# Details on the data analyses and implementation of the IMP models:

In this notebook, we provide a detailed description of our modeling framework and the running of the code:

## Part I: Overview of the Integral Projection Models (IPM)

```
1. Running and construction of the Bayesian model and extracting the posterior
2. Constructing the IPMs
3. Vital functions and validation
4. Fitness calculations
5. Life Table Response Experiment (LTRE) Analyses
```

## Part II: Analyses for the manuscript

```
1. Running the full IPMS and LTREs
2. Plots and interpretation
```

# Part I: Overview of the Integral Projection Models (IPM)

# 1. Running and construction of the Bayesian model and extracting the posterior

## Estimation of the demographic rates

We estimated the demographic rates used for the vital rate functions of the IPMs using bayesian methods via Stan as follow:

## Survival

$$S_i \approx Binomial(1, p_i)$$

$$p_i = \alpha_{S(z)} + \beta_{S(z),NK} NK_i + \beta_{S(z),z}(z_i - \bar{z}) +$$

$$\beta_{S(z),zNK} NK_i(z_i - \bar{z}) + \beta_{S(z),area} Area_i + \beta_{S(z),canopy} + Canopy_i + v_{S(z),[stream,i]}$$

Priors:

$$\sigma_{stream} \approx Cauchy(0, 1)$$

$$v_{S(z)} \approx Normal(0, \sigma_{stream})$$

$$\beta_{S(z),NK} \approx Normal(0, 1)$$

$$\beta_{S(z),z} \approx Normal(0, 1)$$

$$\beta_{S(z),zNK} \approx Normal(0, 1)$$

$$\beta_{S(z),area} \approx Normal(0, 1)$$

$$\beta_{S(z),canopy} \approx Normal(0, 1)$$

$$\alpha_{S(z)} \approx Normal(2.6, 3)$$

## Somathic growth

$$z'_i \approx Normal(\mu_i, \sigma)$$

$$\mu_i = \alpha_{G(z',z)} + \beta_{G(z',z),NK} NK_i + \beta_{G(z',z),z}(z_i - \bar{z}) +$$

$$\beta_{G(z',z),zNK} NK_i(z_i - \bar{z}) + \beta_{G(z',z),area} Area_i + \beta_{G(z',z),canopy} + Canopy_i + v_{G(z',z),[stream,i]}$$

Priors:

$$\sigma_{G(z',z)} \approx Cauchy(0, 2)$$

$$v_{G(z',z)} \approx Normal(0, \sigma_{stream})$$

$$\beta_{G(z',z),NK} \approx Normal(0,1)$$

$$\beta_{G(z',z),z} \approx Normal(0,1)$$

$$\beta_{G(z',z),zNK} \approx Normal(0,1)$$

$$\beta_{G(z',z),area} \approx Normal(0,1)$$

$$\beta_{G(z',z),canopy} \approx Normal(0,1)$$

$$\alpha_{G(z',z)} \approx Normal(18,10)$$

## Fecundity

$$R_i \approx Poisson(\lambda_i)$$

$$\lambda_i = \alpha_{G(z',z)} + \beta_{G(z',z),NK} NK_i + \beta_{G(z',z),z}(z_i - \bar{z}) +$$

$$\beta_{R(z),zNK} NK_i (z_i - \bar{z}) + \beta_{R(z),area} Area_i + \beta_{R(z),canopy} + Canopy_i + v_{R(z),[stream,i]}$$

Priors:

$$\sigma_{R(z)} \approx Cauchy(0,2)$$

$$v_{R(z)} \approx Normal(0, \sigma_{stream})$$

$$\beta_{R(z),NK} \approx Normal(0,1)$$

$$\beta_{R(z),z} \approx Normal(0,1)$$

$$\beta_{R(z),zNK} \approx Normal(0,1)$$

$$\beta_{R(z),area} \approx Normal(0,1)$$

$$\beta_{R(z),canopy} \approx Normal(0,1)$$

$$\alpha_{R(z)} \approx Normal(18,10)$$

To combine demographic rates and to control for any covariation among demographic rates, we use the Bayesian Hierarchical Centering (BHC) method following Elderd and Miller (2016) by simultaneously running all vital rate models for both species and estimating a global random stream drainage effect parameter for each species at the intercept ($\sigma_{stream}$). We use the same model structure for both species, but only the priors for survival were different between guppies and killifish.

## Model executions

In the next cell, we run the Bayesian model via R, using the rstan and rethinking packages as interface.

```
# Load the necessary modules
begin
    using Distributions
    using CSV
    using DataFrames
    using LinearAlgebra
    using StatsPlots
    using JLD2
    using RCall
end
```

### If you wish to run the model on your computer run the cell below:

```
# R"""
#source("R/MainScript.R") # Runs the model via R
# """
```

## Alternatively, you can load an already run model and start using the posteriors. Next lines...

```
R"""
library("rethinking")
mod <- readRDS("Model_KG.RDS")
sum = as.data.frame(precis(mod, digits = 5, prob = .95, depth = 2))
names = rownames(sum)
post <- as.data.frame(extract(mod))
""";
```

```
RCall.jl: Loading required package: rstan
Loading required package: StanHeaders
Loading required package: ggplot2
rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)
For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
Loading required package: parallel
rethinking (Version 2.13)

Attaching package: ‘rethinking’

The following object is masked from ‘package:stats’:

    rstudent
```

| | Pars | mean | sd | 2_5% | 97_5% | n_eff |
|---|---|---|---|---|---|---|
| 1 | "Intercept_survG" | 0.545292 | 0.518297 | -0.511814 | 1.57515 | 4286.31 |
| 2 | "b_NK_survG" | -0.0727618 | 0.287809 | -0.636109 | 0.497028 | 7439.62 |
| 3 | "b_z_survG" | -0.00713631 | 0.0383379 | -0.0837725 | 0.0671609 | 9357.49 |
| 4 | "b_zNK_survG" | 0.0786045 | 0.0568138 | -0.0330623 | 0.190775 | 8552.36 |
| 5 | "b_area_survG" | 0.0591517 | 0.221399 | -0.384254 | 0.496422 | 6846.66 |
| 6 | "b_canopy_survG" | -0.507205 | 0.231561 | -0.960817 | -0.0538452 | 7566.06 |
| 7 | "v_Intercept_survG[1]" | -0.421199 | 0.540052 | -1.47456 | 0.652695 | 4335.06 |
| 8 | "v_Intercept_survG[2]" | 0.119572 | 0.563038 | -0.968271 | 1.24971 | 5163.35 |
| 9 | "v_Intercept_survG[3]" | 0.562973 | 0.520348 | -0.426189 | 1.58824 | 4622.91 |
| 10 | "v_Intercept_survG[4]" | -0.391 | 0.620631 | -1.67252 | 0.813455 | 5353.11 |
| | more | | | | | |
| 64 | "b_canopy_recrK" | 1.23238 | 1.35982 | -1.32222 | 3.96687 | 4074.93 |

```julia
# get the summary table, the names of the rows, and the posterios
begin
    @rget sum names post;
    sumTab = DataFrame(sum)
    sumTab.Pars = names
    sumTab[:, [7,1,2,3,4,5,6] ]
end
```

## 2. Constructing the IPMs

### Integral Projection model for guppies

The model for guppies is a single-sex (female only) integral projection model (IPM) written:

$$n(z', t+1) = \int_L^U K(z', z) n(z, t) dz,$$

The model is a discrete-time population projection model with the population structured by body size ($z$). The function $n(z', t+1)$ describes the density of individuals in the population at the beginning of the experiment, such that $\int_4^{45} n(z, t)dz$ is the number of individuals between standard length 4 mm and 45mm. $K(z', z)$ is the complete IPM kernel, which is composed of the survival and fecundity kernels:

$$n(z', t+1) = \int_L^U [P(z', z) + F(z'z)] n(z, t) dz$$

The kernel, $P(z', z)h$, is the probability that an individual of size $z$ survives between $t$ and $t+1$ and grows to size z'. $F(z', z)h$ is the number of offspring of size $z'$ produced by an individual of size $z$ survives between $t$ and $t+1$. Where $h$ is the interval between size classes and if $h \to 0$ the approximation becomes exact. Hence, the survival kernel, $P(z', z)$ describes the survival and possible growth or shrinkage of individuals of size $z$ within the census interval (here, 25 days). This kernel is composed of two vital functions: $P(z', z)$ = $S(z)G(z', z)$. Where $S(z)$ is a continuous function describing the probability of an individual with body length $z$ at the beginning of the experiment surviving to the end experiment. $G(z', z)$, on the other hand, is the conditional probability density function describing transitions from length $z$ at the beginning of the experiment to length $z'$ at the end of the experiment.

The fecundity or reproduction kernel, $F(z', z) = C(z', z)R(z)S(z)$ is composed of the survival function $S(z)$ and two extra vital functions. $C(z', z)$ is the conditional probability density function describing the distribution of offspring of body length $z'$ at the end of the experiment produced by parents with body length $z$ at the beginning of the experiment. $R(z)$ is a continuous function describing the mean number of offspring produced by an individual with body length $z$ at reproduction. The model assumes that all reproducing individuals do so at the end of the interval, immediately preceding the next census. The IMP model can therefore be written as:

$$n(z', t+1) = \int_L^U [G(z', z)S(z) + C(z', z)R(z)S(z)] n(z, t) dz$$

## 3. Vital functions and validations

## Survival function, S(z):

$$S(z) = \frac{1}{1 + 10^{-[\alpha + \beta_z (z - \bar{z})]}}$$

```
s_z (generic function with 1 method)
```

```
function s_z(df::AbstractDataFrame, z::AbstractVector,
    size_cen::AbstractFloat, row::Integer)
    α= df.α_surv[row]
    β= df.βz_surv[row]
    linear_p = α .+ β * (z .- size_cen)  # linear predictor
    p = 1 ./(1 .+ exp.(-linear_p))
    p = diagm(p)
    return(p)
end
```

## Growth function, G(z',z):

Probability of growint to $z'$ given $z$

$$G(z', z) = \frac{1}{\sqrt[2]{2\pi\sigma^2}} e^{\frac{-[z' - (\alpha + \beta_Z (z - \bar{z}))]^2}{2\sigma^2}}$$

```
g_z1z (generic function with 1 method)
```

```
function g_z1z(df::AbstractDataFrame, z1::AbstractVector, z::AbstractVector,
    size_cen::AbstractFloat, row::Integer)
    α= df.α_grow[row]
    β= df.βz_grow[row]
    σ= df.σ_grow[row]
    p = zeros(size(z)[1],size(z)[1])
    μ = α .+ β * (z .- size_cen )
    for i in 1:nBigMatrix
        p[:,i] = pdf.(Normal(μ[i], σ), z1).*h
    end
    return(p)
end
```

## Recruitment function, R(z):

Number of offspring for an individual of size $z$, divided by two to account for females

$$R(z) = e^{\alpha + \beta_z (z - \bar{z})} \frac{1}{2}$$

```
R_z (generic function with 1 method)
```

```
function R_z(df::AbstractDataFrame, z::AbstractVector, size_cen::AbstractFloat,
row::Integer)
    α= df.α_fec[row]
    β= df.βz_fec[row]
    linear_p = α .+ β * (z .- size_cen)      # linear predictor
    p = exp.(linear_p)*(1/2)
    p = diagm(p)
    return(p)
end
```

## Offspring size, C(z',z):

Probability of having offspring with mean size $z'$ given the size of the mother $z$

$$C(z', z) = \frac{1}{\sqrt[2]{2\pi\sigma^2}} e^{\frac{-[z' - (\alpha + \beta_z(z - \bar{z}))]^2}{2\sigma^2}}$$

Because we have no good estimation of the parameters of this function, we assumed that size has no effect on the size of offspring and when born they are approximately $7.0mm$ $(\pm 0.4)$

```
c_z1z (generic function with 1 method)
```

```
function c_z1z(df::AbstractDataFrame, z1::AbstractVector, z::AbstractVector,
        size_cen::AbstractFloat, row::Integer)
    #α= df.rcz_int[row]
    #β= df.rcz_z[row]
    σ= 0.4 #pars.rcz_sd[row]
    #   βa= df.β_area_rcz[row]
    #   βc= df.β_canopy_rcz[row]
    p_den_grow = zeros(nBigMatrix,nBigMatrix)
    μ = 7 .+ 0*(z .- size_cen)
    for i in 1:nBigMatrix
        p_df = pdf.(Normal(μ[i], σ), z1)*h
        for j in 1:nBigMatrix
        p_den_grow[j,i] = p_df[j]
        end
    end
    return(p_den_grow)
end
```

# Kernel calculations

```
P_z1z (generic function with 1 method)
```

```
## Survival kernel
function P_z1z(df::AbstractDataFrame, z1::AbstractVector, z::AbstractVector,
    size_cen::AbstractFloat, row::Integer)
    out = g_z1z(df, z1, z, size_cen, row) * s_z(df, z, size_cen, row)
    return(out)

end
```

```
F_z1z (generic function with 1 method)
```

```
# Reproduction kernel
function F_z1z(df::AbstractDataFrame, z1::AbstractVector, z::AbstractVector,
    size_cen::AbstractFloat, row::Integer)
    out = c_z1z(df, z1, z, size_cen, row) * R_z(df, z, size_cen, row) * s_z(df, z,
    size_cen, row)
    return(out)
end

```

## 4. Fitness calculations

### IPM construction

The continuous size IPM's are converted to matrix models prior to analysis using the midpoint rule of calculus (Easterling et al. 2000). The guppy model is a standard pre-breeding census model with the matrix model approximation:

$$n(t+1) = An(t),$$

where $A$ and $n(t)$ are structured by body size ($z$). The minimum (L) and maximum (U) size ranges for guppies were 2mm and 45mm with 100 mesh points. The projection matrix, A, is made up of matrices that describe the individual demographic rates:

$$A = P + CRS$$

```
begin
    U= 45
    L=2
    nBigMatrix = 100
    m= nBigMatrix
    h = (U - L)/m
    z1 = zeros(nBigMatrix)
    z1 =  L .+ (collect(1:m) .- 0.5) * h
    z = z1
end;
```

```
mk_K (generic function with 1 method)
```

```
# IPM kernel. Put together the kernels
function mk_K(df::AbstractDataFrame, z1::AbstractVector, z::AbstractVector,
    size_cen::AbstractFloat, row::Integer)
    F = F_z1z(df, z1, z, size_cen, row)
    P = P_z1z(df, z1, z, size_cen, row)
    K = P + F
    out = Dict("K"=> K, "meshpts" => z, "P" => P, "F"=> F)
    return(out)
end
```

To convert the functions into matrices, we calculate the matrix for each vital rate for both NK and KG guppies using the posterior parameters. Therefore, we got the posterior for each treatment as separate dataframes:

## Getting the parameter values

We have 6000 posterior samples from the Bayesian model to run the IPMs 6000 times. To run the guppy IPM, for now, we just use one of those 6000 posteriors:

```
begin
    pars_KG_G = select(post, # Posteriors for the Killifish-Guppy fish
        :Intercept_survG => :"α_surv",
        :b_z_survG => :"βz_surv",
        :Intercept_growG => :"α_grow",
        :b_z_growG => :"βz_grow",
        :sigma_growG => :σ_grow,
        :Intercept_recrG => :"α_fec",
        :b_z_recrG => :"βz_fec"
    )
    pars_NK_G = DataFrame(  # posteriors for the No-Killifish fish
        α_surv = post.Intercept_survG .+ post.b_NK_survG,
        βz_surv = post.b_z_survG .+ post.b_zNK_survG,
        α_grow = post.Intercept_growG .+ post.b_NK_growG,
        βz_grow= post.b_z_growG .+ post.b_zNK_growG ,
        σ_grow = post.sigma_growG,
        α_fec = post.Intercept_recrG .+ post.b_NK_recrG,
        βz_fec = post.b_z_recrG + post.b_zNK_recrG
    )

end;
```

# IPM for KG and NK guppies

In the next lines, we will build an IPM model for guppies

```
# Here, we choose the first posterior values to make and IMP and test our code
# we center our analyses at 18.0 mm
begin
        size_cen = 18.0
        row = 1
        IPM_KG = mk_K(pars_KG_G, z1, z, size_cen, row)
        IPM_NK = mk_K(pars_NK_G, z1, z, size_cen, row)
end;
```

```
100×100 Matrix{Float64}:
 1.20581e-9    2.00061e-10   3.04692e-11   …  2.48168e-31  2.56028e-31  2.64136e-31
 1.33327e-8    2.49782e-9    4.29554e-10      5.35423e-26  5.52379e-26  5.69873e-26
 1.24082e-7    2.62488e-8    5.09711e-9       3.63719e-21  3.75238e-21  3.87121e-21
 9.71955e-7    2.3217e-7     5.0907e-8        7.77953e-17  8.0259e-17   8.28007e-17
 6.40816e-6    1.72842e-6    4.27938e-7       5.23913e-13  5.40505e-13  5.57622e-13
 3.55607e-5    1.08304e-5    3.02789e-6    …  1.11092e-9   1.1461e-9    1.1824e-9
 0.00016613    5.71568e-5    1.80697e-5       7.41696e-7   7.65185e-7   7.89418e-7
 ⋮                                         ⋱
 1.02056e-238  1.54161e-234  2.13758e-230     1.761e-6     6.57398e-6   2.25274e-5
 1.03864e-244  1.77157e-240  2.77373e-236  …  2.34914e-7   9.90229e-7   3.83157e-6
 8.89695e-251  1.71353e-246  3.02939e-242     2.63759e-8   1.25543e-7   5.48519e-7
 6.41453e-257  1.395e-252    2.78481e-248     2.49261e-9   1.33967e-8   6.60929e-8
 3.89258e-263  9.55882e-259  2.15469e-254     1.98267e-10  1.20324e-9   6.70296e-9
 1.9882e-269   5.51294e-265  1.40321e-260     1.32738e-11  9.09607e-11  5.72173e-10
```

```
# see the kernel (K)
IPM_KG["K"]
```

After constructing the IMP, we can estimate some interesting values. Such as the asymptotic growth rate (λ), our measurement of fitness.

```
begin
    # calculate the population growth rate (λ)
    vv_KG = eigen(IPM_KG["K"])
    vv_NK = eigen(IPM_NK["K"])
    λ_KG = real(vv_KG.values[end])
    λ_NK = real(vv_NK.values[end])
end;
```

```
[1.08983, 0.929413]
```

```
# let's check our results
[λ_KG,λ_NK]
```

This suggest that the fitness of KG guppies is slightly higher than that of NK guppies.

## Life Table Response Experiment (LTRE) Analyses

We used LTRE analyses to decompose the fitness contrasts into effects caused by the individual demographic rates (Caswell 2001). The first-order approximation of the decomposition for each contrast for each demographic rate is:

$$\lambda_V \approx (V_i - V_j)\frac{d\lambda}{d^T\bar{V}}$$

where, $e^T$ is a vector of ones, $vecV_i$ and $vecV_j$ are the vector transformed demographic rates (e.g., S(z)) for treatments $i$ and $j$. $\frac{d\lambda}{dvec^T\bar{V}}$ is the sensitivity of absolute fitness (λ) to the vector transformed matrix. The sensitivity of guppy fitness to the demographic rates in the guppy model was calculated as:

$$\frac{d\lambda}{d^T\bar{V}} = \frac{d\lambda}{d^TA}\frac{dA}{d^TV}$$

where $\frac{d\lambda}{d^TA}$ is the vector transformed sensitivity of absolute fitness to the matrix elements of $A$. $\frac{dA}{d^TV}$ is the sensitivity of the elements of A to each demographic rate (V).

```
[-0.167642, -0.167642]
```

```julia
## LTRE
begin
    ## calculate average matrix
    K_avg = (IPM_NK["K"] + IPM_KG["K"])./2
    vv_avg = eigen(K_avg)

    # Normalize stable size distribution
    ω_avg = real(vv_avg.vectors[:, end])

    # Reproductive value
    a_avg = eigen(transpose(K_avg))
    v_avg = real(a_avg.vectors[:, end]) ./ Base.sum(real(a_avg.vectors[:, end]))
    v_avg = v_avg ./ dot(transpose(v_avg), ω_avg)

    ## Sensitivity matrix
    sens_avg = v_avg * ω_avg' # this equivalent to outer() in R
    ΔK = IPM_NK["K"] - IPM_KG["K"]
    λ_eff = ΔK .* sens_avg
    Δλ = Base.sum(λ_eff)
    [Base.sum(λ_eff),Δλ]
end
```

```
-0.16764172911365804
```

```
begin
    ## Make life-response table
    one_mat = ones(nBigMatrix, nBigMatrix)

    # Function differences
    Δ_grow = g_z1z(pars_NK_G, z1, z, size_cen, row) - g_z1z(pars_KG_G, z1, z,
size_cen, row)
    Δ_fec = one_mat*(R_z(pars_NK_G, z, size_cen, row) - R_z(pars_KG_G, z, size_cen,
row))
    Δ_rcz = (c_z1z(pars_NK_G, z1, z, size_cen, row) - c_z1z(pars_KG_G, z1, z,
size_cen, row))
    Δ_sur = one_mat*(s_z(pars_NK_G, z, size_cen, row) - s_z(pars_KG_G, z, size_cen,
row))

    # Function averages
    grow_avg = (g_z1z(pars_NK_G, z1, z, size_cen, row) + g_z1z(pars_KG_G, z1, z,
size_cen, row))/2
    fec_avg = (one_mat*(R_z(pars_NK_G, z, size_cen, row) + R_z(pars_KG_G, z,
size_cen, row)))/2
    rcz_avg = ((c_z1z(pars_NK_G, z1, z, size_cen, row) + c_z1z(pars_KG_G, z1, z,
size_cen, row)))/2
    sur_avg = (one_mat*(s_z(pars_NK_G, z, size_cen, row) + s_z(pars_KG_G, z,
size_cen, row)))/2

    # derivates
    δ_grow= sur_avg
    δ_fec=  rcz_avg .* sur_avg
    δ_sur = grow_avg .+   fec_avg .* rcz_avg
    δ_rcz = fec_avg .* sur_avg

    λ_grow = Δ_grow .* sens_avg .* δ_grow
    λ_fec = Δ_fec .* sens_avg .* δ_fec
    λ_rcz = Δ_rcz .* sens_avg .* δ_rcz
    λ_sur = Δ_sur .* sens_avg .* δ_sur

    # to put in a DataFrame
    sur_con = Base.sum(λ_sur)
    grow_con = Base.sum(λ_grow)
    fec_con = Base.sum(λ_fec)
    rcz_con = Base.sum(λ_rcz)
    sum_con = sur_con + grow_con + fec_con + rcz_con

end

```

## Integral Projection model for killifish

The model for killifish is also a single-sex (female only) IPM but is slightly different than the one for guppies. Because killifish lay eggs that hatch at two weeks, two broods are produced per time-step. This means that the overall projection kernel for a month is a composite of two, two-week projection kernels. Each two-week projection equation is:

$$n(t+1/2)_E = \int R(z)S(z)n(z,t)_A dz,$$

$$n(z,t+1/2)_A = \int C(z',z)vn(t)_E + \int G(z',z)S(z)n(z,t)_A dz,$$

the subscripts $E$ and $A$ denote which equation describes the production of eggs or the transition from eggs to adults or adults to adults. $v$ is the probability of egg survival and hatching, here set to 0.75 for all treatments as we do not have information about this transition. Additionally, because killifish have a larger size range than guppies we set the L and U size range between 2 mm and 110 mm. The composite functions for the entire 28 days are then:

$$n(t+1)_E = \int R(z)S(z)[C(z',z)vn(t)_E + \int G(z',z)S(z)n(z,t)_A dz]dz,$$

$$n(z,t+1)_A = \int C(z',z)v \int R(z)S(z)n(z,t)_A dz + \int G(z',z)S(z)[C(z',z)S(z)vn(t)_E + \int$$

```
md"""
### Integral Projection model for killifish
The model for killifish is also a single-sex (female only) IPM but is slightly
different than the one for guppies. Because killifish lay eggs that hatch at two
weeks, two broods are produced per time-step. This means that the overall
projection kernel for a month is a composite of two, two-week projection kernels.
Each two-week projection equation is:

$n(t+1/2)_E = \int R(z)S(z)n(z,t)_A dz,$

$n(z, t+1/2)_A = \int C(z',z)vn(t)_E + \int G(z',z)S(z)n(z,t)_A dz,$

the subscripts $E$ and $A$ denote which equation describes the production of eggs
or the transition from eggs to adults or adults to adults. $v$ is the probability
of egg survival and hatching, here set to 0.75 for all treatments as we do not have
information about this transition. Additionally, because killifish have a larger
size range than guppies we set the L and U size range between 2 mm and 110 mm.  The
composite functions for the entire 28 days are then:

$n(t+1)_E = \int R(z)S(z)[C(z',z) v n(t)_E + \int G(z',z)S(z)n(z,t)_A dz ]dz,$

$n(z, t+1)_A = \int C(z',z) v  \int R(z)S(z)n(z,t)_A dz + \int G(z',z)S(z)
[C(z',z)S(z) v n(t)_E + \int G(z',z)S(z)n(z,t)_A dz]dz,$
"""
```

```
[2.54, 3.62, 4.7, 5.78, 6.86, 7.94, 9.02, 10.1, 11.18, 12.26, 13.34, 14.42, 15.5, 16.58,
```

```
begin
  # nBigMatrix = 100
  Uk= 110.0
  Lk= 2.0
  mk = nBigMatrix
  hk = (Uk - Lk)/mk

  z1k =  Lk .+ (collect(1:mk) .- 0.5) .* hk
  z1k = round.(z1k, digits = 6)
  zk = z1k
  meshptsk = z1k
end
```

```
begin
    pars_KG_K = select(post, # Posteriors for the Killifish-Guppy fish
        :Intercept_survK => :"α_surv",
        :b_z_survK => :"βz_surv",
        :Intercept_growK => :"α_grow",
        :b_z_growK => :"βz_grow",
        :sigma_growK => :σ_grow,
        :Intercept_recrK => :"α_fec",
        :b_z_recrK => :"βz_fec"
    )
    pars_NG_K = DataFrame(  # posteriors for the No-Killifish fish
        α_surv = post.Intercept_survK .+ post.b_NG_survK,
        βz_surv = post.b_z_survK .+ post.b_zNG_survK,
        α_grow = post.Intercept_growK .+ post.b_NG_growK,
        βz_grow= post.b_z_growK .+ post.b_zNG_growK ,
        σ_grow = post.sigma_growK,
        α_fec = post.Intercept_recrK .+ post.b_NG_recrK,
        βz_fec = post.b_z_recrK + post.b_zNG_recrK
    )

end;
```

To acomodate the egg stage in the killfish matrices, we added an extra column and extra row. See the structure of each matrix.

```
g_z1zK (generic function with 1 method)
```

```
function g_z1zK(df::AbstractDataFrame, z1::AbstractVector, z::AbstractVector,
size_cen::AbstractFloat, row::Integer)
    α= df.α_grow[row]
    β= df.βz_grow[row]
    σ= df.σ_grow[row]
    p_den_grow = zeros(size(z)[1],size(z)[1])
    μ = ((α .+ β .* (z .- size_cen ) .- (z))./2 .+ z) # average growth in two weeks
    for i in 1:length(z)
        p_den_grow[:,i] = pdf.(Normal(μ[i], σ), z1).*h
    end
    matex = zeros(length(z)+1, length(z)+1)
    matex[2:end,2:end] = p_den_grow
    return(matex)
  end
```

```
101×101 Matrix{Float64}:
 0.0  0.0        0.0         0.0         …  0.0        0.0        0.0
 0.0  0.0118776  0.00417295  0.00116668     0.0        0.0        0.0
 0.0  0.0277924  0.0124014   0.00440359     0.0        0.0        0.0
 0.0  0.0506361  0.0286967   0.0129419      0.0        0.0        0.0
 0.0  0.071834   0.0517047   0.0296157      0.0        0.0        0.0
 0.0  0.079348   0.0725377   0.0527697   …  0.0        0.0        0.0
 0.0  0.0682461  0.079238    0.0732119      0.0        0.0        0.0
 ⋮                                       ⋱                        ⋮
 0.0  0.0        0.0         0.0         …  0.0345386  0.0166254  0.0063684
 0.0  0.0        0.0         0.0            0.0581276  0.0355367  0.0172888
 0.0  0.0        0.0         0.0            0.0761721  0.059145   0.0365455
 0.0  0.0        0.0         0.0            0.0777222  0.0766471  0.0601505
 0.0  0.0        0.0         0.0            0.0617491  0.0773408  0.0770869
 0.0  0.0        0.0         0.0         …  0.038199   0.0607656  0.0769232
```

```
g_z1zK(pars_KG_K, z1k, zk, 18.0, 1)
```

```
s_zK (generic function with 1 method)
```

```
## Surival function
function s_zK(df::AbstractDataFrame, z::AbstractVector,
    size_cen::AbstractFloat, row::Integer)
    α= df.α_surv[row]
    β= df.βz_surv[row]
    linear_p = α .+ β * (z .- size_cen)      # linear predictor
    p = 1 ./(1 .+ exp.(-linear_p))
    p = diagm(sqrt.(p))
    matex = zeros(length(z)+1, length(z)+1)
    matex[2:end,2:end] = p
    return(matex)
end
```

```
101×101 Matrix{Float64}:
 0.0  0.0       0.0       0.0       …  0.0       0.0       0.0       0.0
 0.0  0.964314  0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.964862  0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.965403     0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0       …  0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 ⋮                                  ⋱                                ⋮
 0.0  0.0       0.0       0.0       …  0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.992244  0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.992368  0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.992491  0.0
 0.0  0.0       0.0       0.0       …  0.0       0.0       0.0       0.992611
```

```
s_zK(pars_KG_K, zk, 18.0, 1)
```

```
R_zK (generic function with 1 method)
```

```
## Recruitment function (N.B - from birth in spring to first summer), logistic
regression
function R_zK(df::AbstractDataFrame, z::AbstractVector, size_cen::AbstractFloat,
row::Integer)
    α= df.α_fec[row]
    β= df.βz_fec[row]
    linear_p = α .+ β * (z .- size_cen)   # linear predictor  # linear predictor
    p = exp.(linear_p).*(1/2)
    matex = zeros(length(z)+1, length(z)+1)
    matex[1,2:end] = p
    return(matex)
end
```

```
101×101 Matrix{Float64}:
 0.0  0.208571  0.209137  0.209705  …  0.270527  0.271261  0.271996  0.272734
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0       …  0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 ⋮                                  ⋱                                ⋮
 0.0  0.0       0.0       0.0       …  0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0          0.0       0.0       0.0       0.0
 0.0  0.0       0.0       0.0       …  0.0       0.0       0.0       0.0
```

```julia
R_zK(pars_KG_K, zk, 18.0, 1)
```

```
c_z1zK (generic function with 1 method)
```

```julia
## Recruit size function
function c_z1zK(df::AbstractDataFrame, z1::AbstractVector, z::AbstractVector,
    size_cen::AbstractFloat, row::Integer)

    σ= 0.6 #pars.rcz_sd[row]
    p_den_grow = zeros(length(z)+1,length(z)+1)
    μ = 4.4 .+ 0#*(z .- size_cen)
    p_den_grow[2:end, 1] =  pdf.(Normal(μ, σ), z).*h
    return(p_den_grow)
end
```

```
101×101 Matrix{Float64}:
 0.0          0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.000377119  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0039664    0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0249607    0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0939852    0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.21174      0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.285423     0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                                ⋮         ⋱            ⋮                        ⋮
 0.0          0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0          0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0          0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0          0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0          0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0          0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```julia
c_z1zK(pars_KG_K,z, z, 18.0, 1)
```

```
mk_KK (generic function with 1 method)
```

```julia
## Functions to build IPM kernels P, F, and K
function mk_KK(df::AbstractDataFrame, z1::AbstractVector, z::AbstractVector,
size_cen::AbstractFloat, row::Integer, V::AbstractFloat)
    F = R_zK(df, z, size_cen, row) * s_zK(df, z, size_cen, row)
    P = g_z1zK(df, z, z, size_cen, row) * s_zK(df, z, size_cen, row)
    A = (P + F) + (V .* c_z1zK(df, z1, z, size_cen, row))
    K = A * A
    out = Dict("K"=> K, "meshpts" => z, "P" => P, "F"=> F, "A" => A)
    return(out)
end
```

## IPM for KG and NK killifish

In the next lines, we will build an IPM model for killifish

```
[0.938159, 1.14624, -0.208081]
```

```
begin
# Make projection kernels
    IPM_GR = mk_KK(pars_KG_K, z1, z, size_cen, row, 0.7)
    IPM_NG = mk_KK(pars_NG_K, z1, z, size_cen, row, 0.7)
    # calculate the population growth rate (λ)
    vv_GR = eigen(IPM_GR["K"])
    vv_NG = eigen(IPM_NG["K"])
    λ_GR = real(vv_GR.values[end])
    λ_NG = real(vv_NG.values[end])
    [λ_NG, λ_GR, λ_NG-λ_GR] # fitness differences
end
```

## Life Table Response Experiment (LTRE) Analyses for killifish

The first-order approximation of the decomposition for each contrast for each demographic rate is:

$$\lambda_V \approx e^T (vecV_i - vecV_j) \frac{d\lambda}{dvec^T \bar{V}}$$

where, $e^T$ is a vector of ones and $vec$ represents vector transformed matrices (e.g., S(z)). The sensitivity for killifish requires an extra derivate to relate how changes in $A$ alters $\bar{A}$. This derivate was calculated:

$$\frac{dvec\bar{A}}{dvec^T A} = (I \otimes A) + (\bar{A}^T \otimes I)$$

where $I$ is an identity matrix and $otimes$ means to take the Kronecker product. This derivate is inserted into the sensitivity equation:

$$\frac{d\lambda}{dvec^T \bar{V}} = \frac{d\lambda}{dvec^T \bar{A}} \frac{dvec\bar{A}}{dvec^T A} \frac{dvecA}{dvec^T V}$$

```
# ## Killifish LTRE
# begin
#   ## calculate average matrix
#   K_avg = (IPM_NG["K"] + IPM_GR["K"])./2
#   vv_avg = eigen(K_avg)

#   # Normalize stable size distribution
#   ω_GR = real(vv_GR.vectors[:, end]) ./ Base.sum(real(vv_GR.vectors[:, end]))
#   ω_NG = real(vv_NG.vectors[:, end]) ./  Base.sum(real(vv_NG.vectors[:, end]))
#   ω_avg = real(vv_avg.vectors[:, end]) ./ Base.sum(real(vv_avg.vectors[:, end]))

#   # Reproductive value
#   a_avg = eigen(K_avg')
#   v_avg = real(a_avg.vectors[:, end])
#   v_avg = v_avg ./ dot(transpose(v_avg), ω_avg)

#   ## Sensitivity matrix
#   sens_avg = v_avg * ω_avg'  # this equivalent to outer() in R
#   ΔK = IPM_NG["K"] .- IPM_GR["K"]
#   λ_eff = ΔK .* sens_avg
#   Δλ = sum(λ_eff)

#   ## Make life-response table

#   # Function differences
#   Δ_grow = g_z1zK(pars_NG_K, z1, z, size_cen, row) .- g_z1zK(pars_GR_K, z1, z, size_cen, row)
#   Δ_fec = (pr_zK(pars_NG_K, z, size_cen, row) .- pr_zK(pars_GR_K, z, size_cen, row))
#       Δ_rcz = (c_z1zK(pars_NG_K, z1, z, size_cen, row) .- c_z1zK(pars_GR_K, z1, z, size_cen, row))
#   Δ_sur = (s_zK(pars_NG_K, z, size_cen, row) .- s_zK(pars_GR_K, z, size_cen, row))

#   # Function averages
#   grow_avg = (g_z1zK(pars_NG_K, z1, z, size_cen, row) + g_z1zK(pars_GR_K, z1, z, size_cen, row))/2

#   fec_avg = ((pr_zK(pars_NG_K, z, size_cen, row) + pr_zK(pars_GR_K, z, size_cen, row)))/2

#   rcz_avg = ((c_z1zK(pars_NG_K, z1, z, size_cen, row) + c_z1zK(pars_GR_K, z1, z, size_cen, row)))/2

#   sur_avg = ((s_zK(pars_NG_K, z, size_cen, row) + s_zK(pars_GR_K, z, size_cen, row)))/2

#   # derivates
#   I_mat = diagm(ones(nBigMatrix+1))
#   δ_grow = kron(transpose(sur_avg), I_mat)
#   δ_sur  = kron(transpose(I_mat),grow_avg) +  kron(transpose(I_mat),(fec_avg * rcz_avg))
#   δ_fec  = kron(transpose(sur_avg), rcz_avg)
#   δ_rcz  = kron(transpose(fec_avg * sur_avg), I_mat)

#   δA = kron(I_mat, K_avg) + kron(transpose(K_avg), I_mat)

#   λ_grow = Δ_grow .* reshape((([transpose(sens_avg[:]) ; ] * δA )* δ_grow), (nBigMatrix+1,nBigMatrix+1))
#   λ_fec = Δ_fec .*  reshape((([transpose(sens_avg[:]) ; ] * δA )* δ_fec),
(nBigMatrix+1,nBigMatrix+1))

#   λ_rcz = Δ_rcz .*  reshape((([transpose(sens_avg[:]) ; ] * δA )* δ_rcz),
(nBigMatrix+1,nBigMatrix+1))

#   λ_sur = Δ_sur .* reshape((([transpose(sens_avg[:]) ; ] * δA )* δ_sur),
(nBigMatrix+1,nBigMatrix+1))



#   # to put in a DataFrame
#   sur_con = Base.sum(λ_sur)
#   grow_con = Base.sum(λ_grow)
#   fec_con = Base.sum(λ_fec)
#   rcz_con = Base.sum(λ_rcz)
#   sum_con = sur_con + grow_con + fec_con + rcz_con

# end
```

# Putting things together

To use the IPMs and make them meaningful, we have to run them with each iteration from the posterior samples. To facilitate that process you only have to run the *".../Julia/MainScript.jl"* file. This script will call other scripts where the IMP for guppies and killifish have been compiled into functions. Thes analyses will basically run the analyses 6000 times. The *"MainScript.jl"* file will also produce the figures and estimate all the necessary stats. Writing these IPMs in the Julia language has been easy and straightforward, but the major advantage is its speed. If you prefer to run the analyses in "R" you can also do so, but running the script "R/MainScript.R" the results are basically the same, there are some small differences but it is more because of the differences in the decimal digits use by the two programing lenguages.

Running the killifish IPM in R took around 3 days on my computer, during that time I translated the code to Julia. Then, in Julia, the same code run in 4 hours !!!