highways, and then sampling inside

the buffer with the help of `spsample`. We also add a z-value to the grid, corresponding to the "average" human height of 1.8 m. See Figure 3 for a depiction of the grid.

```
> library(rgeos)
> rgeos::setScale(1e+06)
> buffer <- gBuffer(highways, width = 1000)
> cartesian.grid <- spsample(buffer, cellsize = c(500,
    500), type = "regular")
> coordnames(cartesian.grid) <- c("x", "y")
> receptors <- SpatialPointsDataFrame(cartesian.grid,
    data = data.frame(z = rep(1.8, length(cartesian.grid))))
```

In later sections, we'll see how to improve the efficiency of Rcaline—and the quality of our figures—by making estimates at more carefully chosen locations.

2.2 Running the model

For illustration's sake, we'll start by running the model using only the first of the hourly meteorology conditions. `CALINE3.predict` always returns a NxM array, where N is the number of receptors, and M is the number of meteorological conditions. Each cell in the array corresponds to the sum of the incremental contributions from all roadway segments to that receptor under those conditions. If you want to compute the 1-hour maximum, simply `apply` the `max` function across the columns of the array. Similarly, if you want to use the annual average, you can compute the column-wise means instead.[5] Here we compute both the max and the mean (they'll be identical), and then re-bind the result to the SpatialPointsDataFrame called `receptors`.

```
> predicted.array <- CALINE3.predict(receptors,
    highway.segments, meteorology[1, ], surface.roughness = 100)
> predicted.data.frame <- data.frame(mean.value = apply(predicted.array,
    1, mean), max.value = apply(predicted.array,
    1, max))
> receptors <- SpatialPointsDataFrame(receptors,
    data = predicted.data.frame)
```

In Figure 3, we use a bubble plot to show the predicted concentration at each receptor. But are those truly "hot spots" in Figure 3, or are they artifacts of our sampling process? It turns out it's the latter, and we can fix it. Because of the way we constructed a regular grid, some of the receptors just happen to be really close to, and sometimes right on top of, the highway network.

2.3 Setting up the model: the right way

2.3.1 Sampling as a function of distance

To generate a better grid—one that will be more efficient, as well as more representative—we need to do denser sampling closer to the roadways, and

[5] Things get a little more complicated when wind speeds are less than 1.0 m/s. `CALINE3.predict` won't accept such inputs. You can treat calm winds as a "zero" concentration, or you can impute them in another way.

```
> segment.aes <- aes(x = x0, y = y0, xend = x1,
    yend = y1)
> highway.centerlines <- geom_segment(segment.aes,
    highway.segments)
> bubble.aes <- aes(x = x, y = y, size = mean.value,
    color = mean.value, order = mean.value)
> receptor.concentrations <- geom_point(bubble.aes,
    as.data.frame(receptors))
> show(map + highway.centerlines + receptor.concentrations)
```

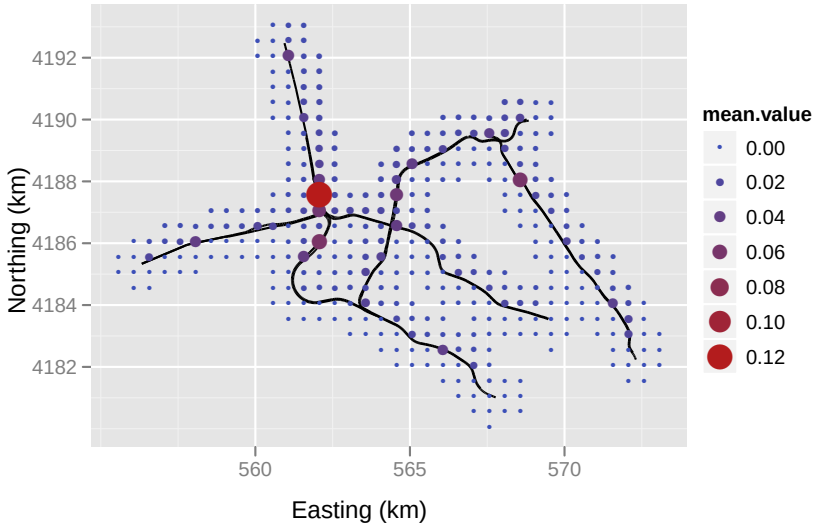

Figure 3: Example highway geometry, receptor grid, and predictions for grid coordinates.

sparser sampling further away. Moreover, none of our sampled locations should be too close to the roadway (like right on top of it). To accomplish this, first we'll construct buffers at varying distances from the highways. Then, instead of sampling *within* these buffers, we'll sample along their edges.

```
> buffer.widths <- c(50, 100, 250, 500, 750, 1000)
> buffers <- lapply(buffer.widths, function(x) gBuffer(highways,
    width = x))
```

Again we use the `gBuffer` function from the `rgeos` package. We coerce the returned SpatialPolygons objects to SpatialLines objects; then we invoke `sp-sample`, using sampling intervals that decrease with distance from the roadway. The effect is to create a sampling distribution that generally decreases in density as one moves away from the roadway. We could add a bit of noise, if we really wanted to mix it up a bit, but for now this looks pretty good (Figure 4).

```
> rings <- lapply(buffers, as.SpatialLines)
> perimeter <- function(ring) sum(unlist(lapply(ring@lines,
    LinesLength)))
> spsample.ring <- function(ring, ring.width, spacing) {
    pts <- spsample(ring, type = "regular", n = perimeter(ring)/spacing)
    coordnames(pts) <- c("x", "y")
    d <- rep(ring.width, nrow(pts@coords))
    SpatialPointsDataFrame(pts, data.frame(distance = d,
        spacing = spacing))
 }
> receptors <- do.call(rbind, mapply(spsample.ring,
    rings, buffer.widths, spacing = c(100, 150,
        250, 500, 750, 1000)))
> receptors$z <- 1.8
```

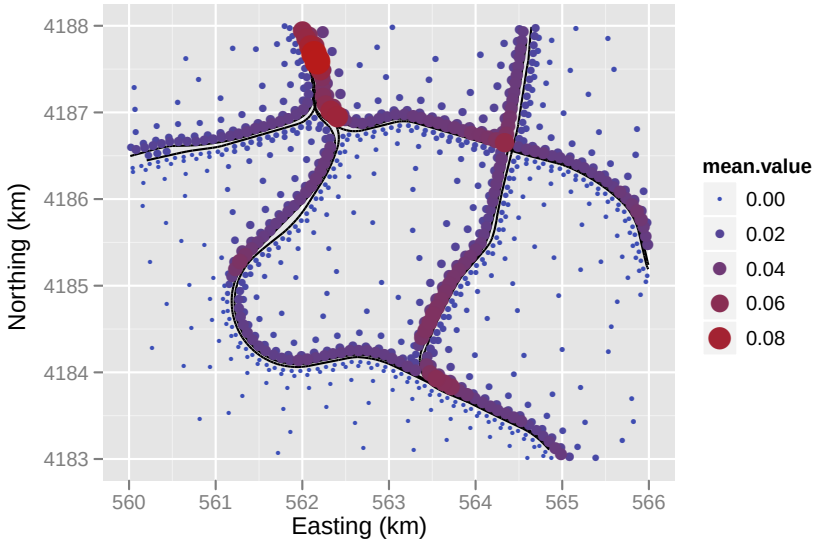

Figure 4: Close-up of receptors and predicted concentrations, this time sampling from rings around the roadways instead of a Cartesian grid.

2.3.2 Interpolating results

As a final exercise, we'll use the `gstat` package to krige the mean predicted concentrations, from the receptors sampled around the roadways, onto a regular grid of moderate resolution (Figure 5). This isn't necessarily the best way to go about it; it's just illustrative.

```
> span <- function(x) diff(range(x))
> xy.grid <- function(extent.x, extent.y, cellsize) {
    cellcentre.offset <- c(min(extent.x), min(extent.y))
    cells.dim <- ceiling(diff(cbind(extent.x,
```

```
        extent.y))/cellsize)
    g <- SpatialGrid(GridTopology(cellcentre.offset,
        cellsize, cells.dim))
    coordnames(g) <- c("x", "y")
    return(g)
}
```

```
> library(automap)
> grid <- xy.grid(extent.x, extent.y, c(50, 50))
> k <- autoKrige(mean.value ~ x + y, receptors,
    grid, nmax = 25)
[using universal kriging]
> summary(k)
krige_output:
Object of class SpatialGridDataFrame
Coordinates:
      min     max
x  559975  565975
y 4182975 4187975
Is projected: TRUE
proj4string :
[+proj=utm +zone=10 +ellps=WGS84 +units=m]
Number of points: 2
Grid attributes:
  cellcentre.offset cellsize cells.dim
x            560000       50       120
y           4183000       50       100
Data attributes:
   var1.pred            var1.var           var1.stdev
 Min.   :-0.009704   Min.   :0.0001409   Min.   :0.01187
 1st Qu.: 0.001120   1st Qu.:0.0001511   1st Qu.:0.01229
 Median : 0.005972   Median :0.0001600   Median :0.01265
 Mean   : 0.008626   Mean   :0.0001623   Mean   :0.01271
 3rd Qu.: 0.014502   3rd Qu.:0.0001643   3rd Qu.:0.01282
 Max.   : 0.058136   Max.   :0.0004579   Max.   :0.02140

exp_var:
       np      dist        gamma dir.hor dir.ver   id
1    4776  113.5989 0.0001147750       0       0 var1
2   14893  241.0483 0.0001900289       0       0 var1
3   16832  396.8528 0.0002032219       0       0 var1
4   30324  591.5468 0.0001889206       0       0 var1
5   34107  825.9271 0.0001927952       0       0 var1
6   38000 1058.7926 0.0001979690       0       0 var1
7  147342 1584.7505 0.0002006589       0       0 var1
8  204752 2374.2390 0.0002042694       0       0 var1
9  387301 3342.6444 0.0002038467       0       0 var1
10 410751 4517.8473 0.0001949326       0       0 var1
11 430337 5683.7020 0.0001922825       0       0 var1
```

```
12 488063 7015.0631 0.0001945243       0       0 var1

var_model:
  model        psill    range kappa
1   Nug 0.0001147750    0.000   0.0
2   Ste 0.0000847239 2240.581   0.2
Sums of squares betw. var. model and sample var.[1] 1.149241e-09
```

As you can see, kriging resulted in some estimates that were negative. Clearly this is wrong: no concentration of pollutant is ever less than zero. A better approach might be to perform trans-Gaussian kriging—using a Box-Cox transformation, kriging the transformed values, and then transforming back to our original, non-negative scale. We'll leave that for another vignette.

```
> interpolated.values <- geom_tile(aes(x = x, y = y,
     fill = var1.pred, alpha = var1.pred), as.data.frame(k$krige_output))
> receptor.locations <- geom_point(aes(x = x, y = y),
     shape = 3, size = 1, alpha = 0.5, as.data.frame(receptors))
> zoom <- function(x, a) (x - mean(x)) * a + mean(x)
> map <- map + easting(limits = zoom(extent.x, 1.1)) +
     northing(limits = zoom(extent.y, 1.1))
> show(map + highway.centerlines + receptor.locations +
     interpolated.values + scale_fill_gradient2(low = "yellow",
     mid = "orange", high = "brown", midpoint = 0.025) +
     scale_alpha(to = c(0, 0.8)))
```

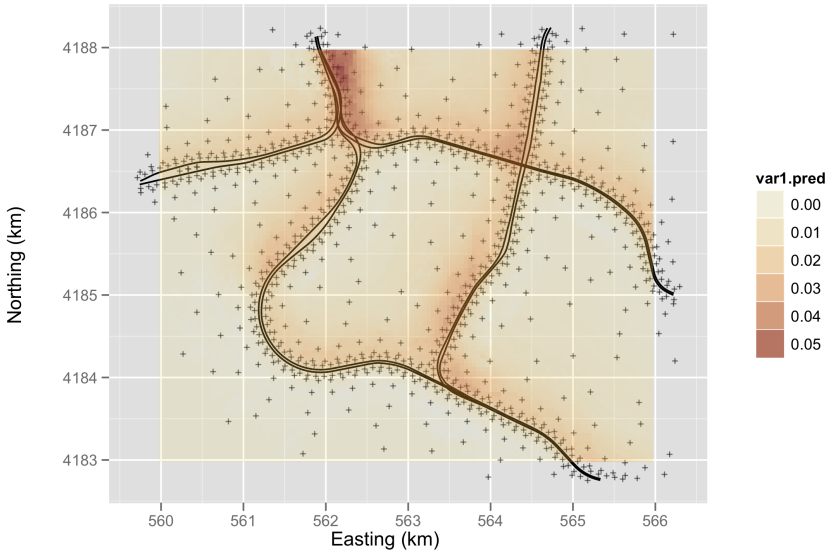

Figure 5: Concentrations interpolated with the use of `automap`.

Before concluding anything, we should also double-check what happened when we attributed traffic volumes at the feature (polyline) level. Recall that we

assigned traffic volumes on the basis of AADT (Annual Average Daily Traffic), which was an attribute of the polyline features in our original shapefile. If there were segments with suspiciously high AADT—as might occur in the case of offramps or onramps—they should be screened more carefully. Predicted concentrations near these features could be biased upward, simply by virtue of there being many child segments having the AADT that was assigned to the parent polyline feature, rather than having the total AADT divided realistically amongst them. Figure 6 maps a color spectrum to AADT quantiles.

```
> AADT.breaks <- with(highway.segments, c(0, quantile(AADT)[-1]))
> AADT.scale <- scale_colour_manual("AADT", value = c("green",
    "yellow", "orange", "red"))
> highway.traffic <- geom_segment(aes(x = x0, y = y0,
    xend = x1, yend = y1, color = cut(AADT, AADT.breaks,
        right = TRUE)), highway.segments)
> show(map + easting(limits = extent.x) + northing(limits = extent.y) +
    highway.traffic + AADT.scale)
```

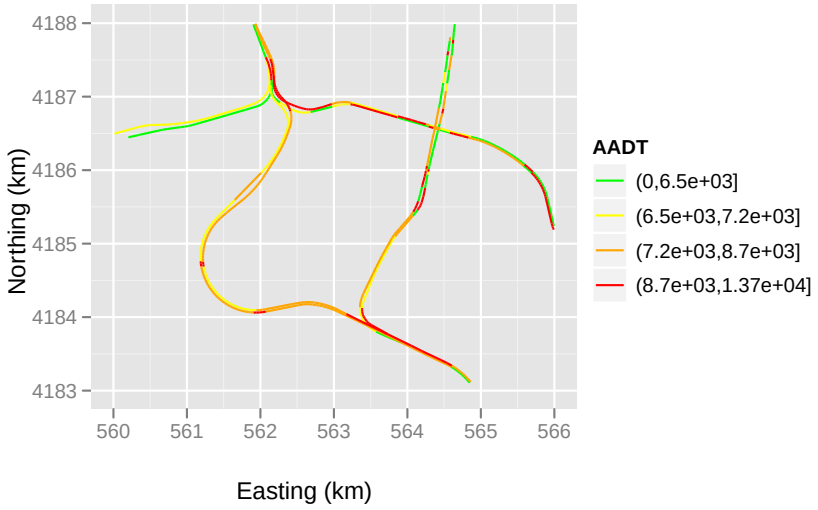

Figure 6: Traffic volumes (AADT).

3 Conclusion

3.1 Issues and future work

For brevity's sake, we've glossed some important issues in the parameterization of the model. For example, how does one determine an appropriate surface roughness length? What to do about calm winds (<1.0 m/s), when terms in the CALINE model start to become ill-defined? What about geocoding errors

in the locations of roadway segments? Using an alternate roadway geometry or different model parameters will change the predicted values, sometimes in a significant way. A future vignette is planned to explore and illustrate the sensitivity of Rcaline (or rather, CALINE3) to a range of factors such as these.

More refinement of the model is also possible. For example, it is relatively easy to integrate a diurnal profile of traffic volumes, a useful feature of the later CAL3QHCR variant of the CALINE3 model. Conveniently, the returned array from `CALINE3.predict` is an array of hourly values at each receptor (if a full year's worth of meteorological data is supplied, as is customary). Assuming the distribution of AADT from segment to segment does not also vary by hour, one can simply scale the predicted values by the hourly proportions of traffic before taking an average or doing other sorts of aggregation.

Improving performance with large datasets is also possible. Multiple Rcaline runs also be set up and executed on subsets of the data, then merged. This kind of flexibility allows for the use of distributed processing. For example, we could partition the data by meteorological condition, allocating one-quarter of each year to each of four processors, then combine the results. If the data were partitioned by links or by receptors, we might also reduce the total CPU cycles required by restricting model estimates to only those link-receptor pairs within a given distance (say, 1 km). Both of these techniques have proven useful, although the details are outside the scope of this document.

3.2 Feedback welcome

Rcaline is still in development, and feedback is welcome. Please contact david.holstius@berkeley.edu if you have questions, suggestions, or related work to discuss. If you have publicly available data that you would like to contribute to the `Rcaline` package, that would also be welcome. Many people could benefit from the inclusion of more example datasets, including those that have been used in previously published research.

4 Acknowledgments

The author thanks Profs. Edmund Seto and Michael Jerrett of the UC Berkeley School of Public Health, Division of Environmental Health Sciences, as well as the UC Berkeley Center for Information Technology in the Interests of Society (CITRIS), for their generous support of the development of `Rcaline` and `libcaline`. Sincere thanks also go to Phil Martien of the Bay Area Air Quality District (BAAQMD), who generously contributed user testing, feedback, and support leading to the 1.0 release.

References

[1] P.E. Benson. CALINE3: a versatile dispersion model for predicting air pollutant levels near highways and arterial streets. Interim report. Technical report, PB-80-220841, California State Dept. of Transportation, Sacramento (USA). Transportation Lab., 1979.

[2] P.E. Benson. A review of the development and application of the CALINE3 and 4 models. *Atmospheric Environment. Part B. Urban Atmosphere*, 26(3):379–390, 1992.

[3] California Air Resources Board. Estimated Annual Average Emissions, 2005.

[4] California Air Resources Board. Motor Vehicle Emission Factor / Emission Inventory Model, EMFAC, 2007.

[5] M.M. Haklay and P. Weber. OpenStreetMap: user-generated street maps. *IEEE Pervasive Computing*, pages 12–18, 2008.