Harmonizing the Lotka-Volterra and MESS models

A. J. Rominger

5/25/2022

The Lotka-Volterra competition model

The Lotka-Volterra competition model for two species (here after refered to as LV) looks like this

$$\frac{dx_1}{dt} = r_1 x_1 \left(1 - \frac{x_1 + \alpha_{12} x_2}{K_1}\right) \tag{1}$$

$$\frac{dx_2}{dt} = r_2 x_2 \left(1 - \frac{x_2 + \alpha_{21} x_1}{K_2}\right) \tag{2}$$

Our goal is to put this in terms of birth and death rates so we can convert these differential equations to an individual-based model like MESS.

We'll look at just the equation for species 1, recognizing the two equations are the same except for subscripts. The equation we want is simply the per capita birth rate (times population size) minus the per capita death rate (time population size)

$$\frac{dx_1}{dt} = B(x_1)x_1 - D(x_1)x_1 \tag{3}$$

Now we need to find meaningful forms for $B(x_1)$ and $D(x_1)$ that are consistent with the LV equations. A sensible option is

$$B(x_1) = \lambda_1^{(0)} \tag{4}$$

$$D(x_1) = \mu_1^{(0)} + \mu_1^{(1)} x_1 + \mu_1^{(2)} x_2 \tag{5}$$

A note on the subscripts and superscripts: the subscript refers to the target species (e.g. the equations we wrote out are for species 1, thus the subscript is 1); the superscript in parentheses refers to the origin of effect (e.g. a superscript of (0) is a zeroth order effect—like an intercept—while the term $\mu_1^{(0)}$ is the effect of species 2 on species 1).

These equations say that the *per capita* birth rate is constant across x_1 while the *per capita* death rate increases linearly with x_1 and with x_2.

Plugging these form of $B(x_1)$ nd $D(x_1)$ into eq. (3) we get

$$\frac{dx_1}{dt} = \lambda_1^{(0)} x_1 - \mu_1^{(0)} x_1 - \mu_1^{(1)} x_1^2 - \mu_1^{(2)} x_1 x_2$$

Rearranging eq. (1) we can quickly see the equivalent terms

$$\frac{dx_1}{dt} = r_1 x_1 - \frac{r_1}{K_1} x_1^2 - \frac{r_1 \alpha_{12}}{K_1} x_1 x_2$$

Which gives us

$$r_1 = \lambda_1^{(0)} - \mu_1^{(0)} \tag{6}$$
$$\frac{r_1}{K_1} = \mu_1^{(1)} \tag{7}$$
$$\frac{r_1 \alpha_{12}}{K_1} = \mu_1^{(2)} \tag{8}$$

A few nice things to note: we confirm that r_1 is the "rate of increase when rare" (i.e. $\lambda_1^{(0)} - \mu_1^{(0)}$); and we see that "carrying capacity" is inversely proportional to the slopes of the death rates with respect to x_1 and x_2, i.e., density dependence.

We can also see that the standard LV parameterization has 3 parameters, while the $B - D$ parameterization has 4 parameters. Specifically, $\lambda_1^{(0)}$ and $\mu_1^{(0)}$ are not distinguishable, i.e., infinitely many different combinations of $\lambda_1^{(0)}$ and $\mu_1^{(0)}$ will yield the same outcome as long as those combinations obey $r_1 = \lambda_1^{(0)} - \mu_1^{(0)}$. Luckily this can be easily corrected in the two species system, specifically set

$$\lambda_i^{(0)} = \lambda_j^{(0)} = B_0 \tag{9}$$
$$\mu_i^{(0)} = 0 \tag{10}$$

This gives us

$$r_i = B_0 \tag{11}$$
$$r_j = B_0 - \mu_j^{(0)} \tag{12}$$

For the complete 2 species system we now have 6 parameters for the classic LV parameterization and 6 parameters for the $B - D$ parameterization. We can choose whether $r_1 = B_0$ or $r_2 = B_2$ such that $\mu_j^{(0)}$ is a positive number.

Taking the LV model to an individual-based framework

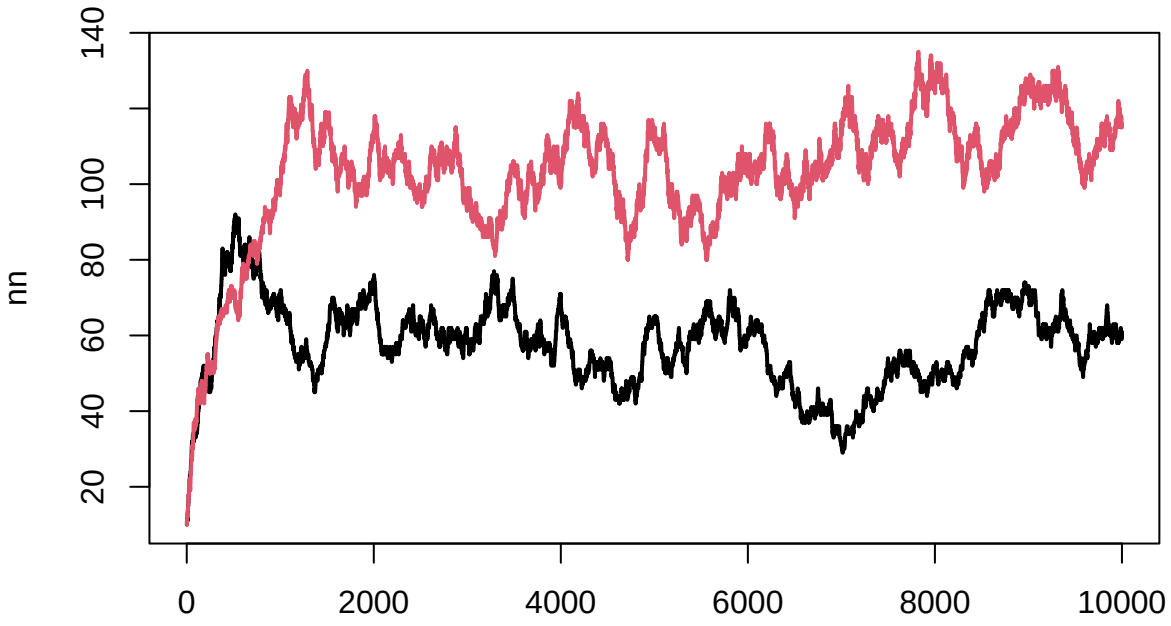
To start, let's convert the above birth and death rates to birth and death probabilities. This is done simply by normalizing the rates

$$P_B(x_1) = \frac{B(x_1)}{B(x_1) + D(x_1)} \tag{13}$$
$$= \frac{\lambda_1^{(0)}}{\lambda_1^{(0)} + \mu_1^{(0)} + \mu_1^{(1)} x_1 + \mu_1^{(2)} x_2} \tag{14}$$
$$P_D(x_1) = \frac{D(x_1)}{B(x_1) + D(x_1)} \tag{15}$$
$$= \frac{\mu_1^{(0)} + \mu_1^{(1)} x_1 + \mu_1^{(2)} x_2}{\lambda_1^{(0)} + \mu_1^{(0)} + \mu_1^{(1)} x_1 + \mu_1^{(2)} x_2} \tag{16}$$

Because we normalized the *per capita* rates, we actually now have the probability of birth or death for one individual of species 1. That means we've arrived at an individual-based model.

```
#' simulate individual-based lotka-voltera
#'
#' @param x0 initial condition, should be 2-long int vector
#' @param params a named list of the model params:
#'     * `r1` growth rate for sp 1
#'     * `K1` carrying capacity for sp 1
#'     * `alpha12` effect of sp 2 on sp 1
#'     * `r2` growth rate for sp 2
#'     * `K2` carrying capacity of sp 2
#'     * `alpha21` effect of sp 1 on 2
#' @param nreps single int, the number of timesteps
#' @return matrix of nrow equal to `nreps` and ncol equal to 2 (one for each spp)



lvclassic <- function(x0, params, nreps) {
    # re-parameterize
    imax <- which.max(c(params$r1, params$r2))
    B0 <- max(params$r1, params$r2)

    mu10 <- ifelse(imax == 1, 0, B0 - params$r1)
    mu20 <- ifelse(imax == 2, 0, B0 - params$r2)

    la10 <- params$r1 + mu10
    la20 <- params$r2 + mu20

    mu11 <- params$r1 / params$K1
    mu22 <- params$r2 / params$K2

    mu12 <- params$r1 * params$alpha12 / params$K1
    mu21 <- params$r2 * params$alpha21 / params$K2

    # initialize
    xt <-  matrix(NA, nrow = nreps, ncol = 2)
    xt[1, ] <- x <- x0

    for(b in 2:nreps) {
        # note: current population state is `x`

        if(all(x <= 0)) break

        # rates
        births <- c(la10, la20) * x
        deaths <- c(mu10 + mu11 * x[1] + mu12 * x[2],
                    mu20 + mu22 * x[2] + mu21 * x[1]) * x

        # which event happened
        # 1 = sp 1 birth
        # 2 = sp 2 birth
        # 3 = sp 1 death
        # 4 = sp 2 death
```

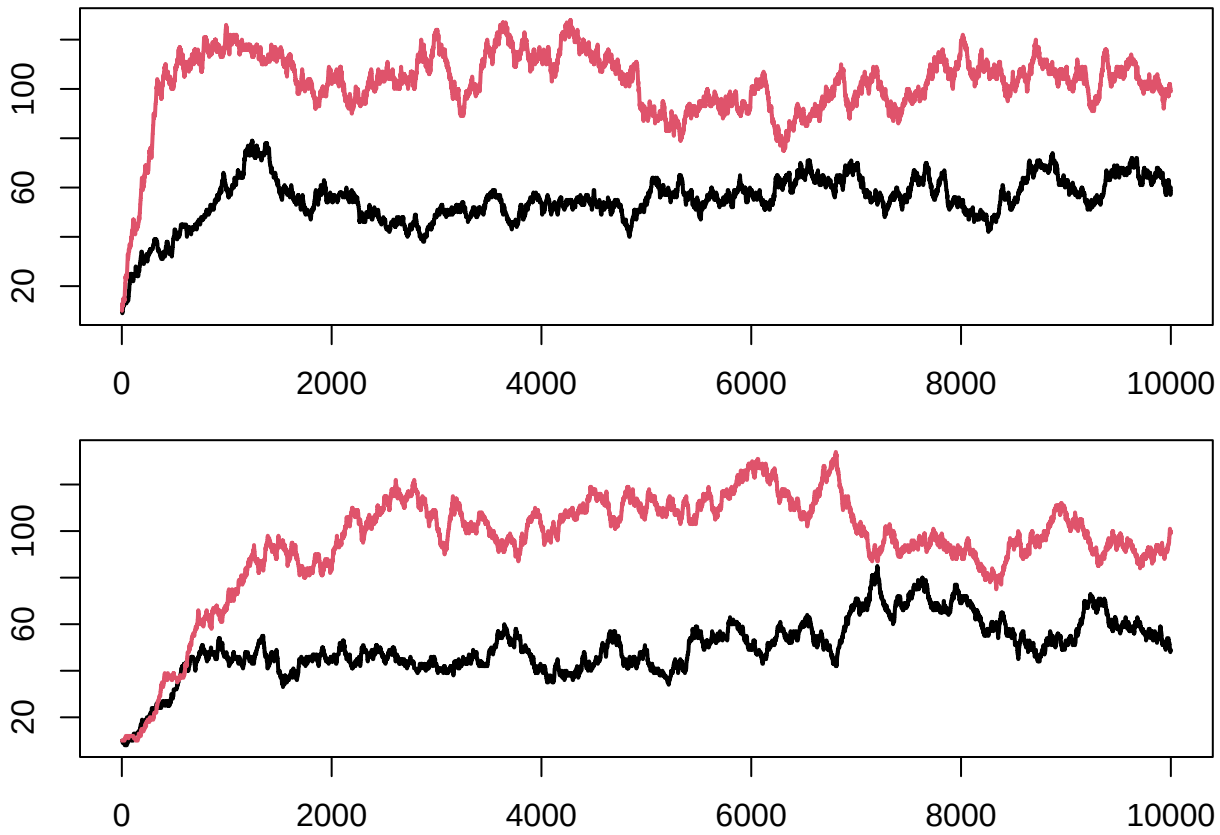
```
        e <- sample(1:4, 1, prob = c(births, deaths))

        # update
        if(e <= 2) {
            # birth
            i <- e
            x[i] <- x[i] + 1
        } else {
            # death
            i <- e - 2
            x[i] <- x[i] - 1
        }

        xt[b, ] <- x
    }

    return(xt)
}


p <- list(r1 = 1.2,
          K1 = 100,
          alpha12 = 0.4,
          r2 = 1.1,
          K2 = 120,
          alpha21 = 0.2)

nn <- lvclassic(x0 = c(10, 10), params = p, nreps = 10000)

matplot(nn, type = 'l', lty = 1, lwd = 2)
```



Now make a function where time steps are replacement events (e.g. death then birth)

```r
#' simulate individual-based lotka-voltera
#'
#' @param x0 initial condition, should be 2-long int vector
#' @param params a named list of the model params:
#'     * `r1` growth rate for sp 1
#'     * `K1` carrying capacity for sp 1
#'     * `alpha12` effect of sp 2 on sp 1
#'     * `r2` growth rate for sp 2
#'     * `K2` carrying capacity of sp 2
#'     * `alpha21` effect of sp 1 on 2
#' @param nreps single int, the number of timesteps
#' @return matrix of nrow equal to `nreps` and ncol equal to 2 (one for each spp)



lvrep <- function(x0, params, nreps) {
    # re-parameterize
    imax <- which.max(c(params$r1, params$r2))
    B0 <- max(params$r1, params$r2)

    mu10 <- ifelse(imax == 1, 0, B0 - params$r1)
    mu20 <- ifelse(imax == 2, 0, B0 - params$r2)

    la10 <- params$r1 + mu10
    la20 <- params$r2 + mu20

    mu11 <- params$r1 / params$K1
    mu22 <- params$r2 / params$K2

    mu12 <- params$r1 * params$alpha12 / params$K1
    mu21 <- params$r2 * params$alpha21 / params$K2

    J <- params$K1 + params$K2
    x0 <- c(x0, J - sum(x0)) # add "rocks"

    # initialize
    xt <-  matrix(NA, nrow = nreps, ncol = 3)
    xt[1, ] <- x <- x0

    for(b in 2:nreps) {
        # note: current population state is `x`

        if(all(x <= 0)) break

        # death rates
        deaths <- c(mu10 + mu11 * x[1] + mu12 * x[2],
                    mu20 + mu22 * x[2] + mu21 * x[1],
                    1) * x

        # who dies
        idead <- sample(1:3, 1, prob = deaths)
        x[idead] <- x[idead] - 1
```

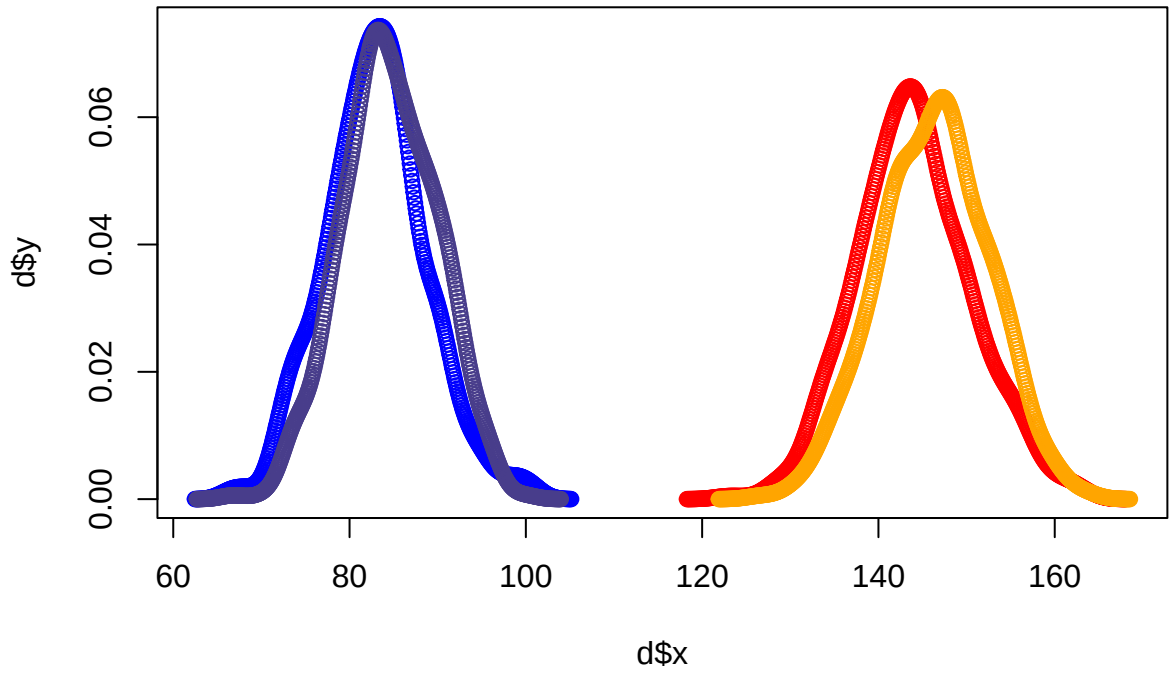
```
        # birth rates
        births <- c(la10, la20, 1) * (x + c(0, 0, 1)) # the added one
                                                      # incorporates the extra
                                                      # immigration prob

        # who replaces
        iborn <- sample(1:3, 1, prob = births)
        x[iborn] <- x[iborn] + 1

        # update
        xt[b, ] <- x
    }

    return(xt)
}


p <- list(r1 = 1.2,
          K1 = 100,
          alpha12 = 0.4,
          r2 = 1.1,
          K2 = 120,
          alpha21 = 0.2)

nnZS <- lvrep(x0 = c(10, 10), params = p, nreps = 10000)
nnLV <- lvclassic(x0 = c(10, 10), params = p, nreps = 10000)

par(mfcol = c(2, 1), mar = c(2, 2, 0, 0) + 0.5)
matplot(nnLV[, 1:2], type = 'l', lty = 1, lwd = 2)
matplot(nnZS[, 1:2], type = 'l', lty = 1, lwd = 2)
```



The raw timing is clearly different, but let's see if equilib is the same. We'll say equilib is achieved post 4000 iterations for both versions

```
p <- list(r1 = 1.4,
          K1 = 180,
          alpha12 = 0.4,
          r2 = 1.1,
          K2 = 100,
          alpha21 = 0.1)

sims <- parallel::mclapply(1:500, mc.cores = 10, FUN = function(i) {
    nnZS <- lvrep(x0 = c(10, 10), params = p, nreps = 10000)[-(1:4000), 1:2]
    nnLV <- lvclassic(x0 = c(10, 10), params = p, nreps = 10000)[-(1:4000), ]

    return(c(colMeans(nnZS), colMeans(nnLV)))
})

sims <- do.call(rbind, sims)

d <- lapply(1:ncol(sims), function(i) {
    thisD <- density(sims[, i])
    o <- as.data.frame(thisD[c('x', 'y')])

    return(cbind(ID = i, o))
})

d <- do.call(rbind, d)
```

```
plot(d$x, d$y, col = c('red', 'blue', 'orange', 'darkslateblue')[d$ID])
```



Converting LV parameters to MESS parameters

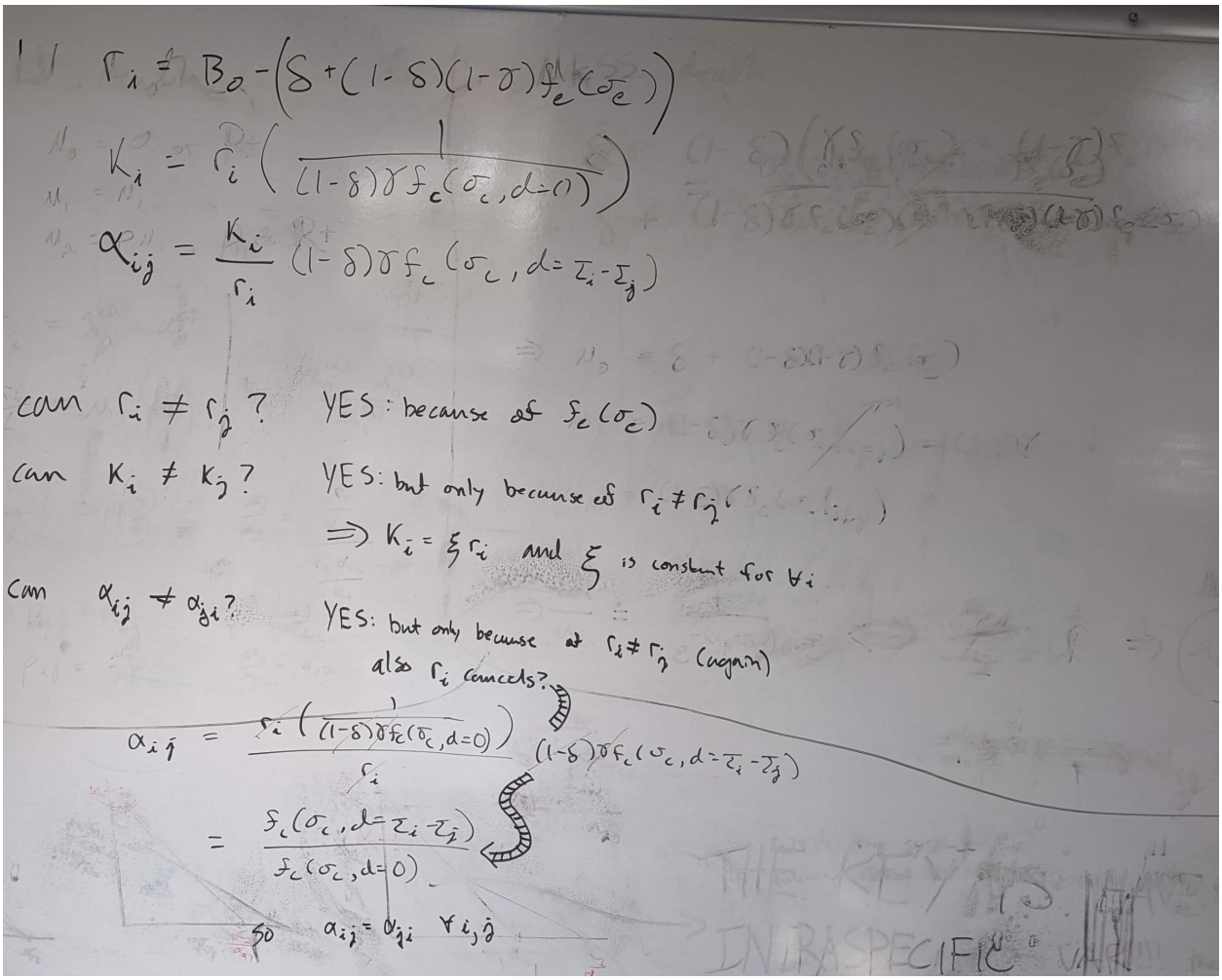
Let's assume that the MESS death rate for a two species system looks like

$$D_{MESS}(x_i) = \delta + (1 - \delta)\left(\gamma f_c(\sigma_c, d) + (1 - \gamma) f_e(\sigma_e)\right)$$

Where:

- δ is the proportion of death due to pure chance ($\delta = 1$ means full neutral)
- γ controls the relative contributions of death due to competition versus environmental filtering
- $f_c(\sigma_c, d)$ is the Gaussian curve that determines death due to competition (d is the trait distance between species, $d = 0$ is intra-specific competition)
- $f_e(\sigma_e)$ is the Gaussian curve that determines death due to environmental filtering (which also depends on trait distance to the optimum, but for notational simplicity I'm not including that d term).

Then we have:


$r_i = B_0 - \left(\delta + (1-\delta)(1-\gamma) f_c(\sigma_c)\right)$
$K_i = r_i \left(\frac{1}{(1-\delta)\gamma f_c(\sigma_c, d=0)} \right)$
$\alpha_{ij} = \frac{K_i}{r_i} (1-\delta)\gamma f_c(\sigma_c, d = z_i - z_j)$
can $r_i \neq r_j$? YES: because of $f_c(\sigma_c)$
can $K_i \neq K_j$? YES: but only because of $r_i \neq r_j$
$\Rightarrow K_i = \xi r_i$ and ξ is constant for $\forall i$
can $\alpha_{ij} \neq \alpha_{ji}$? YES: but only because of $r_i \neq r_j$ (again)
also r_i cancels?
$\alpha_{ij} = \frac{r_i \left(\frac{1}{(1-\delta)\gamma f_c(\sigma_c, d=0)} \right)}{r_i} (1-\delta)\gamma f_c(\sigma_c, d = z_i - z_j)$
$= \frac{f_c(\sigma_c, d = z_i - z_j)}{f_c(\sigma_c, d=0)}$
so $\alpha_{ij} = \alpha_{ji} \quad \forall i, j$