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

David Holstius

February 23, 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 distinct 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 the reader is well advised to consult the CALINE User's Guide [1] for additional detail.

2.1 Preparing input

2.1.1 Reading from an ESRI shapefile

Contemporary GIS data is often stored in the ESRI shapefile format. The `Rcaline` package includes an example of such a shapefile with highway data for West Oakland, California.

```
> library(Rcaline)
> shpfile <- system.file("extdata", "WestOakland",
      "highways.shp", package = "Rcaline")
```

We can read it into R by using the `readShapeSpatial` function from the `maptools` package, as follows:

```
> library(maptools)
> highways <- readShapeSpatial(shpfile)
```

Note that `Rcaline` expects coordinates to be *projected*: that is, to be given in meters, rather than in degrees latitude/longitude. If you need to (re)project your input shapefile, so that the coordinates are in meters, consult the documentation for the `sp` or `rgdal` packages.

2.1.2 Segmenting roadway geometry

In preparation for a CALINE model run, we need to break the roadway geometry up into individual segments. This only needs to be done once; you can save the result for future use.

```
> segments <- Rcaline:::segmentize(highways)
> link.data <- merge(segments, highways, by.x = "marks",
      by.y = "row.names")
```

In this example, we're using the `as.psp` function from the `spatstat` package. The code above re-assigns to each segment the attributes associated with its parent polyline feature. If that's a bit confusing, don't worry. It allows us to retain attributes like traffic volume, link elevation, number of lanes, or other data that might have been present in the shapefile.

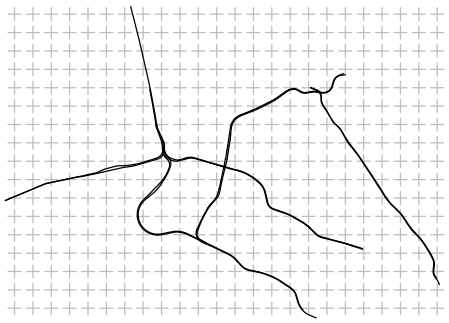

Figure 1: Example highway geometry and receptor grid.

2.1.3 Imputing missing variables

In this example, we're provided with an estimate of traffic volume: AADT (Annual Average Daily Traffic). However, we'll need to impute or assume default values for a few other variables. For example, the `LANES` variable, which supplies the number of lanes, has some NA (missing) values. One way to deal with that is to replace those with the median of the other values, as follows:

```
> link.data[is.na(link.data$LANES), "LANES"] <- median(link.data$LANES,
    na.rm = TRUE)
```

Using the number of lanes as a guide, we can calculate the highway width (a required input) using a rough estimate of 10 meters per lane.

```
> link.data <- transform(link.data, width = LANES *
    10)
```

Another required input is the height of each link above ground level, as well as a *link classification*. Links can be classified as: At Grade (AG); Fill (FL); Bridge (BR); or Depressed (DP). This allows for a correction to be made depending on whether air can flow below as well as above the roadway link. It also allows for a correction for links built on raised ground or in a trench—in which case air will also flow differently.

```
> link.data <- transform(link.data, height = 0,
    classification = "AG")
```

Lacking any other information, we'll assume all of the links are At Grade. (A more thorough analysis might perform a sensitivity test or include an adjustment factor to reflect the uncertainty introduced by this assumption.)

Finally, we assign to each link an estimate of the *flow*, or traffic volume (vehicles per hour), as well as an *emissions factor*, given in grams per mile [per vehicle].

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

Because we are interested in an hourly estimate, we divide the average daily traffic by 24. We don't have an emission factor available, so we assume uniform emissions of 10 g /veh-mi. 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]

2.1.4 Constructing a receptor grid

Now that we have the traffic-related input prepared, we need to define a *receptor grid*. These are the locations at which we will use the model to 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, they can be constructed on-the-fly, as follows:

```
> library(sp)
> receptor.grid <- sample.Spatial(highways, n = 400,
     type = "regular")
> gridded(receptor.grid) <- TRUE
```

The receptor grid is fully three-dimensional. A conventional approach is to assume a z-value of 1.8 meters above ground, which is about the same height as an average person's head. This helps if you are estimating a pollutant surface for exposure assessment purposes.

2.1.5 Providing meteorology

CALINE3 requires four variables corresponding to the prevailing meteorology:

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

Hourly values for these are sometimes available in the form of an "ISC-ready" input file, often with a .MET file extension. `Rcaline` provides a helpful function, `read.ISC`, for reading these files as input. (See ?read.ISC for more.) For this example, we'll just assume some reasonable values for a single hour.

```
> meteorology <- list(wind.bearing = 330, wind.speed = 1.5,
     stability.class = 4, mixing.height = 1000)
```

If you are interested in computing annual averages, 8-hour maximum concentrations, etc., it's quite easy to use the scripting capabilities of R to process many different scenarios and summarize the results.

2.2 Running the model

Now that the roadway geometry, emissions, receptor locations, and prevailing meteorology have been established, we can run the model.

2.2.1 Model parameters

CALINE3 also requires several "job parameters", including:

- averaging time;
- surface roughness;
- pollutant settling velocity; and
- pollutant deposition velocity.

We'll use some common values for averaging time (60 min) and surface roughness (100 cm). When modeling carbon monoxide, it's conventional to specify the settling velocity and deposition velocity as 0.0 m/s. For more on these parameters, including reasonable ranges and representative values, see [1, 2] or the documentation for `CALINE3.predict`.

2.2.2 Computing receptor totals

The following code computes the predicted aerosol concentration, given the prevailing conditions and the parameters we've chosen.

```
> n.receptors <- 10000
> receptor.grid <- sample.Spatial(highways, n = n.receptors,
      type = "regular")
> predicted <- CALINE3.predict(receptor.grid, link.data,
      meteorology, averaging.time = 60, surface.roughness = 100)
```

2.3 Visualization

As a quick check on the model output, it can be good to create a visualization. This most easily takes the form of a map. Since we have computed model results on a regular grid, it's fairly easy to construct such a map.

The map in Figure 2 was produced with a higher-resolution receptor grid than that depicted in Figure 1 (10,000 receptors vs 400 receptors). This is one approach to achieving higher-resolution estimates of pollutant concentrations. Another would be to interpolate a lower-resolution receptor grid, using, for example, kriging or bicubic interpolation. Many R packages and functions are available to assist in this task!

2.4 Improving performance

Running CALINE can be a computationally demanding task, but it's possible to speed up the clock time. First, you can restrict the receptor grid to areas of interest—often those close to the roadway network. Second, you can distribute your job(s) among multiple cores or even multiple machines.

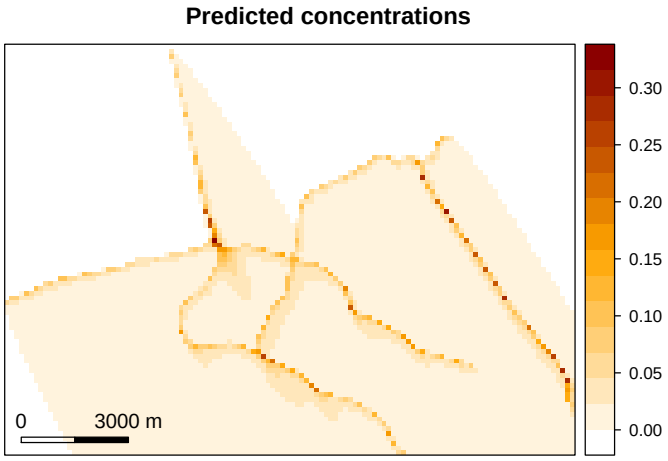

Figure 2: Example heatmap at moderate resolution (10,000 receptors).

2.4.1 Predicting near-roadway concentrations

To follow this example, you'll need to have enabled the GPC library by calling

```
> spatstat.options(gpclib = TRUE)
```

Now we can use the spatstat library to compute a dilation (what geographers call a buffer) of the roadway geometry (Figure 3):

We could use the simpler and more precise `dilation(..., polygonal=TRUE)` instead of `Rcaline:::fast.dilation()`—but it's a bit too slow for this example.

2.4.2 Distributed computation

[TODO]

3 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`.

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.

```
> buffer.radius <- 1000
> buffer <- Rcaline:::fast.dilation(highways, buffer.radius,
     eps = 100)
> receptor.grid <- spsample(buffer, n = n.receptors,
     type = "regular")
> print(plot(buffer))
NULL
```

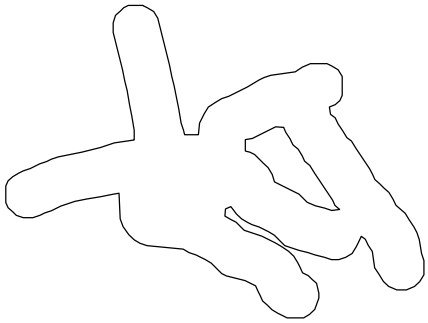

Figure 3: Approximate 1000 m buffer around roadways.

[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.

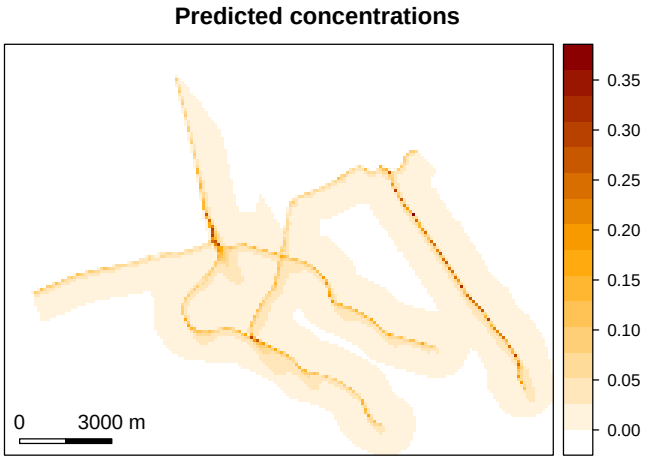

Figure 4: Higher-resolution map created by sampling near roadways.