

Appendix A: Detailed visualization of fish heatmaps over eight body regions of fish over time

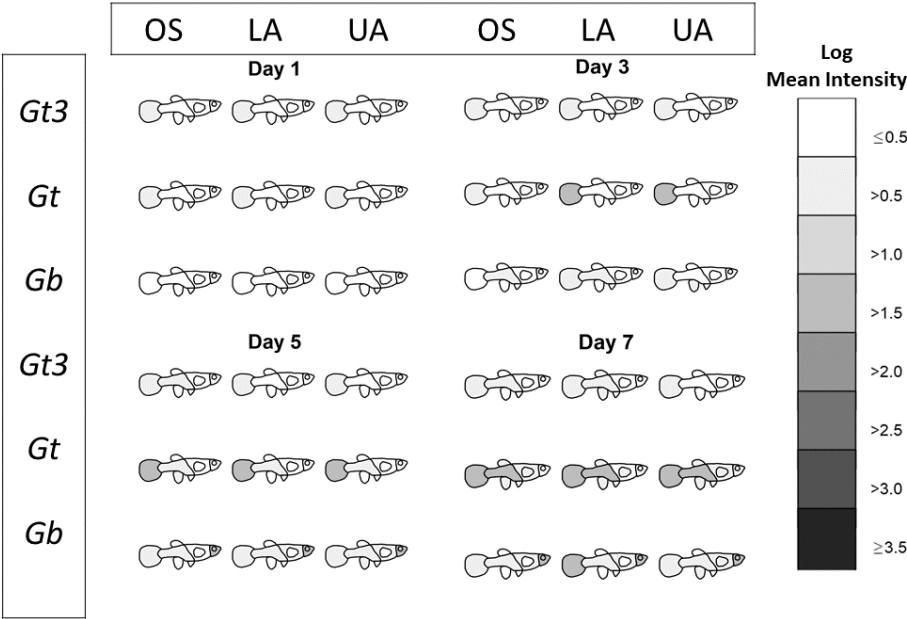


Figure A.1: Detailed visualization of fish heatmaps over eight body regions of fish across parasite strains and fish stocks from day 1 to 7 [1, in press].

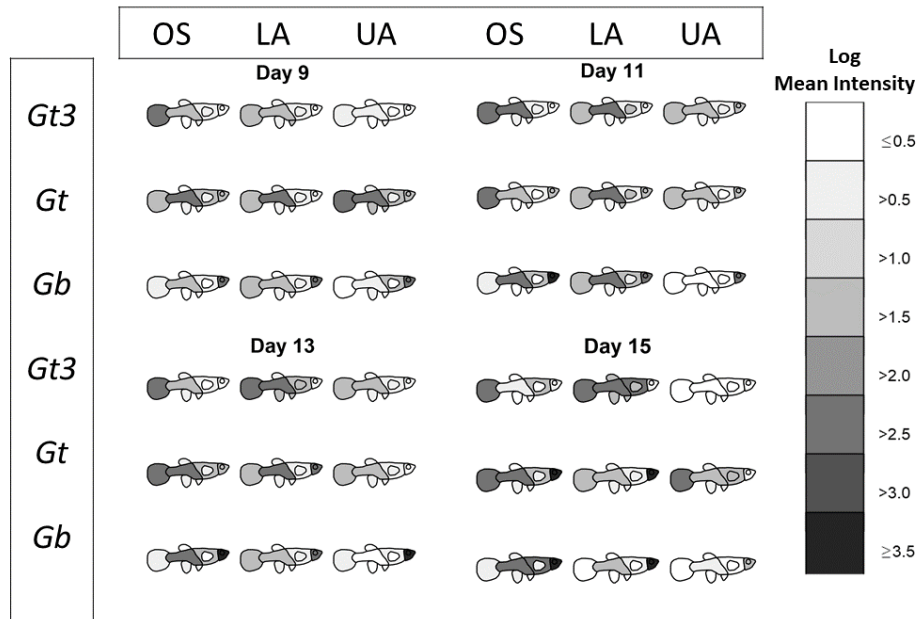


Figure A.2: Detailed visualization of fish heatmaps over eight body regions of fish across parasite strains and fish stocks from day 9 to 15 [1, in press].

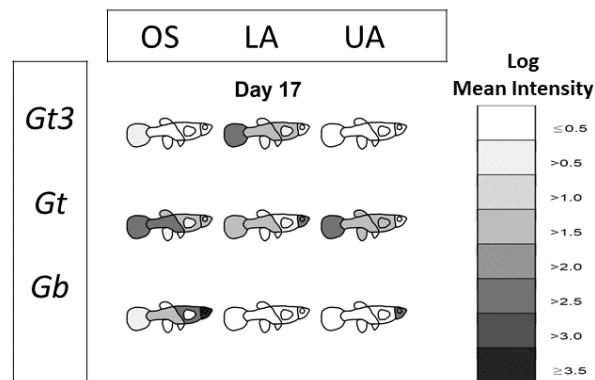


Figure A.3: Detailed visualization of fish heatmaps over eight body regions of fish across parasite strains and fish stocks on day 17 [1, in press] .

Appendix B: Grouped barcharts showing variations in mean intensities at four main body regions of fish over time

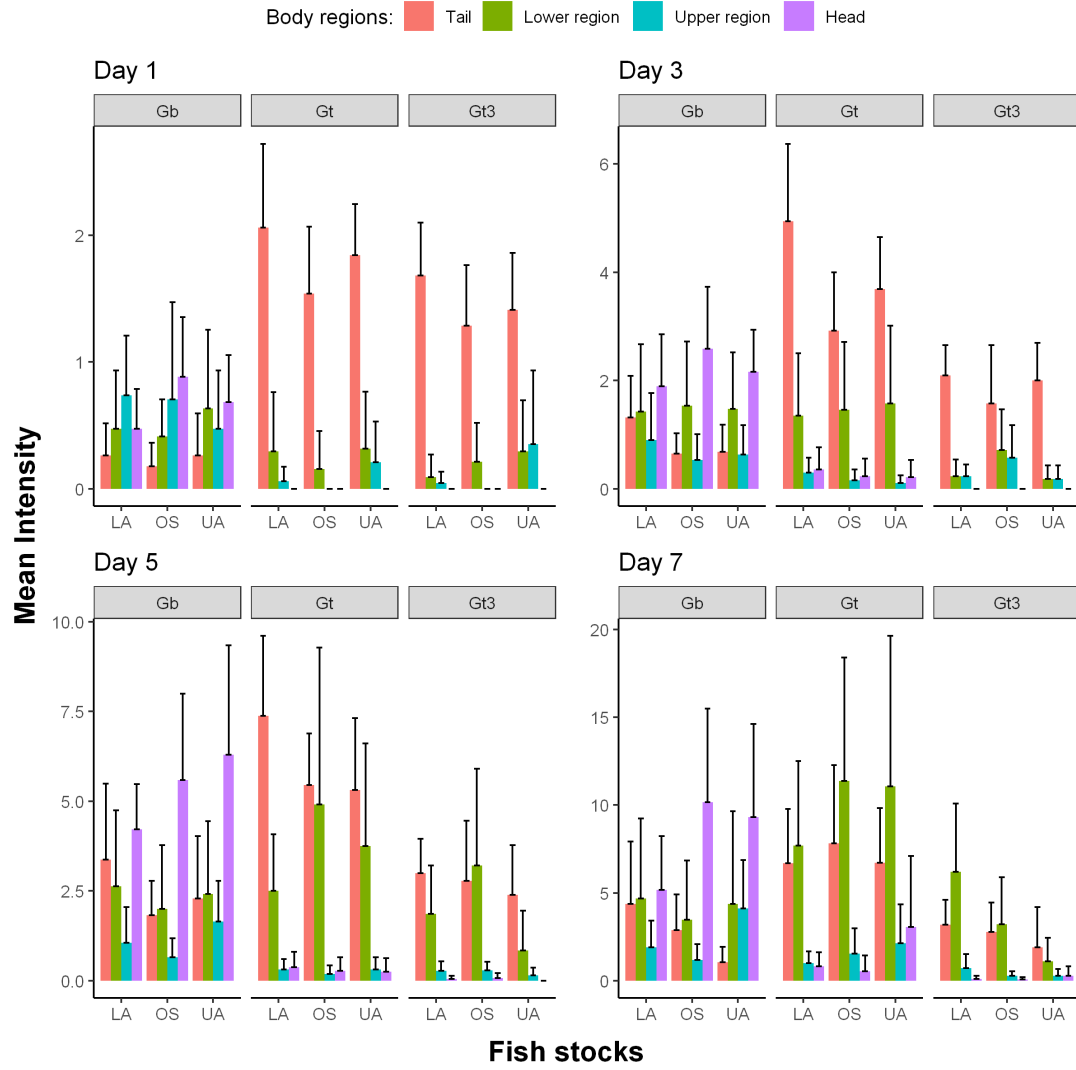


Figure A.4: Grouped barcharts showing variations in mean intensities at four main body regions of fish across parasite strains and fish stocks over surviving fish from day 1 to 7 [1, in press].

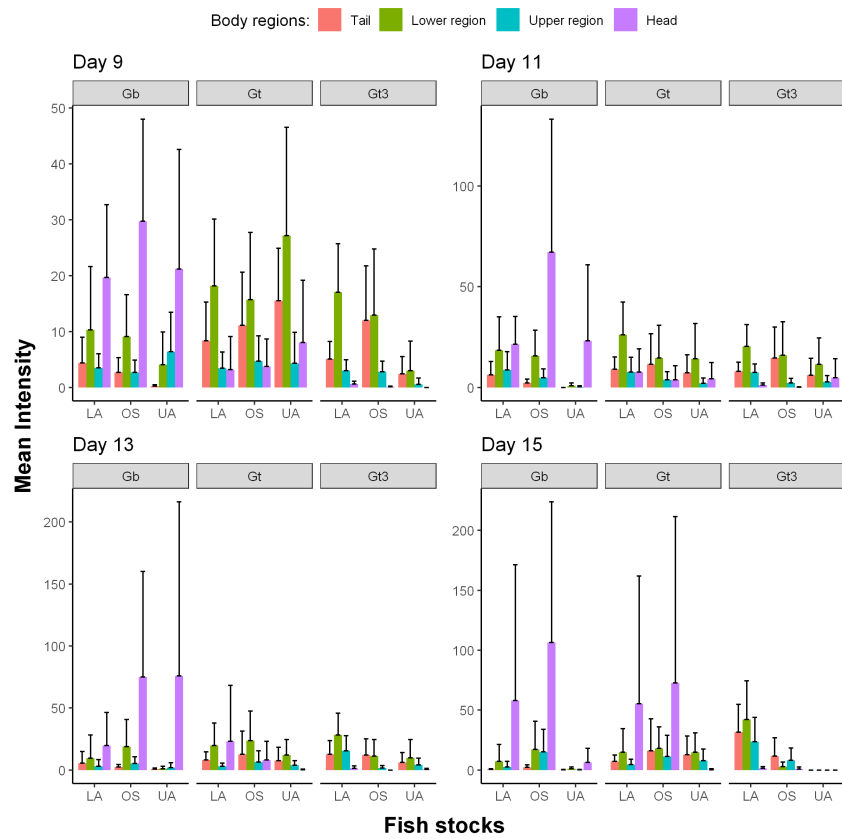


Figure A.5: Grouped barcharts showing variations in mean intensities at four main body regions of fish across parasite strains and fish stocks over surviving fish from day 9 to 15 [1, in press].

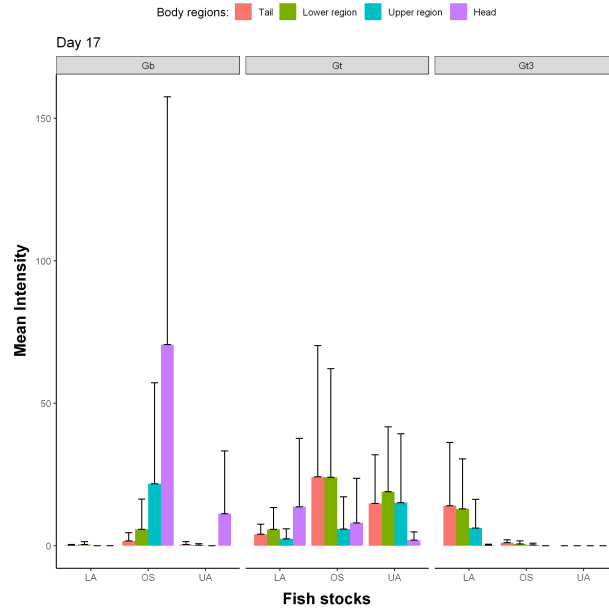


Figure A.6: Grouped barcharts showing variations in mean intensities at four main body regions of fish across parasite strains and fish stocks over surviving fish on day 17 [1, in press].

Appendix C: R Codes for Exact SSA of the B-D-C process

C.1: Function for updating events of the B-D-C process via exact SSA

```
SSA_update_event=function(X,fish_status,rate,total_rate){  
  #Let b,d & c be the birth, death & catastrophe parameters  
  #X be the number of parasites  
  #fish_status <- 1 # fish starts out alive  
  
  if(total_rate == 0) {  
    return(list(X = X, t_incr = Inf)) # zero population  
  }  
  
  #Determine event occurrence from single draw  
  u<-runif(1,0, total_rate)  
  if (u<abs(rate[1])){  
    #birth of parasites  
    X<-X+1  
  } else if(u<abs(rate[1]+rate[2])){  
    #death of parasites  
    X<-X-1  
  } else {  
    #catastrophe or death of fish  
    X<-0  
    fish_status<-2  
  }  
  t_incr <- rexp(1, total_rate) # time increment  
  #Returns parasite numbers,time step and survival status  
  return(list(X = X,t_incr=t_incr,fish_status=fish_status))  
}
```

C.2: Function for exact stochastic simulation (SSA)

```
#Function for exact simulation of the B-D-C process
Exact_BDC<-function(X0,b,d,c,ti=0,tmax=30){
  #Let ti be the initial time (set at 0)
  #tfinal be the final simulation time
  rate<-numeric(3) #event rates
  #stop simulation if total population exceeds this limit
  pop_max <- 10000
  #Time variable; ti<- 0; tmax<-30;
  save_ti <- 1:tmax #Discrete times to store simulation
  save_TF <- rep(FALSE, length(save_ti))
  #parasite pop over time
  pop <-matrix(NA,1,length(save_ti))
  # host host status at each time point
  alive <- rep(2, length(save_ti))
  alive_ti <- 1 #fish starts out alive
  X<-X0;pop_ti <- sum(X)
  while(sum(save_TF) < length(save_ti)){
    #Calculate rate of events
    #probability of birth
    rate[1]<- b*X
    #probability of death
    rate[2]<- d*X
    #probability of catastrophe
    rate[3]<- c*X
    #total rate
    total_rate<- rate[1]+rate[2]+rate[3]
    if(sum(pop_ti) > pop_max){
      cat("Popmax_exceeded","\n")
      break
    }
    if(alive_ti == 2) break
    output <-SSA_update_event(X,fish_status=alive_ti,
      rate=rate,total_rate=total_rate)
    #Update time to next event
    ti <- ti + output$t_incr
    #break if there is negative population
    if (X < 0) break
    # Events to occur
    save_new <- which((ti >= save_ti) & !save_TF)
    for (i in save_new){
      pop[,i]<- pop_ti
      alive[i] <- alive_ti
    }
    save_TF <- (ti >= save_ti)
    X<- output$X
    pop_ti <- sum(X)
    alive_ti <- output$fish_status
  }
  #Returns the parasite numbers & survival status over time
  return(list(pop=pop,alive = alive))
}
```

Appendix D: Julia codes for computing the log-likelihood function

D.1: Computing constants of the B-D-C transition function

```
module BDCfit #begin module
using PolynomialRoots
export logL
function BDCconsts(lambda, mu, rho, t)
#lambda, mu, rho are B-D-C parameters respectively
# Computing constants of BDC process at time t
rts = sort(real(roots([mu, -(lambda+mu+rho), lambda])))
v0 = rts[1]
v1 = rts[2]
sigma = exp(-lambda*(v1 - v0)*t)
k1 = v0*v1*(1 - sigma)/(v1 - sigma*v0)
k2 = (v1*sigma - v0)/(v1 - sigma*v0)
k3 = (1 - sigma)/(v1 - sigma*v0)
return [k1, k2, k3]
end
function gamma_n_j(nmax)
# calculates gamma^n_j for n = 1, ..., nmax & j = 1, ..., n
# used by ProbBDC
gnj = zeros{BigInt, nmax, nmax}
gnj[1,1] = 1
if nmax > 1
for n = 2:nmax
gnj[n,1] = n*gnj[n-1,1]
end
for j = 2:nmax
for n = j:nmax
gnj[n,j] = gnj[n-1,j-1] + (n+j-1)*gnj[n-1,j]
end
end
end
return gnj
end
function delta_m_j(mmax, k1, k2, k3)
# calculates delta^m_j for n = 1, ..., mmax & j = 1, ..., n
# used by ProbBDC; k1, k2, k3 will be output from BDCconsts
k = (k2 + k1*k3)/k1/k3
dmj = zeros{BigFloat, mmax, mmax}
dmj[1,1] = k
if mmax == 1
return dmj
else
for m = 2:mmax
dmj[m,1] = k*m
for j = 2:m
dmj[m,j] = k*(m - j + 1)*dmj[m,j-1]
end
end
return dmj
end
end
```


D.2: Computing the B-D-C transition function

```
function ProbBDC(lambda, mu, rho, t, mmax, nmax)
# P(X_t=n | X_0=m) for -1 <= m <= mmax and
# -1 <= n <= nmax
# where -1 indicates extinction by catastrophe
cc = BDCconsts(lambda, mu, rho, t)
k1 = cc[1]
k2 = cc[2]
k3 = cc[3]
k4 = (k1 + k2) / (1 - k3)
P = zeros(Float64, mmax+2, nmax+2)
P[1,1] = 1
P[2,2] = 1
k1_powers = zeros(BigFloat, mmax)
k3_powers = zeros(BigFloat, nmax)
k4_powers = zeros(BigFloat, mmax)
facts = zeros(BigFloat, nmax)
k1_powers[1] = k1
k4_powers[1] = k4
P[3,1] = Float64(1 - k4)
P[3,2] = Float64(k1)
for m = 2:mmax
k1_powers[m] = k1*k1_powers[m-1]
k4_powers[m] = k4*k4_powers[m-1]
P[m+2,1] = Float64(1 - k4_powers[m])
P[m+2,2] = Float64(k1_powers[m])
end
k3_powers[1] = k3
facts[1] = 1
for n = 2:nmax
k3_powers[n] = k3*k3_powers[n-1]
facts[n] = n*facts[n-1]
end
gnj = gamma_n_j(nmax)
dmj = delta_m_j(mmax, k1, k2, k3)
for m = 1:mmax
for n = 1:nmax
x = BigFloat(0)
for j = 1:(min(m,n))
x = x + gnj[n,j]*dmj[m,j]
end
P[m+2,n+2] = Float64(x*k1_powers[m]*k3_powers[n]/facts[n])
end
end
return P
end
```

D.3: Computing the B-D-C log-likelihood function

```
function logL(lambda, mu, rho, x)
# calculate the log likelihood for params:
# lambda, mu, rho and data x
# each row of x are population at times: t= 1, 3, 5,7,..17
# assume population at time 0 is 2;
# state -1 indicates catastrophe
mmax1 = 2
nmax1 = Int64(max(maximum(x[:,1]), 2))
P1 = ProbBDC(lambda, mu, rho, 1, mmax1, nmax1)
mmax2 = Int64(max(maximum(x[:,1:8]), 2))
nmax2 = Int64(max(maximum(x), 2))
P2 = ProbBDC(lambda, mu, rho, 2, mmax2, nmax2)
el = 0
for i = 1:size(x, 1) # logL for observation i
# time 0 to time 1 transition
el = el + log(P1[4, Int64(x[i,1]+2)])
for j = 1:8
# time 2j-1 to time 2j+1 transition
el = el + log(P2[Int64(x[i,j]+2), Int64(x[i,j+1]+2)])
end
end
return el
end

end #module
```

E.1: Function for updating B-D-C Hybrid τ -leaping simulation

(see Appendix E)

```
#Function to update tau-leaping

tauleap_update<-function(X,tau,fish_status,
rate,total_rate){
#Inputs:
#X=parasite number, tau=leap size, rate=event rates
#fish_status=survival status, total_rate=total rate
if(runif(1) < rate[3]*tau){ # catastrophe

X <- 0
fish_status<-2
}else{ # births and deaths
X <- X + rpois(1, rate[1]*tau) - rpois(1,rate[2]*tau)
}
#Returns the parasite numbers & survival status
return(list(X = X,fish_status=fish_status))
}
```

Appendix E: R Codes for B-D-C Hybrid τ -leaping algorithms

E.2: Function for τ -leaping based on Gillespie 2001

```
HTL2001<-function(X0,b,d,c,error,ti=0,tmax=30){
  #ti<-0 #initial time, X0=initial population size
  #tmax<-30 #final time
  rate<-numeric(3) #store event rates
  save_ti <- 1:tmax #Times to simulate
  # host fish status at each time point
  alive <- rep(2, length(save_ti))
  alive_ti <- 1 #fish starts out alive
  save_TF <- rep(FALSE, length(save_ti))
  # parasite pop at observed time point
  pop <-matrix(NA,1,length(save_ti))
  X<-X0;pop_ti <- sum(X)
  while (ti<tmax){
    #Computing event rates (birth,death & catastrophe)
    rate[1]<- b*X;rate[2]<- d*X;rate[3]<- c*X
    #representing a0(x) or total rate
    total_rate<- rate[1]+rate[2]+rate[3]
    #Computing tau on Gillespie (2001)
    tau<-(error*(b+d))/(abs(b-d)*max(b,d))
    #Switching condition
    leap_condition<- 2/total_rate #leap condition
    #Running Tau-leaping
    if (tau>leap_condition){#Execute tau-leaping
      ti <- ti + tau #update time
      output<-tauleap_update(X,tau=tau,fish_status=alive_ti,
        rate=rate,total_rate=total_rate)
      X<-output$X
    } #end of tau-leaping
    #Running exact SSA algorithm if tau<=leap_condition
    else {#Execute exact SSA
      output<-SSA_update_event(X,fish_status=alive_ti,
        rate=rate,total_rate=total_rate)
      X<- output$X;ti <- ti +output$t_incr# update time
      if (X < 0) break #break if there is negative population
      if (alive_ti == 2) break
      # saving output
      save_new <- which((ti >= save_ti) & !save_TF)
      for (i in save_new){
        pop[,i]<- pop_ti; alive[i] <- alive_ti
      }
      save_TF <- (ti >= save_ti)
      X<- X;pop_ti<- sum(X);alive_ti <- output$fish_status
    }
    #Returns parasite numbers & survival status over time
    return(list(pop=pop,alive=alive))
  }
}
```

E.3: Function for τ -leaping based on Gillespie and Petzold (2003)

```
HTL2003<-function(X0,b,d,c,error,ti=0,tmax=30){
  #ti<-0 #initial time, X0=initial population size
  #tmax<-30 #final time
  rate<-numeric(3) #store event rates
  Leap_sizes<- NULL #store leap size
  save_ti <- 1:tmax #Times to simulate
  # host fish status at each time point
  alive <- rep(2, length(save_ti))
  alive_ti <- 1 #fish starts out alive
  save_TF <- rep(FALSE, length(save_ti))
  # parasite pop at observed time point
  pop <-matrix(NA,1,length(save_ti))
  X<-X0;pop_ti <- sum(X)
  while (ti<tmax){
    #Computing event rates (birth,death & catastrophe)
    rate[1]<- b*X;rate[2]<- d*X;rate[3]<- c*X
    #representing a0(x) or total rate
    total_rate<- rate[1]+rate[2]+rate[3]
    #Computing tau on Gillespie & Petzold 2003
    Leap_sizes[[1]]<- (error*(b+d))/(abs(b-d)*max(b,d))
    Leap_sizes[[2]]<- X*(error*(b+d))^2/((b+d)*max(b^2,d^2))
    tau<- min(Leap_sizes[[1]],Leap_sizes[[2]])#leap size
    #Switching condition
    leap_condition<- (1/(10*total_rate)) #leap condition
    #Running Tau-leaping
    if (tau>leap_condition){#Execute tau-leaping
      ti <- ti + tau #update time
      output<-tauleap_update(X,tau=tau,fish_status=alive_ti,
        rate=rate,total_rate=total_rate)
      X<-output$X
    } #end of tau-leaping
    #Running exact SSA algorithm if tau<=leap_condition
    else {#Execute exact SSA
      output<SSA_update_event(X,fish_status=alive_ti,
        rate=rate,total_rate=total_rate)
      X<- output$X; ti <- ti +output$t_incr# update time
      if (X < 0) break#break if there is negative values
      if (alive_ti == 2) break
      # saving output
      save_new <- which((ti >= save_ti) & !save_TF)
      for (i in save_new){
        pop[,i]<- pop_ti;alive[i] <- alive_ti
      }
      save_TF <- (ti >= save_ti)
      X<- X;pop_ti<- sum(X);alive_ti <- output$fish_status
    }

    #Returns parasite numbers & survival status over time
    return(list(pop=pop, alive=alive))
  }
}
```

Appendix F: R Codes for the modified weighted-iterative ABC & ABC Post-Processing Regression Analysis

F.1: Functions for population projection & weighted distances

```
## 1. Function for population projection
#until day 17 after host mortality

#ga= gamma which is tuning parameter (set at 0.9)
project <- function(pop_single, alive_single, ga) {

  # project parasite numbers beyond fish mortality
  n <- length(alive_single)
  k <- sum(alive_single == 1)
  if (k == n) return(pop_single)
  if (k == 0) return(matrix(0, nrow=4, ncol=n))
  if (k == 1) return(matrix(pop_single[,1], nrow=4, ncol=n))
  z <- log(colSums(pop_single[,1:k], na.rm=T))
  al <- sum( (z[k] - z[1:(k-1)]) * ((k-1):1)
  * ga^((k-1):1), na.rm=T) /
  sum( ((k-1):1)^2 * ga^((k-1):1), na.rm=T)

  pop_single[, (k+1):n] <- pop_single[, k] %*%
  t( exp( (1:(n-k))*al ) )
  return(pop_single)
}

#converting function to byte-code compilation
project_compiler=cmpfun(project)

## 2. Function for computing weighted distance
#between simulated and observed summary statistics

w_distance <- function(S1, S2, weight) {
  n<- dim(S1)[1]
  #squared difference between matrix S1 & S2
  Squared_diff_mat<- (S1-S2)^2
  #Multiplying vector to weights
  Weighted_sq_diff<- lapply(1:dim(S1)[1],
  function(k) weight*Squared_diff_mat[k, ])
  #total weighted distances (WSS)
  WSS<- do.call("sum",Weighted_sq_diff)
  #return a scaled weighted sum of squares distance
  return(sqrt(WSS/n))
}

#converting function to byte-code compilation
distance_compiler=cmpfun(w_distance)
```

F.2(i): External functions for Galton-Watson & GMM estimators for B-D-C parameter estimation

```
# 1. Function for computing BDC constants and PGF
BDCconsts <- function(lambda, mu, rho, t) {
  # Constants used in calculating distribution of BDC process at time t

  roots <- sort(Re(polyroot(c(mu, -(lambda+mu+rho), lambda))))
  v0 <- roots[1]
  v1 <- roots[2]
  sigma <- exp(-lambda*(v1 - v0)*t)
  k1 <- v0*v1*(1 - sigma)/(v1 - sigma*v0)
  k2 <- (v1*sigma - v0)/(v1 - sigma*v0)
  k3 <- (1 - sigma)/(v1 - sigma*v0)
  return(list(k1=k1, k2=k2, k3=k3, sigma=sigma, v0=v0, v1=v1))
}

# 2. Function for the probability generating function G(z,t)
PGF_z <- function(lambda, mu, rho, t, z, m) {
  #v0<-((lambda+mu+rho)-sqrt((lambda+mu+rho)^2-4*mu*lambda))/(2*lambda)
  #v1<-((lambda+mu+rho)+sqrt((lambda+mu+rho)^2-4*mu*lambda))/(2*lambda)
  constants=BDCconsts(lambda, mu, rho, t)
  v0<- constants$v0
  v1<- constants$v1
  sigma<- constants$sigma
  num<-(v0*v1*(1-sigma))+(z*(v1*sigma-v0))
  den<- v1-(sigma*v0)-(z*(1-sigma))
  return( (num/den)^m)
}

#3. Analytical probability of death due to catastrophe
#Estimating C(t)=P(catastrophe resulting in 0 population|host death)
Prob_catastrophe<- function(lambda, mu, rho, t, z=1, m=2) {
  constant <- 1-PGF_z_compiler(lambda=lambda,
    mu=mu, rho=rho, t=t, z=z, m=m)
  #return the probability of catastrophic extinction
  return(constant)
}

#4. Function of the Exact mean/1st moment of the BDC process
First_moment<-function(b, d, c, t, m) {
  #b,d,c are the birth,death and catastrophe rates; m=X0=2 and t=time
  roots <- sort(Re(polyroot(c(d, -(b+d+c), b))))
  v0 <- roots[1]
  v1 <- roots[2]
  sigma<-exp(-b*(v1-v0)*t)
  k1<-(v0*v1*(1-sigma))/(v1-(sigma*v0))
  k2<-((v1*sigma)-v0)/(v1-(sigma*v0))
  k3<-(1-sigma)/(v1-(sigma*v0))
  expectation=m*((k1+k2)/(1-k3))^(m-1)*(k2+(k1*k3))*(1-k3)^-2
  return(expectation)#returns 1st moment
}
```

F.2(ii): External functions for Galton-Watson & GMM estimators for B-D-C parameter estimation

```
# 1. Function of the 2nd moment of the BDC process
Second_moment<-function(b,d,c,t,m){
  roots <- sort(Re(polyroot(c(d, -(b+d+c), b))))
  v0 <- roots[1]
  v1 <- roots[2]
  sigma<-exp(-b*(v1-v0)*t)
  k1<- (v0*v1*(1-sigma))/(v1-(sigma*v0))
  k2<-((v1*sigma)-v0)/(v1-(sigma*v0))
  k3<-(1-sigma)/(v1-(sigma*v0))
  expectation<- m*((k1+k2)/(1-k3))^(m-1)*(k2+(k1*k3))*(1-k3)^-2

  Second_derivative_pgf<-((2*m*k3*(k2+k1*k3))*
  ((k1+k2)/(1-k3))^(m-1)*(1-k3)^-3 +
  m*(m-1)*(k2+k1*k3)^2*
  ((k1+k2)/(1-k3))^(m-2)*(1-k3)^-4)
  Variance<-(Second_derivative_pgf+ expectation)-(expectation)^2

  Second_moment_results<- Variance + expectation^2
  return(Second_moment_results)#returns 2nd moment
}

# 2. Function of the 3rd moment of the BDC process
Third_moment<-function(b,d,c,t,m){
  roots <- sort(Re(polyroot(c(d, -(b+d+c), b))))
  v0 <- roots[1]
  v1 <- roots[2]
  sigma<-exp(-b*(v1-v0)*t)
  k1<-(v0*v1*(1-sigma))/(v1-(sigma*v0))
  k2<-((v1*sigma)-v0)/(v1-(sigma*v0))
  k3<-(1-sigma)/(v1-(sigma*v0))
  expectation<- m*((k1+k2)/(1-k3))^(m-1)*(k2+(k1*k3))*(1-k3)^-2

  Second_derivative_pgf<-((2*m*k3*(k2+k1*k3))
  *((k1+k2)/(1-k3))^(m-1)*(1-k3)^-3 +
  m*(m-1)*(k2+k1*k3)^2*
  ((k1+k2)/(1-k3))^(m-2)*(1-k3)^-4)

  Third_derivative_pgf<- 6*m*(k2+k1*k3)*(k3^2)*
  (((k1+k2)/(1-k3))^(m-1))*(1-k3)^(-4)+
  6*m*(m-1)*((k2+k1*k3)^2)*k3*
  (((k1+k2)/(1-k3))^(m-2))*(1-k3)^(-5)+
  m*(m-1)*(m-2)*((k2+k1*k3)^3)*
  (((k1+k2)/(1-k3))^(m-3))*(1-k3)^(-6)

  Variance<-(Second_derivative_pgf+ expectation)-(expectation)^2

  Second_moment_results<- Variance + expectation^2

  Third_moment_results<- Third_derivative_pgf + (3*Second_moment_results)-
  (2*expectation)

  return(Third_moment_results)#returns 3rd moment
}
```

F.2(iii): External functions for Galton-Watson & GMM estimators for B-D-C parameter estimation

```
# 1. Set the catastrophe state -1 to 0
zero.catastrophe <- function (x) {
  x[x<0] <- 0
  return(x)
}

# 2. Set the ratio Z(i)/Z(i-1) to 1 if NA
#(due to case of 0/0 in Z(i)/Z(i-1))
one.ratio <- function (x) {
  x[is.na(x)|x==Inf|x==Inf] <- 1
  return(x)
}

# 3. functions for sample moments
sample_mean_1st<- function(x) sum(x)/length(x)
sample_mean_2nd<- function(x) sum(x^2)/length(x)
sample_mean_3rd<- function(x) sum(x^3)/length(x)

### Computing the 2-step GMM estimates ###

time<-seq(1,17,by=2)

# 4. Objective function for 1st step of GMM
g_objectivefunc_firstStep <- function(x,prob_sample,
fixed=c(FALSE,FALSE,FALSE)) {
  Prob_catastrophe_analytical<- rep(NA,length=length(time))
  params<-fixed
  function(p){
    params[!fixed]<-p
    #The three parameters to be optimized
    b1<-params[1]
    d1<-params[2]
    c1<-params[3]

    #Computing theoritical prob of catastrophe
    for(i in seq_along(time)){
      Prob_catastrophe_analytical[i]<-Prob_catastrophe(lambda=b1,mu=d1,rho=c1,
        ,t=time[i])
    }

    m1 <- First_moment(b=b1,d=d1,
      c=c1,t=seq(1,17,by=2),m=2) -
      apply(zero.catastrophe(x),1,sample_mean_1st)
    m2 <- Second_moment(b=b1, d=d1,
      c=c1,t=seq(1,17,by=2),m=2) -
      apply(zero.catastrophe(x),1,sample_mean_2nd)
    m3 <- Third_moment(b=b1, d=d1,
      c=c1,t=seq(1,17,by=2),m=2) -
      apply(zero.catastrophe(x),1,sample_mean_3rd)

    Catastrophe_Prob<- Prob_catastrophe_analytical- prob_sample

    gbar_theta<-c(mean(m1),mean(m2),mean(m3),mean(Catastrophe_Prob))

    Objective_func<- t(gbar_theta)%*%gbar_theta

  }
}
```


F.2(iv): External functions for Galton-Watson & GMM estimators for B-D-C parameter estimation

```
### Computing the 2-step GMM estimates(continued) ###

#First step of GMM
GMM_firstStep<-function(prob_sample,x){
  objec_func<- g_objectivefunc_firstStep(x=x,prob_sample=prob_sample)
  initial<-c(2, 1, 0.001)# initial values to optimize over
  estimates<-constrOptim(initial, objec_func, NULL,
    ui=rbind(c(1,0,0), # lambda>0
      c(0,1,0), # mu >0
      c(0,0,1) # rho > 0
    ),
    ci=c(0,0,0),method='Nelder-Mead')$par

  return(estimates)
}

# Second step of GMM
# Second-step of the GMM optimization: Function to calculating the weight
#matrix
Weight<-function(x,prob_sample,estimate1){
  est_step1<- c(estimate1)
  Prob_catastrophe_analytical1<- rep(NA,length=length(time))
  #Computing theoretical prob of catastrophe
  for(i in seq_along(time)){
    Prob_catastrophe_analytical1[i]<- Prob_catastrophe(
      lambda=est_step1[1],mu=est_step1[2],
      rho=est_step1[3],t=time[i])
  }

  m1 <- First_moment(b=est_step1[1],
    d=est_step1[2],c=est_step1[3],t=seq(1,17,by=2),m=2)
  -apply(zero.catastrophe(x),1,sample_mean_1st)
  m2 <- Second_moment(b=est_step1[1],
    d=est_step1[2],c=est_step1[3],t=seq(1,17,by=2),m=2)
  -apply(zero.catastrophe(x),1,sample_mean_2nd)
  m3 <- Third_moment(b=est_step1[1],d=est_step1[2],
    c=est_step1[3],t=seq(1,17,by=2),m=2)
  -apply(zero.catastrophe(x),1,sample_mean_3rd)
  Catastrophe_Prob<- Prob_catastrophe_analytical1- prob_sample

  g<-cbind(m1,m2,m3,Catastrophe_Prob)

  covariance_matrix<- cov(g)
  #Setting off-diagonals to 0 to obtain an
  #invertible weighting (diagonal) matrix
  #by assuming that the moment conditions are uncorrelated
  covariance_matrix[lower.tri(covariance_matrix)] <- 0
  covariance_matrix[upper.tri(covariance_matrix)] <- 0

  #Finding inverse for the covariance diagonal matrix
  #finding reciprocal of entries
  weightmatrix<- 1/covariance_matrix
  weightmatrix[lower.tri(weightmatrix)] <- 0
  weightmatrix[upper.tri(weightmatrix)] <- 0
  weightmatrix

}
```

F.2(v): External functions for Galton-Watson & GMM estimators for B-D-C parameter estimation

```

### Computing the 2-step GMM estimates(continued) ####
#Second optimization step
g_objectivefunc_2ndStep <- function(x, prob_sample,
weighting_matrix, fixed=c(FALSE, FALSE, FALSE)) {
  Prob_catastrophe_analytical<-rep(NA, length=length(time))
  params<-fixed
  function(p){
    params[!fixed]<-p
    #The three parameters to be optimized
    b1<-params[1]
    d1<-params[2]
    c1<-params[3]

    #Computing theoritical prob of catastrophe
    for(i in seq_along(time)){
      Prob_catastrophe_analytical[i]<- Prob_catastrophe(lambda=b1, mu=d1,
        rho=c1, t=time[i])
    }

    m1 <-First_moment(b=b1, d=d1, c=c1, t=seq(1, 17, by=2), m=2)
    -apply(zero.catastrophe(x), 1, sample_mean_1st)
    m2 <-Second_moment(b=b1, d=d1, c=c1, t=seq(1, 17, by=2), m=2)
    -apply(zero.catastrophe(x), 1, sample_mean_2nd)
    m3 <-Third_moment(b=b1, d=d1, c=c1, t=seq(1, 17, by=2), m=2)
    -apply(zero.catastrophe(x), 1, sample_mean_3rd)

    Catastrophe_Prob<-Prob_catastrophe_analytical- prob_sample

    gbar_theta<-c(mean(m1), mean(m2), mean(m3),
      mean(Catastrophe_Prob))

    Objective_func<- t(gbar_theta)%*%
      weighting_matrix%*%gbar_theta

  }

}

#second step of GMM
GMM_2ndStep<-function(prob_sample, x, weighting_matrix){
  objec_func<- g_objectivefunc_2ndStep(x=x,
    prob_sample=prob_sample, weighting_matrix=
    weighting_matrix)
  # initial values to optimize over
  initial<-c(2, 1, 0.001)
  estimates=constrOptim(initial, objec_func, NULL,
    ui=rbind(c(1,0,0), # lambda>0
      c(0,1,0), # mu >0
      c(0,0,1) # rho > 0
    ),
    ci=c(0,0,0), method='Nelder-Mead')$par
  return(estimates)
}

```

F.2(vi): External functions for Galton-Watson & GMM estimators for B-D-C parameter estimation

```
#Restructuring data format for GW-GMM BDC estimation
RestructureData_BDC<- function(pop, alive, group){
  #Inputs:pop=parasite population per
  #region over time
  #alive= survival status over time
  # group=parasite-fish groups

  # to store parasite numbers over
  #time as a dataframe for each parasite-fish
  ParasiteData_combined<- NULL
  # to store survival status as a
  #dataframe for each parasite-fish
  SurvStatus_combined<- NULL

  #Set NA in pop to state 0 denoting host
  # death for the B-D-C estimation

  for(pf in seq_along(group)){

    ParasiteData_combined[[pf]]<- matrix(NA,
      nrow=9, ncol=numF[[pf]])
    #Array for time steps fish was alive
    #for each combination
    SurvStatus_combined[[pf]]<- matrix(NA,
      nrow=9, ncol=numF[[pf]])
    for(i in 1:numF[[pf]]){
      # total parasites over time for each fish belonging to
      #each parasite-fish group
      # state -1 in the BDC denote host death
      ParasiteData_combined[[pf]][,i]<-
        na.zero(apply(pop[[pf]][i,,], 2, sum))
      SurvStatus_combined[[pf]][,i]<-
        alive[[pf]][i, ]
    }
  }
  return(list(PopTime_group=ParasiteData_combined,
    SurvTime_group=SurvStatus_combined))
}
```

F.2(vii): External functions for Galton-Watson & GMM estimators for B-D-C parameter estimation

```
#Function for finding Maximum likelihood estimates for the
#catastrophe rate given the GW estimate of
#birth and death rates for the B-D-C model
MLE_catastrophe<-function(b_est,d_est,dead_fish_time){
  log_like<-0
  #LogLikelihood function to maximize
  Catastrophe_Loglik<-function(param){
    rho<-param[1]

    #log likelihood function for catastrophe rate
    for(i in dead_fish_time){
      #sum across all dead fish for each group
      if(i>=3){#if the time to death>=3
        log_like<-log_like+
          na.zero(log(Prob_catastrophe(lambda=b_est,
            mu=d_est,rho=rho,t=i))-
            Prob_catastrophe(lambda=b_est,
            mu=d_est,rho=rho,t=(i-2)))
      }else{#if the time to death=1
        log_like<-log_like+
          na.zero(log(Prob_catastrophe(lambda=b_est,
            mu=d_est,rho=rho,t=i)-
            Prob_catastrophe(lambda=b_est,mu=d_est,
            rho=rho,t=0)))
      }
    }
    log_like
  }

  Catastrophe_Loglik_compiler=cmpfun(Catastrophe_Loglik)

  ## Inequality constraints: rho>0

  estimates<-maxLik(logLik=Catastrophe_Loglik_compiler,
    start=c(rho= 1e-5))

  #returning estimates of catastrophe rate
  return(as.vector(estimates$estimate))
}

#External scripts
source("MLE_catastrophe-script.r")
source("GMM-1st2nd-Steps-script.r")
```

F.3: Function for computing the B-D-C model parameters as extra ABC summary statistics using Galton-Watson & GMM estimations

```

GW_GMM_BDCestimator<-function(X0,pop,alive,group){
  #X0= initial parasites
  Parasite_data<- NULL; survival_data<- NULL
  #re-structuring the format of the data into the
  # 9 parasite-fish groups
  data<-RestructureData_BDC(pop=pop, alive=alive, group=group)

  time<-seq(1,17,by=2)
  # Parasite_data[[pf]][, fish_index]
  Prob_catastrophe_analytical=Prob_catastrophe_sample=
  matrix(0,nrow=length(time),ncol=length(group))
  ## Initialize GMM ###
  #Computing catastrophic probability analytically
  # & based on the sample data

  #computing sample probability of catastrophe

  time_index<- seq_along(time)
  for (pf in seq_along(group)){
    Parasite_data[[pf]]<- data$PopTime_group[[pf]]
    survival_data[[pf]]<- data$SurvTime_group[[pf]]
    for(i in time_index){
      if(any(survival_data[[pf]][i,]==2)==TRUE){
        #print(paste("time=",time[i]))
        fish_dead_sim<-length(which(
          survival_data[[pf]][i,]==2))
        #print(fish_dead_sim)
        Prob_catastrophe_sample[i,pf]<-
          fish_dead_sim/dim(survival_data[[pf]])[2]
      }
    }

  }
  #Let Zi_t be the population for fish i at time t
  # Let alive_status be the survival status of each fish
  Z=NULL; alive_status=NULL
  for(pf in seq_along(group)) {
    Z[[pf]]<-list()
    alive_status[[pf]]<-list()
  }

  ##GW_GMM_BDCestimator function continues at the next page##
  +++

```

```

#Continuation of GW_GMM_BDCestimator function
for(pf in seq_along(group)){
  for(k in 1:numF[[pf]]){
    Z[[pf]][[k]]<- Parasite_data[[pf]][,k]
    alive_status[[pf]][[k]]<-survival_data[[pf]][,k]
  }
}
#Computing the mean and variance for the
#Galton-Watson process based on fish survival
# And for each k replicate
mean_GW=NULL; var_GW=NULL; mean_sum_num=NULL;
mean_sum_den=NULL; var_sum=NULL
#Computing the mean of GW process
for(pf in seq_along(group))
#initial summation for the GW mean
mean_sum_num[[pf]]=mean_sum_den[[pf]]=0
for(pf in seq_along(group)){
  for(k in 1:numF[[pf]]){
    if(all(survival_data[[pf]][,k]==1)==TRUE){

      mean_sum_num[[pf]]<-mean_sum_num[[pf]]+
      sum(Z[[pf]][[k]][1:9])# sum from t1-t17
      mean_sum_den[[pf]]<-mean_sum_den[[pf]]+
      sum(Z[[pf]][[k]][1:8])+X0 #sum from t0-t15
    }
  }
  mean_GW[[pf]]<- one.ratio(mean_sum_num[[pf]]/
  mean_sum_den[[pf]])#if 0/0=1
}
#computing the variance of GW process
#initial summation for GW variance
for(pf in seq_along(group)) var_sum[[pf]]<-0
for(pf in seq_along(group)){
  for(k in 1:numF[[pf]]){
    if(all(survival_data[[pf]][,k]==1)==TRUE){
      var_sum[[pf]]<-var_sum[[pf]]+
      sum(Z[[pf]][[k]][1:9]*
      (one.ratio(Z[[pf]][[k]][1:9]/
      c(X0,Z[[pf]][[k]][1:8])) -
      mean_GW[[pf]])^2)
    }
  }
  var_GW[[pf]]<- var_sum[[pf]]/
  (numF[[pf]]*length(time))
}
### GMM estimation ###
birth_rate=NULL;death_rate=NULL; c_estimates<-
NULL;delta_t=2;
BDC_estimates=NULL
GMM_resultsStep1=NULL; GMM_resultsStep2=NULL;
weighting_matrix_cov=NULL; method=NULL

#Estimating the catastrophe rate
#using MLE when m>1 for GW estimation
#time at death for each fish i and
#replicate/simulation run k
t_death<-NULL;
for(pf in seq_along(group)){ t_death[[pf]]<-rep(NA, length=numF[[pf]]) }
for(pf in seq_along(group)){
  for(k in 1:numF[[pf]]){
    #time to death
    t_death[[pf]][k]<-time[which(
    survival_data[[pf]][,k]==2)[1]]
  }
}
##GW_GMM_BDCestimator function continues at the next page##
+++

```

```

#Continuation of GW_GMM_BDCestimator function ### begining of GW and GMM
for(pf in seq_along(group)){### begining of GW and GMM
  if(mean_GW[[pf]]>1){####Consider GW if mean_GW>1
    method[[pf]]<-"GW estimation"
    birth_rate[[pf]]<-((log(mean_GW[[pf]]))
    /(2*delta_t))*(one.ratio(var_GW[[pf]]
    /(mean_GW[[pf]]*(mean_GW[[pf]]-1))) +1))
    death_rate[[pf]]<- ((log(mean_GW[[pf]]))
    /(2*delta_t))*(one.ratio(var_GW[[pf]]/
    (mean_GW[[pf]]*(mean_GW[[pf]]-1))) -1))

    #Computing MLE of catastrophe rate
    #based on estimated birth and death rates
    if(all(is.na(t_death[[pf]]))==FALSE){
      #if at least some fish are dead
      #estimates of the catastrophe rate
      c_estimates[[pf]]<-MLE_catastrophe_compiler(
      b_est=birth_rate[[pf]],d_est=death_rate[[pf]],
      dead_fish_time=na.omit(t_death[[pf]][k]))
    }else if(all(is.na(t_death[[pf]]))==TRUE){
      #if no fish is dead
      c_estimates[[pf]]<-0
    }

    BDC_estimates[[pf]]<-c(birth_rate[[pf]],
    death_rate[[pf]],c_estimates[[pf]])
  }else if(mean_GW[[pf]]<=1){ #Consider GMM
    method[[pf]]<-"GMM estimation"
    #First stage of GMM
    GMM_resultsStep1[[pf]]<- GMM_firstStep(
    prob_sample=Prob_catastrophe_sample[,pf],
    x=as.data.frame(Parasite_data[[pf]]))

    weighting_matrix_cov[[pf]]<-Weight(x=
    as.data.frame(Parasite_data[[pf]]),
    prob_sample=Prob_catastrophe_sample[,pf],
    estimate1=GMM_resultsStep1[[pf]])

    #Second stage of GMM
    GMM_resultsStep2[[pf]]<- GMM_2ndStep(
    prob_sample=Prob_catastrophe_sample[,pf],
    x=as.data.frame(Parasite_data[[pf]]),
    weighting_matrix=weighting_matrix_cov[[pf]])

    BDC_estimates[[pf]]<- GMM_resultsStep2[[pf]]
  } ####GMM estimation ends
} #### end of GW and GMM
#Returning the B-D-C parameters and method used
BDC_estimates_df<-do.call("rbind", BDC_estimates)
return(list(BDC_estimates=BDC_estimates_df,
method_used=unique(unlist(method))))
}

```

F.4(i): Functions for initial prior & sampling proposals

```
#Prior distribution of model parameters (on log scale)

prior<- function() {
  lb1<- runif(2, -4, 1) # birth of parasites (Gt3)
  # birth rate for young parasites based on lb (Gt3)
  logb11 <- max(lb1)
  logb12<- min(lb1) # birth rate for older parasites based on lb1 (Gt3)

  lb2<- runif(2, -4, 1) # birth of parasites (Gt)
  logb21 <- max(lb2) # birth rate for young parasites based on lb2 (Gt)
  logb22<- min(lb2) # birth rate for older parasites based on lb2 (Gt)

  lb3<- runif(2, -4, 1) # birth of parasites (Gb)
  logb31 <- max(lb3) # birth rate for young parasites based on lb3 (Gb)
  logb32<- min(lb3) # birth rate for older parasites based on lb3 (Gb)

  ld1 <- runif(2, -5, 2) #death rates (Gt3)
  logd11 <- min(ld1) # death rate without an immune response (Gt3)
  logd12 <- max(ld1) # death rate with immune response (Gt3)

  ld2 <- runif(2, -5, 2) # death rates (Gt)
  logd21 <- min(ld2) #death rate without an immune response (Gt)
  logd22 <- max(ld2) # death rate with immune response (Gt)

  ld3 <- runif(2, -5, 2) # death rates (Gb)
  logd31 <- min(ld3) # death rate without an immune response (Gb)
  logd32 <- max(ld3) # death rate with immune response (Gb)

  logm<- runif(1, -4, 1) #movement rate

  logr <- runif(1, -10, 1) #immune response rate (base rate)

  # immune response (adjustment for LA fish)
  logr1 <- runif(1, -10, 1)
  logr2 <- runif(1, -10, 1) # immune response rate (adj for OS fish)
  logr3 <- runif(1, -10, 1) # immune response rate (adj for male fish)

  logs <- runif(1, -8, -2) #fish mortality rate (base rate)

  logs1 <- runif(1, -8, -2) #fish mortality (adj for male fish)

  loge1 <- runif(1, -8, log(2)) #movement rate adj (Gt3)
  loge2 <- runif(1, -8, log(2)) #movement rate adj (Gt)
  loge3 <- runif(1, -8, log(2)) #movement rate adj (Gb)

  log_kappa <- runif(1, 4.5, 6.5) #effective carrying capacity

  #Returns the prior samples on log scale
  return(c(logb11, logb12, logb21, logb22, logb31,
  logb32, logd11, logd12, logd21, logd22, logd31, logd32,
  logm, logr, logr1, logr2, logr3, logs, logs1, loge1,
  loge2, loge3, log_kappa))
}

#view next page for the perturbation kernel function
```



```

#MVN kernel given optimal bandwidth matrix H For peturba-
#tion
MultivNorm_rkernel<- function(Num,bandwidth_matrix){
  dim_k<- dim(bandwidth_matrix)[2]
  mean_vector<- rep(0,dim_k)
  #return random noise from MVN kernel
  return(tmvtnorm::rtmvnorm(n=1, mean=mean_vector,
  sigma=bandwidth_matrix,
  lower=rep(-.1,dim_k), upper=rep(.1,dim_k),
  algorithm=c("gibbs"))
)

## Function for importance proposal sampling
post <- function(samp=tha_post,importance_weight=weight,
optimal_bw_matrix=Sigma_optimal_t){

  ##new proposal based on accepted priors (samp)##
  #number of previous accepted samples
  n <- dim(samp)[1]
  sample.particle<-sample(n, 1,prob=importance_weight)
  # Perturbing sampled particle based on MVN kernel
  KDE_sampler<- samp[sample.particle, ]
  +MultivNorm_rkernel(Num=1,
  bandwidth_matrix=optimal_bw_matrix)

  new_proposal<- KDE_sampler; x<- new_proposal

  # birth rate of young>old
  x[1:2] <- sort(x[1:2], decreasing=TRUE)
  x[3:4] <- sort(x[3:4], decreasing=TRUE)
  x[5:6] <- sort(x[5:6], decreasing=TRUE)
  #death rates (without and with immune response)
  x[7:8] <- sort(x[7:8],decreasing=FALSE)
  x[9:10] <- sort(x[9:10],decreasing=FALSE)
  x[11:12] <- sort(x[11:12],decreasing=FALSE)
  return(x)
}

```

F.4(ii): Functions for computing initial summary statistics weights & setting other initial conditions for the modified ABC

```
## Computing initial weights ###
A0 <- matrix(0, 4, 2)
A0[1, 1] <- 2 #Initial parasites at the tail
B0 <- rep(1, 4) #initial immune response at 4 body regions

#Transition matrix
J<- matrix(c(0, 1, 0, 0,
1/2, 0, 1/2, 0,
0, 1/2, 0, 1/2,
0, 0, 1, 0), 4, 4, byrow=TRUE)

# initial summary statistics weights estimate
dimS<-17 #length of summary statistics for ABC
n0 <- 100 #number of simulations for initial weights
#saving summary statistic for each group sim realisation
#for computing initial weights for ABC fitting
SummaryStats_sim <- NULL; SummaryStats_sim_combined<-NULL

for (i in 1:n0) {
  theta<- prior()
  output<- SimGroup_taleap(theta1=theta,
  fish_sex=fishSex, fish_type=Fish_stock,
  strain=Strain, fish_size=fishSize, error=0.01)

  #B-D-C parameter estimates for the
  #parasite-fish groups based on simulated data
  #for each simulation realisation
  BDC_estimates_sim<-GW_GMM_BDCestimator(X0=2,
  pop=output$pop_sim, output$alive_sim,
  group=parasite_fish)$BDC_estimates

  #Computing the summary stats for each sim realisation
  SummaryStats_sim[[i]] <- Summary_stats(
  pop=output$pop_sim, alive=output$alive_sim,
  BDC_estimates=BDC_estimates_sim)
  #combining for all summary stats of
  #parasite-fish groups for each simulation realisation
  SummaryStats_sim_combined[[i]] <-do.call("rbind",
  SummaryStats_sim[[i]])
}
#dimension is rows=(n0*total_fish) by cols=17
S0<- do.call("rbind", SummaryStats_sim_combined)
#initial weight (inverse of summary statistics)
w <- 1/apply(S0, 2, var, na.rm = TRUE)
print(w) # printing initial weights
```

F.4(iii): Functions for returning priors, summaries and distances

```
#Function for returning priors, summaries and distances
ABC <- function(fork, pftn , n, w ) {
  # pftn is prior function or sampling proposals
  # n is number of samples or proposals
  # w are summary statistics weights
  dimS<-17 #dimension or number of ABC summary statistics
  number_of_parameters<- 23 #number of model parameters
  # matrix of prior distributions
  theta <- matrix(nrow = n, ncol = number_of_parameters)
  #storing the summary stats across all simulations
  S_i <- NULL
  #S is a matrix(nrow = n*total_fish, ncol = dimS)
  d <- rep(NA, n)# weighted distance
  SummaryStats_sim <- NULL
  w<- w/sum(w) #normalising summary statistics weights

  for (i in 1:n) {
    theta[i,] <- pftn()
    output<-SimGroup_taleap(theta1=theta[i,], fish_sex=
      fishSex, fish_type=Fish_stock, strain=Strain, fish_size=
      fishSize, error=0.01)
    #B-D-C parameter estimates for the parasite-fish groups
    # based on simulated data & simulation realisations
    BDC_estimates_sim<- GW_GMM_BDCestimator(X0=2,
      pop=output$pop_sim, output$alive_sim,
      group=parasite_fish)$BDC_estimates
    #Computing the all summary stats for each group
    SummaryStats_sim[[i]]<-Summary_stats(pop=output$pop_sim,
      alive=output$alive_sim, BDC_estimates=BDC_estimates_sim)
    #combining for all summary stats of parasite-fish
    #groups for each simulation realisation
    SummaryStats_sim_combined<-do.call("rbind",
      SummaryStats_sim[[i]])
    #Combining the observed summaries for the groups
    SummaryStats_obs_combined<- do.call("rbind",
      summaries_obs)

    #Storing weighted distances between summaries
    #of observed and simulated data
    S_i[[i]] <- SummaryStats_sim_combined
    d[i] <- w_distance(S1=S_i[[i]],
      S2=SummaryStats_obs_combined, weight=w)
  }
  # summary stats matrix(nrow = n*total_fish,
  #ncol = dimS)
  S<-do.call("rbind", S_i)
  #returns priors (theta), simulated summaries (S)
  #& distances (d)
  return(list(theta=theta, S=S, d=d))
}
```

F.4(iv): The modified weighted-iterative ABC (with SMC & SIS)

```
#Function to obtain the final posterior distribution iter-
#atively using the ABC() function
Weighted_iterative_ABC<- function(N=500,dimS=17,
fish_total=Total_fish,numCores=numCores,
ABC_time_steps=10){
  # N= total number of samples
  n_cores <- numCores;n<- N/n_cores #Run on n cores
  #number of parameters to be estimated
  number_of_parameters<- 23
  #Storing importance weight for sequential sampling
  import_weights<- NULL
  #Storing weights corresponding to accepted samples
  w_accepted<- NULL
  #saving number of particles for each iteration
  dim_tha_post<-NULL
  #saving summaries of all fish for each simulation
  S_i <- NULL
  #ABC_time_steps= time for the algorithm to terminate
  eps<-NULL # storage for index of accepted particles
  #proportion of sample to retain during SIS
  if(N<1000){
    epsilon<- c(0.5,0.43,0.4,0.35,0.3,
0.2,0.1,0.08,0.06,0.02)
  }else if(N>=1000){
    epsilon<-c(0.5,0.3,0.2,0.1,0.08,
0.07,0.06,0.03,0.02,0.01)
  }
  d_i<-NULL;d<-NULL#storing weighted distances
  #For storing parameter values at time t
  theta_i<- NULL;theta<-NULL
  # for density plots (256 used here is
  #the number of equally spaced points
  #at which the density is to be estimated)
  #range of prior distribution (on log scale)
  x <- seq(from = -10, to = 7, length.out = 256)
  fx <- array(dim=c(ABC_time_steps+1,
number_of_parameters, 256))
  time0<- proc.time()
  for (t in 1:ABC_time_steps) {
    cat("ABC_time_steps", t, "\n")
    if (t == 1) {
      pftn <- prior
      ABC_out <- mclapply(1:n_cores, ABC,
pftn=pftn, n=n, w=w, mc.cores=n_cores)
      for (i in 1:n_cores) {
        theta_i[[i]] <- ABC_out[[i]]$theta
        S_i[[i]] <- ABC_out[[i]]$S
        d_i[[i]] <- ABC_out[[i]]$d
      }
    }else{#if t>1
      #Calculate optimal MVN kernel bandwidth matrix
      #parameter values for a new proposal sample
      #N0= number of accepted particles
      #N1= total number of proposal samples
      +++
    }
  }
}
```

```

#Calculate optimal MVN kernel bandwidth matrix H
# denoted by Sigma_optimal_t
#eps[[t]]= index of accepted samples
#w_accepted[[t]]= weights of accepted particles
#tha_post= accepted proposals at time t
Sigma_optimal_t<- matrix(0,nrow =number_of_parameters,
ncol=number_of_parameters)
N1<-dim(theta[[t-1]])[1]
N0<- dim(tha_post)[1]
for(i in 1:N1) {
  for(k in 1:N0){
    Sigma_optimal_t<- Sigma_optimal_t+
      (import_weights[[t-1]][i]*w_accepted[[t-1]][k]
      *(matrix(tha_post[k,]-theta[[t-1]][i, ])
      %*%t(matrix(tha_post[k,]-theta[[t-1]][i, ])))
  }
}

#Sampling from MVN Perturbation kernel
weight<-w_accepted[[t-1]]
pftn <- function() post(tha_post,weight,
Sigma_optimal_t)
ABC_out <- mclapply(1:n_cores, ABC,
pftn=pftn, n=n, w=w, mc.cores=n_cores)
for (i in 1:n_cores) {
  theta_i[[i]] <- ABC_out[[i]]$theta
  S_i[[i]] <- ABC_out[[i]]$S
  d_i[[i]] <- ABC_out[[i]]$d
}
#Combining theta at time t
theta[[t]]<- as.matrix(na.zero(
do.call("rbind",theta_i)))#N by 23 matrix
#Re-weighting for importance sampling
import_weights[[t]]<-rep(NA,
length=dim(theta[[t]))[1])

#Evaluating the perturbation kernel
#for each particle at time t
dMVN_func<- function(i)
mvtnorm::dmvnorm(x=theta[[t]][i, ],
mean = theta[[t-1]][i, ],sigma =Sigma_optimal_t)
K_normal_kernel<- mclapply(1:dim(theta[[t]))
[1],dMVN_func,mc.cores=n_cores)
#### KDE of proposal distn####
#Estimating the optimal bandwidth
density_proposals<- matrix(NA,
nrow=length(import_weights[[t]])
,ncol=number_of_parameters)
N1<-length(unlist(K_normal_kernel))

+++

```

```

for(i in seq_along(import_weights[[t]])){
  density_proposals[i, ] <-
    ks::kde(x = theta[[t]][i, ], eval.points =
      theta[[t]][i, ]) $estimate
  #KDE value for each proposal sample
  par.weight.numerator <- mean(density_proposals[i, ])
  par.weight.denominator <- sum(import_weights[[t-1]]
    [1:N1]*unlist(K_normal_kernel))
  import_weights[[t]][i] <- par.weight.numerator /
    par.weight.denominator
}
#normalizing weights
import_weights[[t]] <- import_weights[[t]] /
  sum(import_weights[[t]])
}
#Combining results from the ncores
# N by 23 matrix
theta[[t]] <- as.matrix(
  na.zero(do.call("rbind", theta_i))) #N by 23 matrix
d[[t]] <- na.zeros(do.call("c", d_i)) #length of N
#number of draw for posterior samples
small_draws <- epsilon[t]*N
#adding the computed distance as extra column of theta
theta_dist <- cbind(theta[[t]], d[[t]])
#smallest distance index
eps[[t]] <- order(theta_dist[, 24])[1:small_draws]

# choose posterior samples
tha_post <- theta_dist[eps[[t]], , -24]
dim_tha_post[[t]] <- dim(tha_post)[1]
#initialize importance weight for sequential sampling
if(t==1) import_weights[[1]] <- rep(1/N, length=N)

#Weights corresponding to accepted proposal samples
w_accepted[[t]] <- import_weights[[t]][eps[[t]]]
w_accepted[[t]] <- w_accepted[[t]] /
  sum(w_accepted[[t]]) #normalising accepted weights

# update summary statistics weights
#max least distance
eps_dist_max <- sort(d[[t]])[small_draws]
#combining the summaries[(N*fish_total) by 17 matrix]
S <- na.omit(do.call("rbind", S_i))
wlinv <- apply(S[rep(d[[t]], fish_total) <= eps_dist_max, ],
  2, var, na.rm = TRUE)
w <- na.zero(2 / (1 / w + wlinv))

+++

```

```

# densities
if (t == 1) {
  for (k in 1:number_of_parameters){
    fx[1,k,]<- density(theta[[1]][ , k], from=-10, to=7, n=256)$y
    #saving the densities for each iteration
    write.csv(fx[1, ,], file = paste0("density_post_", 1, ".csv"))
  }
}
for (k in 1:number_of_parameters){
  fx[t+1,k,]<- density(tha_post[, k], from=-10, to=7, n=256)$y
  #saving the densities for each iteration
  write.csv( fx[t+1, ,], file = paste0("density_post_", t+1, ".csv"))
}
###saving importance weights
write.csv(import_weights[[t]],
file = paste0("importance_weights_",t, ".csv"))
#accepted particles at each iteration
write.csv(tha_post, file = paste0("theta_post_", t, ".csv"))
#saving weighted distance
write.csv(d[[t]], file = paste0("weighted_distance_", t, ".csv"))
}
timef<- proc.time()-time0
CPUtime<-sum(as.vector(timef)[-3])
write.csv(CPUtime, file=paste0("CPUtime_", N, ".csv"))
#Returns estimated densities & final posterior
return(list(fx=fx, final_posterior=tha_post))
} #end of the weighted-iterative ABC algorithm

```

```

#External functions in the posterior adjustment func.

#Gaussian kernel with bandwidth delta
guass_kernel<- function(dist,delta){
  #bandwidth=delta for regression adjustment is optimally determined using
  #the kedd package
  kern<-(sqrt(2*pi*delta))*exp(-(dist^2)/(2* delta^2))
  return(kern)
}

#To deal with any possible unknown irregularity
na.inf.zero<- function(x){
  x[is.na(x)|is.finite(x)==FALSE]<- 0
  return(x)
}

```

F.4(v): Function for proposed ABC post-processing analysis

```
#Function for the modified local-linear regression
# based on weighted ridge regression
require("kedd")
Post_Ridge_reg_adj<- function(post_distn,summary_obs){
  #k=biassing parameter or penalty parameter
  # post_dtn is the posterior sample
  # w are summary statistics weights
  #storing the summary stats across simulations
  S_i <- NULL
  no_of_parameters<- 23
  #storing adjusted posterior means
  posterior_mean_adj<- rep(NA,no_of_parameters)
  #Combining the summary stats for
  # the observed data for the parasite-fish groups
  SummaryStats_obs_combined<- do.call("rbind",
    summaries_obs)
  m<- dim(post_distn)[1]# m=number of posterior samples
  d<- rep(0, m)#weighted distances given observed data
  p<- dim(summary_obs)[2] #dimension of summary statistics
  Unadj_dist<- post_distn
  SummaryStats_sim <- NULL
  X_Design_matrix<- matrix(NA,ncol=p,nrow=m)#design matrix
  # Weights based on Gaussian kernel
  #for local-linear regression adjustment
  W<- matrix(0, ncol=m,nrow=m)
  #saving weighted column means of design matrix
  X_bar=numeric(length=p)
  for (i in 1:m) {
    theta<- as.vector(unlist(post_distn[i,]))
    output_sim<- SimGroup_taleap(theta1=theta,
      fish_sex=fishSex,fish_type=Fish_stock,
      strain=Strain,fish_size=fishSize,error=0.01)
    #B-D-C parameter estimates for the
    #parasite-fish groups based on simulated data
    #for each simulation realisation
    BDC_estimates_sim<- GW_GMM_BDCestimator(X0=2,
      pop=output_sim$pop_sim,
      output_sim$alive_sim,
      group=parasite_fish)$BDC_estimates

    #Computing the summary stats for each
    #group simulation realisation
    SummaryStats_sim[[i]] <- Summary_stats(
      pop=output_sim$pop_sim,alive=output_sim$alive_sim,
      BDC_estimates=BDC_estimates_sim)

    #combining for all summary stats of
    #parasite-fish groups for each simulation realisation
    SummaryStats_sim_combined<-do.call("rbind",
      SummaryStats_sim[[i]])
    mean_diff<- apply(SummaryStats_sim_combined-
      summary_obs,2,mean,na.rm = TRUE)
  }
  +++
```



```

#storing each row of design matrix X
X_Design_matrix[i, ] <- mean_diff

# Computing weights based on
#Storing weighted distances between summaries of observ-
#ed and simulated data)
S_i[[i]] <- SummaryStats_sim_combined
#Updating summary statistics weights
w <- apply(S_i[[i]], 2, var, na.rm = TRUE)
w <- w/sum(w) #normalising summary weights
d[i] <- w_distance(S1=S_i[[i]],
S2=summary_obs, weight=w)
}

distances<-na.inf.zero(d)
#Adaptively choosing the bandwidth
#of the Gaussian kernel based on the distances
bandwidth<- kedd::h.amise(x=distances,
deriv.order =0, kernel = c("gaussian"))$h
diag(W) <- guass_kernel(dist= distances, delta=bandwidth)
theta_post<- as.matrix(post_distn)
# (normalising) main diagonal of Weighting matrix
weights<- diag(W)/sum(diag(W))

#Transforming X and Y (posterior distribution
#and summary statistics)
for(j in seq_along(posterior_mean_adj)){
  #For each jth model parameter, j=1,2,...23
  X<- X_Design_matrix
  Y<- theta_post[,j]

  #Step 1 (Mean centring X and Y)
  for (k in 1:p) X_bar[k]<- sum(weights*X[,k])

  for (k in 1:p) {
    X[, k]<- X[, k]-X_bar[k]
  }
  #finding the weighted mean of Y and mean centring
  Y_bar <- sum(weights*Y)
  Y<- Y- Y_bar

+++

```

```

#Step 2: scaling (centred X and Y) by weights

for(k in 1:p) X[, k] <- sqrt(weights)*X[, k]
Y<- sqrt(weights)*Y

#Choose optimal value of k (the penalty paramters)
# Using cross validation glmnet
# Setting the range of lambda values
options(warn = -1)
lambda_seq <- 10^seq(2, -2, by = -.1)
ridge_cv <- cv.glmnet(X, Y, alpha = 0,
lambda =lambda_seq)
# Best lambda value
best_lambda <- ridge_cv$lambda.min
k<-best_lambda
# calculate beta estimates corresponding
#to summary statistics X (standardised coefficients)
beta_ridge_std <- solve(t(X) %*%W%*X+
k*diag(p)) %*% t(X)%*%W%*Y

# calculating beta estimates of predictors
beta_ridge <- solve(t(X) %*%W%*X+ k*diag(p))
%*% t(X)%*%W%*Y

#calculate intercept estimates (adjusted posterior mean)
posterior_mean_adj[j]<- exp(Y_bar - X_bar%*%beta_ridge)

#Adjusting the posterior distribution
Unadj_dist[, j]<- post_distn[, j]-X_Design_matrix
%*%beta_ridge
}

posterior_mean_uadj<- exp(apply(post_distn,2,mean))
Posterior_mean_output<- data.frame(Adj_posterior_mean=
posterior_mean_adj,Uadj_posterior_mean=
posterior_mean_uadj)

#returns the design data matrix, adjusted & unadjusted
# means, and the adjusted posterior distribution
return(list(X_Design_matrix=X,
Posterior_mean_output=Posterior_mean_output,
Adjusted_posterior_dist=Unadj_dist))
}

```

Appendix G: R Codes for the novel individual-based simulation model

G.1: Description of state variables and simulation parameters

```
## 1. State variables ##

# A[j,k] gives the number of parasites at location j, age k, where
#   j = 1 for Tail population
#   j = 2 for Lower region population
#   j = 3 for Upper region population
#   j = 4 for head population
#   k = 1 for young parasites (yet to give birth)
#   k = 2 for old parasites (have given birth)

# B[j]=immune response at location j (1 for no response; 2 for a response)

# X = state of fish (1 for alive; 2 for dead)

## 2. Base simulation parameters ##

#b1[k,e1]= birth rate for parasites age k,when immune state is e1(for Gt3)
#b2[k,e1]= birth rate for parasites age k,when immune state is e1(for Gt)
#b3[k,e1]= birth rate for parasites age k,when immune state is e1(for Gb)
#d1[k,e1]= death rate for parasites age k,when immune state is e1(for Gt3)
#d2[k,e1]=death rate for parasites age k,when immune state is e1(for Gt)
# d3[k,e1]=death rate for parasites age k,when immune state is e1(for Gb)
#m[k,e1] = movement rate for parasites age k, when immune state is e1
# e = the adjustment to the movement rate for forward/backward movement
# r = rate a single parasite increases immune state (base rate)
# kappa = effective carrying capacity per unit area of each body region
# s = rate a single parasite causes fish mortality

## 3. Additional simulation parameters ##

# r1 = immune response rate adjustment for LA fish (ref: UA fish)
# r2 = immune response rate adjustment for OS fish (ref: UA fish)
# r3 = immune response rate adjustment for male fish (ref: female fish)
# e1, e2, e3 = movement rate adjustment depending on
parasite type (Gt3,Gt, Gb respectively)
#s = must depend on total parasite numbers, fish sex and fish size
#s1 = host mortality rate with adjustment for male fish (ref: female)

## 4. Experiment descriptors ##

# fish_type (1 for UA, 2 for LA & 3 for OS)
# Parasite type (Gt3, Gt & Gb)
# fish_sex (1 for female fish & 2 for male fish)
# f= area of each body part (depends on size and gender)
# a= fish size

#Function to convert NA's to 0 where necessary
na.zero<-function(x){
  x[is.na(x)]<-0
  return(x)
}

#Loading packages (R packages to install)
library(transport)#for Wasserstein distance computation
library(parallel)# for parallizing R codes
RNGkind("L'Ecuyer-CMRG")#Dealing with distinct seed numbers
library(compiler)# byte code compilation
library("maxLik")#for MLE/optimization
library("R.utils")
```

G.2: Function for computing event rates

```
# Function for computing rates based on fish sex, fish type and parasite
#strain
compute_rates_func<- function(A, B, b1,b2,b3, d1,d2,d3, m,
r,r1,r2,r3,s,s1,e1,e2,e3,
kappa,f,a,fish_sex,fish_type,strain){

  #Matrix of immune rates (additive effect of covariates)
  r_matrix<- matrix(c(r,r+r1,r+r2,r+r3,r+r1+r3,r+r2+r3),nrow=2,ncol=3,
    byrow=T)
#r_selected=selected rate based on adjustments(adj) for fish sex&fish type
#selecting the immune response rate depending on fish sex and fish type
if(fish_sex=="F"& fish_type=="UA"){r_selected<- r_matrix[1,1]}#base rate
if(fish_sex=="F"& fish_type=="LA"){r_selected<-r_matrix[1,2]}#adj for LA
if(fish_sex=="F"& fish_type=="OS"){r_selected<-r_matrix[1,3]}#adj for OS
if(fish_sex=="M"& fish_type=="UA"){r_selected<-r_matrix[2,1]}#adj for M
if(fish_sex=="M"& fish_type=="LA"){r_selected<-r_matrix[2,2]}#adj for M&LA
if(fish_sex=="M"& fish_type=="OS"){r_selected<-r_matrix[2,3]}#adj for M&OS

  # selecting which host mortality rate & body areas given fish sex
  if(fish_sex=="F"){
    s_selected<- s #base host mortality rate
    #f=body_area
    f<-as.vector(f[,1])# body areas for female fish
  }
  if(fish_sex=="M"){
    s_selected<- s+s1 #host mortality rate with adjustment for male fish
    #f=body_area
    f<-as.vector(f[,2])## body areas for male fish
  }
#selecting microhabitat preference rate depending on parasite strain
if(strain=="Gt3"){e_selected<- e1}
if(strain=="Gt"){e_selected<- e2}
if(strain=="Gb"){e_selected<- e3}
#selecting birth and deaths rates depending on parasite strain
if(strain=="Gt3"){b_selected<-b1; d_selected<-d1}
if(strain=="Gt"){b_selected<- b2; d_selected<- d2}
if(strain=="Gb"){b_selected<- b3; d_selected<- d3}

  # birth rates; death rates; movement rates;immune response
QB<-matrix(0, 4, 2) # QB[k,j]=birth rate for parasites location j age k
QD<-matrix(0, 4, 2) # QD[k,j]=death rate for parasites location j age k
QM_forward <- matrix(0, 4, 2) # QM[k,j] = movement rate for j age k
QM_backward <- matrix(0, 4, 2)
QI <-rep(0, 4) #QI[j]=rate at which location j increases immune response
for (j in 1:4) {
  QI[j] <- sum(A[j, ]) * r_selected
  for (k in 1:2) {
    QB[j, k] <- A[j, k] * (1-(A[j, k]/(f[j]*a*kappa)))*b_selected[k, B[j]]
    QD[j, k] <- A[j, k] * (1-(A[j, k]/(f[j]*a*kappa)))*d_selected[k, B[j]]
    QM_forward[j, k] <- A[j, k] *m[k, B[j]]*e_selected
    QM_backward[j, k] <- A[j, k] *m[k, B[j]]*(1-e_selected)
  }
}
# total rates
laB <- sum(QB) #total birth rate
laD <- sum(QD) #total death rate
laM_forward <- sum(QM_forward) # total rate for forward movement
laM_backward <- sum(QM_backward) # total rate for backward movement
laI <- sum(QI)# total rate of immune response
laX <- sum(A) * s_selected # host fish mortality rate
#overall total
la <- na.zero(abs(laB + laD + laM_forward+laM_backward+ laI + laX))

#Returns rates in relation to birth, death, movement,
# immune response, host mortality and total rate (la)
return(list(laB=laB,laD=laD,laM_forward=laM_forward,
laM_backward=laM_backward,laI=laI,laX=laX,
la=la,QB=QB,QD=QD,QM_forward=QM_forward,
QM_backward=QM_backward,QI=QI))
}
```

G.3: Function for extracting parasite numbers & experimental descriptors of the empirical data

```
Experiment_descriptors<-function(empirical_data){
  ###Fish-parasite combinations/groups###
  parasite_fish<- c("Gt3-OS", "Gt3-LA", "Gt3-UA", "Gt-OS", "Gt-LA", "Gt-UA", "Gb-OS", "Gb-LA", "Gb-UA") #groups
  levels(empirical_data$Sex_fish)<-c("F", "M")

  empirical_data$LowerRegion<-empirical_data$LB+empirical_data$Pelvic+
  empirical_data$Anal+empirical_data$Dorsal
  empirical_data$UpperRegion<-empirical_data$UB +Combined_data$Pectoral

  ### Data across the four recategorized body regions###
  Data_fourRegions<-empirical_data[, c(1,15,16,8,9,10,12,13,11,14)]
  #head(Data_fourRegions, n=4)

  ###Data across parasite strains###
  Gt3_data<-Data_fourRegions[Data_fourRegions$Parasite_strain=="Gt3",]
  Gt_data<-Data_fourRegions[Data_fourRegions$Parasite_strain=="Gt",]
  Gb_data<-Data_fourRegions[Data_fourRegions$Parasite_strain=="Gb",]

  #To store extracted information
  Parasite_fish_data=NULL; fishID=NULL;
  numF=NULL; pop_obs=NULL; alive_obs=NULL;
  fishSize=NULL; fishSex=NULL; Size=NULL; Sex=NULL; Parasite_strain=NULL; Strain=NULL;
  Fish_type=NULL; Fish_stock=NULL

  ##Data of each parasite strain across fish stocks ##
  Parasite_fish_data[[parasite_fish[1]]] <-split(Gt3_data, Gt3_data$Fish_strain)$"OS"
  Parasite_fish_data[[parasite_fish[2]]] <-split(Gt3_data, Gt3_data$Fish_strain)$"LA"
  Parasite_fish_data[[parasite_fish[3]]] <-split(Gt3_data, Gt3_data$Fish_strain)$"UA"
  Parasite_fish_data[[parasite_fish[4]]] <-split(Gt_data, Gt_data$Fish_strain)$"OS"
  Parasite_fish_data[[parasite_fish[5]]] <-split(Gt_data, Gt_data$Fish_strain)$"LA"
  Parasite_fish_data[[parasite_fish[6]]] <-split(Gt_data, Gt_data$Fish_strain)$"UA"
  Parasite_fish_data[[parasite_fish[7]]] <-split(Gb_data, Gb_data$Fish_strain)$"OS"
  Parasite_fish_data[[parasite_fish[8]]] <-split(Gb_data, Gb_data$Fish_strain)$"LA"
  Parasite_fish_data[[parasite_fish[9]]] <-split(Gb_data, Gb_data$Fish_strain)$"UA"

  for (pf in 1:length(parasite_fish)){
    #Assigning unique ID for data
    fishID[[pf]]<- unique(Parasite_fish_data[[parasite_fish[pf]]]$Fish_ID)
    #Total number of fish used for data
    numF[[pf]] <- length(fishID[[pf]])
    #Observed data or matrix across 4 regions
    pop_obs[[pf]] <- array(dim = c(numF[[pf]], 4, 9))
    #Array for time steps fish was alive for each combination
    alive_obs[[pf]] <- array(dim = c(numF[[pf]], 9))
    #NB: Fish size & sex over time
    #Array of fish size across the 9 time steps for each combination
    Size[[pf]]<- array(dim = c(numF[[pf]], 9))
    Sex[[pf]]<- array(dim = c(numF[[pf]], 9))
    Parasite_strain[[pf]]<- array(dim = c(numF[[pf]], 9))
    Fish_type[[pf]]<- array(dim = c(numF[[pf]], 9))

    for(i in 1:numF[[pf]]){
      pop_obs[[pf]][i,,] <-
      t(Parasite_fish_data[[parasite_fish[pf]]]
      [Parasite_fish_data[[parasite_fish[pf]]
      $Fish_ID==fishID[[pf]][i], 1:4])
      alive_obs[[pf]][i, ] <-ifelse(is.na(pop_obs[[pf]][i,1,]), 2, 1)
      Size[[pf]][i, ]<-
      Parasite_fish_data[[parasite_fish[pf]]]
      [Parasite_fish_data[[parasite_fish[pf]]
      $Fish_ID==fishID[[pf]][i], 9]
      #1=Female fish & 2=Male fish
      Sex[[pf]][i, ]<-
      paste(Parasite_fish_data[[parasite_fish[pf]]]
      [Parasite_fish_data[[parasite_fish[pf]]
      $Fish_ID==fishID[[pf]][i], 10]
      Parasite_strain[[pf]][i, ]<-paste(Parasite_fish_data[[parasite_fish[pf]]]
      [Parasite_fish_data[[parasite_fish[pf]]
      $Fish_ID==fishID[[pf]][i], 7])
      Fish_type[[pf]][i, ]<-
      paste(Parasite_fish_data[[parasite_fish[pf]]]
      [Parasite_fish_data[[parasite_fish[pf]]
      $Fish_ID==fishID[[pf]][i], 8])
    }

    ### Experiment descriptors ###
    fishSize[[pf]]<-apply(Size[[pf]],1,unique)
    fishSex[[pf]]<- apply(Sex[[pf]],1,unique)
    Strain[[pf]]<- apply(Parasite_strain[[pf]],1,unique)
    Fish_stock[[pf]]<- apply(Fish_type[[pf]],1,unique)
  }

  #return data on experiment descriptors (fish size, sex,
  #fish type & strain) for each parasite-fish group
  return(list(fishSize=fishSize, fishSex=fishSex,
  Strain=Strain, Fish_stock=Fish_stock, numF=
  numF, fishID=fishID, pop_obs=pop_obs,
  alive_obs=alive_obs, fishID=fishID))
}
```

G.4: Function for updating exact SSA

```
# Function for updating exact SSA
#For for updating simulation events across the 4 body regions
(Tail, Lower region, Upper region, Head)

SSA_update_event <- function(A,B,J,X,laB,laD,laM_forward,
laM_backward,laI,laX,la,QB,QD,QM_forward,QM_backward,QI) {

  #Inputs:
  # A[j,k] gives the number of parasites at location j, age k, where
  #B[j]= immune response at body location j(1 for no response;2 for a response)
  #J is transition matrix
  #X is survival status (1= alive, 2=dead)
  #And all rates in relation to birth,
  #death, movement, immune response, host mortality and total rate (la)

  if (la == 0) {
    return(list(A = A, B = B, t_incr_SSA = Inf, X = X)) # zero population
  }

  U <- runif(1, 0, la) #uniform random number/generator

  if (U < laB) {# birth
    i <- sample(8, 1, prob = abs(QB))
    j <- ((i-1) %% 4) + 1 # location
    k <- ((i-1) %/% 4) + 1 # age
    if (k == 1) {
      A[j, 2] <- A[j, 2] + 1
    } else {
      A[j, 1] <- A[j, 1] + 1
    }
  } else if (U < sum(c(laB,laD))) {# death
    i <- sample(8, 1, prob = abs(QD))
    j <- ((i-1) %% 4) + 1 # location
    k <- ((i-1) %/% 4) + 1 # age
    A[j, k] <- A[j, k] - 1
  } else if(U < sum(c(laB,laD,laM_forward))){#forward movement

    i <- sample(8, 1, prob = abs(QM_forward))
    j <- ((i-1) %% 4) + 1 # location
    k <- ((i-1) %/% 4) + 1 # age
    j_new <- sample(4, 1, prob =abs(J[j,]))#new location
    A[j, k] <- A[j, k] - 1
    A[j_new, k] <- A[j_new, k] + 1
  } else if (U < sum(c(laX,laI,laB,laD,laM_forward)) ) {#backward movement
    i <- sample(8, 1, prob = abs(QM_backward))
    j <- ((i-1) %% 4) + 1 #location
    k <- ((i-1) %/% 4) + 1 # age
    j_new <- sample(4, 1, prob =abs(J[j,]))#new location
    A[j, k] <- A[j, k] - 1
    A[j_new, k] <- A[j_new, k] + 1
  }else if(U<sum(c(laB,laD,laM_forward,laM_backward,laI))){#immune response
    i <- sample(4, 1, prob = abs(QI))
    B[i] <- 2
  } else {# fish death
    X <- 2
  }

  t_incr_SSA <- rexp(1, la) #time increment for exact SSA
  #Output: returns A[j,k] the number of parasites
  # at location j, age k
  # where B[j] = immune response at location j
  #(1 for no response; 2 for a response)
  # t_incr_SSA= time increment for exact SSA
  # X survival status
  return(list(A = A, B = B,t_incr_SSA=t_incr_SSA, X = X))
}
```

G.5: Function for updating hybrid τ -leaping

```
#Function for updating tau-leaping
taulealping_update_event <-function(A,B,J,X,laB,laD,
laM_forward,laM_backward,laI,laX,la
,QB,QD,QM_forward,QM_backward,QI,tau){

  #Inputs:
  # A[j,k] gives the number of parasites at location j, age k, where
  # B[j]=immune response at body location j(1 for no response;2 for a response)
  # J is transition matrix
  # X is survival status (1= alive, 2=dead)
  # tau is the leap size
  # And all rates in relation to birth, death, movement, immune response, host
  # mortality and total rate (la)

  U <- runif(1, 0, la)
  if(U<laX) X <-2 #Fish mortality
  else if(U<sum(c(laX,laI))){#Immune response

    j <- sample(4, 1, prob = abs(QI))
    B[j] <- 2
  } else if (U<sum(c(laX,laI,laB,laD,laM_forward))){
    #birth, death or forward movement
    i <- sample(8, 1, prob = abs(QB+ QD+QM_forward))
    j <- ((i-1) %% 4) + 1 # current location
    k <- ((i-1) %/% 4) + 1 # age
    j_new <- sample(4, 1, prob =abs(J[j,]))# new location
    A[j,k]<- A[j,k] + rpois(1,abs(laB*tau))-
      rpois(1,abs(laD*tau))
    -rpois(1,abs(laM_forward*tau))
    A[j_new,k]<- A[j_new,k]
    +rpois(1,abs(laB*tau))
    -rpois(1,abs(laD*tau))
    +rpois(1,abs(laM_forward*tau))
  } else if (U< sum(c(laX,laI,laB,laD,laM_forward,laM_backward))){
    #birth, death or backward movement
    i <- sample(8, 1, prob = abs(QB+ QD+QM_backward))
    j <- ((i-1) %% 4) + 1 # current location
    k <- ((i-1) %/% 4) + 1 # age
    j_new <- sample(4, 1, prob =abs(J[j,]))# new location
    A[j,k]<- A[j,k]+rpois(1,abs(laB*tau))-
      rpois(1,abs(laD*tau))-
      rpois(1,abs(laM_backward*tau))
    A[j_new,k]<-A[j_new,k]
    +rpois(1,abs(laB*tau))-rpois(1,abs(laD*tau))
    +rpois(1,abs(laM_backward*tau))
  }

  #Output: returns A[j,k] gives the number of parasites at location j,age k,
  #where B[j] = immune response at location j
  #(1 for no response; 2 for a response)
  # X=survival status
  return(list(A = A, B = B, X = X))

}
```

G.6: Function for simulating infection dynamics for a single fish

```
#tau-leaping simulation for a single fish
sim_tauleap_singlefish<- function(A0, B0,J, b1,b2,b3,
d1,d2,d3, m, r,r1,
r2,r3,s,s1,e1,e2,e3,kappa,
f,a,fish_sex,fish_type,strain,error){
  #Inputs:  inital conditions,  parameter values,  fish sex,
  #fish type,  parasite strain and error bound
  #f=body area (dependent on fish sex and size)
  #parasite_fish=c("Gt3-OS","Gt3-LA","Gt3-UA",
  #"Gt-OS","Gt-LA","Gt-UA","Gb-OS","Gb-LA","Gb-UA")
  #strain-parasite type to be simulated
  parasite_fish<-paste(strain,"-",fish_type)
  pop=NULL; alive =NULL; exploded=NULL;Leap_sizes=NULL;
  A<-A0; B<- B0
  #observed discrete times
  save_ti <- c(1, 3, 5, 7, 9, 11, 13, 15, 17)
  save_TF <- rep(FALSE, length(save_ti))
  ti<- 0 # initial time
  #parasite pop at each location (rows) & timepoint (cols)
  pop[[parasite_fish]] <- matrix(NA, 4, length(save_ti))
  # host fish status at each time point
  alive[[parasite_fish]] <- rep(2, length(save_ti))
  pop_ti <- rowSums(A)
  # host survival status (alive=1; dead=2)
  alive_ti <- 1
  exploded[[parasite_fish]] <- FALSE
  # stop the simulation if total population>pop_max
  pop_max <- 10000
  X <- 1 # fish starts out alive
  while(sum(save_TF) < length(save_ti)){
    ##### Computing the rates #####
    computed_rates<-compute_rates(A=A, B=B,b1=b1,b2=b2,
    b3=b3, d1=d1,d2=d2,d3=d3,m=m,r=r,r1=r1,r2=r2,r3=r3
    ,s=s,s1=s1,e1=e1, e2=e2,e3=e3,kappa=kappa,f=f,a=a,
    fish_sex=fish_sex,fish_type=fish_type,strain=strain)
    laB<-computed_rates$laB
    laD<-computed_rates$laD
    laM_forward<-computed_rates$laM_forward
    laM_backward<-computed_rates$backward
    laI<-computed_rates$laI
    laX<-computed_rates$laX
    la<-computed_rates$la
    QB<-computed_rates$QB
    QD<-computed_rates$QD
    QM_forward<-computed_rates$QM_forward
    QM_backward<-computed_rates$QM_backward
    QI<-computed_rates$QI

    ##sim_tauleap_singlefish function continues at next page##
    +++++
  }
```



```

#sim_tauleap_singlefish function continuation

# Determining the switching condition between the exact SSA & the Tau-
#leaping algorithm
#selecting birth and deaths rates
#depending on parasite strain for leap size
if(strain=="Gt3"){b_selected<-b1; d_selected<-d1}
if(strain=="Gt"){b_selected<- b2; d_selected<-d2}
if(strain=="Gb"){b_selected<- b3; d_selected<-d3}
#finding average birth & death rates (eqn 6.1)
b_avg<- mean(b_selected[,1]);d_avg<-mean(d_selected[, 1])
Leap_sizes[[1]]<-(error*(b_avg+d_avg))/
(abs(b_avg-d_avg)*max(b_avg,d_avg))
Leap_sizes[[2]]<- sum(A)*(error*(b_avg+d_avg))^2
/((b_avg+d_avg)* max(b_avg^2,d_avg^2))
# the leap size: time increment for tau-leaping
tau<- na.zero(min(Leap_sizes[[1]],Leap_sizes[[2]]))
leap_condition<- na.zero((1/(10*la)))
if (sum(pop_ti) > pop_max) {
  exploded[[parasite_fish]] <- TRUE
  break }
if (alive_ti == 2) break
#Running tau-leaping if tau >leap_condition
if(tau >leap_condition){ #Execute tau-leaping
  out<-tauleaping_update_event(A=A,
  B=B,J=J,X=X,laB=laB,laD=laD,laM_forward=laM_forward,
  laM_backward=laM_backward,laI=laI,laX=laX,
  la=la,QB=QB,QD=QD,QM_forward=QM_forward,
  QM_backward=QM_backward,QI=QI,tau=tau)
  X<- out$X;A<- out$A;B<- out$B;
  time_increment=tau
} #end of tau-leaping
else if(tau <=leap_condition){
  #Execute exact SSA if tau <=leap_condition
  out <- SSA_update_event(A=A,B=B,J=J,X=X,
  laB=laB,laD=laD,laM_forward=laM_forward,
  laM_backward=laM_backward,laI=laI,laX=laX,
  la=la,QB=QB,QD=QD,QM_forward=QM_forward,
  QM_backward=QM_backward,QI=QI)
  #time increment for SSA
  time_increment<- out$t_tincr_SSA
  X<- out$X; A<- out$A;B<- out$B
} #end of exact SSA
ti <- ti +time_increment #updating time ti
save_new <- which((ti >= save_ti) & !save_TF)
for (i in save_new) {
  pop[[parasite_fish]][,i] <- pop_ti
  alive[[parasite_fish]][i] <- alive_ti}
save_TF <- (ti >= save_ti)
pop_ti <- rowSums(A)
alive_ti <- X
#break if parasite number<0 at any body region
if(any(pop_ti<0) ==TRUE) break
}
#Output: returns pop (parasite pop at each location and time)
#alive: survival status of fish
# exploded: explosion status
#(whether parasite numbers>pop_max=10000)
# parasite_fish: the host-parasite group being simulated
return(list(pop = pop[[parasite_fish]],
  alive = alive[[parasite_fish]],
  exploded =exploded[[parasite_fish]],
  parasite_fish=parasite_fish))
}

```

G.7: Exporting external scripts and extracting relevant information from the empirical data for group simulation

```
#tau-leaping simulation for a group of fish
###exporting external scripts###
#Script of function for computing event rates
source("Computing-rates-script.r")
# Script of function for updating exact SSA
source("Update-exactSSA-script.r")
# Script of function for updating tau-leaping
source("Update-tauleaping-script.r")
# Script of function experimental descriptors
# (fish type, strain, fish size, fish sex &
# areas of the 4 body regions)
source("Descriptors-Data-script.r")
# Script of function for simulating parasites
#only a single fish over time and across body regions
source("Simulation-single-fish-script.r")

#Importing empirical data
Combined_data <- read.csv(file="Parasite_Data.csv")

#Importing data for area of the 8
#body parts across 18 fish (measured in mm^2)
Bodyparts_area<- read.csv(file ="Area_Fish_bodyParts.csv")
#Experimental descriptors
Descriptors<- Experiment_descriptors(empirical_data =
Combined_data)
fishSize <- Descriptors$fishSize #fish size
fishSex <- Descriptors$fishSex #fish sex
Strain <- Descriptors$Strain # parasite strain
Fish_stock<-Descriptors$Fish_stock #fish stock
# total fish for each parasite-fish group
numF <- Descriptors$numF
# fish IDs for each parasite-fish group
fishID <- Descriptors$fishID
#observed parasite numbers for each parasite-fish group
pop_obs <- Descriptors$pop_obs
# observed survival status for each parasite-fish group
alive_obs<- Descriptors$alive_obs
#body areas for female (column 1) & male (column 2) fish
Area_normalized<-Body_area(Area_data=Bodyparts_area)

#Initial simulation inputs for A (parasite numbers)
#and B (immune status)
A0 <- matrix(0, 4, 2)
A0[1, 1] <- 2 #Intial parasites at the tail
#initial immune response at 4 body regions
#(1=no response, 2=response)
B0 <- rep(1, 4)
#Transition matrix(between body regions)
J<- matrix(c(0, 1, 0, 0,
1/2, 0, 1/2, 0,
0, 1/2, 0, 1/2,
0, 0, 1, 0), 4, 4, byrow=TRUE)
```

G.8: Function for simulating infection dynamics for a group of fish corresponding to the empirical data

```
#To simulate group of fish for each parasite-fish combination as observed
# in the empirical data

SimGroup_tauleap<-function(thetal,fish_sex,
fish_type,strain,fish_size,error){
  #Inputs: thetal= parameter values from prior distribution
  #fish_sex= sex of fish
  #fish_type= type of fish
  #strain= parasite strain
  #fish_size= fish size
  #error= error bound of tau-leaping
  pop_sim<-NULL; alive_sim<- NULL;
  exploded_sim<-NULL; results<-NULL; group<-NULL

  for(pf in 1:9){
    pop_sim[[pf]]<- array(dim = c(numF[[pf]], 4, 9))
    #Array for time steps fish was alive for each combination
    alive_sim[[pf]]<-array(dim = c(numF[[pf]], 9))
    #Array for time steps parasites>pop_max for each combination
    exploded_sim[[pf]]<-array(dim = c(numF[[pf]], 9))

    for(i in 1:numF[[pf]]){
      results[[pf]]<-sim_tauleap_singlefish(A0=A0,
      B0=B0,J=J,b1=matrix(exp(thetal[1:2]), 2,2),
      b2=matrix(exp(thetal[3:4]), 2, 2),
      b3=matrix(exp(thetal[5:6]), 2,2),
      d1=matrix(exp(thetal[7:8]), 2, 2, byrow=TRUE),
      d2=matrix(exp(thetal[9:10]), 2,2,byrow=TRUE),
      d3=matrix(exp(thetal[11:12]),2, 2,byrow=TRUE),
      m=matrix(exp(thetal[13]), 2,2),
      r=exp(thetal[14]),r1=exp(thetal[15]),
      r2=exp(thetal[16]),r3=exp(thetal[17]),
      s=exp(thetal[18]),s1=exp(thetal[19]),
      e1=exp(thetal[20]),e2=exp(thetal[21]),
      e3=exp(thetal[22]),kappa=exp(thetal[23]),
      f=Area_normalized,a=fish_size[[pf]][i],
      fish_sex=fish_sex[[pf]][i],
      fish_type=fish_type[[pf]][i],
      strain=strain[[pf]][i],error=error)
      pop_sim[[pf]][i, ]<- results[[pf]]$pop
      alive_sim[[pf]][i, ]<- results[[pf]]$alive
      exploded_sim[[pf]][i, ]<- results[[pf]]$exploded
      group[[pf]]<-results[[pf]]$parasite_fish
    }
  }
  #Output: returns
  #(pop_sim=parasite pop per region and time)
  #alive_sim: survival status of fish
  # exploded_sim: explosion status
  #(whether parasite numbers>pop_max=10000)
  # group: the host-parasite groups being simulated
  return(list(pop_sim=pop_sim,alive_sim=alive_sim,
  exploded_sim=exploded_sim, group= unlist(group)))
}
```

G.9: Function for performing ROPE+HDI Bayesian hypothesis testing

```
# Function to perform Region of Practical Equivalence (ROPE) and Highest
#Density Interval (HDI)
require(bayestestR)
ROPE_Cred_Int<-function(theta_distn_diff, parameter_labels, ci_percent=0.89){

  if(is.list(theta_distn_diff)==FALSE){
    #standard deviation of differenced posterior samples
    sigma_d<- sd(theta_distn_diff)
    output<-bayestestR::equivalence_test(

      #ROPE range recommended by Norman et al (2003)
      theta_distn_diff, range =c(-.5*sigma_d,.5*sigma_d),
      ci = ci_percent, ci_method = "HDI")
    final_output<- cbind(parameter_labels, output)
    names(final_output)[1]<- "Parameter"
    return(final_output)
  }

  #theta_distn_diff=a list of posterior samples of
  # differences of parameters of interest
  output<-list() #save ROPE+HDI results
  for(i in seq_along(parameter_labels)){
    #standard deviation of differenced posterior samples
    sigma_d<- sd(theta_distn_diff[[i]])

    #ROPE range is recommend by Norman et al (2003)
    output[[i]]<- bayestestR::equivalence_test(
      theta_distn_diff[[i]], range =c(-.5*sigma_d,.5*sigma_d),
      ci = ci_percent, ci_method = "HDI")
  }

  #Function returns ROPE interval,
  #ROPE Percentage or coverage probability,
  #ROPE equivalence decision and the corresponding HDI

  final_output<- do.call("rbind", output)
  final_output<- cbind(parameter_labels, final_output)
  names(final_output)[1]<- "Parameters"
  return(final_output)
}
```

Bibliography

1. Twumasi, C., Cable, J., and Jones, O. (in press 2022). Spatial and temporal parasite dynamics: microhabitat preferences and infection progression of two co-infecting gyrodactylids. Preprint available at: <https://doi.org/10.13140/RG.2.2.33220.71044>.