PADRINO digitization guide

Sam Levin

Last updated: 2021-04-05

Contents

PADRINO

Welcome to the project! PADRINO is an open access database that aims to store text representations of Integral Projection Models (IPMs). An important goal is to ensure these models can be rebuilt *as published*,

though there are exceptions in some cases (see here). Beyond the functional forms and parameters, we also store metadata to help users select models for synthesis.

It is important to note that PADRINO does not store raw data (e.g. individual level data collected from field studies). Other projects have already set out to accomplish this goal (e.g. `BIEN`, `popler`, `BioTime`), and we do not believe in duplicating those efforts specifically for IPMs. However, there is still a gap to fill in terms of generating sets of kernels, population states, and life history traits from published models for pressing synthetic questions in ecology.

Throughout the course of this guide and the Writing PADRINO Expressions Guide, references to `ipmr` will pop up. `ipmr` is the *R* package that powers the re-implementation of PADRINO models in *R*. There is an introduction to `ipmr` here.[1]

Required tools

For digitizers, this project makes heavy use of GitHub, RStudio, Google Sheets, and Slack. If you do not already have a GitHub account, you can make one here and walk through the steps of linking GitHub and RStudio here. For the most part, you will only need to know how to clone a repository, and then how to keep your local version in sync with the online version (via pulling, pushing, and branching). Please let me know if you'd like a more thorough introduction to this during on boarding and I'll make sure to go through the linking/syncing steps with you! A technical setup guide is availabe here.

Initial setup

Once you've set up GitHub and RStudio to talk to each other, it will be useful to create a local copy of the Padrino repository on your computer. This will give you access to, among other things, local copies of this guide and the others, as well as a the "Email an Author" template. To do this:

1. Open Rstudio
2. In the top right corner, there will be a "Project: (None)" tab. Click this dropdown, and select "New Project".
3. Select "Version Control" from the first menu.
4. Select "Git" from the second menu.
5. Enter `https://github.com/levisc8/Padrino` into the "Repository URL" bar. Select a directory to place your version in using the "Create project as subdirectory of:".
6. Click "Create" button.
7. Once all files are downloaded and the new R session is started, click on the Git tab. This will usually be in the same Pane as the Environment and History tabs in RStudio.
8. Once in that tab, click "New Branch" and give your branch an easily recognizable name (e.g. "sam_branch"). The name should not have any spaces or apostrophes.

You can now safely edit documents and add notes as you like without affecting the main branch!

Steps to digitizing

There a few steps in digitizing an IPM in PADRINO. These are outlined briefly here, and each has a deeper dive (when applicable) somewhere below (with links to each).

[1] PADRINO is designed to be (pretty) language agnostic, and we would like to add ports to other languages (e.g. Julia, Python) after an initial stable release. However, for now, for now, there is an only *R*-based engine to use PADRINO.

Identify whether a publication contains an IPM

The first step of entering a paper is to find papers and store them in a place that the PADRINO team can find later. Below, we go through the steps this involves, in order.

1. Tracking our incoming `literature` channel on Slack, as well as checking other sources for papers we may have missed (e.g. Google Scholar, Web of Science, Twitter (seriously, it's a good source)).[2]
2. If a paper has an IPM, then we first add it to our shared GoogleSheet containing citation information, species names, author contact info, and digitization progress. This can be accessed here. Please email me at levisc8@gmail.com to get access.
3. Download a PDF of the paper into the shared Dropbox literature folder. Additionally, if the publication contains an Appendix with further information, store that as well. The folder's relative path within the PADRINO dropbox folder is: `Dropbox/PADRINO/Literature/PDF/KINGDOM-NAME`.
4. If you are ready to begin digitizing the paper, then update the "Digitized" and "Digitizer" columns in the Google Sheet to reflect that you are working on this paper.

Determine the number of `ipm_id`'s in the paper

The `ipm_id` currently takes the place of an SQL key. `ipm_id` is the only column that appears in every table in PADRINO. Some models may contain many rows in some tables while others contain none. PADRINO needs some piece of information to make sure we do not accidentally pull in information from a different model at build time. The `ipm_id` column fills this role. Because of the way that `ipmr` handles models with some grouping effects (e.g. plots, years, populations), it may be possible to re-construct many kernels using a slightly modified functional form (i.e. appending a suffix corresponding to the group effects). Grouping models saves us a lot of typing/copy+pasting (which can be error prone). The rules for when models can be grouped with the same `ipm_id` are as follows:

1. If there are many versions of the same sub-kernel, and all of these have the same functional form and differ only in their parameter values, we can use a single `ipm_id`.
    - The above situation is commonly the result of mixed effects models, or models where something like an experimental treatment is a fixed effect. More generally, if an underlying regression model contains some mixture of discrete and continuous predictors, then the result is going to be multiple kernels - 1 for each level of the discrete predictor. For example, a growth model fit to data from many transitions may use "year" as fixed or random effect. In this case, the functional form of the growth model is the same for all years, but the parameter values differ, generating a unique kernel for each year. We can use the same functional form to generate these unique kernels, so we can use a single `ipm_id` for this IPM.
2. If any of the underlying kernels have different functional forms across a discrete grouping variable, then we cannot combine them, and they must be split into separate `ipm_id`'s.
    - Unique functional forms in the same sub-kernel may happen when, for example, different sites or years generate such different demographic responses that the authors could not find a way to keep everything in one model. Generally, if the authors report using different regression models, or different kernels, for the same vital rate, then we will need to split up the model into different `ipm_id`'s.[3]

Once we have decided how many entries the paper is going to get, then we can begin entering the model. Since there may be multiple people working on PADRINO in parallel, we will discuss a system to keep your `ipm_id`'s unique from others during your first training session.

[2] on the slack channel, papers that have been posted there and contain an IPM get a thumbs up reaction, all others get a thumbs down. You can "reply in thread" to open a discussion about whether or not to include a paper.

[3] This does not refer to model selection procedures with multiple candidate models, only the final models used in the IPM!

Begin digitizing the model

Details on each column and each table are provided below. Some details on functional form syntax are given below, but a much longer introduction is available here, and an introduction to `ipmr` (which powers all of PADRINO) is available here.

Test the model

Testing the model requires testing the output against some expected target value. To do this, you'll need to install the `pdbDigitUtils` package with the following snippet:

```
if(!requireNamespace("remotes")) {
  install.packages("remotes")
}

remotes::install_github("levisc8/pdbDigitUtils")
```

This package contains a couple helpful functions for loading a locally stored development version of the database, and testing outputs from individual models. `read_pdb` loads the Excel version of the database into *R*, and `test_model` lets you test a newly entered model. The package is still under development, so additional functionality can be added as requested. Please use the Issues Tab on the Github repository to create function requests and report bugs.

If the newly entered model passes all tests, then it is ready to enter the production version of the database. At this point, I am still working out who should have access to that, so for now, just send me your version of the database via email (levisc8@gmail.com).

The tables

Next, we will walk through individual tables and the columns within them. The first is `Metadata`.

Metadata

The metadata table contains information on species taxonomy, publication information, study duration, ecoregion, and any treatments applied to the study population. The columns are:

1. `ipm_id`: This is a unique identifier for each model. It is 6 alphanumeric characters with no spaces.

Taxonomic information

2. `species_author`: This the Latin species name as used by the author in the manuscript. Note that this name may no longer be the accepted name. This has the format `genus_species` (e.g. `Ligustrum_obtusifolium`, `Ovis_aries`). Some authors may include additional information (sub-species, varieties, etc). These can be appended using underscores (e.g. `genus_species_subspp_var`).
    - NB: If you are already familiar with COMPADRE, we DO NOT use the the extra `_#` to distinguish between models/publications. For example, two publications in COMPADRE on `Lonicera_maackii` would be entereed as `Lonicera_maackii` and `Lonicera_maackii_2`. We just use different `ipm_id`'s and `apa_citation`'s to distinguish these models.
3. `species_accepted`: The accepted name of the species (currently from Global Names Resolver and Catalog of Life, but may switch to the Leipzig List for Plantae soon). This follows the format `genus_species`, but does not contain any varietal or sub-species information. You can use the

experimental `resolve_pdb_tax` function in `pdbDigitUtils` for this, though it may not work perfectly, and results should be checked.

The following taxonomic categories contain the `"tax_"` prefix to prevent naming ambiguity with some R functions.

4. `tax_genus`: The accepted genus.
5. `tax_family`: The accepted family.
6. `tax_order`: The accepted order.
7. `tax_class`: The accepted class.
8. `tax_phylum`: The accepted phylum.
9. `kingdom`: The kingdom.
10. `organism_type`: The type of organism. For plants, this is usually something like `"Herbaceous perennial"`, or `"Shrub"`. For animals, this could be, for example, `"mammal"` or `"reptile"`. See here for more details (but also do not hesitate to contact me if there instances that fall outside of the classification given there).
11. `dicot_monocot`: Whether the species is a dicotyledon or a monocotyledon (only applies to plants).
12. `angio_gymno`: Whether the species is a angiosperm or a gymosperm (only applies to plants).

Publication information

13. `authors`: The last names of each author on the manuscript, separated by a semicolon.
14. `journal`: The abbreviated name of the journal that the model appears in. This follows the BIOSIS format. Exceptions are when the source is not a journal (e.g. a PhD/MSc thesis, government report). In that case, we use something like `"PhD Thesis"` and then include a link in the `remark` column.
15. `pub_year`: The year the article was published.
16. `doi`: The DOI of the publication (NOT THE doi.org URL though!!).
17. `corresponding_author`: The last name of the corresponding author.
18. `email_year`: The corresponding author's email, along with the year of publication in parentheses to denote how old (and possibly inaccessible) it is. For example, this could `levisc8@gmail.com (2020)`. If you are able to find a more recent email address via Google, then this can also be used (this isn't necessarily expected though).
19. `remark`: Any qualitative comments you may have on the model. These can range from comments to accuracy of GPS coordinates to descriptions of the different levels of a treatment that was applied.
20. `apa_citation`: The full APA style citation for the paper.
21. `demog_appendix_link`: If there is one, a link to the Electronic Supplementary Material that contains further details/parameter values for the model.

Data collection information

21. `duration`: The duration of data collection used to implement the model. This is a crude measure, defined as `end_year - start_year + 1`, and does not account for years where data collection may have been skipped.
22. `start_year`: The year that demographic data collection began. Formatted `YYYY` (e.g. `1990`).

23. `start_month`: The month of the year that demographic data collection began. This is an integer between 1 and 12, where 1 corresponds to January, and 12 corresponds to December.
24. `end_year`: The final year of demographic data collection. Formatted `YYYY`.
25. `end_month`: The month of the year that demographic data collection concluded.
26. `periodicity`: Indicates the time step (periodicity) for which the seasonal, annual, or multi-annual IPM was constructed. For example, 1 indicates that the IPM iteration period is 1 year; 0.5 indicates that the IPM iterates once every 0.5 years or 6 months; 2 indicates that the IPM iteration occurs every 2 years.
27. `population_name`: The name of the population given by the author. For example, `"Bear Creek"`, or `"Havatselet"`. If the population names are missing, use sequential names in alphabetical order (e.g. `"A"`, `"B"`, `"C"`, etc).
28. `number_populations`: Sometimes, a `population_name` may encompass multiple sub-populations that are located close by. This integer specifies the number of populations/sub-populations that are described by the model.
29. `lat`: The decimal latitude of the population. Use the `dms_deg` function from `pdbDigitUtils` to generate this.
30. `lon`: The decimal longitude of the population. Use the `dms_deg` function from `pdbDigitUtils` to generate this.
31. `altitude`: The altitude above/below sea level, in meters.
32. `country`: The ISO3 country code for the country in which the population is located. These are defined here.
33. `continent`: The continent that the population is located on. Options are `n_america`, `s_america`, `oceania`, `asia`, `europe` and `africa`. Others may be added as needed.
34. `ecoregion`: The ecoregion, as defined by the World Wildlife Fund (see here).

Model information

35. `studied_sex`: The sex of the population studied. Options are `M` (male only), `F` (female only), `H` (hermaphrodites), `M/F` (males and females modeled separately, but in the same IPM), and `A` (all sexes studied together).
36. `eviction_used`: Whether or not the authors account for unintentional eviction. This should either be `TRUE` or `FALSE`. Only put `TRUE` if an eviction correction is explicitly discussed, or you find model code in the supplementary materials and are able to discern the correct `evict_type`.
37. `evict_type`: If an eviction correction was used, then the name of the method to correct it. Options are `stretched_domain`, `rescale_kernel`, `truncated_distributions` and `discrete_extrema`. `rescale_kernel` is the same as `truncated_distributions` and a relic of the past. Feel free to update those entries to `truncated_distributions` as you go.
    - NB: `stretched_domain` does not indicate that any function was applied to the kernels to return individuals. Rather, it just means that the authors extended the size boundaries some amount beyond the observed size distributions and relied on vital rate functions to prevent individuals from ever getting to the bounds. This information is included in PADRINO so users can understand what approach was used to deal with the problem, and, theoretically, try implementing models with different eviction corrections.
38. `treatment`: A brief description of any experimental treatment that the authors applied to a given population.

39. `has_time_lag`: Indicates whether any vital rates have a time lag (i.e. are a function of $n(z^*, t-1)$ rather than $n(z,t)$).
40. `has_age`: Indicates whether the model has age structure.
41. `has_dd`: Indicates whether the model incorporates density dependence.

Database specific information

42 `.embargo`: Have the authors requested an embargo period for the model?

43. `.embargo_date`: If so, when have they agreed to allow us to release it?

You've made it through the metadata table! Next, we will get into IPM specific details, starting with the state variables in use.

StateVariables

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.
2. `state_variable`: the name of the of state variable that the model uses. This is largely up to you to choose. It can be descriptive (e.g. `"dbh"`, `"leaf_area"`), or vague (`"size"`).
3. `discrete`: Whether or not the state variable is discretely or continuously distributed.

DiscreteStates

Ignore this table for now, I'm fairly certain it will be removed soon.

ContinuousDomains

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.
2. `state_variable`: the name of the continuous state variable. Should match the value from StateVariables table.
3. `domain`: likely not useful and slated for deletion once I confirm this, skip for now.
4. `lower`: the lower bound of the domain.
5. `upper`: the upper bound of the domain.
6. `kernel_id`: The name of the kernels that it appears in. Because of the way `ipmr` builds these models, you can actually omit the "K" values for new entries. They are present as a historical artefact, but will be removed eventually. The sub-kernel names should be separated by a semicolon (";").
7. `notes`: any qualitative observations you have about the domain itself.

IntegrationRules

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.
2. `state_variable`: the name of the continuous state variable. Should match the value from StateVariables table.
3. `domain`: likely not useful and slated for deletion once I confirm this, skip for now.

4. `n_meshpoints`: the number of meshpoints used for integration. The information this represents will vary depending on the integration rule. For now, the midpoint rule is the only one that's implemented in `ipmr`. This document will get updated to include formats for other integration rules as they are implemented in that package.
5. `integration_rule`: the name of the integration rule. If a paper uses something besides `"midpoint"`, write enter the name here, but do not worry about the `n_meshpoints` for now.
6. `kernel_id`: the kernel name that the integration rule applies to. I cannot think of any cases where a model uses different integration rules for different sub-kernels, but I do not *think* it would be problematic either, and tree IPMs may go that direction as they become increasingly sophisticated.

StateVectors

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.
2. `expression`: This is the name of the `state_variable` from the from the StateVariables table, with an `"n_"` appended to it to denote that it is a population state vector.
3. `n_bins`: The number of bins it will be discretized into. For discrete states, this should always be 1. For continuous states, it should match the `n_meshpoints` value from the IntegrationRules table.
4. `comment`: qualitative comments on the population state distribution function.

IpmKernels

Because of the way that `ipmr` implements models, we only need to digitize sub-kernels to generate a complete IPM. In other words, there shouldn't be any need to enter expressions that create the $K(z', z)$ iteration kernel, we only the need the $P(z', z)$ and $F(z', z)$. The rest of this guide is a sort of a quick-reference to remind you what goes where - for help writing kernel formulae, you should consult the Writing kernels, vital rate, and environmental stochasticity expressions guide.

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.
2. `kernel_id`: The name of the IPM sub-kernel.
3. `formula`: The formula describing how the vital rates combine to generate a sub-kernel. This field can make use of `ipmr`'s suffix syntax, so kernels with identical functional forms do not need to be re-entered to work with different parameter values.
4. `model_family`: One of 4 options:
    - `"CC"`: describes a continuous -> continuous transition.
    - `"DC"`: describes a discrete -> continuous transition.
    - `"CD"`: describes a continuous -> discrete transition.
    - `"DD"`: describes a discrete -> discrete transition.
5. `domain_start`: the name of the state variable that the sub-kernel acts on.
6. `domain_end`: the name of the state variable that the sub-kernel produces.

VitalRateExpr

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.

2. `demographic_parameter`: The demographic process that the vital rate relates to. For example, could be "Survival", "Growth", "Fecundity", or "Clonal".
3. `formula`: The mathematical formula for the vital rate. This field can make use of `ipmr`'s suffix syntax, so vital rates with identical functional forms do not need to be re-entered to work with different parameter values. These can be split out into different cells if they are too long to safely write in a single cell. Vital rates that include probability distributions (e.g. growth, recruitment) must have their own line, and only take the parameters that the distribution accepts. For example:
    - A growth kernel with a Gaussian distribution parameterized by a linear model must have two lines:
        - `"mu_g = int_g + slope_g * z_1"`
        - `"g = Norm(mu_g, sigma_g)"`
    - The following will not work:
        - `"g = Norm(int_g + slope_g * z_1, sigma_g)"`
    - See Writing kernels, vital rate, and environmental stochasticity expressions guide for more details. See PADRINO's Probability Distribution Dictionary for each distribution's notation.
4. `model_type`: Should be either "Evaluated" or "Substituted". Anything that contains a probability density function should be "Substituted", and everything else will be "Evaluated".
5. `kernel_id`: The sub-kernel(s) that use the vital rate expression. If it appears in more than one kernel, you can put the sub-kernel names here separated by a semicolon (e.g. "P; F").

ParameterValues

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.
2. `demographic_parameter`: The demographic process that the parameter relates to. For example, could be "Survival", "Growth", "Fecundity", or "Clonal". Use "General" for a parameter that appears in multiple sub-kernels.
3. `state_variable`: ignore for now. This column is probably going to get deleted.
4. `parameter_name`: The name of the parameter. This should match the name that appears in `VitalRateExpr$formula` or `IpmKernels$formula`. The exception here is when using the suffix syntax. In this case, the actual value of the suffix must replace the suffix itself, and the parameter value should change from level to level. Consider the following example:
    - In `VitalRateExprformula`, `"mu_g_yr = int_g_yr + slope_g * z_1"`. The `"yr"` suffix can take on values 2008:2010. We would need to enter 3 values in the ParamterValues table: `"int_g_2008"`, `"int_g_2009"`, and `"int_g_2010"`.
5. `parameter_value`: The numeric value for the parameter.

EnvironmentalVariables

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.
2. `env_variable`: Qualitative description of the environmental variable.
3. `vr_expr_name`: The name of the variable as it appears in either the `IpmKernels$formula`, `VitalRateExpr$formula`, or `EnvironmentalVariables$env_function`.
4. `env_range`: Three possibilities:

- A. Two numbers separated by a semicolon (";") denoting the minimum and maximum values that the environmental variable can take on. Use this when `env_function` is `"sample"`.
- B. A single number corresponding to the value that the parameter can take. Use this when the `env_function` is `NULL`.
- C. `NULL`. Use this when `env_function` is something other than `sample` or `NULL`.

5. `env_function`: Either the name of a function, or a mathematical expression to compute some value as a function of the parameters listed in this table or in ParameterValues. These will usually be either simple arithmetic (e.g. `"SE_rain * sqrt(n_env)"`), or a function that samples randomly from some distribution. This table also makes use of the Probability Distribution Dictionary, but substitutes `rdist` instead of `ddist`. There are couple additional functions that may appear:
   - `c`: used to generate vectors of parameters. This is most commonly used to create a vector of means to pass to a multivariate normal distribution.
   - `sig_mat`: used to generate a variance-covariance matrix to pass to a multivariate normal distribution. It takes a set of numbers and converts it to matrix in **ROW MAJOR** order (i.e. `matrix(..., byrow = TRUE)`).
6. `model_type`: Should be either "Evaluated", "Parameter", or "Substituted". Anything that contains a probability distribution should be "Substituted". Anything that is not probability distribution or raw parameter value should be "Evaluated" (i.e. `env_range` is `NULL` and `env_function` is something other than a probability distribution). Anything that is a parameter value should be "Parameter" (i.e. `env_range` is a single number, and `env_function` is `NULL`).

HierarchTable

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.
2. `env_variable`: a qualitative explanation of the suffix.
3. `vr_expr_name`: the suffix that is used to abbreviate the `env_variable` in the `IpmKernels` formula/kernel_id, `VitalRateExpr` formula/kernel_id.
4. `range`: An expression denoting the different levels that the each suffix can take on. For example, `2008:2011` to denote sampling years, or `c("GNone", "GLow", "GMedium", "GHigh")` to denote grazing levels.
5. `kernel_id`: the sub-kernel(s) that are modified by the suffix.
6. `drop_levels`: This is used to denote which levels in a continuous sequence do not actually appear in the model. `ipmr` assumes the suffixes get fully crossed, and so we need to indicate that some levels are missing. For example, say a study sampled multiple sites (`site = c("A", "B", "C")`) in multiple years (`yr = 2010:2014`). However, site A didn't get sampled in 2012 for some reason. We would add `c("A_2012")` to the `drop_levels` column to make sure `ipmr` doesn't try to find parameters/sub-kernels for that level when rebuilding the model.

UncertaintyTable

This table is not yet active, so skip for now.

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.

TestTargets

This table is not user-facing, but we keep it to validate the models numerically. Basically, we want to make sure that the model, as we've entered it, can reproduce some target metric when re-constructed by `ipmr`. The target is usually λ, because it's quick and easy to compute, and it's a single number (as opposed to, say, the right eigenvector or λ_s).

1. `ipm_id`: The 6 digit alphanumeric `ipm_id` from the Metadata table.
2. `target_name`: Usually `"lambda"` or `"lambda_suffixValue"` (if working with a grouped model). If you find that λ values are hard to come by for some publications, but other values may work, let me know and we will figure out a syntax for supporting those.
3. `target_value`: The numerical value of the target.
4. `precision`: The number of digits that the `target_value` is reported to. Used to make sure floating point/rounding error doesn't cause us to exclude working models from database builds.

Summary

You've made it this far! Good work! There are a number of potential pitfalls one may encounter when digitizing. These are described below.

Common issues

There are a number of issues that can (and probably will) arise when digitizing papers. Many authors do not include all the information needed to fully rebuild their models. Wrangling with $LaTeX$ when creating pretty functional forms can introduce accidental typos which do not reflect the code that was used to actually build and analyze the model. Below, there are some guideline for how to handle these situations.

Missing information

The vast majority of papers published do not contain all the information we need to re-build these IPMs. Therefore, we also contact authors to request the missing details. The most common ones are things like the numerical integration rule, the number of meshpoints, and the upper/lower bounds for the state variables used. Functional forms for some vital rates are also pretty common. We have a template for requesting information available in here (at some point, I'll convert that to a function that automatically generates email text given an `ipm_id` from a `pdb_raw` object. For now, you'll need to edit it manually).

Exceptions to our digitization rules

There are only a few times when we might want to enter a model differently from how it appears in the publication. The main way is when there is a typo in a functional form in the manuscript/appendix that doesn't match what is happening in the code the authors provide. For example, consider a survival model with a logistic regression of `survival ~ size`. The correct form would be:

$$s_z = \frac{1}{1 + exp(-(\alpha_s + \beta_s * z))}$$

However, the paper may contain the following form (notice the missing $-$ in the denominator):

$$s_z = \frac{1}{1 + exp(\alpha_s + \beta_s * z)}$$

In this case, it is appropriate to contact the author to double check what the exact functional form used in the model is. If this turns out to be a typo, then we would update the functional form in the database to reflect the code used, rather than the text used in the manuscript.