Using Rcaline: An Illustrated Example

David Holstius

December 6, 2011

1 Introduction

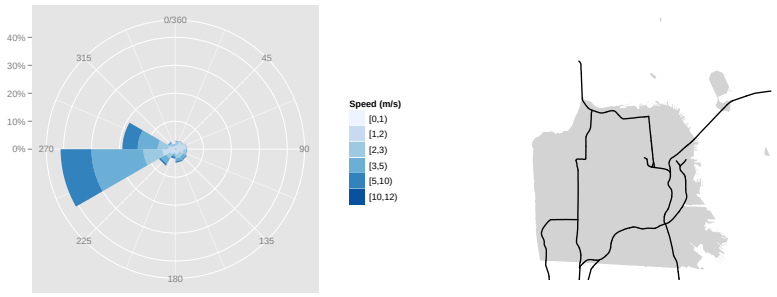
Here we apply `Rcaline` to San Francisco data provided by the Bay Area Air Quality Management District (BAAQMD). This example illustrates the basic steps of constructing, running, and visualizing a model with Rcaline.

2 Model construction

In this example, the traffic data and the meteorological data have already been imported. If you're unfamiliar with this step, please consult the accompanying vignette, *Importing Data for use with Rcaline*.

There are three objects in the imported data: an `ISCFile`, containing hourly meteorological records; a `SpatialPolygonsDataFrame`, representing the extent of San Francisco county; and a `SpatialLinesDataFrame`, which describes the roadway geometry and the Annual Average Daily Traffic (AADT) counts.

```
> require(Rcaline)
> data(SanFrancisco, package='Rcaline')
> ls()
[1] "met_5801.isc"         "SF_county.shp"
[3] "STRte_BAAQMD_v2.shp"
```



(a) Meteorology (b) Highway network

Figure 1: San Francisco data included with the `Rcaline` package.

2.1 Meteorology

The conventional way to supply hourly meteorological data is to use an "ISC-ready" meteorology file containing records of wind speed, wind bearing, atmospheric stability, and mixing height. We need only specify whether to use urban or rural mixing heights, as follows:

```
> met <- Meteorology(met_5801.isc, use='urban')
> show(ggplot(met))
```

The hourly resolution ensures that we can adequately model the seasonal and diurnal variability. Figure 1a illustrates the joint distribution of wind speed and wind bearing; the wind here is predominantly coming from the south-west.

2.2 Roadways and traffic

Importing shapefiles (with a package like `maptools` or `rgdal`) is usually the easiest way to get GIS data into R. Figure 1b shows the San Franciso highway network. This network is composed of multiple *polylines*, each of which is composed of one or more segments or *links*.

Each polyline in `STRte_BAAQMD_v2.shp` has an associated attribute, `TRVol2009`, that represents the Annual Average Daily Traffic (AADT) volume on that stretch of highway. In order to convert the traffic volume to vehicles per hour, we need to divide the AADT by 24. We also need to specify the road width (in meters). If we had link-level data on the number of lanes, we could use that to impute widths instead of supplying a single value.

2.3 Emission factors

We also need to supply emission factors. The units for an emission factor should be *grams per vehicle per mile*. Emission factors can vary by link, just like traffic volume or road width. Inclined stretches of road, for example, might merit a higher emission factor.

For the sake of this example, we'll use a constant "unit emission factor" of 1.0 across the entire highway network. In California, emission factors for specific pollutants can be obtained from models like EMFAC [2, 3, 4].

```
> lnk <- FreeFlowLinks(STRte_BAAQMD_v2.shp,
          vehiclesPerHour = TRVol2009 / 24,
          emissionFactor = 1.0,
          width = 30.0)
> plot(SF_county.shp, col="light gray", border=NA)
> lines(lnk, lwd=2)
```

The emission rate calculated by CALINE3 for any given link (in grams per mile per hour) is exactly equal to that link's emission factor multiplied by its traffic volume. Further, the incremental concentration contributed by any given link to a downwind receptor is directly proportional to that link's emission rate. This means that the resulting dispersion surface can be linearly scaled to reflect different emissions scenarios, without needing to run the dispersion step again, as long as the emission factors and/or traffic volumes are scaled uniformly.

(a) Grid-based (b) Distance-based

Figure 2: Receptor locations sampled using two different methods.

2.4 Receptors

Receptors are the locations at which the dispersed pollutant concentration will be calculated. CALINE3 is a steady-state Gaussian plume model, rather than a numerical grid-based model. Hence, it is up to us to determine the exact nature of the receptor locations. We can essentially do one of two things: (1) generate a set of receptor locations algorithmically; or (2) supply a predefined set of locations.

2.4.1 Constructing a receptor grid algorithmically

Given that modelers often use a regular Cartesian grid (Figure 2a), here we show how to generate such a grid. Because we do not expect significant concentrations beyond 1 km, we restrict the grid to locations within 1 km of the highway centerlines.[1]

```
> rcp <- ReceptorGrid(lnk, resolution=250, maxDistance=1e3)
> plot(SF_county.shp, col="light gray", border=NA)
> lines(lnk)
> points(rcp, pch='+', cex=0.5)
```

Relying on a Cartesian grid can induce artificial "hot spots" where the grid just happens to fall closer to the road. A different approach (Figure 2b) is to create receptors at specific distances from the road network. This approach also allows receptors to be packed more densely close to the roadway, and concentrates our modeling efforts (e.g., CPU time) on the more interesting parts of the dispersion field.

```
> rcp <- ReceptorRings(lnk, distances=c(100, 250, 500, 1000))
> plot(SF_county.shp, col="light gray", border=NA)
> lines(lnk)
> points(rcp, pch='+', cex=0.5)
```

If you were interested in estimating aggregate exposures at a population level, you could also construct an irregular grid by sampling locations from predefined regions, like ZIP codes or census tracts.

[1] To see the code, which relies on `rgeos`, just type `ReceptorGrid` and hit Enter.

2.4.2 Supplying a set of receptors instead of a grid

Receptor locations need not be constructed algorithmically. For example, you might want to compute predicted concentrations at a set of geocoded street addresses. As long as you have these in a format that you can import into R, you can supply them instead. Just replace the `rcp` variable in the code that follows with your `SpatialPointsDataFrame` or `SpatialPoints` object containing the locations of interest.

2.5 Other model parameters

Additional model parameters, including terrain and pollutant characteristics, also need to be specified. For detailed information, consult the CALINE3 User's Guide [1]. Here we supply some reasonable default values:

```
> ter <- Terrain(surfaceRoughness=80.0)
> CO <- Pollutant("Carbon monoxide",
          molecularWeight = 28.0,
          settlingVelocity = 0.0)
```

3 Predicting concentrations

The CALINE3 algorithm is CPU-intensive. If you're using R on a console (not, say, the Mac R.app GUI), `Rcaline` will automatically attempt to use the `foreach` package to do the computations in parallel, using however many cores you have. The speedup should be nearly linear in the number of cores.[2]

3.1 Running the model

We use the `predict` method to actually run the model. Since the model will actually be run once for every meteorological condition we supply (N=8760), it can be quickest to use only a small sample for the first pass. Here we use 1% of the meteorology, sampled at random:

```
> mod <- Caline3Model(lnk, sample.rows(met, p=0.01), rcp, ter, CO)
> pred <- predict(mod)
```

The result of running the model is an $M \times N$ array, where M is the number of meteorological conditions and N is the number of receptors. Each cell indicates the concentration, in g/m^3, at that receptor during those conditions.

3.2 Computing summary statistics

By computing summary statistics such as the mean or maximum, we can treat the result as a sample from a theoretical annual distribution, and estimate its properties. The `aggregate` function computes several statistics by default. Multiplying by 1.0×10^3 converts the results to mg/m^3, and casting the result to a `SpatialPointsDataFrame` re-binds the statistics to the receptor locations:

```
> agg <- aggregate(pred) * 1.0e3
> spdat <- as(agg, 'SpatialPointsDataFrame')
> spdat[1:3, c('distance', 'mean', 'max')]
               coordinates distance   mean  max
RECP. 1 (15011.2, 4188660)      100 0.0670 1.20
RECP. 2 (14968.7, 4188750)      100 0.0675 1.29
RECP. 3 (14957.1, 4188850)      100 0.0708 1.38
```

[2] You can also configure the `foreach` package to distribute computation across multiple host machines, although that is outside the scope of this example.

4 Analyzing results

After aggregation, we can select a statistic of interest and explore the distribution. Here we focus on exploring results graphically, although they could also be tabulated or subjected to statistical tests.

4.1 Upwind vs. downwind concentrations

Within the results is a variable, `distance`, that contains the distance-to-roadway for each receptor. (Recall that we specified these distances when constructing the receptor grid.) We can use this to divide the receptors into specific classes and explore the distribution of predicted concentrations in each.

```
> ggplot(aes(x=mean), data=as(spdat, 'data.frame')) +
        geom_histogram(binwidth=5e-2) +
        stat_density(aes(y=5e-2*..count..), color='red', geom='path') +
        facet_wrap(~ distance)
```

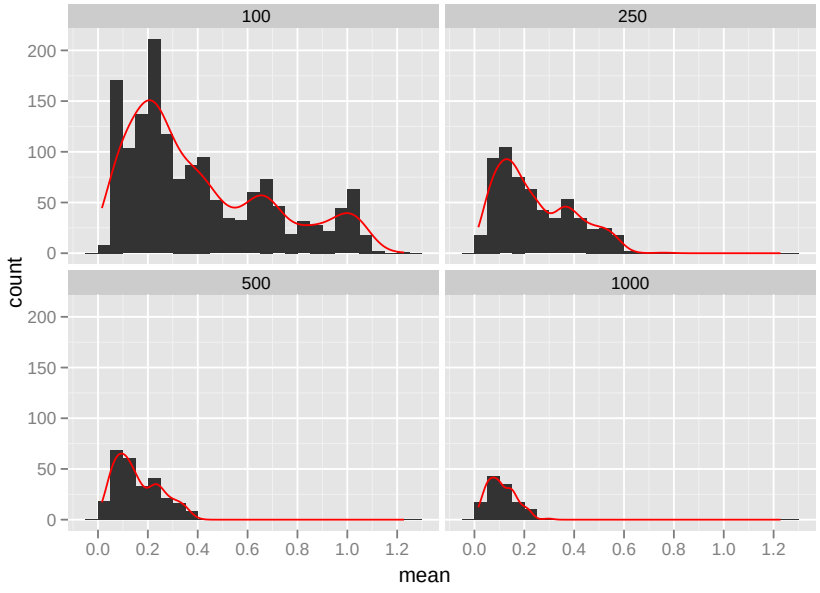
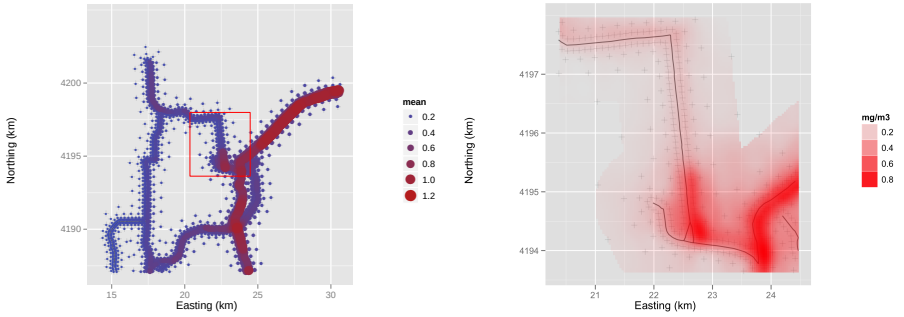


Figure 3: Mean predicted concentration, by distance-to-roadway.

4.2 Mapping



(a) Bubble plot

(b) Interpolated subregion

Figure 4: Estimated concentrations, with receptors shown as crosshairs.

The `ggplot` package makes it easy to generate maps (Figure 4a). First we define the bounds of the map (San Francisco county); then we add a `geom_point` layer, binding the size, color, and stacking order of the points to our variable of interest. (For more information on using `ggplot`, consult the `ggplot` documentation.) We also add a rectangle to highlight a sub-region of interest.

```
> bounds <- Rectangle(SF_county.shp)
> map <- ggplot(agg, bounds=bounds)
> bubbles <- geom_point(aes(x, y, size=mean, color=mean, order=mean))
> region <- resize(bounds, 0.25)
> box <- geom_rect(aes(xmin=xmin, ymin=ymin, xmax=xmax, ymax=ymax),
          fill=NA, color='red', data=as(region, 'data.frame'))
> show(map + bubbles + box)
```

4.3 Interpolation

As an alternative to the bubble plot, it's possible to construct a raster image by interpolating a summary statistic back to a regular grid (Figure 4b). We can also change the extent of the map, so that we zoom in on our region of interest.

Here we intersect the region of interest defined above (red box, Figure 4a) with a 1 km buffer constructed around the highways. Then we sample the resulting sub-region with `spsample`, such that we obtain a regular grid having 10,000 points:

```
> buf <- gBuffer(centerlines(lnk), width=1e3)
> buf <- gIntersection(buf, as(region, 'SpatialPolygons'))
> grd <- spsample(buf, n=1e4, type='regular', offset=c(0.5, 0.5))
> coordnames(grd) <- c('x', 'y')
```

After the grid has been constructed, a method for interpolation must be selected. Here, we use multilevel B-splines [5], as implemented by the `MBA` package.

Note that we replace the primary plot data by using the `%+%` operator. We also use `geom_tile`, instead of `geom_point`, to construct the "heatmap".

```
> require(MBA)
> obs <- cbind(coordinates(spdat), z=spdat$mean)
> fit <- mba.points(obs, coordinates(grd), verbose=FALSE)
> srf <- with(fit, as.data.frame(xyz.est))
> map <- ggplot(agg, bounds=region) %+% srf
> show(map + geom_tile(aes(x, y, alpha=z), fill='red') +
         scale_alpha('mg/m3', to=c(0, 1)))
```

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., 1979.

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

[3] CARB. EMFAC2007 Version 2.30 User Guide: Calculating Emission Inventories for Vehicles in California. Technical report, California Air Resources Board, 2006.

[4] CARB. Motor Vehicle Emission Factor / Emission Inventory Model, EMFAC, 2007.

[5] S. Lee, G. Wolberg, and S. Y. Shin. Scattered data interpolation with multilevel B-splines. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):229–244, 1997.