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

David Holstius

May 12, 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 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 is easy to use basic R commands—or third-party R packages—to visualize, compare, and export model results.

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).

1.2 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

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

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. The CALINE User's Guide [1] gives some additional guidance beyond what we'll cover in this example.

2.1 Preparing input

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 `readShapeSpatial` function from 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))
> 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 by using Rcaline's `as.segments` function. The CALINE model computes a prediction for each individual segment of a roadway, then sums them together at the end.

A basic map of the highway data is shown in Figure 1. 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. The colors are mapped to the quantiles of the AADT (Annual Average Daily Traffic) on each stretch of highway. (Because the lengths of the segments vary, this isn't necessarily the best way to color the map; but it's a start.)

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

```
> 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)
> print(map + highway.centerlines)
```

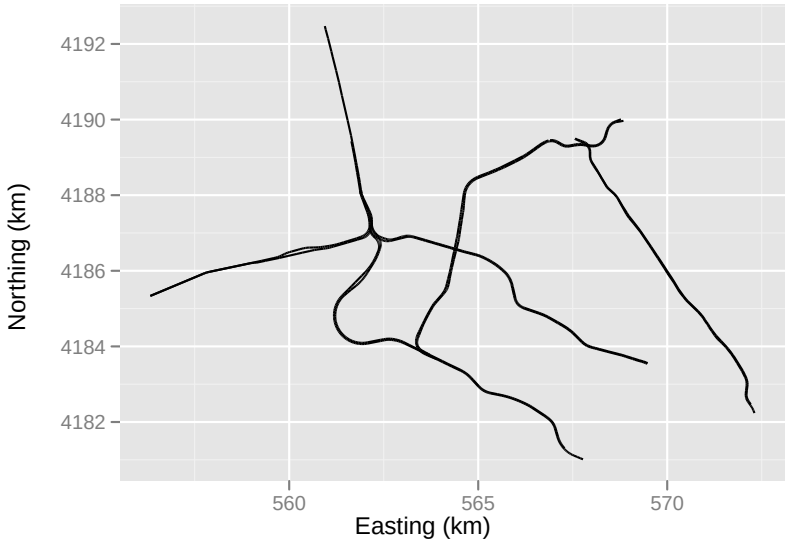

Figure 1: Example highway geometry.

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

[2]This classification system allows for a correction to be made depending on whether air

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
```

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.

---

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.

```
> 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)
> print(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"))
```

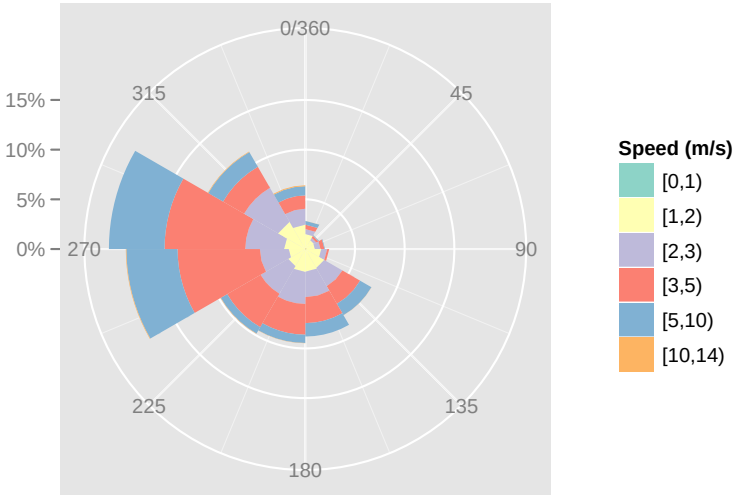

Figure 2: Wind rose for West Oakland meteorology.

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`. See Figure 3 for a depiction of the grid.

```
> library(rgeos)
> buffer <- gBuffer(highways, width = 1000)
> xy.locations <- spsample(buffer, cellsize = c(500,
```

```
    500), type = "regular")
> receptors <- SpatialPointsDataFrame(xy.locations,
    data = data.frame(z = rep(1.8, length(xy.locations))))
```

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. In Figure 3, we use a bubble plot to show the predicted concentration at each receptor.

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

```
> highway.centerlines <- geom_segment(aes(x = x0,
    y = y0, xend = x1, yend = y1), highway.segments)
> receptor.concentrations <- geom_point(aes(x = x1,
    y = x2, size = predicted, color = predicted,
    order = predicted), as.data.frame(receptors))
> print(map + highway.centerlines + receptor.concentrations)
```

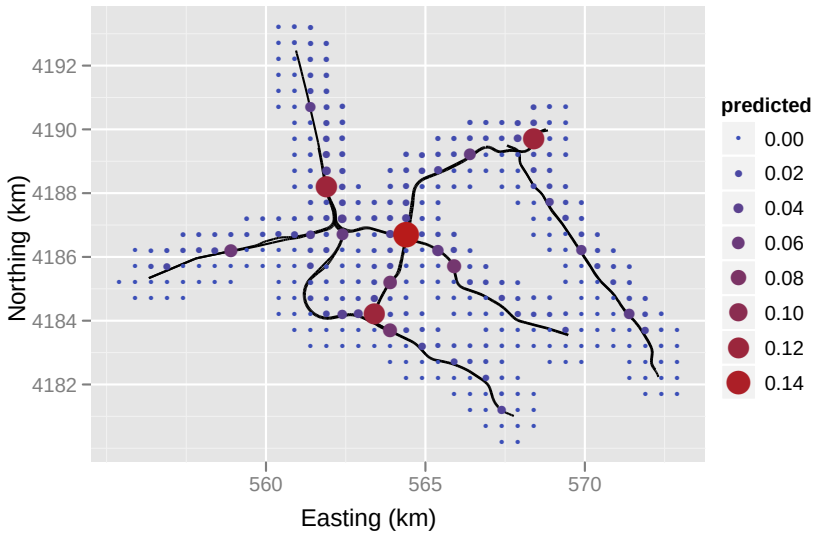

Figure 3: Example highway geometry, receptor grid, and predictions for grid coordinates.

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 we sampled without respect to the highways, 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 sparser sampling further away. To accomplish this, first we'll construct buffers at varying distances from the highways.

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

Then, instead of sampling *within* these buffers, we'll sample along their edges. First we coerce them to SpatialLines objects; then we invoke `spsample`, using sampling intervals that decrease with distance from the roadway. The effect is to create a sampling distribution that generally increases in density as one approaches 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.

```
> 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)
      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 shows an up-close look at the resulting receptor grid and predicted values:

2.3.2 Interpolating results

As a final exercise, we'll use the `intamap` package to interpolate the CALINE predictions onto a regular grid of moderate resolution.

```
> library(intamap)
> cellsize <- c(50, 50)
> cellcentre.offset <- c(min(x.range), min(y.range))
> span <- function(x, ...) max(x, ...) - min(x,
      ...)
```

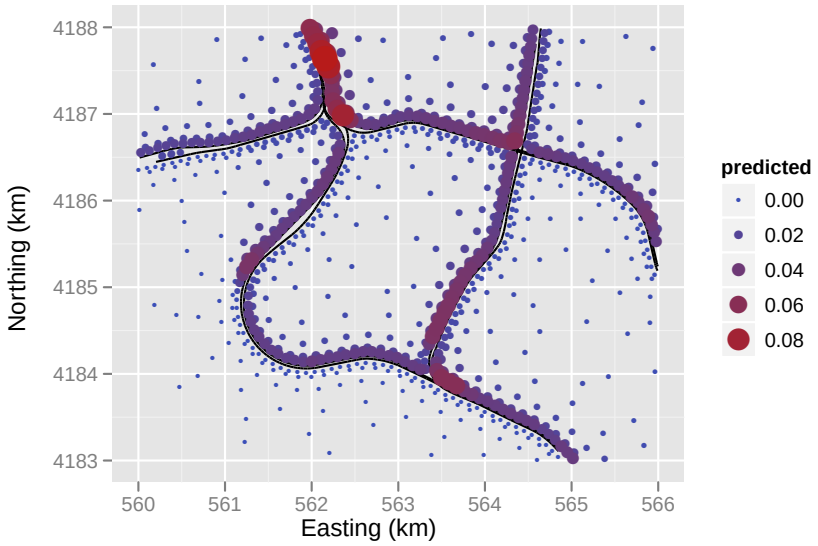

Figure 4: Close-up of receptor rings and predicted concentrations.

```
> cells.dim <- as.integer(c(span(x.range), span(y.range))/cellsize)
> grid <- SpatialGrid(GridTopology(cellcentre.offset,
      cellsize, cells.dim))
> proj4string(grid) <- proj4string(receptors)
> receptors$value <- receptors$predicted
> interpolated <- interpolate(receptors, grid, maximumTime = 5)
> normalize <- function(x) scale(x, 0, max(x))
> normalized.estimates <- transform(data.frame(interpolated$predictions),
      Y = normalize(var1.pred))
```

Figure 5 shows the result:

One final thing to check for is bias caused by attributing traffic volumes at the feature (polyline) level. Recall that we assigned traffic volumes on the basis of AADT, which was an attribute in our original shapefile. Here's an illustration of the traffic volumes (Figure 6).

If there were segments with suspiciously high AADT—as might occur in the case of offramps—they should be investigated. Geometric complexity could be biasing the results upward, simply by virtue of there being many segments carrying the "total" AADT for the parent polyline feature, rather than having the total AADT divided realistically amongst them (as when a roadway forks in two).

3 Conclusion

Rcaline is still in development, and feedback is welcomed. Please contact david.holstius@berkeley.edu if you have questions, suggestions, or related work to discuss!

```
> library(colorspace)
> n.colours <- 10
> predicted.raster <- geom_tile(aes(x = s1, y = s2,
    fill = Y), data = normalized.estimates, alpha = 0.75)
> predicted.scale <- scale_fill_gradientn(aes = aes(fill = Y),
    breaks = seq(0, 1, length = n.colours + 1),
    colours = rev(heat_hcl(n.colours)))
> receptor.locations <- geom_point(aes(x, y), alpha = 0.5,
    size = 1, shape = 3, data = as.data.frame(receptors))
> print(map + easting(limits = x.range) + northing(limits = y.range) +
    highway.centerlines + receptor.locations +
    predicted.raster + predicted.scale)
```

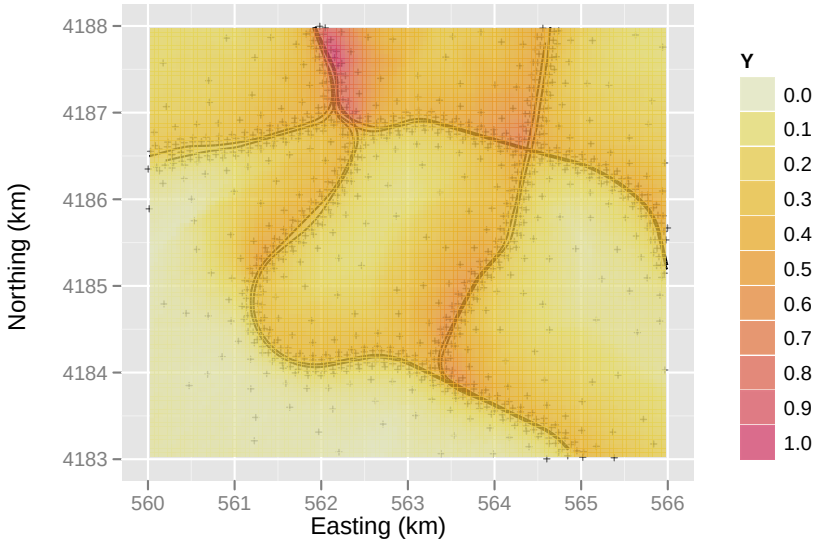

Figure 5: Close-up of receptor rings and interpolated concentrations.

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`. Thanks go also to Phil Martien of the Bay Area Air Quality District (BAAQMD), who generously provided user testing, feedback, and support leading to the 1.0 release.

```
> 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)
> print(map + easting(limits = x.range) + northing(limits = y.range) +
    highway.traffic + AADT.scale)
```

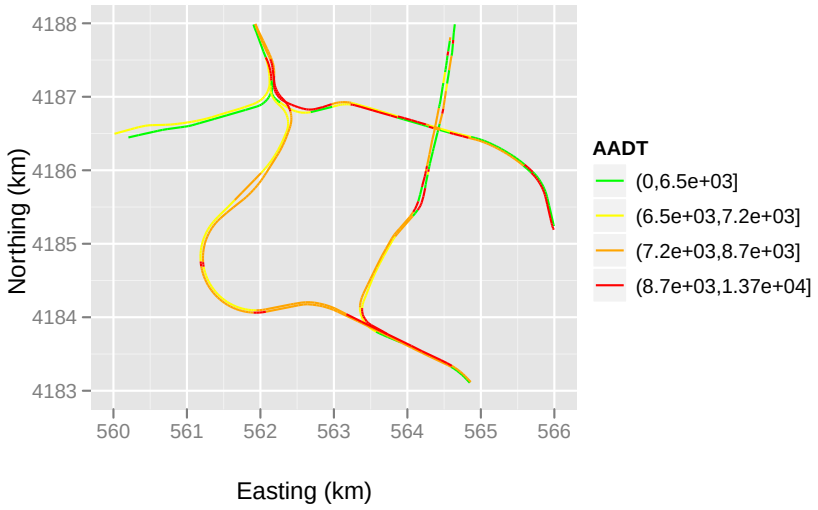

Figure 6: Traffic volumes (AADT).

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.