CONCEPTUAL FRAMEWORK

- Density dependent matrices require 3 things to generate V'
  - Density dependent expressions
    - E.g. $s_i = \frac{1}{1+ e^{-(\beta_1 * U + \beta_0)}}$ where U = $\sum_{i=1}^{5} V_i$
  - Population vector (V)
  - Fixed parameters (F, T, etc)

$$\overset{V'}{\begin{bmatrix} Seed1 \\ Seed2 \\ Small \\ Medium \\ Big \end{bmatrix}_{t+1}} = \overset{A}{\begin{bmatrix} 0 & 0 & F_s & F_m & F_b \\ S_{seed} & 0 & 0 & 0 & 0 \\ g_1 & g_2 & S_s T_{s,s} & S_m T_{m,s} & S_b T_{b,s} \\ 0 & 0 & S_s T_{s,m} & S_m T_{m,m} & S_b T_{b,m} \\ 0 & 0 & S_s T_{s,b} & S_m T_{m,b} & S_b T_{b,b} \end{bmatrix}} \overset{V}{\begin{bmatrix} Seed1 \\ Seed2 \\ Small \\ Medium \\ Big \end{bmatrix}_{t}}$$

CONCEPTUAL FRAMEWORK

- Density dependent matrices require 3 things to generate V'
  - Density dependent expressions
    - E.g. $s_i = \frac{1}{1+ e^{-(\beta_1 * U + \beta_0)}}$ where U = $\sum_{i=1}^{5} V_i$
    - The matrix may contain any number of these
    - We only need tables for
      - βs/constants and
      - Density dependent expressions and the matrix itself

$$\overset{V'}{\begin{bmatrix} Seed1 \\ Seed2 \\ Small \\ Medium \\ Big \end{bmatrix}_{t+1}} = \overset{A}{\begin{bmatrix} 0 & 0 & F_s & F_m & F_b \\ S_{seed} & 0 & 0 & 0 & 0 \\ g_1 & g_2 & S_s T_{s,s} & S_m T_{m,s} & S_b T_{b,s} \\ 0 & 0 & S_s T_{s,m} & S_m T_{m,m} & S_b T_{b,m} \\ 0 & 0 & S_s T_{s,b} & S_m T_{m,b} & S_b T_{b,b} \end{bmatrix}} \overset{V}{\begin{bmatrix} Seed1 \\ Seed2 \\ Small \\ Medium \\ Big \end{bmatrix}_{t}}$$

CONCEPTUAL FRAMEWORK

- Density dependent matrices require 3 things to generate V’
  - Population vector
    - We only need to store the initial one (V), we generate V’ via iteration
    - This can be stored in string format (e.g. “c(seed1 = 10, seed2 = 5, small = 2, medium = 2, large = 1)”)
    - Alternatively, we can store them as constants with a metadata indicator that these values are part of the population vector (whichever is easier on the SQL/Compadrino training side of things)

$$\overset{V'}{\begin{bmatrix} Seed1 \\ Seed2 \\ Small \\ Medium \\ Big \end{bmatrix}_{t+1}} = \overset{A}{\begin{bmatrix} 0 & 0 & F_s & F_m & F_b \\ S_{seed} & 0 & 0 & 0 & 0 \\ g_1 & g_2 & S_sT_{s,s} & S_mT_{m,s} & S_bT_{b,s} \\ 0 & 0 & S_sT_{s,m} & S_mT_{m,m} & S_bT_{b,m} \\ 0 & 0 & S_sT_{s,b} & S_mT_{m,b} & S_bT_{b,b} \end{bmatrix}} \overset{V}{\begin{bmatrix} Seed1 \\ Seed2 \\ Small \\ Medium \\ Big \end{bmatrix}_{t}}$$

IMPLEMENTATION IN RCOMPADRE

- Flexibility is critical
  - Currently, this focuses on the generic function: *iterate_dd_mat()*
  - Methods for *iterate_dd_mat.CompadreDDM ()* and *iterate_dd_mat.list()*
    - *Iterate_dd_mat.CompadreDDM()* is for usage on matrices stored in Compadre
    - *Iterate_dd_mat.list()* is for user-generated data (which may end up in Compadre one day!)
    - These are really just intermediate steps to get the data into a consistent format, then an internal call to *.iterate_dd_mat_impl()*
    - *.iterate_dd_mat_impl()* does the iterations and stores desired outputs
  - Helper functions *make_mat_exprs()* and *make_data_list()* to assist with generating correct data formats
    - Since this is implemented using the Tidy Eval framework, *make_mat_exprs* will need to be a fairly smart (and probably somewhat complicated) function to figure out what needs quoting and what needs evaluating!
    - Quoting and evaluating? Return to that later…

IMPLEMENTATION IN RCOMPADRE

- Flexibility is critical
  - Benefits to this approach
    - Won't require re-designing the existing Rcompadre implementation
    - Probably will only require one additional metadata column (e.g. has_dd or something like that)
    - I don't *think* this will require too much effort from Tony and Austin either, but I know a lot less about how that works
    - Minimal re-training of Compadrinos
      - We really just need to teach them how to translate math in the papers into pseudo R-code
    - Providing a suite of functions for fitting these creates a pipeline for new data to be incorporated after publication

IMPLEMENTATION IN RCOMPADRE

- Flexibility is critical
  - Currently, this focuses on the generic function: *iterate_dd_mat()*
  - Currently writing methods for *iterate_dd_mat.CompadreDDM ()* and *iterate_dd_mat.list()*
    - *Iterate_dd_mat.CompadreDDM()* is for usage on matrices stored in Compadre
    - List w/ components:
      - All density dependent expressions in the following format:
        - s_2 = quo(1/(1 + exp(bs2_2 * eval_tidy(u_i) + bs2_1 * eval_tidy(t_i) + bs2_0)))
        - This can be repeated as many times as needed to get all the necessary expressions (see example here)
        - Final slot is the expression for the matrix itself
        - mat_expr = quo(

```
                 matrix(
                        c(
                          1 - g_2, 0, v * (1-g_1) * eval_tidy(f),
                          g_2 * s_1, 0, v * g_1 * s_1 * eval_tidy(f),
                          0, eval_tidy(s_2) * eval_tidy(s_3), 0
                          ),
                        nrow = 3,
                        byrow = TRUE
                      )
                    )
```

IMPLEMENTATION IN RCOMPADRE

- Flexibility is critical
  - Currently, this focuses on the generic function: *iterate_dd_mat()*
  - Currently writing methods for *iterate_dd_mat.CompadreDDM ()* and *iterate_dd_mat.list()*
    - *Iterate_dd_mat.CompadreDDM()* is for usage on matrices stored in Compadre
    - List w/ components:
      - *data_list:*
        - *v = 0.8228,*
        - *g_1 = 0.5503,*
        - *g_2 = 0.3171,*
        - *bs2_2 = 0.0016,*
        - *bs2_1 = -0.0664,*
        - *bs2_0 = -0.156,*
        - *bs3_1 = -0.289,*
        - *bf_1 = -0.0389,*
        - *bf_0 = 7.489,*
        - *s_1 = 0.5,*
        - *initial_population_vector = c(s = 10, r = 0, a = 0)*

IMPLEMENTATION IN RCOMPADRE

- Flexibility is critical
  - These expressions aren't simple! They're hideous! WTF???
    - s_2 = quo(1/(1 + exp(bs2_2 * eval_tidy(u_i) + bs2_1 * eval_tidy(t_i) + bs2_0)))
    - mat_expr = quo(

```
                    matrix(
                          c(
                            1 - g_2, 0, v * (1-g_1) * eval_tidy(f),
                            g_2 * s_1, 0, v * g_1 * s_1 * eval_tidy(f),
                            0, eval_tidy(s_2) * eval_tidy(s_3), 0
                           ),
                          nrow = 3,
                          byrow = TRUE
                         )
                        )
```

    - *Quo()* and *eval_tidy()* are used to capture without evaluating and evaluate expressions, respectively
      - This would force our package(s) to depend on *rlang* ☹ but *rlang* itself has no external dependencies ☺
    - We can use the left hand and right hand sides in *mat_exprs* to figure out which elements need *eval_tidy()* and then wrap those programmatically, so Compadrinos don't need to understand these concepts!*

- *still very much a work in progress ;)

IMPLEMENTATION IN RCOMPADRE

- Flexibility is critical
  - *Iterate_dd_mat.list()* is for user-generated data (which may end up in Compadre one day!)

  - This is essentially the same, except the user will need to specify their own data and matrix.
  - This works kind of like this (parentheses may be a little off…):

```
        dd_data_list <- list(mat_exprs = make_mat_exprs(foo = exp(b_1 * U + b_0),
                                                        bar = 1/(1+exp(-(b_1 * R + b_0))),
                                                        U = a + b + c,
                                                        R = a*b*c,
                                                          mat_expr = matrix(c(0, 2, 4,
                                                                              foo, s_1, 1,
                                                                              0, s_2, bar),
                                                                             nrow = 3,
                                                                             byrow = TRUE)),
                             data_list = make_data_list(s_1 = 0.7, s_2 = 0.9,
                                                              initial_population vector = c(a = 20, b = 2, c = 3))

            Iterate_dd_mat(data_list,…)
```

IMPLEMENTATION IN RCOMPADRE

- Flexibility is critical
  - Quoting with *rlang::quo()*
    - Basically, we capture an expression without forcing R to actually evaluate it
      - We can modify user defined expressions (*iterate_dd_mat.list()*) or modify database entries (*iterate_dd_mat.CompadreDDM()*)
      - Thus, we can intelligently generate a sequence of evaluation ensuring that data is always present *before* evaluation and is safely insulated from the users environment (where we don't know what variables have been created!)

  - Evaluating with *rlang::eval_tidy()*
    - If an expression in the matrix or in the *right hand side* of an assignment in density dependent expression appears as the *left hand side* of another expression, then it must be wrapped in a call to *eval_tidy()*
      - This is only true for expressions in the *mat_exprs* object, *not the data_list*

  - Employing this framework means we *should* be able to capture any type of matrix that a user might wish to fit, or has already fitted in the past

THE WAY FORWARD/STILL UP IN THE AIR

- More test cases and experimental integration into the database

- Somebody please check my logic and make sure I haven't totally botched it!

- Up in the air (feel free to add more!)
  - Default population vectors?
    - What do we do when those aren't supplied?

  - Additional covariates?
    - E.g. DEB models that have multiple variables and domains?

  - Current outputs for *iterate_dd_mat()* are stage vectors and lambda
    - What else to add?
  - Current settings for *iterate_dd_mat()* are # of generations, target output, and initial population vector.
    - What else to add?