An Introduction to Steady-State Gaussian Dispersion Modeling in R

David Holstius

October 16, 2012

1 Introduction

This vignette is a brief introduction to `plume`, a package that illustrates the basics of atmospheric dispersion modeling in R. The package itself is an example of a tool for reproducible environmental modeling research, complete with test cases and documentation (including this document).

Throughout this vignette we will make use of the excellent `ggplot2` package created by Hadley Wickham. For more information on creating plots with `ggplot2`, type `help(package='ggplot2')` at the R prompt. We will also work toward integration with the `sp` library, a foundational package for anyone working with spatial data in R.

2 Steady-State Gaussian Dispersion

2.1 Modeling a Point Source

Assuming a source of emissions with constant emission rate Q, and a steady wind field with speed u, the concentration C at any location (x, y, z) downwind can be conceptualized as the product of three terms.

$$C = \frac{Q}{u} \cdot \frac{f}{\sigma_y \sqrt{2\pi}} \cdot \frac{g}{\sigma_z \sqrt{2\pi}} \tag{1}$$

where

$$f = exp\left[\frac{-y^2}{2\sigma_y^2}\right]$$

and

$$g = exp\left[\frac{-(z-H)^2}{2\sigma_z^2}\right] + exp\left[\frac{-(z+H)^2}{2\sigma_z^2}\right]$$

The orientation of the wind field is, by definition, along the positive x-axis. From the first term we can see that C is directly proportional to Q, and inversely proportional to u. Thus, the equation is ill-defined for low wind speeds (as u goes to 0, C goes to infinity). Q is given in grams per second (g/s), and H in meters (m) above ground level.

The second term captures the crosswind diffusivity, and the third term captures the vertical diffusivity. Both σ_y and σ_z are curves parameterized by x. At any fixed distance x along the wind vector, the cross-section of the plume is a continuous two-dimensional Gaussian distribution described by these two curves.

In this very basic example, we imagine the ground to be like a mirror: perfectly level, frictionless, and flat. However, rather than absorbing the plume aerosol, the ground will "reflect" it back toward the sky. The sky, in contrast, is an infinite sink; there is no inversion layer to reflect the plume back down.

The difference between the height of the emission source—which might be something like an incinerator stack—and the receptor height is expressed by $(z - H)$. The curious $(z + H)$ term actually captures the plume's reflection. It can be imagined as the contribution from an identical emissions source which is located at $-H$ meters; that is, H meters *below* the ground plane.

A rule of thumb is that the wind must be blowing at least 1.0 meters per second (m/s) for our model to be sensible. Under these conditions, the transport of the plume is dominated by advection, and diffusion along the wind vector is negligible. Diffusion in the crosswind and vertical directions, however, gives the plume its characteristic shape. The diffusivity field σ describes the lateral (σ_y) and vertical (σ_z) diffusivity at distance x along the wind vector.

2.2 Parameterizing the Plume

We use the `GaussianPlume` function to construct a particular plume fitting the general form of Eqn (1):

```
> require(plume)
> Q <- 50.0   # emission rate, g/s
> H <- 10.0   # source height, m
> u <- 2.0    # wind speed, m/s
> plume <- GaussianPlume(Q, H, u, sigma=PasquillGifford('D'))
```

`GaussianPlume` is a *factory function*: a function that returns another function. The object `plume`, to which we assign the result, is now an instance of a new function. To inspect the function body, just type the following:

```
> show(plume)
```

In the code, you can see there are references to `sigma` as well as Q, u, and H, although none are passed as arguments. The trick here is that we have taken advantage of R's *lexical scoping*. When the `plume` function was constructed by `GaussianPlume`, all of these variables were in scope. They have therefore been

"captured" within the scope of this particular function instance, and we can refer to them there.

`PasquillGifford` is also a factory function. It takes a single parameter, `stability`, which is the Pasquill stability class corresponding to prevailing meteorological conditions. This in turn parameterizes the curves σ_y and σ_z, which are obtained when the function returned by `PasquillGifford` is invoked.[1] Underlying these curves are the same experimentally derived σ parameters used in the U.S. EPA's Industrial Source Control (ISC) family of models.

Any `sigma` function may be supplied as an argument to `GaussianPlume`, as long as it accepts a numeric vector x and returns a list with numeric y and z components. You might experiment by using `PasquillGifford('B')` or `PasquillGifford('E')` instead of `PasquillGifford('D')`.

2.3 Predicting Concentrations at Receptors

Having defined our plume, the next thing to do is specify the locations at which we would like to compute dispersed concentrations. This is our *receptor grid*.

Here, we construct a regular Cartesian grid with 20-meter spacing. We set the receptor height z to 1.8 meters, corresponding to the height of an "average" adult. Our coordinate frame is in meters; all locations are relative to the emission source, which is at $(0, 0)$. The positive x-axis corresponds to the direction the wind is blowing.

```
> resolution <- 20
> locations <- expand.grid(
          x = seq(20, 1500, by=resolution),
          y = seq(-500, 500, by=resolution),
          z = 1.8)
```

We use the function `plume`, created earlier with a call to `GaussianPlume`, to compute the predicted concentration at each location of interest. Because the concentration is computed independently for each receptor, the time is essentially linear in the number of locations:

```
> system.time(gm3 <- plume(locations))
   user  system elapsed
  0.027   0.001   0.031
```

`plume(locations)` returns a vector, which we re-bind to to the receptor locations with a call to `data.frame`. At the same time we multiply by 1×10^3, converting the units from g/m^3 to mg/m^3.

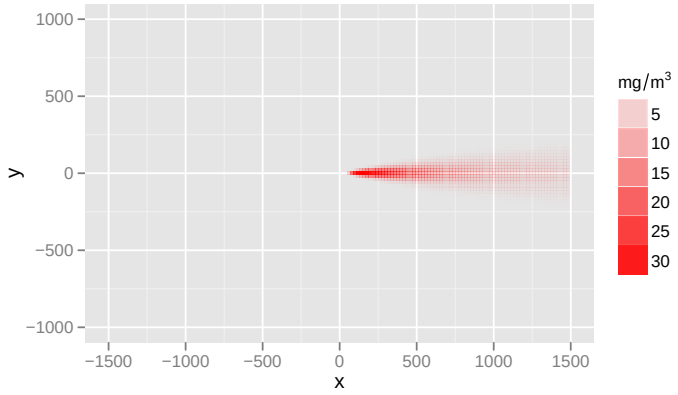
```
> cells <- data.frame(locations, mgm3 = gm3 * 1e3)
```

[1] For a diagram of curves corresponding to different Pasquill classes, see the Appendix.

2.4 Plotting Results

A useful visualization is obtained by plotting the concentrations in the form of a *heat map*, where the intensity of the color corresponds to the predicted concentration. This is a "view from above" the predicted dispersion field.

```
> require(ggplot2)
> p <- ggplot(subset(cells, mgm3 > 0.1), aes(x, y)) +
      geom_tile(aes(alpha=mgm3), fill='red') +
          scale_alpha(expression(mg/m^3), range=c(0, 1))
> show(p + xlim(c(-1500, 1500)) + ylim(c(-1000, 1000)) + coord_equal())
```



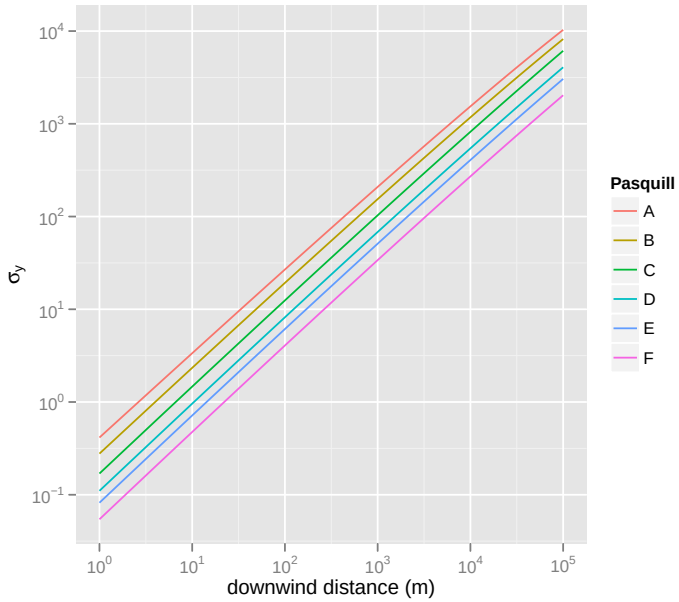
Plotting is sped up substantially by dropping all of the cells with a negligible concentration (less than 0.1 mg/m^3).

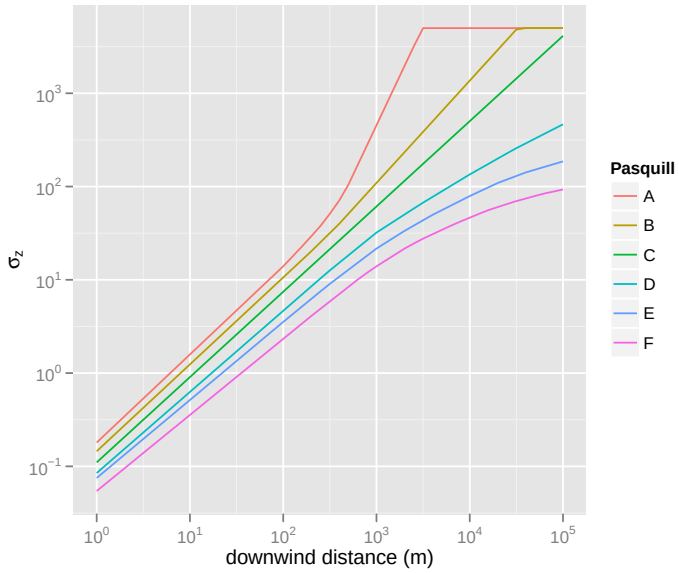
A Pasquill-Gifford Curves

This code demonstrates how to construct the `data.frame` of values used to generate the following plots of σ_y and σ_z:

```
> curves <- lapply(LETTERS[1:6], PasquillGifford)
> names(curves) <- LETTERS[1:6]
> x <- 10 ^ seq(0, 5, by=0.1)
> values <- ldply(curves, function(sigma) data.frame(sigma(x)))
> values <- rename(values, c(.id='Pasquill'))
> values$x <- x
```

To see the code used to generate the plots themselves, take a look at `inst/doc/plume.Rnw` in the `plume` package source!


10^4
10^3
10^2
10^1
10^0
10^{-1}
σ_y
10^0
10^1
10^2
10^3
10^4
10^5
downwind distance (m)
Pasquill
A
B
C
D
E
F


10^3
10^2
10^1
10^0
10^{-1}
σ_z
10^0
10^1
10^2
10^3
10^4
10^5
downwind distance (m)
Pasquill
A
B
C
D
E
F