Supplementary material I

Efficient Approximation of Interval Specific Estimates for Biological Growth Curve Models using Localized Likelihood Maximization and Its Applications to Real Data Analysis

Md Aktar Ul Karim[1], Amiya Ranjan Bhowmick[1,*]

1. Null distribution

```
Initial data set up
K = 100
x0 = 10
r = 0.4
t = seq(from = 0, to = 19, by = 1)
q = length(t)
n= 100 # Trajectories

h = numeric(length = length(t)-1)
for(i in 1:(length(t)-1)){
  h[i] = t[i+1] - t[i]
}
time_int=length(t)-2

mu= K*(1 + ((x0/K)^(1/3) - 1)*exp(-r*t))^3  # Von Bertalanffy mean function
```

Change the mean function mu for different growth model.

```
plot(mu)
rep= 1000
# The ISRP estimators derived by Bhowmick et al.,(2014)'s method.
r_hat_vals = matrix(data = NA, nrow = rep, ncol = time_int)

K_hat_vals = matrix(data = NA, nrow = rep, ncol = time_int)

modified_RGR = matrix(data = NA, nrow = rep, ncol = time_int)

sigma2 = 0.1
rho = 0.1
library(mvtnorm) # To generate multivariate normal distribution data
x0_mean_vals = numeric(length = rep)
cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))

for (i in 1:length(t)) {
```

*Corresponding author
Email addresses: mdaktarulkarim829@gmail.com (Md Aktar Ul Karim), amiyaiitb@gmail.com (Amiya Ranjan Bhowmick)
[1]Department of Mathematics, Institute of Chemical Technology, Mumbai

```
  for (j in 1:length(t)) {
    cov_mat[i,j] = sigma2*rho^(abs(i-j))
  }
}

for (i in 1:rep) {
  library(mvtnorm)
  data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat)
  data_mean = colMeans(data)
  x0_mean_vals[i] = mean(data_mean[1])


  for(j in 1:time_int){
    d1= (data_mean[j+1])^(1/3) - (data_mean[j])^(1/3)
    d2= (data_mean[j+2])^(1/3) - (data_mean[j+1])^(1/3)
    d12=d1/d2
    r_hat_vals[i,j]=(3/h[j])*log(d12)
    K_hat_vals[i,j]= ((x0_mean_vals[i])^(1/3)-(d1/(exp(-r_hat_vals[i,j]*t[j]/3)*
                                     (exp(-r_hat_vals[i,j]*h[j]/3)-1))))^3
    modified_RGR[i,j]= (r_hat_vals[i,j]*exp(-r_hat_vals[i,j]*(t[j]/3))*
                        ((K_hat_vals[i,j])^(1/3) - (x0_mean_vals[i])^(1/3)))/
      ((K_hat_vals[i,j])^(1/3) + exp(-r_hat_vals[i,j]*(t[j]/3))*
         ((x0_mean_vals[i])^(1/3)-(K_hat_vals[i,j])^(1/3)))
  }
}

V = matrix(data = NA, nrow = rep, ncol = time_int)

test_stat = matrix(data = NA, nrow = rep, ncol = time_int-2)

for (i in 1:rep) {
  for (j in 1:time_int) {
    V[i,j ] = log((1/modified_RGR[i,j]) + (1/ r_hat_vals[i,j]))
  }

  for (j in 1:(time_int-2)) {
    test_stat[i,j] = V[i,j+2] - 2*V[i,j+1] + V[i,j]
  }
}

par(mfrow = c(1,1))

#View(test_stat)

test_stat_final=na.omit(test_stat) # to delete the replications containing at-least one NaN entries.
View(test_stat_final)

# Histogram for the test statistics
# We have to change the time point to get different test statistics for different time points.
hist( test_stat_final[,11]/sd(test_stat_final[,11]), probability = TRUE, main ='',
        col = "lightgreen", border = "black", xlab = expression(T[t]), breaks = 10, ylim=c(0,0.4))
curve(dnorm(x), add = TRUE, col = "red", lwd = 2, lty = 1)


```

```
# The ISRP estimators of the parameter derived by localized MLE method
n = 100    # number of independent trajectories
data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat)
matplot(t, t(data), type = "l")

# Likelihood function for computing overall estimates that is it will
# give a single estimate of the parameters using the data at all time
# points 1 to q.
fun_likelihood = function(param){
  r = param[1]
  K = param[2]
  sigma2  = param[3]
  rho = param[4]
  mu = K*((1 + ((x0/K)^(1/3) - 1)*exp(-r*t/3))^3)

  cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:q) {
    for (j in 1:q) {
      cov_mat[i,j] = sigma2*rho^(abs(i-j))
    }
  }
  exponent = 0
  for(i in 1:n){
    exponent = exponent + t(data[i,]-mu)%*%(solve(cov_mat))%*%(data[i,]-mu)
  }


  log_lik = - (n*q/2)*log(2*pi) - (n/2)*log(det(cov_mat)) - (1/2)*exponent
  return(-log_lik)
}
param_0 = c(0.4, 100, 3, 0.1)
out = optim(param_0, fun_likelihood, method = "L-BFGS-B",
            hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001), upper = c(3, 150, 10, 0.999))
out$par
solve(out$hessian)
```

The Main programme starts here for ISRP estimates by maximizing localized likelihood. The following program will estimate the ISRP of r, K, σ^2 and ρ by maximizing localized likelihood function that will use only the data for three consecutive time points. Therefore, for q = 20 time points, we will obtain 18 estimates of r and K. Note that we did not provide the ISRP estimate of X_0. We estimated X_0 by taking the average of all the observations at time point $t = 1$.

```
est_local = matrix(data = NA, nrow = q-2, ncol = 4)

fun_local_likelihood = function(param){
  r = param[1]
  K = param[2]
  sigma2  = param[3]
  rho = param[4]
  mu = K*((1 + ((x0/K)^(1/3) - 1)*exp(-r*t/3))^3)

  cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
```

```
  for (i in 1:q) {
    for (j in 1:q) {
      cov_mat[i,j] = sigma2*rho^(abs(i-j))
    }
  }

  # localized estimates
  mu_local = mu[ind:(ind + 2)]
  data_local = data[,ind:(ind + 2)]
  cov_mat_local = cov_mat[ind:(ind+2), ind:(ind+2)]

  exponent = 0
  for(i in 1:n){
    exponent = exponent + t(data_local[i,] - mu_local)%*%(solve(cov_mat_local))%*%(data_local[i,] - mu_local)
  }

  log_lik = - (n*length(ind:(ind+2))/2)*log(2*pi) - (n/2)*log(det(cov_mat_local)) - (1/2)*exponent
  return(-log_lik)
}

# The following is for verification purpose for whether we are able to get the
# ISRP estimates by maximizing the localized likelihood function
for(j in 1:(q-2)){
  ind = j
  param_0 = c(0.4, 100, 3, 0.1)
  out = optim(param_0, fun_local_likelihood, method = "L-BFGS-B",
              hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001),
              upper = c(3, 150, 10, 0.999))
  est_local[j, ] =  out$par
}
colnames(est_local) = c("r", "K", "sigma2", "rho")

par(mfrow = c(2,2))
plot(1:(q-2), est_local[,1], type = "b", col = "red", lwd = 2,
     main = expression(widehat(r(Delta~t))[MLE]),
     ylab = expression(hat(r)))
plot(1:(q-2), est_local[,2], type = "b", col = "red", lwd = 2,
     main = expression(widehat(K(Delta~t))[MLE]),
     ylab = expression(hat(K)))
plot(1:(q-2), est_local[,3], type = "b", col = "red", lwd = 2,
     main = expression(widehat(sigma^2)),
     ylab = expression(hat(sigma^2)))
plot(1:(q-2), est_local[,4], type = "b", col = "red", lwd = 2,
     main = expression(widehat(rho)),
     ylab = expression(hat(rho)))
```

In this section we validate the asymptotic distribution of the test statistic $(T_1, T_2, \ldots, T_{q-4})$ when the null hypothesis is true. The following experiment will be carried for different n values replicate this process for $n = 100$ times and simulation of the distribution of the test statistics under the null hypothesis.

```
n=100
rep = 500
est_local_list = list()
```

```
x0_mean_vals = numeric(length = rep)
for(l in 1:rep){
  set.seed(l)
  print(l)
  data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat)
  x0_mean_vals[l] = mean(data[,1])
  est_local = matrix(data = NA, nrow = q-2, ncol = 4)

  for(j in 1:(q-2)){
    ind = j
    param_0 = c(0.4, 100, 3, 0.1)
    out = optim(param_0, fun_local_likelihood, method = "L-BFGS-B",
                hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001),
                upper = c(3, 150, 10, 0.999))
    est_local[j, ] =  out$par
  }
  colnames(est_local) = c("r", "K", "sigma2", "rho")
  est_local_list[[l]] = est_local
}

# Plot the estimates based on replication
# Distribution of ISRP estimates of r using localized maximum likelihood
mat_est_r_MLE = matrix(data = NA, nrow = q-2, ncol = rep)
for (j in 1:rep) {
  mat_est_r_MLE[,j] = est_local_list[[j]][,1]
}
# Distribution of ISRP estimates of K using localized maximum likelihood
mat_est_K_MLE = matrix(data = NA, nrow = q-2, ncol = rep)
for (j in 1:rep) {
  mat_est_K_MLE[,j] = est_local_list[[j]][,2]
}

# Estimation of modified RGR and its distribution
modified_RGR = matrix(data = NA, nrow = q-2, ncol = rep)
for (i in 1:(q-2)) {
  numerator = mat_est_r_MLE[i,]*exp(-mat_est_r_MLE[i,]*t[i]/3)*((mat_est_K_MLE[i,])^(1/3) - (x0_mean_vals)^(1
  denominator = (mat_est_K_MLE[i,])^(1/3) + exp(-mat_est_r_MLE[i,]*t[i]/3)*((x0_mean_vals)^(1/3) - (mat_est_K
  modified_RGR[i,] = numerator/denominator
}

# Computation of the Test Statistics and its distribution
V = matrix(data = NA, nrow = q-2, ncol = rep)
for (i in 1:(q-2)) {
  V[i, ] = log(1/modified_RGR[i,] + 1/ mat_est_r_MLE[i,])
}
print(head(V))
sum(is.na(V))

test_stat_mle = matrix(data = NA, nrow = q-4, ncol = rep)
for (i in 1:(q-4)) {
  test_stat_mle[i,] = V[i+2,] - 2*V[i+1,] + V[i,]
}
# Path setting
setwd("D:/My Paper/3rd Paper MLE vs ISRP/MLE ISRP/Von_bertalanffy_0-19/sigma = 0.1 _0.4_3")
```

```
write.table(test_stat_mle, file = "VB_Test_STAT_500.txt", row.names = FALSE,
            col.names = FALSE)
# Histogram of the test statistics under the null distribution (required)
par(mfrow = c(1,1))
for(i in 1:(q-4)){

  hist( test_mle_final[i,]/sd(test_mle_final[i,]), probability = TRUE,
        main = '' , xlab = expression(widehat(T[t])), breaks = 10)
  curve(dnorm(x), add = TRUE, col = "red", lwd = 2, lty = 2)
}
```

2. Power curve plotting

```
# The following parameters will be considered as the fixed set up for simulation.
K = 100
x0 = 10
r = 0.4
t = seq(from = 0, to = 19, by = 1)
q = length(t)

h = numeric(length = length(t)-1)
for(i in 1:(length(t)-1)){
  h[i] = t[i+1] - t[i]
}


mu = K*((1 + ((x0/K)^(1/3) - 1)*exp(-r*(t/3)))^3)   # Von Bertalanffy mean function
par(mfrow = c(1,1))
plot(t, mu, col = "red", lwd=2, type = "b", pch = 19,
     ylab = "size, X(t)", xlab = "time (t)")

sigma2 = 0.1   # variance of X_t
rho = 0.1    # corr(X_t, X_{t+1})
cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
for (i in 1:length(t)) {
  for (j in 1:length(t)) {
    cov_mat[i,j] = sigma2*rho^(abs(i-j))
  }
}
n = 100    # number of independent trajectories
data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat)
matplot(t, t(data), type = "l")

abline(v = c(3, 6, 9, 12, 15, 18), lwd = 2, col = "red", lty = 2)
text(4.5, 70, expression(widehat(r(Delta~ t))))
# Likelihood function for computing overall estimates that is it will give a single estimate of the
# parameters using the data at all time points 1 to q.
fun_likelihood = function(param){
  r = param[1]
  K = param[2]
  sigma2  = param[3]
  rho = param[4]
  mu = K*((1 + ((x0/K)^(1/3) - 1)*exp(-r*(t/3)))^3)

```

```
  cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:q) {
    for (j in 1:q) {
      cov_mat[i,j] = sigma2*rho^(abs(i-j))
    }
  }
  exponent = 0
  for(i in 1:n){
    exponent = exponent + t(data[i,]-mu)%*%(solve(cov_mat))%*%(data[i,]-mu)
  }


  log_lik = - (n*q/2)*log(2*pi) - (n/2)*log(det(cov_mat)) - (1/2)*exponent
  return(-log_lik)
}
param_0 = c(0.3, 100, 3, 0.3)
out = optim(param_0, fun_likelihood, method = "L-BFGS-B",
            hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001), upper = c(3, 150, 10, 0.999))
out$par
solve(out$hessian)
```

The Main programme starts here ISRP estimates by maximizing localized likelihood. The following program will estimate the ISRP of r, K, σ^2 and ρ by maximizing localized likelihood function that will use only the data for three consecutive time points. Therefore, for $q = 20$ time points, we will obtain 18 estimates of r and K. Note that we did not provide the ISRP estimate of X_0. We estimated X_0 by taking the average of all the observations at first time point. Here, we consider Von Bertalanffy model as the test bed model. For other model, we need to change the mean (mu) according to the different growth model. Also for different σ^2 and ρ values, we need to set different σ^2 and ρ values.

```
est_local = matrix(data = NA, nrow = q-2, ncol = 4)

fun_local_likelihood = function(param){
  r = param[1]
  K = param[2]
  sigma2  = param[3]
  rho = param[4]
  mu = K*((1 + ((x0/K)^(1/3) - 1)*exp(-r*(t/3)))^3)

  cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:q) {
    for (j in 1:q) {
      cov_mat[i,j] = sigma2*rho^(abs(i-j))
    }
  }

  # localized estimates
  mu_local = mu[ind:(ind + 2)]
  data_local = data[,ind:(ind + 2)]
  cov_mat_local = cov_mat[ind:(ind+2), ind:(ind+2)]

  exponent = 0
```

```
  for(i in 1:n){
    exponent = exponent + t(data_local[i,] - mu_local)%*%(solve(cov_mat_local))%*%(data_local[i,] - mu_local)
  }

  log_lik = - (n*length(ind:(ind+2))/2)*log(2*pi) - (n/2)*log(det(cov_mat_local)) - (1/2)*exponent
  return(-log_lik)
}

# The following is for verification purpose for whether we are able to get the
# ISRP estimates by maximizing the localized likelihood function
for(j in 1:(q-2)){
  ind = j
  param_0 = c(0.3, 100, 3, 0.3)
  out = optim(param_0, fun_local_likelihood, method = "L-BFGS-B",
              hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001),
              upper = c(3, 150, 10, 0.999))
  est_local[j, ] =  out$par
}
colnames(est_local) = c("r", "K", "sigma2", "rho")

par(mfrow = c(2,2))
plot(1:(q-2), est_local[,1], type = "b", col = "red", lwd = 2,
     main = expression(widehat(r(Delta~t))[MLE]),
     ylab = expression(hat(r)))
plot(1:(q-2), est_local[,2], type = "b", col = "red", lwd = 2,
     main = expression(widehat(K(Delta~t))[MLE]),
     ylab = expression(hat(K)))
plot(1:(q-2), est_local[,3], type = "b", col = "red", lwd = 2,
     main = expression(widehat(sigma^2)),
     ylab = expression(hat(sigma^2)))
plot(1:(q-2), est_local[,4], type = "b", col = "red", lwd = 2,
     main = expression(widehat(rho)),
     ylab = expression(hat(rho)))


# In this section we validate the asymptotic distribution of the test statistic (T_1, T_2,...,T_{q-4})
# when the null hypothesis is true. The following experiment carried for n = 100 but we run the program
# for different values of n (n = 10, n= 20, n = 50, n = 70, n = 100). Replicate this process
# for 1000 times and simulation of the  distribution of the test statistics under the null hypothesis.

# We create a path to save the data
setwd("C:/Users/student/Desktop/Aktar/VON BERTALANFFY/n = 100")
n = 100      # As folder changes, change the value of n
rep = 1000
est_local_list = list()
x0_mean_vals = numeric(length = rep)
for(l in 1:rep){
  set.seed(l)
  print(l)
  data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat)
  x0_mean_vals[l] = mean(data[,1])
  est_local = matrix(data = NA, nrow = q-2, ncol = 4)

  for(j in 1:(q-2)){
```

```
    ind = j
    param_0 = c(0.3, 100, 3, 0.3)
    out = optim(param_0, fun_local_likelihood, method = "L-BFGS-B",
                hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001),
                upper = c(3, 150, 10, 0.999))
    est_local[j, ] =  out$par
  }
  colnames(est_local) = c("r", "K", "sigma2", "rho")
  est_local_list[[l]] = est_local
}

# Plot the estimates based on replication
# Distribution of ISRP estimates of r using localized maximum likelihood
mat_est_r_MLE = matrix(data = NA, nrow = q-2, ncol = rep)
for (j in 1:rep) {
  mat_est_r_MLE[,j] = est_local_list[[j]][,1]
}
write.table(mat_est_r_MLE, file = "r_MLE_ISRP.txt", row.names = FALSE,
            col.names = FALSE)
matplot(1:(q-2), mat_est_r_MLE, type = "l",
        main = expression(widehat(r(Delta~t))[MLE]),
        ylab = expression(hat(r)))
legend("topright", legend = paste("n = ", n), bty = "n", cex = 1.5)

# Distribution of ISRP estimates of K using localized maximum likelihood
mat_est_K_MLE = matrix(data = NA, nrow = q-2, ncol = rep)
for (j in 1:rep) {
  mat_est_K_MLE[,j] = est_local_list[[j]][,2]
}
write.table(mat_est_K_MLE, file = "K_MLE_ISRP.txt", row.names = FALSE,
            col.names = FALSE)
matplot(1:(q-2), mat_est_K_MLE, type = "l",
        main = expression(widehat(K(Delta~t))[MLE]),
        ylab = expression(hat(K)))
legend("topright", legend = paste("n = ", n), bty = "n", cex = 1.5)

# Estimation of modified RGR and its distribution
modified_RGR = matrix(data = NA, nrow = q-2, ncol = rep)
for (i in 1:(q-2)) {
  numerator = mat_est_r_MLE[i,]*exp(-mat_est_r_MLE[i,]*(t[i]/3))*
              ((mat_est_K_MLE[i,])^(1/3) - (x0_mean_vals)^(1/3))
  denominator = (mat_est_K_MLE[i,])^(1/3) + exp(-mat_est_r_MLE[i,]*(t[i]/3))*
                ((x0_mean_vals)^(1/3) - (mat_est_K_MLE[i,])^(1/3))
  modified_RGR[i,] = numerator/denominator
}
write.table(modified_RGR, file = "modified_RGR.txt", row.names = FALSE,
            col.names = FALSE)

par(mfrow = c(2,3))
for(i in 1:(q-2)){
  hist(modified_RGR[i,], probability = TRUE, main = paste("t = ", i),
       xlab = expression(hat(R[t])))
}
```

```
# Computation of the Test Statistics and its distribution
V = matrix(data = NA, nrow = q-2, ncol = rep)
for (i in 1:(q-2)) {
  V[i, ] = log(1/modified_RGR[i,] + 1/ mat_est_r_MLE[i,])
}
print(head(V))
sum(is.na(V))

test_stat = matrix(data = NA, nrow = q-4, ncol = rep)
for (i in 1:(q-4)) {
  test_stat[i,] = V[i+2,] - 2*V[i+1,] + V[i,]
}
write.table(test_stat, file = "test_stat.txt", row.names = FALSE,
            col.names = FALSE)

# Histogram of the test statistics under the null distribution (required)
par(mfrow = c(3,3))
for(i in 1:(q-4)){

  hist( test_stat[i,]/sd(test_stat[i,]), probability = TRUE, main = paste("t = ", i),
        xlab = expression(widehat(T[t])), breaks = 10)
  curve(dnorm(x), add = TRUE, col = "red", lwd = 2, lty = 2)
}

# Following we simulate the observations of size n = 100 from the Generalized Von Bertalanffy model
# f or different choices of theta. We simulate the power curves for each theta values based on 500 replicatio
# The null hypothesis is $theta = 1/3$, that is the Von Bertalanffy model. The other parameters' choices are
# $K = 100, x_0 = 10, r = 0.3$ $q = 20$ (number of time points), $sigma2 = 2, rho = 0.5$
# There will be q-4 = 16 power curves and the first power curves based on the statistics T_1 will test
#whether the data followed Von Bertalanffy model (theta = 1/3) in the time interval {1,2,3,4,5},
# similarly T_2 will check in the interval {2,3,4,5,5} and so on till T_{q-4}.
# All the power curves are generated based on 1000 replications.
# Simulation of Power function for different values of theta.

setwd("C:/Users/student/Desktop/Aktar/VON BERTALANFFY/n = 100/Power Curves")

theta_vals = seq(from = 0.23, to = 0.80, length.out = 30)
power_vals = matrix(data = NA, nrow = length(theta_vals),
                    ncol = q-4)

for (k in 1:length(theta_vals)) {
  theta = theta_vals[k]
  mu_theta_GVB = K*((1 + ((x0/K)^(theta) - 1)*exp(-r*t*theta))^(1/theta))
  # Generalized Von Bertalanffy mean function

  rep = 1000
  est_local_list = list()
  x0_mean_vals = numeric(length = rep)
  for(l in 1:rep){
    set.seed(l)
    print(l)
    data = mvtnorm::rmvnorm(n, mean = mu_theta_GVB, sigma = cov_mat)
    x0_mean_vals[l] = mean(data[,1])
    est_local = matrix(data = NA, nrow = q-2, ncol = 4)
```

```
    for(j in 1:(q-2)){
      ind = j
      param_0 = c(0.3, 100, 3, 0.3)
      out = optim(param_0, fun_local_likelihood, method = "L-BFGS-B",
                  hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001),
                  upper = c(3, 150, 10, 0.999))
      est_local[j, ] =  out$par
    }
    colnames(est_local) = c("r", "K", "sigma2", "rho")
    est_local_list[[l]] = est_local
  }
  mat_est_r_MLE = matrix(data = NA, nrow = q-2, ncol = rep)
  for (j in 1:rep) {
    mat_est_r_MLE[,j] = est_local_list[[j]][,1]
  }

  mat_est_K_MLE = matrix(data = NA, nrow = q-2, ncol = rep)
  for (j in 1:rep) {
    mat_est_K_MLE[,j] = est_local_list[[j]][,2]
  }

  # Estimation of modified RGR
  modified_RGR = matrix(data = NA, nrow = q-2, ncol = rep)
  for (i in 1:(q-2)) {
    numerator = mat_est_r_MLE[i,]*exp(-mat_est_r_MLE[i,]*(t[i]/3))*
      ((mat_est_K_MLE[i,])^(1/3) - (x0_mean_vals)^(1/3))
    denominator = (mat_est_K_MLE[i,])^(1/3) + exp(-mat_est_r_MLE[i,]*(t[i]/3))*
      ((x0_mean_vals)^(1/3) - (mat_est_K_MLE[i,])^(1/3))
    modified_RGR[i,] = numerator/denominator
  }

  V = matrix(data = NA, nrow = q-2, ncol = rep)
  for (i in 1:(q-2)) {
    V[i, ] = log(1/modified_RGR[i,] + 1/ mat_est_r_MLE[i,])
  }

  test_stat = matrix(data = NA, nrow = q-4, ncol = rep)
  for (i in 1:(q-4)) {
    test_stat[i,] = V[i+2,] - 2*V[i+1,] + V[i,]
  }

  alpha = 0.05
  prop_reject = numeric(length = q-4)
  for (i in 1:(q-4)) {
    prop_reject[i] = sum(abs(test_stat[i,]/sd(test_stat[i,]))>qnorm(1-alpha/2))/rep
  }
  power_vals[k, ] = prop_reject
}

write.table(power_vals, file = "power.txt", row.names = FALSE,
            col.names = FALSE)
```

```
par(mfrow = c(2,3))
for (i in 1:(q-4)) {
  plot(theta_vals, power_vals[,i], type = "l", lwd = 2, col = "red",
       ylab = expression(beta(theta)), xlab = expression(theta),
       main = paste("t = ", i))
}
```

3. stability

Stability of the estimators of the parameter r and K derived by Bhowmick et al.,(2014)'s method.

```
# The following parameters will be considered as the fixed set up for simulation.
K = 100 # carrying capacity
x0 = 10 # initial population size
r = 0.4 # initial intrinsic growth rate
t = seq(from = 0, to = 19, by = 1) # time # change for different time series
time_int=length(t)-2

h = numeric(length = length(t)-1) # difference between two consecutive time points
for(i in 1:(length(t)-1)){
  h[i] = t[i+1] - t[i]
}


mu= K*((1 + ((x0/K)^(1/3) - 1)*exp(-r*t/3))^3) # Generalized VB mean # change for different growth models
RGR= (r*exp(-r*t/3)*(K^(1/3)-x0^(1/3)))/(K^(1/3)+exp(-r*t/3)*(x0^(1/3)-K^(1/3)))
par(mfrow = c(1,1))
par(mar = c(5.1, 5.1, 4.1, 2.1))

plot(t, mu, col = "red", lwd=2, type = "b", pch = 19, ylab = "size, X(t)", xlab = "time (t)")

rep = 1000   # number of simulations
n = 50     # number of trajectories
sigma2 = 1    # variance of X(t) # Change for different values
rho = 0.1   # Correlation between X(t) and X(t+1)
x0_mean_vals = numeric(length = rep)
cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))   # covariance matrix
for (i in 1:length(t)) {
  for (j in 1:length(t)) {
    cov_mat[i,j] = sigma2*rho^(abs(i-j))
  }
}

r_hat_vals = matrix(data = NA, nrow = rep, ncol = time_int)

K_hat_vals = matrix(data = NA, nrow = rep, ncol = time_int)

modified_RGR = matrix(data = NA, nrow = rep, ncol = time_int)

# simulation of artificial data sets
for (i in 1:rep) {
```

```
  library(mvtnorm)
  data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat)
  x0_mean_vals[i] = mean(data[,1])
  data_mean = colMeans(data)

  # Interval estimates of r
  for(j in 1:(length(t)-2)){
    d1= (data_mean[j+1])^(1/3) - (data_mean[j])^(1/3)
    d2= (data_mean[j+2])^(1/3) - (data_mean[j+1])^(1/3)
    d12=d1/d2
    r_hat_vals[i,j]=(3/h[j])*log(d12)
    K_hat_vals[i,j]= ((x0_mean_vals[i])^(1/3)-(d1/(exp(-r_hat_vals[i,j]*(t[j]/3))*
                                                    (exp(-r_hat_vals[i,j]*(h[j]/3))-1))))^(3)
    modified_RGR[i,j]= (r_hat_vals[i,j]*exp(-r_hat_vals[i,j]*(t[j]/3))*
                          ((K_hat_vals[i,j])^(1/3) - (x0_mean_vals[i])^(1/3)))/
      ((K_hat_vals[i,j])^(1/3) + exp(-r_hat_vals[i,j]*(t[j]/3))*
         ((x0_mean_vals[i])^(1/3)-(K_hat_vals[i,j])^(1/3)))
  }
  }

par(mfrow = c(1,1))
par(mar = c(5.1, 5.1, 4.1, 2.1))

for (j in 1:(length(t)-2)) {
  hist(r_hat_vals[,j], probability = TRUE, col = "lightgreen",
       xlab = expression(widehat(r[t])), main = '', breaks = 10, cex.axis=1.9, cex.lab=1.9)
  points(r, 0, col = "red", pch = 19, cex = 2.8)

}

for (j in 1:(length(t)-2)) {
  hist(K_hat_vals[,j], probability = TRUE, col = "lightgreen",
       xlab = expression(widehat(K[t])), main = '', breaks = 10, cex.axis=1.9, cex.lab=1.9)
  points(K, 0, col = "red", pch = 19, cex = 2.8)

}
```

Stability of the estimators of the parameter r K derived by localized MLE method.

```
# The following parameters will be considered as the fixed set up for simulation.
K = 100 # carrying capacity
x0 = 10 # initial population size
r = 0.4 # initial intrinsic growth rate
t = seq(from = 0, to = 19, by = 1) # time # change for different time series
q = length(t)

h = numeric(length = length(t)-1) # difference between two consecutive time points
for(i in 1:(length(t)-1)){
  h[i] = t[i+1] - t[i]
}


# For other growth model change mu according to that growth model
```

```
mu= K*((1 + ((x0/K)^(1/3) - 1)*exp(-r*t/3))^3) # Generalized VB mean
RGR= (r*exp(-r*t/3)*(K^(1/3)-x0^(1/3)))/(K^(1/3)+exp(-r*t/3)*(x0^(1/3)-K^(1/3)))
par(mfrow = c(1,1))
plot(t, mu, col = "red", lwd=2, type = "l", pch = 19,
     ylab = "Size X(t)", xlab = "Time (t)")

# For different sigma2 and rho values, we have to change the values and set new values here.

sigma2 = 1   # variance of X(t) # Change for different values
rho = 0.1    # corr(X_t, X_{t+1})
cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
for (i in 1:length(t)) {
  for (j in 1:length(t)) {
    cov_mat[i,j] = sigma2*rho^(abs(i-j))
  }
}

n = 50    # number of independent trajectories
data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat)
matplot(t, t(data), type = "l")

# Likelihood function
fun_likelihood = function(param){
  r = param[1]
  K = param[2]
  sigma2  = param[3]
  rho = param[4]
  mu = K*((1 + ((x0/K)^(1/3) - 1)*exp(-r*t/3))^3)

  cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:q) {
    for (j in 1:q) {
      cov_mat[i,j] = sigma2*rho^(abs(i-j))
    }
  }
  exponent = 0
  for(i in 1:n){
    exponent = exponent + t(data[i,]-mu)%*%(solve(cov_mat))%*%(data[i,]-mu)
  }


  log_lik = - (n*q/2)*log(2*pi) - (n/2)*log(det(cov_mat)) - (1/2)*exponent
  return(-log_lik)
}
param_0 = c(0.4, 100, 3, 0.1)
out = optim(param_0, fun_likelihood, method = "L-BFGS-B",
            hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001), upper = c(3, 150, 10, 0.999))
out$par
solve(out$hessian)

# Localized likelihood function

est_local = matrix(data = NA, nrow = q-2, ncol = 4)

```

```r
fun_local_likelihood = function(param){
  r = param[1]
  K = param[2]
  sigma2  = param[3]
  rho = param[4]
  mu = K*((1 + ((x0/K)^(1/3) - 1)*exp(-r*t/3))^3)

  cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:q) {
    for (j in 1:q) {
      cov_mat[i,j] = sigma2*rho^(abs(i-j))
    }
  }

  # localized estimates
  mu_local = mu[ind:(ind + 2)]
  data_local = data[,ind:(ind + 2)]
  cov_mat_local = cov_mat[ind:(ind+2), ind:(ind+2)]

  exponent = 0
  for(i in 1:n){
    exponent = exponent + t(data_local[i,] - mu_local)%*%(solve(cov_mat_local))%*%(data_local[i,] - mu_local)
  }

  log_lik = - (n*length(ind:(ind+2))/2)*log(2*pi) - (n/2)*log(det(cov_mat_local)) - (1/2)*exponent
  return(-log_lik)
}

# Verification of localized likelihood

for(j in 1:(q-2)){
  ind = j
  param_0 = c(0.4, 100, 3, 0.1)
  out = optim(param_0, fun_local_likelihood, method = "L-BFGS-B",
              hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001),
              upper = c(3, 150, 10, 0.999))
  est_local[j, ] =  out$par
}
colnames(est_local) = c("r", "K", "sigma2", "rho")

par(mfrow = c(2,2))
plot(1:(q-2), est_local[,1], type = "b", col = "red", lwd = 2,
     main = expression(widehat(r(Delta~t))[MLE]),
     ylab = expression(hat(r)))
plot(1:(q-2), est_local[,2], type = "b", col = "red", lwd = 2,
     main = expression(widehat(K(Delta~t))[MLE]),
     ylab = expression(hat(K)))
plot(1:(q-2), est_local[,3], type = "b", col = "red", lwd = 2,
     main = expression(widehat(sigma^2)),
     ylab = expression(hat(sigma^2)))
plot(1:(q-2), est_local[,4], type = "b", col = "red", lwd = 2,
     main = expression(widehat(rho)),
     ylab = expression(hat(rho)))
```

```
775
776 # Localized MLE of the parameters
777
778 # Path to save the values
779 setwd("D:/My Paper/3rd Paper MLE vs ISRP/Final_Plot_file/Von/1-20_1")
780 n=50 # Number of trajectories
781 rep = 1000 # Replication no.
782 est_local_list = list()
783 x0_mean_vals = numeric(length = rep)
784 for(l in 1:rep){
785   set.seed(l)
786   print(l)
787   data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat)
788   x0_mean_vals[l] = mean(data[,1])
789   est_local = matrix(data = NA, nrow = q-2, ncol = 4)
790
791   for(j in 1:(q-2)){
792     ind = j
793     param_0 = c(0.4, 100, 3, 0.1)
794     out = optim(param_0, fun_local_likelihood, method = "L-BFGS-B",
795                 hessian = TRUE, lower = c(0.001, 50, 0.0001, -0.001),
796                 upper = c(3, 150, 10, 0.999))
797     est_local[j, ] =  out$par
798   }
799   colnames(est_local) = c("r", "K", "sigma2", "rho")
800   est_local_list[[l]] = est_local
801 }
802
803 # Plot the estimates based on replication
804 # Distribution of ISRP estimates of r using localized maximum likelihood
805 mat_est_r_MLE = matrix(data = NA, nrow = q-2, ncol = rep)
806 for (j in 1:rep) {
807   mat_est_r_MLE[,j] = est_local_list[[j]][,1]
808 }
809 write.table(mat_est_r_MLE, file = "r_MLE_ISRP.txt", row.names = FALSE,
810             col.names = FALSE)
811 matplot(1:(q-2), mat_est_r_MLE, type = "l",
812         main = expression(widehat(r(Delta~t))[MLE]),
813         ylab = expression(hat(r)))
814 legend("topright", legend = paste("n = ", n), bty = "n", cex = 1.5)
815
816 # Distribution of ISRP estimates of K using localized maximum likelihood
817 mat_est_K_MLE = matrix(data = NA, nrow = q-2, ncol = rep)
818 for (j in 1:rep) {
819   mat_est_K_MLE[,j] = est_local_list[[j]][,2]
820 }
821 write.table(mat_est_K_MLE, file = "K_MLE_ISRP.txt", row.names = FALSE,
822             col.names = FALSE)
823 matplot(1:(q-2), mat_est_K_MLE, type = "l",
824         main = expression(widehat(K(Delta~t))[MLE]),
825         ylab = expression(hat(K)))
826 legend("topright", legend = paste("n = ", n), bty = "n", cex = 1.5)
827
828 # Estimation of modified RGR and its distribution
```

```
modified_RGR = matrix(data = NA, nrow = q-2, ncol = rep)
for (i in 1:(q-2)) {
  numerator = mat_est_r_MLE[i,]*exp(-mat_est_r_MLE[i,]*t[i]/3)*((mat_est_K_MLE[i,])^(1/3) - (x0_mean_vals)^(1
  denominator = (mat_est_K_MLE[i,])^(1/3) + exp(-mat_est_r_MLE[i,]*t[i]/3)*((x0_mean_vals)^(1/3) - (mat_est_K
  modified_RGR[i,] = numerator/denominator
}
write.table(modified_RGR, file = "modified_RGR.txt", row.names = FALSE,
            col.names = FALSE)

par(mfrow = c(1,1))
par(mar = c(5.1, 5.1, 4.1, 2.1))

for(i in 1:(q-2)){
  hist(mat_est_r_MLE[i,], probability = TRUE, main = '', breaks = 10,
       xlab = expression(widehat(r[t])), cex.axis=1.9, cex.lab=1.9)
  points(r, 0, col = "red", pch = 19, cex = 2.8)
}

for(i in 1:(q-2)){
  hist(mat_est_K_MLE[i,], probability = TRUE, main = '', breaks = 10,
       xlab = expression(widehat(K[t])), cex.axis=1.9, cex.lab=1.9)
  points(K, 0, col = "red", pch = 19, cex = 2.8)
}

for(i in 1:(q-2)){
  hist(modified_RGR[i,], probability = TRUE, main = '', breaks = 10,
       xlab = expression(widehat(R[t])), cex.axis=1.9, cex.lab=1.9)
  points(RGR[i], 0, col = "red", pch = 19, cex = 2.8)
}
```

4. Efficiency

```
# Testing of efficiency by using Multivariate Delta Method
# m = 5 ## Number of sigma values
p = 1000 ### repetition
index.sum1 = numeric(p); index.sum1

for(a in 1:p)
{
  K = 100 # carrying capacity
  x0 = 10 # initial population size
  r0 = 0.4 # initial intrinsic growth rate
  t = seq(from = 1, to = 20, by = 1) # time
  q=length(t)

  h = numeric(length = length(t)-1) # difference between two consecutive time points
  for(i in 1:(length(t)-1)){
    h[i] = t[i+1] - t[i]
  }

  mu = K*(1 + ((x0/K)^(1/3) - 1)*exp((-r0*t)/3))^3   # Von Bertalanffy mean function
  par(mfrow = c(1,1))
  plot(t, mu, col = "red", lwd=2, type = "b", pch = 19,
```

```
       ylab = "size, X(t)", xlab = "time (t)")


  sigma1 = 0.1  # variance of X_t
  rho = 0.1      # corr(X_t, X_{t+1})


  cov_mat1 = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:length(t)) {
    for (j in 1:length(t)) {
      cov_mat1[i,j] = sigma1*rho^(abs(i-j))
    }
  }

  n = 100 # number of trajectories
  library(mvtnorm)
  data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat1)
  matplot(t(data), type = "l", ylab = "Size",  xlab = "Time")
  legend("topleft", legend = paste("n = ", n), bty = "n")

  d1 = matrix(data = NA, nrow = n, ncol = length(t)-2)
  d2 = matrix(data = NA, nrow = n, ncol = length(t)-2)

  for(j in 1:(length(t)-2)){
    d1[,j] = ((data[,j+1])^(1/3)) - ((data[,j])^(1/3))
    d2[,j] = ((data[,j+2])^(1/3)) - ((data[,j+1])^(1/3))
  }

  # Computation for r_hat
  r_hat_vals = matrix(data = NA, nrow = n, ncol = length(t)-2)
  for (j in 1:(length(t)-2)) {
    r_hat_vals[,j] = (3/h[j])*log(d1[,j]/d2[,j])
  }

  selected_index = rep(FALSE, n)
  for (i in 1:n) {
    if(sum(is.na(r_hat_vals[i,])) ==0)
      selected_index[i] = TRUE

  }
  selected_index    # List of selected rows for further calculations
  index.sum1[a] = sum(selected_index)
}
index.sum1

# For second sigma value #
index.sum2 = numeric(p); index.sum2
for(a in 1:p)
{
  K = 100 # carrying capacity
  x0 = 10 # initial population size
  r0 = 0.4 # initial intrinsic growth rate
  t = seq(from = 1, to = 20, by = 1) # time
  q=length(t)
```

```
  h = numeric(length = length(t)-1) # difference between two consecutive time points
  for(i in 1:(length(t)-1)){
    h[i] = t[i+1] - t[i]
  }

  mu = K*(1 + ((x0/K)^(1/3) - 1)*exp((-r0*t)/3))^3   # Von Bertalanffy mean function
  par(mfrow = c(1,1))
  plot(t, mu, col = "red", lwd=2, type = "b", pch = 19,
       ylab = "size, X(t)", xlab = "time (t)")


  sigma2 = 0.15  # variance of X_t
  rho = 0.1      # corr(X_t, X_{t+1})

  cov_mat2 = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:length(t)) {
    for (j in 1:length(t)) {
      cov_mat2[i,j] = sigma2*rho^(abs(i-j))
    }
  }

  n = 100 # number of trajectories
  library(mvtnorm)
  data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat2)
  matplot(t(data), type = "l", ylab = "Size",  xlab = "Time")
  legend("topleft", legend = paste("n = ", n), bty = "n")

  d1 = matrix(data = NA, nrow = n, ncol = length(t)-2)
  d2 = matrix(data = NA, nrow = n, ncol = length(t)-2)

  for(j in 1:(length(t)-2)){
    d1[,j] = ((data[,j+1])^(1/3)) - ((data[,j])^(1/3))
    d2[,j] = ((data[,j+2])^(1/3)) - ((data[,j+1])^(1/3))
  }

  # Computation for r_hat
  r_hat_vals = matrix(data = NA, nrow = n, ncol = length(t)-2)
  for (j in 1:(length(t)-2)) {
    r_hat_vals[,j] = (3/h[j])*log(d1[,j]/d2[,j])
  }


  selected_index = rep(FALSE, n)
  for (i in 1:n) {
    if(sum(is.na(r_hat_vals[i,])) ==0)
      selected_index[i] = TRUE

  }
  selected_index    # List of selected rows for further calculations
  index.sum2[a] = sum(selected_index)
}
index.sum2
```

```
# For third sigma value #
index.sum3 = numeric(p); index.sum3
for(a in 1:p)
{
  K = 100 # carrying capacity
  x0 = 10 # initial population size
  r0 = 0.4 # initial intrinsic growth rate
  t = seq(from = 1, to = 20, by = 1) # time
  q=length(t)

  h = numeric(length = length(t)-1) # difference between two consecutive time points
  for(i in 1:(length(t)-1)){
    h[i] = t[i+1] - t[i]
  }

  mu = K*(1 + ((x0/K)^(1/3) - 1)*exp((-r0*t)/3))^3   # Von Bertalanffy mean function
  par(mfrow = c(1,1))
  plot(t, mu, col = "red", lwd=2, type = "b", pch = 19,
       ylab = "size, X(t)", xlab = "time (t)")


  sigma3 = 0.20  # variance of X_t
  rho = 0.1      # corr(X_t, X_{t+1})

  cov_mat3 = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:length(t)) {
    for (j in 1:length(t)) {
      cov_mat3[i,j] = sigma3*rho^(abs(i-j))
    }
  }

  n = 100 # number of trajectories
  library(mvtnorm)
  data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat3)
  matplot(t(data), type = "l", ylab = "Size",  xlab = "Time")
  legend("topleft", legend = paste("n = ", n), bty = "n")

  d1 = matrix(data = NA, nrow = n, ncol = length(t)-2)
  d2 = matrix(data = NA, nrow = n, ncol = length(t)-2)

  for(j in 1:(length(t)-2)){
    d1[,j] = ((data[,j+1])^(1/3)) - ((data[,j])^(1/3))
    d2[,j] = ((data[,j+2])^(1/3)) - ((data[,j+1])^(1/3))
  }

  # Computation for r_hat
  r_hat_vals = matrix(data = NA, nrow = n, ncol = length(t)-2)
  for (j in 1:(length(t)-2)) {
    r_hat_vals[,j] = (3/h[j])*log(d1[,j]/d2[,j])
  }

  selected_index = rep(FALSE, n)
  for (i in 1:n) {
    if(sum(is.na(r_hat_vals[i,])) ==0)
```

```
      selected_index[i] = TRUE

  }
  selected_index    # List of selected rows for further calculations
  index.sum3[a] = sum(selected_index)
}
index.sum3

# For fourth sigma value #
index.sum4 = numeric(p); index.sum4
for(a in 1:p)
{
  K = 100 # carrying capacity
  x0 = 10 # initial population size
  r0 = 0.4 # initial intrinsic growth rate
  t = seq(from = 1, to = 20, by = 1) # time
  q=length(t)

  h = numeric(length = length(t)-1) # difference between two consecutive time points
  for(i in 1:(length(t)-1)){
    h[i] = t[i+1] - t[i]
  }

  mu = K*(1 + ((x0/K)^(1/3) - 1)*exp((-r0*t)/3))^3   # Von Bertalanffy mean function
  par(mfrow = c(1,1))
  plot(t, mu, col = "red", lwd=2, type = "b", pch = 19,
       ylab = "size, X(t)", xlab = "time (t)")


  sigma4 = 0.25  # variance of X_t
  rho = 0.1      # corr(X_t, X_{t+1})

  cov_mat4 = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:length(t)) {
    for (j in 1:length(t)) {
      cov_mat4[i,j] = sigma4*rho^(abs(i-j))
    }
  }

  n = 100 # number of trajectories
  library(mvtnorm)
  data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat4)
  matplot(t(data), type = "l", ylab = "Size",  xlab = "Time")
  legend("topleft", legend = paste("n = ", n), bty = "n")

  d1 = matrix(data = NA, nrow = n, ncol = length(t)-2)
  d2 = matrix(data = NA, nrow = n, ncol = length(t)-2)

  for(j in 1:(length(t)-2)){
    d1[,j] = ((data[,j+1])^(1/3)) - ((data[,j])^(1/3))
    d2[,j] = ((data[,j+2])^(1/3)) - ((data[,j+1])^(1/3))
  }

  # Computation for r_hat
```

```
  r_hat_vals = matrix(data = NA, nrow = n, ncol = length(t)-2)
  for (j in 1:(length(t)-2)) {
    r_hat_vals[,j] = (3/h[j])*log(d1[,j]/d2[,j])
  }

  selected_index = rep(FALSE, n)
  for (i in 1:n) {
    if(sum(is.na(r_hat_vals[i,])) ==0)
      selected_index[i] = TRUE

  }
  selected_index    # List of selected rows for further calculations
  index.sum4[a] = sum(selected_index)
}
index.sum4

# For fifth sigma value #
index.sum5 = numeric(p); index.sum5
for(a in 1:p)
{
  K = 100 # carrying capacity
  x0 = 10 # initial population size
  r0 = 0.4 # initial intrinsic growth rate
  t = seq(from = 1, to = 20, by = 1) # time
  q=length(t)

  h = numeric(length = length(t)-1) # difference between two consecutive time points
  for(i in 1:(length(t)-1)){
    h[i] = t[i+1] - t[i]
  }

  mu = K*(1 + ((x0/K)^(1/3) - 1)*exp((-r0*t)/3))^3   # Von Bertalanffy mean function
  par(mfrow = c(1,1))
  plot(t, mu, col = "red", lwd=2, type = "b", pch = 19,
       ylab = "size, X(t)", xlab = "time (t)")


  sigma5 = 0.30  # variance of X_t
  rho = 0.1      # corr(X_t, X_{t+1})

  cov_mat5 = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:length(t)) {
    for (j in 1:length(t)) {
      cov_mat5[i,j] = sigma5*rho^(abs(i-j))
    }
  }

  n = 100 # number of trajectories
  library(mvtnorm)
  data = mvtnorm::rmvnorm(n, mean = mu, sigma = cov_mat5)
  matplot(t(data), type = "l", ylab = "Size",  xlab = "Time")
  legend("topleft", legend = paste("n = ", n), bty = "n")

  d1 = matrix(data = NA, nrow = n, ncol = length(t)-2)
```

```
  d2 = matrix(data = NA, nrow = n, ncol = length(t)-2)

  for(j in 1:(length(t)-2)){
    d1[,j] = ((data[,j+1])^(1/3)) - ((data[,j])^(1/3))
    d2[,j] = ((data[,j+2])^(1/3)) - ((data[,j+1])^(1/3))
  }

  # Computation for r_hat
  r_hat_vals = matrix(data = NA, nrow = n, ncol = length(t)-2)
  for (j in 1:(length(t)-2)) {
    r_hat_vals[,j] = (3/h[j])*log(d1[,j]/d2[,j])
  }

  selected_index = rep(FALSE, n)
  for (i in 1:n) {
    if(sum(is.na(r_hat_vals[i,])) ==0)
      selected_index[i] = TRUE

  }
  selected_index    # List of selected rows for further calculations
  index.sum5[a] = sum(selected_index)
}
index.sum5

index.sum = rbind(index.sum1/100, index.sum2/100, index.sum3/100, index.sum4/100,
                  index.sum5/100)
colnames(index.sum) = seq(1:p)
rownames(index.sum) = c(0.10, 0.15, 0.20, 0.25, 0.30)
index.sum
boxplot(index.sum~rownames(index.sum), col="lightgreen", ylab = "Proportion of successful simulation",
        xlab = expression(paste(sigma^2)))
```

Similarly we can plot the efficiency boxplot for other growth models. We just have to change the mu value as per the growth model.

5. Sensitivity analysis

Parameter sensitivity analysis using the ISRP of the parameters derived by localized MLE method

```
fun_local_likelihood = function(r,K){
  sigma2  = 1
  rho = 0.5
  mu = K/(1 + (K/x0 - 1)*exp(-r*t)) #change for other growth eqn

  cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:q) {
    for (j in 1:q) {
      cov_mat[i,j] = sigma2*rho^(abs(i-j))
    }
  }

  # localized estimates
  mu_local = mu[ind:(ind + 2)]
```

```
  data_local = data[,ind:(ind + 2)]
  cov_mat_local = cov_mat[ind:(ind+2), ind:(ind+2)]

  exponent = 0
  for(i in 1:n){
    exponent = exponent + t(data_local[i,] - mu_local)%*%(solve(cov_mat_local))%*%(data_local[i,] - mu_local)
  }

  log_lik = - (n*length(ind:(ind+2))/2)*log(2*pi) - (n/2)*log(det(cov_mat_local)) - (1/2)*exponent
  return(-log_lik)
}

r = seq(0.01, 3, length = 10)
K = seq(50, 150, length = 10)

par(mfrow = c(2,3))

par(mfrow = c(1,1))
for (k in 1:(q-2)) {
  ind = k
  val = matrix(data = NA, nrow = length(r), ncol = length(K))
  for(i in 1:length(r)){
    for (j in 1:length(K)) {
      val[i,j] = fun_local_likelihood(r[i], K[j])
    }
  }
  if(k%%3==0){
    persp(r,K,val, col = "lightgrey", main = "",
          theta = 30, phi = 30)
  }


}

library(rgl)
open3d()
mfrow3d(2, 3, sharedMouse = TRUE)

ind = 1
val = matrix(data = NA, nrow = length(r), ncol = length(K))
for(i in 1:length(r)){
  for (j in 1:length(K)) {
    val[i,j] = fun_local_likelihood(r[i], K[j])
  }
}
persp3d(r,K,val, col = "yellow", main = paste0("t=",ind),
        theta = 50, phi = 30)

ind = 4
val = matrix(data = NA, nrow = length(r), ncol = length(K))
for(i in 1:length(r)){
  for (j in 1:length(K)) {
    val[i,j] = fun_local_likelihood(r[i], K[j])
  }
```

```
}
persp3d(r,K,val, col = "yellow", main = paste0("t=",ind),
        theta = 50, phi = 30)

ind = 8
val = matrix(data = NA, nrow = length(r), ncol = length(K))
for(i in 1:length(r)){
  for (j in 1:length(K)) {
    val[i,j] = fun_local_likelihood(r[i], K[j])
  }
}
persp3d(r,K,val, col = "yellow", main = paste0("t=",ind),
        theta = 50, phi = 30)


ind = 11
val = matrix(data = NA, nrow = length(r), ncol = length(K))
for(i in 1:length(r)){
  for (j in 1:length(K)) {
    val[i,j] = fun_local_likelihood(r[i], K[j])
  }
}
persp3d(r,K,val, col = "yellow", main = paste0("t=",ind),
        theta = 50, phi = 30)

ind = 14
val = matrix(data = NA, nrow = length(r), ncol = length(K))
for(i in 1:length(r)){
  for (j in 1:length(K)) {
    val[i,j] = fun_local_likelihood(r[i], K[j])
  }
}
persp3d(r,K,val, col = "yellow", main = paste0("t=",ind),
        theta = 50, phi = 30)


ind = 17
val = matrix(data = NA, nrow = length(r), ncol = length(K))
for(i in 1:length(r)){
  for (j in 1:length(K)) {
    val[i,j] = fun_local_likelihood(r[i], K[j])
  }
}
persp3d(r,K,val, col = "yellow", main = paste0("t=",ind),
        theta = 50, phi = 30)
```

6. Real data analysis

```
REAL Data Set for this analysis:
id G 0 14 28 42 56 70 84 98 112 126 133
1 B 210 215 230 244 259 266 277 292 292 290 264
2 B 230 240 258 277 277 293 300 323 327 340 343
3 B 226 233 248 277 297 313 322 340 354 365 362
4 B 233 239 253 277 292 310 318 333 336 353 338
```

5 B 238 241 262 282 300 314 319 331 338 348 338
6 B 225 228 237 261 271 288 300 316 319 333 330
7 B 224 225 239 257 268 290 304 313 310 318 318
8 B 237 241 255 276 293 307 312 336 336 344 328
9 B 237 224 234 239 256 266 276 300 302 293 269
10 B 233 239 259 283 294 313 320 347 348 362 352
11 B 217 222 235 256 267 285 295 317 315 308 301
12 B 228 223 246 266 277 287 300 312 308 328 333
13 B 241 247 268 290 309 323 336 348 359 372 370
14 B 221 221 240 253 273 282 292 307 306 317 318
15 B 217 220 235 259 262 276 284 305 303 315 317
16 B 214 221 237 256 271 283 287 314 316 320 298
17 B 224 231 241 256 265 283 295 314 313 328 334
18 B 200 203 221 236 248 262 276 294 291 311 310
19 B 238 232 252 268 285 298 303 320 324 320 327
20 B 230 222 243 253 268 284 290 316 314 330 330
21 B 217 224 242 265 284 302 309 324 328 338 334
22 B 209 209 221 238 256 267 281 295 301 309 289
23 B 224 227 245 267 279 294 312 328 329 297 297
24 B 230 231 244 261 272 283 294 318 320 333 338
25 B 216 218 223 243 259 270 270 290 301 314 297
26 B 231 239 254 276 294 304 317 335 333 319 307
27 B 207 216 228 255 275 285 296 314 319 330 330
28 B 227 236 251 264 276 287 297 315 309 313 294
29 B 221 232 251 274 284 295 300 323 319 333 322
30 B 233 238 254 266 282 294 295 310 320 327 326
31 A 233 224 245 258 271 287 287 287 290 293 297
32 A 231 238 260 273 290 300 311 313 317 321 326
33 A 232 237 245 265 285 298 304 319 317 334 329
34 A 239 246 268 288 308 309 327 324 327 336 341
35 A 215 216 239 264 282 299 307 321 328 332 337
36 A 236 226 242 255 263 277 290 299 300 308 310
37 A 219 229 246 265 279 292 299 299 298 300 290
38 A 231 245 270 292 302 321 322 334 323 337 337
39 A 230 228 243 255 272 276 277 289 289 300 303
40 A 232 240 247 263 275 286 294 302 308 319 326
41 A 234 237 259 289 311 324 342 347 355 368 368
42 A 237 235 258 263 282 304 318 327 336 349 353
43 A 229 234 254 276 294 315 323 341 346 352 357
44 A 220 227 248 273 290 308 322 326 330 342 343
45 A 232 241 255 276 293 309 310 330 326 329 330
46 A 210 225 242 260 272 277 273 295 292 305 306
47 A 229 241 252 265 274 285 303 308 315 328 328
48 A 204 198 217 233 251 258 272 283 279 295 298
49 A 220 221 236 260 274 295 300 301 310 318 316
50 A 233 234 250 268 280 298 308 319 318 336 333
51 A 234 234 254 274 294 306 318 334 343 349 350
52 A 200 207 217 238 252 267 284 282 282 284 288
53 A 220 213 229 252 254 273 293 289 294 292 298
54 A 225 239 254 269 289 308 313 324 327 347 344
55 A 236 245 257 271 294 307 317 327 328 328 325
56 A 231 231 237 261 274 285 291 301 307 315 320
57 A 208 211 238 254 267 287 306 312 320 337 338
58 A 232 248 261 285 292 307 312 323 318 328 329

```
59 A 233 241 252 273 301 316 332 336 339 348 345
60 A 221 219 231 251 270 272 287 294 292 292 299

setwd("D:/My Paper/3rd Paper MLE vs ISRP/RDA")

cattle_data_1=read.delim("cattle_data.txt",sep=" ")
colnames(cattle_data_1)[1:13]<-c("id","G","0","14","28","42","56","70","84","98","112","126","133")
cattle_data=cattle_data_1[,1:13]

data_B=cattle_data[1:30,2:12]#for Group B
data_A=cattle_data[31:60,2:13]#for Group A
t=c(0,14,28,42,56,70,84,98,112,126,133)

write.table(data_A[,-1], "data_A.txt", row.names = FALSE, col.names = FALSE)

t = seq(from = 1, to = 11, by = 1)

q=length(t)
data=as.matrix(read.table("data_A.txt", header = FALSE))

n= 30
View(data)

# The following parameters will be considered as the fixed set up for simulation.
r= 0.05
x0 = 220
sigma2= 1
rho= 0.1

mu = x0*exp(r*t)
plot(t,mu)

fun_likelihood = function(param){
  r = param[1]
  sigma2  = param[2]
  rho = param[3]
  mu = x0*exp(r*t)

  cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:q) {
    for (j in 1:q) {
      cov_mat[i,j] = sigma2*rho^(abs(i-j))
    }
  }
  exponent = 0
  for(i in 1:n){
    exponent = exponent + t(data[i,]-mu)%*%(solve(cov_mat))%*%(data[i,]-mu)
  }


  log_lik = - (n*q/2)*log(2*pi) - (n/2)*log(det(cov_mat)) - (1/2)*exponent
  return(-log_lik)
}
param_0 = c(0.1, 1, 0.1)
```

```
out = optim(param_0, fun_likelihood, method = "L-BFGS-B",
            hessian = TRUE, lower = c(0.001, 0.1, -0.001), upper = c(1, 10, 0.999))
out$par
solve(out$hessian)

# Local MLE
est_local = matrix(data = NA, nrow = q-1, ncol = 3)

fun_local_likelihood = function(param){
  r = param[1]
  sigma2  = param[2]
  rho = param[3]
  mu = x0*exp(r*t)

  cov_mat = matrix(data = NA, nrow = length(t), ncol = length(t))
  for (i in 1:q) {
    for (j in 1:q) {
      cov_mat[i,j] = sigma2*rho^(abs(i-j))
    }
  }

  # localized estimates
  mu_local = mu[ind:(ind + 1)]
  data_local = data[,ind:(ind + 1)]
  cov_mat_local = cov_mat[ind:(ind+1), ind:(ind+1)]

  exponent = 0
  for(i in 1:n){
    exponent = exponent + t(data_local[i,] - mu_local)%*%(solve(cov_mat_local))%*%(data_local[i,] - mu_local)
  }

  log_lik = - (n*length(ind:(ind+1))/2)*log(2*pi) - (n/2)*log(det(cov_mat_local)) - (1/2)*exponent
  return(-log_lik)
}


# likelihood function
for(j in 1:(q-1)){
  ind = j
  param_0 = c(0.1, 1, 0.1)
  out = optim(param_0, fun_local_likelihood, method = "L-BFGS-B",
              hessian = TRUE, lower = c(0.001, 0.1, -0.001), upper = c(1, 10, 0.999))
  est_local[j, ] =  out$par
}
colnames(est_local) = c("r", "sigma2", "rho")

par(mar = c(5.1, 5.1, 4.1, 2.1))
plot(1:(q-1), est_local[,1], type = "b", col = "red", lwd = 2,
     ylab = expression(widehat(r(Delta~t))[MLE]),
     xlab = "Time")
r_hat=est_local[,1]
x = 1:10
fit = nls(r_hat ~ b*exp(-c*x)*x, start = list(b=1,c=3))
```

```
points(x,r_hat, col = "red", lwd = 2, cex= 1.5, pch = 19)

curve(coef(fit)[1]*exp(-coef(fit)[2]*x)*x, lwd = 3, col = "blue", add = TRUE)

legend("bottomright", legend = bquote(r(t) == b*e^(-c*t)*t),
       bty = "n", lwd = 3, col = "blue")

b_hat = coef(fit)[1]
c_hat = coef(fit)[2]
AIC(fit)

```