Time-Lagged Integral Projection Model

Study species: *Campanula thyrsoides*

Last update: Sunday, 13.07.2007

Required input files

1. "ct.ipm.txt"
   columns: n == Unique number for each individual
   site == Population abbreviation (here: FU == Furka Pass, SP == Schynige Platte)
   year.t == here: 2004 or 2005
   nl.t == number of rosette leaves in year t
   nl.t1 == number of rosette leaves in year t+1
   ll.t == length of longest rosette leave in year t
   ll.t1 == length of longest rosette leave in year t+1
   surv == survival: 1 == yes, 0 == no
   flow == flowering: 1 == yes, 0 == no
   ros == number of rosettes in individual
2. "ct.IPM.fecundity.txt"
   columns: n == Unique number for each individual
   site == Population abbreviation (here: FU == Furka Pass, SP == Schynige Platte)
   year.t == here: 2004 or 2005
   ll.t == length of longest rosette leave in year t
   nl.t == number of rosette leaves in year t
   si.t1 == seeds per individual in year t+1
   brows.t1 == browsed in year t+1 (here: 1 ==Yes, 0 == No)
   ros == number of rosettes in individual
   size.t == size in year t
3. "CT.IPM.seedlings.txt"
   columns: n == Unique number for each individual
   site == Population abbreviation (here: FU == Furka Pass, SP == Schynige Platte)
   year.t == here: 2004 or 2005
   ll.t == length of longest rosette leave in year t
   nl.t == number of rosette leaves in year t
   size.t == size in year t
4. "IPM.establishment.data.txt"
   columns: site == Population abbreviation (here: FU == Furka Pass, SP == Schynige Platte)
   plot == Number of plot
   year.t == here: 2004 or 2005
   seeds.t == seed production in each plot
   sdl.t1 == seedlings in year t+1 in each plot
5. "ipm.saf.data.txt"
   columns: n == Unique number for each individual
   site == Population abbreviation (here: FU == Furka Pass, SP == Schynige Platte)
   flo == Flowering (here: 1==Yes, 0==NO)
   age == Age 2005

Load packages

```
# Clear environment
rm(list=ls())

# Load packages
library(tidyverse)
library(nlme)
library(MASS)
library(cowplot)

setwd("~/Desktop/IQ/Rotation2/rotation2_thyrsoides")

# Detach any lingering attached datafs
while(any(search()=="dataf_r1")) detach(dataf_r1)
```

Part I. Fitting Models

Read first in dataset

```
dataf = data.frame(read.table("ct.ipm.txt",header=T))
# names(dataf)

# attach(dataf) ## Set reference dataframe

# Filter full dataframe for plants with 1 rosette
dataf_r1 = filter(dataf,ros==1)
attach(dataf_r1)

dataf_r1$size.t.all = log(nl.t*ll.t) ## plant sizes in year t

dataf_r1$size.t1.all = log(nl.t1*ll.t1) ## plant sizes in year t+1

## The next three are unnecessary since they are just duplicating data in another column, but keeping f
dataf_r1$flow.all = flow ## Logical vector if flowering: 1 = yes
dataf_r1$surv.all = surv ## Logical vector of survival: 1 = yes
dataf_r1$site.all = site ## Site vector

site.code.l=c("FU","SP") ## Used later for plots
pch.code=c(19,1)

# Detach old version, attach new version
detach(dataf_r1)
attach(dataf_r1)

all.sizes=c(size.t.all[flow.all==0],size.t1.all[year.t==2005]) ## all plant sizes
all.site=c(site.all[flow.all==0],site.all[year.t==2005]) ## All sites -- how is this one used?
detach(dataf_r1)
```

Calculation: Growth

```
growth_df <- dataf_r1 %>%
  filter(flow.all==0) %>%
  dplyr::select(size.t = size.t.all,
                size.t1 = size.t1.all,
                site.s = site.all) %>%
```

```r
  filter(complete.cases(.))

######### check whether variance structure is needed #########
## AKA check for heterogeneity?
fit.grow.gls.1<-gls(size.t1~size.t+site.s,
                    na.action=na.omit,
                    weight=varExp(form=~fitted(.)|site.s), ## Exponential of the variance covariate gro
                    method="ML",
                    data=growth_df); ## Maximum likelihood
# summary(fit.grow.gls.1)
# plot(fit.grow.gls.1) ##

fit.grow.gls<-gls(size.t1~size.t+site.s,
                  na.action=na.omit,
                  weight=varExp(form=~fitted(.)), ## Exponential variance function structure of fitted
                  method="ML",
                  data=growth_df);
# summary(fit.grow.gls)
# plot(fit.grow.gls) ##

fit.grow.gls.0<-gls(size.t1~size.t+site.s,
                    na.action=na.omit,
                    ## no weight used
                    method="ML",
                    data=growth_df);
# summary(fit.grow.gls.0)
# plot(fit.grow.gls.0) ##

## Compare models
anova(fit.grow.gls.0,fit.grow.gls,fit.grow.gls.1)
```

```
##                Model df      AIC      BIC    logLik   Test   L.Ratio
## fit.grow.gls.0     1  4 1803.644 1823.627 -897.8221
## fit.grow.gls       2  5 1648.528 1673.507 -819.2639 1 vs 2 157.11630
## fit.grow.gls.1     3  6 1627.669 1657.643 -807.8344 2 vs 3  22.85915
##                p-value
## fit.grow.gls.0
## fit.grow.gls    <.0001
## fit.grow.gls.1  <.0001
```

```r
# Remove models from environments
rm(fit.grow.gls,fit.grow.gls.0,fit.grow.gls.1)

######### check whether intercept estimate for habitat is needed #########
fit.grow.gls.0<-gls(size.t1~size.t, ## size only
                    na.action=na.omit,
                    weight=varExp(form=~fitted(.)|site.s),
                    method="ML",
                    data=growth_df);
# plot(fit.grow.gls.0)

fit.grow.gls.1<-gls(size.t1~size.t+site.s, ## size + site as independent terms
                    na.action=na.omit,
                    weight=varExp(form=~fitted(.)|site.s),
```

```r
                     method="ML",
                     data=growth_df)
# plot(fit.grow.gls.1)

fit.grow.gls.2<-gls(size.t1~size.t*site.s, ## size*site as interaction term
                     na.action=na.omit,
                     weight=varExp(form=~fitted(.)|site.s),
                     method="ML",
                     data=growth_df)
# plot(fit.grow.gls.2)

## Compare models
anova(fit.grow.gls.0,fit.grow.gls.1,fit.grow.gls.2)
```

```
##                  Model df      AIC      BIC    logLik   Test   L.Ratio
## fit.grow.gls.0     1  5 1636.038 1661.017 -813.0191
## fit.grow.gls.1     2  6 1627.669 1657.643 -807.8344 1 vs 2 10.369433
## fit.grow.gls.2     3  7 1627.092 1662.062 -806.5459 2 vs 3  2.576921
##                  p-value
## fit.grow.gls.0
## fit.grow.gls.1  0.0013
## fit.grow.gls.2  0.1084
```

```r
# Remove models from environment
rm(fit.grow.gls.0,fit.grow.gls.1,fit.grow.gls.2)

######### refit model with size and site main effects, and site specific decreasing variance #########
fit.grow.gls<-gls(size.t1~site.s+size.t-1, ## Why -1?
                   na.action=na.omit,
                   weight=varExp(form=~fitted(.)|site.s),
                   method="ML",
                   data=growth_df)

summary(fit.grow.gls)
```

```
## Generalized least squares fit by maximum likelihood
##   Model: size.t1 ~ site.s + size.t - 1
##   Data: growth_df
##        AIC      BIC    logLik
##   1627.669 1657.643 -807.8344
##
## Variance function:
##  Structure: Exponential of variance covariate, different strata
##  Formula: ~fitted(.) | site.s
##  Parameter estimates:
##         FU         SP
## -0.2455818 -0.2115804
##
## Coefficients:
##             Value  Std.Error  t-value p-value
## site.sFU 1.175113 0.08097083 14.51280       0
## site.sSP 1.067800 0.08598521 12.41841       0
## size.t   0.882743 0.01269721 69.52259       0
##
##  Correlation:
```

```
##          st.sFU st.sSP
## site.sSP  0.922
## size.t   -0.979 -0.942
##
## Standardized residuals:
##         Min          Q1         Med          Q3         Max
## -4.74270213 -0.64741828 -0.06019704  0.56735452  4.13082080
##
## Residual standard error: 2.145755
## Degrees of freedom: 1092 total; 1089 residual
```

```
intervals(fit.grow.gls) ## Confidence intervals on the parameters associated with the model
```

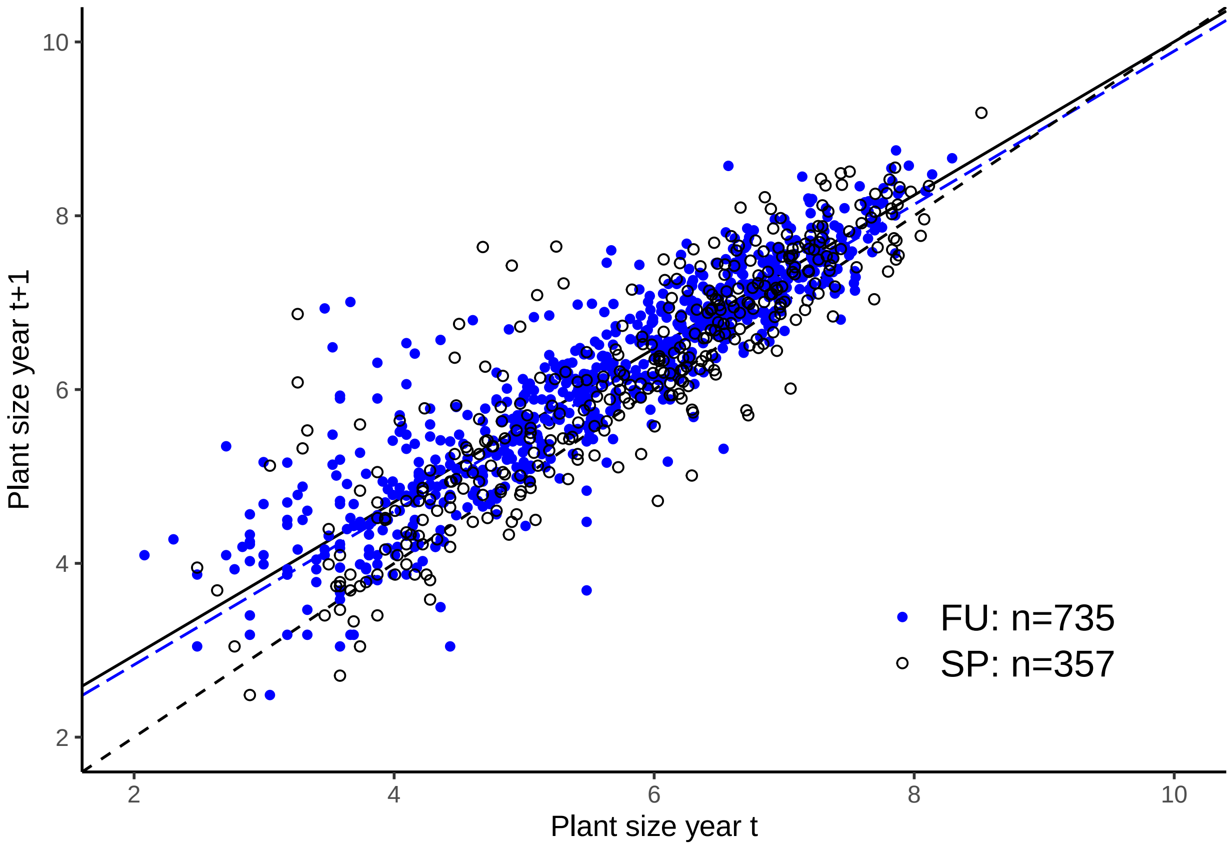
```
## Approximate 95% confidence intervals
##
##  Coefficients:
##              lower     est.     upper
## site.sFU 1.0162366 1.175113 1.3339895
## site.sSP 0.8990843 1.067800 1.2365151
## size.t   0.8578292 0.882743 0.9076568
## attr(,"label")
## [1] "Coefficients:"
##
##  Variance function:
##         lower       est.      upper
## FU -0.2824338 -0.2455818 -0.2087297
## SP -0.2490239 -0.2115804 -0.1741368
## attr(,"label")
## [1] "Variance function:"
##
##  Residual standard error:
##    lower     est.    upper
## 1.707969 2.145755 2.695754
```

```
g.intercepts=fit.grow.gls$coef[1:2] ## Growth intercepts for each site
g.slopes=rep(fit.grow.gls$coef[3],2) ## Growth slopes for each site
var.exp.coef=fit.grow.gls$modelStruct$varStruct ## coef for variance structure
sigma.g=fit.grow.gls$sigma ## Residual standard error
```

Plot: Annual Growth

```
FUcol <- "black"
SPcol <- "blue"

ggplot() +
  geom_abline(intercept = g.intercepts[1],slope = g.slopes[1], color=FUcol)+ # FU line
  geom_abline(intercept = g.intercepts[2],slope = g.slopes[2], linetype='longdash', color = SPcol)+ # S
  geom_abline(linetype='dashed') + # 45degree  line
  geom_point(data=growth_df,aes(x=size.t, y=size.t1, shape=site.s, color=site.s)) + # Size points for e
  scale_shape_manual(name = "",values = c(16,1), labels = c("FU: n=735","SP: n=357")) +
  scale_color_manual(name = "",values = c(SPcol, FUcol), labels = c("FU: n=735","SP: n=357"))+
  theme_classic()+
  labs(x="Plant size year t", y="Plant size year t+1")+
  xlim(2,10)+
  ylim(2,10) +
```

```
  theme(legend.position = c(0.8, 0.2),
        legend.text = element_text(size=14))
```



Calculation: Flowering

```
flower_df <- dataf_r1 %>%
  dplyr::select(flow.s = flow.all,
                site.s = site.all,
                size.t = size.t.all) %>%
  filter(complete.cases(.))

attach(flower_df)
# table(site.s)

store.size.flow=size.t[flow.s==1] # store size of plants that flowered
store.site.flow=site.s[flow.s==1] # store sites of plants that flowered

fit.flow.1=glm(flow.s~size.t*site.s, family=binomial) ## Fit flowering to binomial lm
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
fit.flow=glm(flow.s~size.t+site.s, family=binomial)
fit.flow.0=glm(flow.s~size.t, family=binomial)

anova(fit.flow.0,fit.flow,fit.flow.1, test="Chisq") ## Why test with Chisq here specifically?
```

```
## Analysis of Deviance Table
##
```

```
## Model 1: flow.s ~ size.t
## Model 2: flow.s ~ size.t + site.s
## Model 3: flow.s ~ size.t * site.s
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      1156     186.69
## 2      1155     183.50  1   3.1860 0.074271 .
## 3      1154     175.04  1   8.4631 0.003624 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

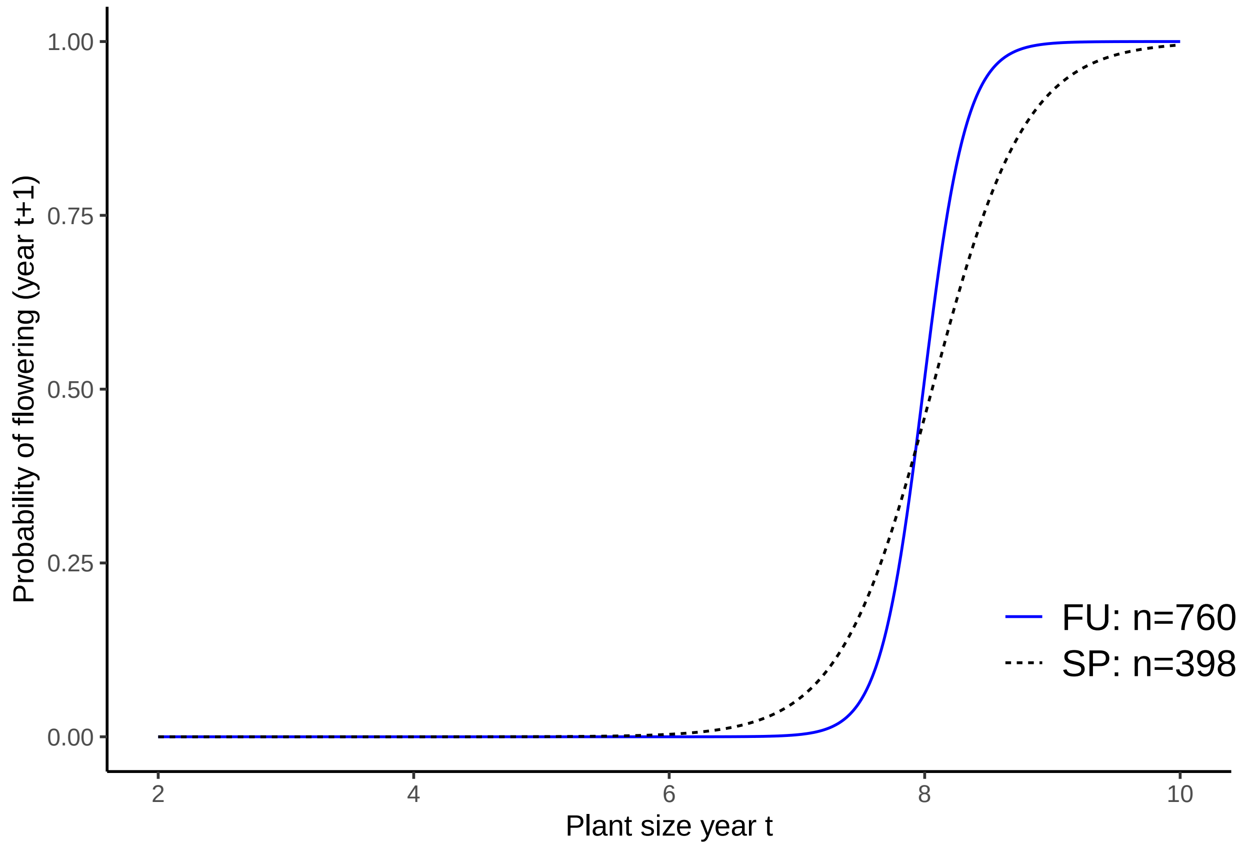
```
# Remove models from environments
rm(fit.flow,fit.flow.1,fit.flow.0)

fit.flow=glm(flow.s~site.s/size.t-1,family=binomial) ## Why '/'  and why -1?
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
f.intercepts=fit.flow$coef[1:2] ## Model intercepts
f.slopes=c(fit.flow$coef[3:4]) ## Model slopes

site.flow.SE=summary(fit.flow)$coef[5:6]
```

Plot: Flowering

```
flower_df$probability <- predict(fit.flow, site.s=flower_df$site.s, size.t = flower_df$size.t, type = "
predict_df <- data.frame(site.s=c(rep("FU",1000),rep("SP",1000)),
                         size.t=rep(seq(2,10,length.out = 1000),2))
predict_df <- cbind(predict_df,predict(fit.flow, newdata = predict_df, type = "response", se.fit = TRUE


ggplot(data = predict_df, aes(x = size.t, y = fit)) +
  geom_line(aes(color = site.s, linetype=site.s)) +
  scale_color_manual(name = "", values=c(SPcol, FUcol),labels = c("FU: n=760","SP: n=398"))+
  scale_linetype_discrete(name = "",labels = c("FU: n=760","SP: n=398"))+
  scale_x_continuous(limits = c(2, 10)) +
  # geom_point(data=flower_df,aes(x=size.t,y=probability))+
  theme_classic()+
  labs(x="Plant size year t", y = "Probability of flowering (year t+1)")+
  theme(legend.position = c(0.9, 0.2),
        legend.text = element_text(size=14))
```




```r
### Don't understand where their points around the lines come from. Is that related to binning?
detach(flower_df)
```

Calculation: Survival

```r
survival_df <- dataf_r1 %>%
  dplyr::select(surv.s = surv.all,
                site.s = site.all,
                size.t = size.t.all) %>%
  filter(complete.cases(.))

attach(survival_df)
# table(site.s)

## 3 models to compare
fit.surv.1=glm(surv.s~size.t*site.s, family=binomial)
fit.surv=glm(surv.s~size.t+site.s, family=binomial)
fit.surv.0=glm(surv.s~size.t, family=binomial)

anova(fit.surv.0,fit.surv,fit.surv.1,test="Chisq")
```

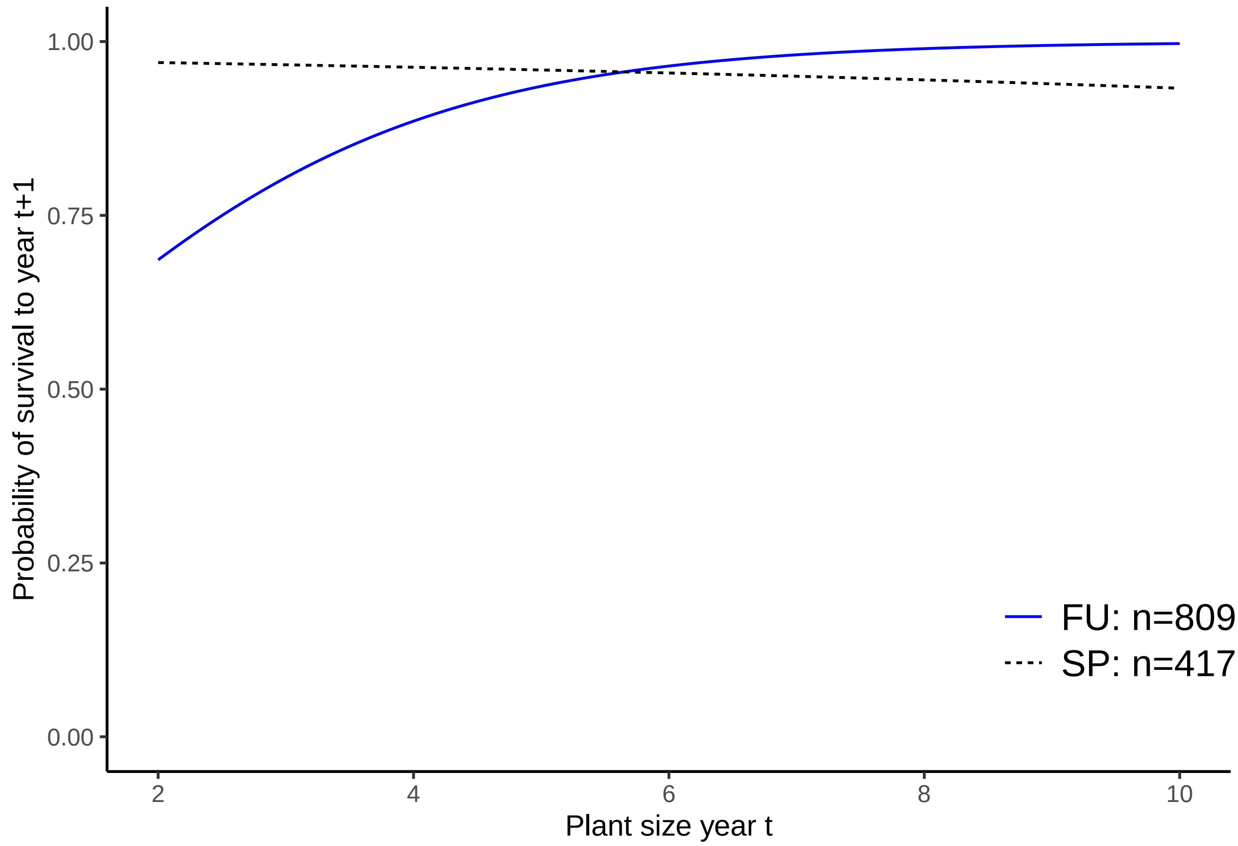
```
## Analysis of Deviance Table
##
## Model 1: surv.s ~ size.t
## Model 2: surv.s ~ size.t + site.s
## Model 3: surv.s ~ size.t * site.s
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
```

```
## 1      1224     505.42
## 2      1223     505.15  1   0.2647 0.606939
## 3      1222     492.80  1  12.3583 0.000439 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
rm(fit.surv.0,fit.surv,fit.surv.1)
fit.surv = glm(surv.s~site.s/size.t-1, family=binomial) ## Same as above -- why '/' and why -1?

s.intercepts = fit.surv$coef[1:2]
s.slopes = c(fit.surv$coef[3:4])
```

Plot: Survival

```
survival_df$probability <- predict(fit.surv, site.s=site.s, size.t = size.t, type = "response")

predict_df <- data.frame(site.s=c(rep("FU",1000),rep("SP",1000)),
                         size.t=rep(seq(2,10,length.out = 1000),2))
predict_df <- cbind(predict_df,predict(fit.surv, newdata = predict_df, type = "response", se.fit = TRUE)

ggplot(data = predict_df, aes(x = size.t, y = fit)) +
  geom_line(aes(color = site.s, linetype=site.s)) +
  scale_color_manual(name = "", values=c(SPcol, FUcol),labels = c("FU: n=809","SP: n=417"))+
  scale_linetype_discrete(name = "",labels = c("FU: n=809","SP: n=417"))+
  scale_x_continuous(limits = c(2, 10)) +
  scale_y_continuous(limits = c(0,1))+
  # geom_point(data=flower_df,aes(x=size.t,y=probability))+
  theme_classic()+
  labs(x="Plant size year t", y = "Probability of survival to year t+1")+
  theme(legend.position = c(0.9, 0.2),
        legend.text = element_text(size=14))
```




```
### Don't understand where their points around the lines come from. Is that related to binning?
detach(survival_df)
```

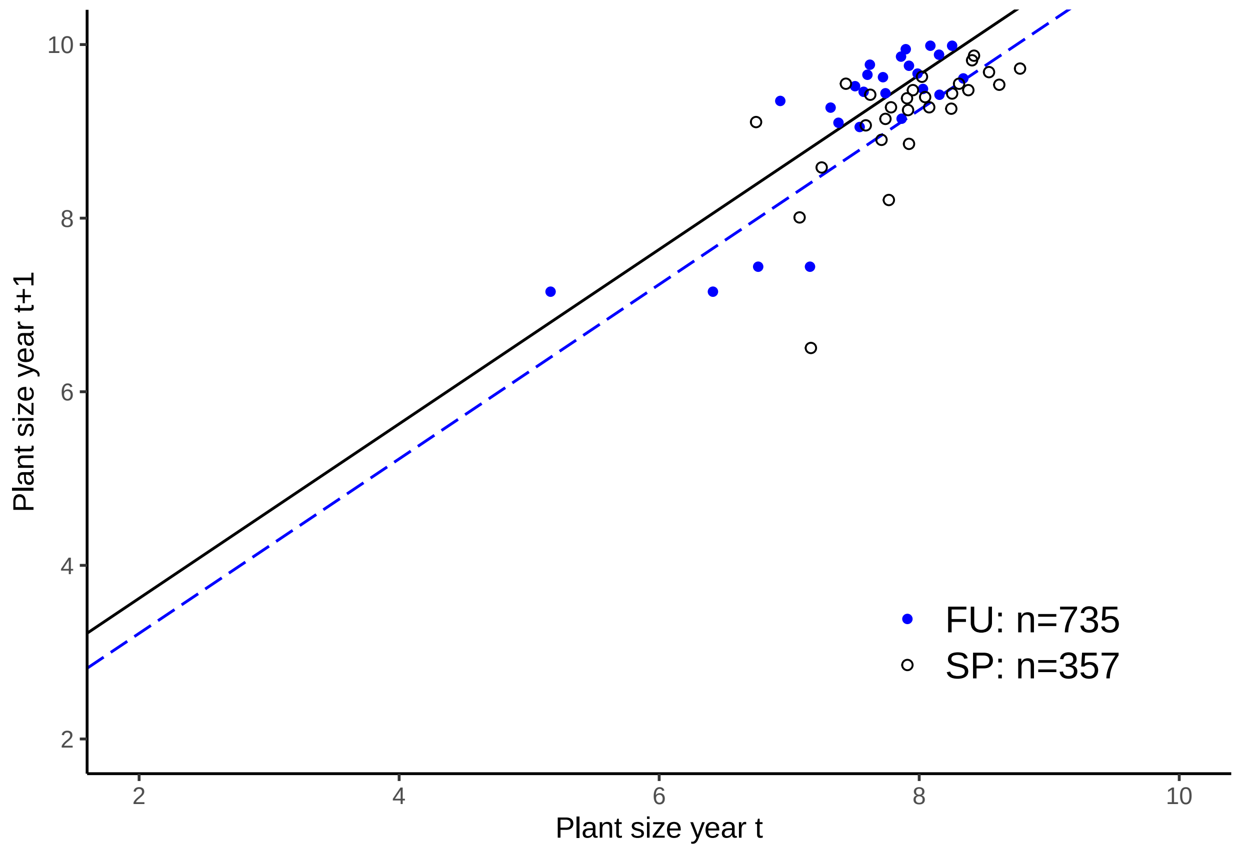
Calculation: Fecundity

```
IPM.fecundity=data.frame(read.table("CT.IPM.fecundity.txt", header=T))
# names(IPM.fecundity)

# Unbrowsed individuals
fecundity_df <- IPM.fecundity %>%
  filter(brows.t1==0) %>%
  dplyr::select(size.t.f=size.t,
                site.t.f=site,
                si.t1.f=si.t1,
                nl.t.f=nl.t)

attach(fecundity_df)

fit.fec = lm(log(si.t1.f)~ ## dep var = log of seeds per individual (unbrowsed) in year t+1
               site.t.f + ## site of unbrowsed individuals
               size.t.f-1) ## size of unbrowsed Why -1? #
```

Plot: Fecundity

```
ggplot() +
  geom_abline(intercept = fit.fec$coefficients[1],slope = fit.fec$coefficients[3], color=FUcol)+ # FU l
  geom_abline(intercept = fit.fec$coefficients[2],slope = fit.fec$coefficients[3], linetype='longdash',
```

```
  geom_point(data=fecundity_df,aes(x=size.t.f, y=log(si.t1.f), shape=site.t.f, color=site.t.f)) + # Siz
  scale_shape_manual(name = "",values = c(16,1), labels = c("FU: n=735","SP: n=357")) +
  scale_color_manual(name = "",values = c(SPcol, FUcol), labels = c("FU: n=735","SP: n=357"))+
  theme_classic()+
  labs(x="Plant size year t", y="Plant size year t+1")+
  xlim(2,10)+
  ylim(2,10) +
  theme(legend.position = c(0.8, 0.2),
        legend.text = element_text(size=14))
```




```
detach(fecundity_df)
```

Seedling sizes

```
####### Seedlings data #######
IPM.seedlings=data.frame(read.table("CT.IPM.seedlings.txt",header=T)) %>%
  filter(site != "JU") %>%
  dplyr::select(seedlings.size.t=size.t,
                seedlings.site=site)

fit.seedlings=lm(seedlings.size.t~seedlings.site-1, data = IPM.seedlings) ## That -1! What is that?
# summary(fit.seedlings)

## Add seedlings to all.sizes vector
size.all.plus.seedlings=c(all.sizes, IPM.seedlings$size.t[IPM.seedlings$year.t!=2003])
## Add site codes for seedlings to all sites vector
site.all.plus.seedlings=c(all.sizes, IPM.seedlings$site[IPM.seedlings$year.t!=2003])
```

```r
####### Establishment data #######
IPM.establishment=data.frame(read.table("IPM.establishment.data.txt",header=T)) %>%
  dplyr::select(p.est.site=site,
                p.est.seeds.t=seeds.t,
                p.est.seedlings=sdl.t1)

attach(IPM.establishment)

## Site specific seedling establishment rates
est.p.est = IPM.establishment %>%
  group_by(p.est.site) %>%
  summarise(est_rate=sum(p.est.seedlings)/sum(p.est.seeds.t))
est.p.est = as.numeric(est.p.est$est_rate)

detach(IPM.establishment)
```

Collect parameters

```r
## Set upper and lower plant size bounds
minsize<-1
maxsize<-12

# Global variables for midpoint rule approximation
# n.big.matrix is the number of mesh points for size, n.age is the number of age classes.
n.big.matrix = 100; n.age = 50; n=n.big.matrix ## I think in paper they said 100 mesh points

L= minsize; U= maxsize;

# boundary points b and mesh points y
b = L+c(0:n)*(U-L)/n
y = 0.5*(b[1:n]+b[2:(n+1)])

# step size for midpoint rule, see equations 4 and 5
h = y[2]-y[1]

## initialize empty parameter matrix
store.p.vec = matrix(NA,12,2)
colnames(store.p.vec) <- c("FU","SP")
p.vec.names <- c("1st survival param","2nd survival param",
                 "1st flow param    ","2nd flow param    ",
                 "ag                ","bg                ",
                 "sigma2 growth     ","intercept seeds   ",
                 "slope seeds       ","mean kids size   ",
                 "sigma2 kids size ","growth variance parameter ")

## initialize empty parameter matrix
store.p.vec = matrix(NA,12,2)
colnames(store.p.vec) <- c("FU","SP")

## Store parameters in store.p.vec 12x2 array
for(i in 1:2){

    store.p.vec[1,i]<- as.numeric(s.intercepts[i])      # ; p.vec.names[1]<-"1st survival param";
```

```
    store.p.vec[2,i]<- s.slopes[i]              # ; p.vec.names[2]<-"2nd survival param";
    store.p.vec[3,i]<- f.intercepts[i]      #   ; p.vec.names[3]<-"1st flow param    ";
    store.p.vec[4,i]<- f.slopes[i]              # ; p.vec.names[4]<-"2nd flow param    ";
    store.p.vec[5,i]<- g.intercepts[i]      #   ; p.vec.names[5]<-"ag                ";
    store.p.vec[6,i]<- g.slopes[i]              # ; p.vec.names[6]<-"bg                ";
    store.p.vec[7,i]<- sigma.g^2                # ; p.vec.names[7]<-"sigma2 growth     ";
    store.p.vec[8,i]<- fit.fec$coef[i]   #          ; p.vec.names[8]<-"intercept seeds   ";
    store.p.vec[9,i]<- fit.fec$coef[3]   #      ; p.vec.names[9]<-"slope seeds       ";
    store.p.vec[10,i]<- fit.seedlings$coef[i]   #   ; p.vec.names[10]<-"mean kids size   ";
    store.p.vec[11,i]<- summary(fit.seedlings)$sigma^2  # ; p.vec.names[11]<-"sigma2 kids size ";
    store.p.vec[12,i]<- var.exp.coef[i]         # ; p.vec.names[12]<-"growth variance parameter ";

}
# summary(store.p.vec)
```

Part II. Compute the kernel component functions from the fitted models

Basic demographic functions

```
## Survival model

sx <- function(x,params) {
    u <- exp(params[2]*x+params[1]); ## Logit model => odds
    return(u/(1+u)); ## odds to prob
}

## Fecundity model
fx<-function(x,params) {
    u<-exp(params[3]+params[4]*x) ## Logit model => odds
      return(u/(1+u))  ## odds to prob
}

## Growth model
gxy<-function(x,y,params) {
    mux<-params[5]+params[6]*x; ## growth slope and intercepts
    sigmax2<-(params[7])*exp(2*(params[12]*mux)) ## Variance around growth curve
    sigmax<-sqrt(sigmax2); ## Standard deviation around growth curve
    fac1<-sqrt(2*pi)*sigmax; ## ??
    fac2<-((y-mux)^2)/(2*sigmax2); ## ??
    return(exp(-fac2)/fac1); ## ??
}

## Survival-growth function
pxy<-function(x,y,params) {
  return(sx(x,params)*(1-fx(x,params))*gxy(x,y,params))}

## Fecundity function
fxy<-function(x,y,params) {
    nkids<-p.est*exp(params[8]+params[9]*x) ## 8: seeds intercept; 9: seeds slope
    kidsize.mean<- params[10] ## Mean seedling size
    kidsize.var<- params[11] ## Sigma seedling size
    fac1<-sqrt(2*pi)*sqrt(kidsize.var) ## Same Q as 483 above
    fac2<-((y-kidsize.mean)^2)/(2*kidsize.var) ## 484 above
    f<-sx(x,params)*fx(x,params)*nkids*exp(-fac2)/fac1 ## End is 485 above
```

```
    return(f);
}
```

The 'big matrix' M of size n x n

```
bigmatrix<-function(n,params) {

  # upper and lower integration limits
    L<-minsize; U<-maxsize;

  # boundary points b and mesh points y
    b<-L+c(0:n)*(U-L)/n; # Boundaries
    y<-0.5*(b[1:n]+b[2:(n+1)]); # Midpoints

  # construct the matrix
  I <- diag(n); ## Identity matrix

  ## Survival-growth matrix for all midpoint sizes
    P<-t(outer(y,y,pxy,params=params)) # P: survival-growth transpose outer product. Why transpose?

    ## fecundity matrix for all midpoint sizes
    B<-t(outer(y,y,fxy,params=params)) # B: fecundity

    ## Empty matrix of zeroes
    M=array(0,dim=c(2*n,2*n))

    ## Insert P matrix
    M[1:n,1:n]=P*(U-L)/n

    ## Insert B matrix to the right of the P matrix
    M[1:n,(n+1):(2*n)]=B*(U-L)/n

    ## Insert the identity matrix
    M[(n+1):(2*n),1:n]=diag(n)
    K<-M ## Call K to match paper?
  P<-(U-L)*P/n
  B<-(U-L)*B/n
    return(list(matrix=M,
                kernel=K,
                meshpts=y,
                Pmatrix=P,
                Bmatrix=B,
                Imatrix=I));
}


R0.calc<-function(n,params){

  M<-bigmatrix(n,params) ## Construct matrix

  ## If any NAs in matrix, return NA for estimates
    if (any(is.na(M$matrix))){
        ave.R0=NA
        lam=NA
```

```
        T=NA }

  else{
        N <- solve(M$Imatrix-M$Pmatrix) ## ??
        R <- M$Bmatrix %*% N ## Matrix multiplication
        ave.R0<-Re(eigen(R)$values[1]) ##
        lam<-Re(eigen(M$matrix)$values[1]);
        T=log(ave.R0)/log(lam)
    }

    return(list(lam=lam,ave.R0=ave.R0,T=T))
}

R0.betas<-function(x){

    p.vec[3] <- x;
    nR0 <- R0.calc(n.big.matrix, p.vec)
    return(nR0$ave.R0)

}
```

Calculation: generation time

```
gen.time=rep(NA,2)

for(i in 1:2){
    #if(i==1) p.est= 8.604605e-05 else p.est=0.0001655622  # assuming dd-reg
    if(i==1) p.est= est.p.est[1] else p.est=est.p.est[2]   # actual
    p.vec=store.p.vec[,i]
    tmp=R0.calc(n.big.matrix,p.vec)
    gen.time[i]=tmp$T
    cat("Site ",i," lambda=",tmp$lam," R0=",tmp$ave.R0," Generation time=",tmp$T,"\n")
    cat("ESS intercept ", optimize(R0.betas, c(-100,10), maximum=T, tol=0.01)$maximum,"\n")
}
```

```
## Site  1  lambda= 1.048766  R0= 1.675448  Generation time= 10.83887
## ESS intercept  -64.51811
## Site  2  lambda= 1.167493  R0= 4.989201  Generation time= 10.37898
## ESS intercept  -26.50841
```

Calculation: Evolutionarily stable strategy

```
n.test <- 100
R0.beta <- array(NA,dim=c(n.test,2));
lam.beta <- array(NA,dim=c(n.test,2));
ESS = rep(NA,2)
```

Plot: Evolutionarily stable strategy

```
dev.new()
par(mfrow=c(2,2), mar=c(3,3,1,2)+0.1, bty="l",pty="s", cex.main=1, cex.axis=1, cex.lab=1, tck=0.02, mgp=

for(i in 1:2){
    p.vec=store.p.vec[,i]
```

```
    #if(i==1) p.est= 8.604605e-05 else p.est=0.0001655622  # assuming dd-reg
    if(i==1) p.est= est.p.est[1] else p.est=est.p.est[2]   # actual
    if(i==1) beta.flow<-seq(-100,0,length=n.test) else beta.flow<-seq(-50,0,length=n.test);

    for(beta.test in 1:n.test){
        p.vec[3]<-beta.flow[beta.test];
        nR0<-R0.calc(n.big.matrix,p.vec)
        R0.beta[beta.test,i]<-nR0$ave.R0
        lam.beta[beta.test,i]<-nR0$lam
        cat(beta.flow[beta.test],"  ",nR0$ave.R0,"  ",nR0$lam,"\n")
    }

    ESS[i]<-beta.flow[R0.beta[,i]==max(R0.beta[,i])]

    plot(beta.flow,R0.beta[,i],type="n",xlab=expression("Intercept of flowering function " * italic(beta
    ylab=expression(italic("R"*scriptstyle(0))))
    min.R0=min(R0.beta[,i]); max.R0=max(R0.beta[,i])
    mean.m2se=fit.flow$coef[i]-2*site.flow.SE[i]
    mean.p2se=fit.flow$coef[i]+2*site.flow.SE[i]

    polygon(c(mean.m2se,mean.p2se,mean.p2se,mean.m2se),c(min.R0,min.R0,max.R0,max.R0), col="grey90",bord
    points(beta.flow,R0.beta[,i],type="l")
    abline(h=1)
    points(beta.flow[R0.beta[,i]==max(R0.beta[,i])],max(R0.beta[,i]),pch=19)
    abline(v=fit.flow$coef[i])
    #abline(v=beta.flow[R0.beta==max(R0.beta)])

    # if (i==1) text(2.3,10,"a)") else text(locator(1),"c)")

    plot(beta.flow,lam.beta[,i],type="n",xlab=expression("Intercept of flowering function " * italic(bet
    ylab=expression(italic(lambda)))
    min.R0=min(lam.beta[,i]);max.R0=max(lam.beta[,i])
    mean.m2se=fit.flow$coef[i]-2*site.flow.SE[i]
    mean.p2se=fit.flow$coef[i]+2*site.flow.SE[i]

    polygon(c(mean.m2se,mean.p2se,mean.p2se,mean.m2se),c(min.R0,min.R0,max.R0,max.R0), col="grey90",bord
    points(beta.flow,lam.beta[,i],type="l")
    abline(h=1)
    abline(v=fit.flow$coef[i])
    points(beta.flow[lam.beta[,i]==max(lam.beta[,i])],max(lam.beta[,i]),pch=19)

    # if (i==1) text(locator(1),"b)") else text(locator(1),"d)")
}
```

```
## -100    1.16473e-13    0.997046
## -98.9899    3.198206e-13    0.997046
## -97.9798    8.781885e-13    0.997046
## -96.9697    2.411399e-12    0.997046
## -95.9596    6.621408e-12    0.997046
## -94.94949    1.818158e-11    0.997046
## -93.93939    4.992441e-11    0.997046
## -92.92929    1.370863e-10    0.997046
## -91.91919    3.764224e-10    0.997046
## -90.90909    1.03361e-09    0.997046
```

```
## -89.89899    2.838168e-09    0.997046
## -88.88889    7.793264e-09    0.997046
## -87.87879    2.139935e-08    0.997046
## -86.86869    5.876003e-08    0.997046
## -85.85859    1.613479e-07    0.997046
## -84.84848    4.430416e-07    0.997046
## -83.83838    1.216538e-06    0.997046
## -82.82828    3.340466e-06    0.997046
## -81.81818    9.17251e-06    0.997046
## -80.80808    2.518657e-05    0.9970461
## -79.79798    6.915904e-05    0.9970462
## -78.78788    0.0001899007    0.9970466
## -77.77778    0.0005214323    0.9970476
## -76.76768    0.001431699    0.9970505
## -75.75758    0.00393059    0.9970582
## -74.74747    0.01078779    0.9970795
## -73.73737    0.0295833    0.9971377
## -72.72727    0.08094238    0.9972961
## -71.71717    0.2201064    0.9977209
## -70.70707    0.5888089    0.998818
## -69.69697    1.510893    1.001419
## -68.68687    3.52426    1.006689
## -67.67677    6.878034    1.01531
## -66.66667    10.54698    1.026729
## -65.65657    12.95905    1.03955
## -64.64646    13.74729    1.052374
## -63.63636    13.4378    1.064228
## -62.62626    12.56721    1.074592
## -61.61616    11.46166    1.083272
## -60.60606    10.29432    1.090246
## -59.59596    9.153663    1.095573
## -58.58586    8.083256    1.099331
## -57.57576    7.10254    1.1016
## -56.56566    6.217766    1.102447
## -55.55556    5.427946    1.101926
## -54.54545    4.728213    1.10008
## -53.53535    4.111768    1.096941
## -52.52525    3.571017    1.092533
## -51.51515    3.098243    1.086875
## -50.50505    2.685985    1.079979
## -49.49495    2.327252    1.071858
## -48.48485    2.015622    1.06252
## -47.47475    1.745281    1.051973
## -46.46465    1.511018    1.040226
## -45.45455    1.3082    1.02729
## -44.44444    1.132732    1.013174
## -43.43434    0.9810127    0.9978946
## -42.42424    0.8498843    0.9814674
## -41.41414    0.7365897    0.9639132
## -40.40404    0.6387255    0.9452567
## -39.39394    0.554202    0.9255267
## -38.38384    0.4812048    0.904757
## -37.37374    0.4181612    0.8829861
## -36.36364    0.3637101    0.8602578
```

```
## -35.35354    0.3166746    0.8366212
## -34.34343    0.2760385    0.8121305
## -33.33333    0.2409241    0.7868449
## -32.32323    0.2105718    0.7608277
## -31.31313    0.1843216    0.7341447
## -30.30303    0.1615966    0.7068613
## -29.29293    0.1418898    0.6790398
## -28.28283    0.1247565    0.6507357
## -27.27273    0.1098094    0.6219962
## -26.26263    0.09671973    0.5928605
## -25.25253    0.08521892    0.5633638
## -24.24242    0.07509906    0.5335447
## -23.23232    0.06620974    0.5034547
## -22.22222    0.05844905    0.4731671
## -21.21212    0.05174941    0.4427836
## -20.20202    0.04606048    0.4124346
## -19.19192    0.04133249    0.3822748
## -18.18182    0.0375035    0.3524763
## -17.17172    0.03449286    0.3232269
## -16.16162    0.03220137    0.2947445
## -15.15152    0.03051691    0.2673229
## -14.14141    0.02932314    0.2414292
## -13.13131    0.02850853    0.21787
## -12.12121    0.02797379    0.1979538
## -11.11111    0.02763632    0.18319
## -10.10101    0.02743164    0.1740208
## -9.090909    0.02731238    0.169147
## -8.080808    0.02724569    0.1667808
## -7.070707    0.02721006    0.1656847
## -6.060606    0.02719211    0.1651964
## -5.050505    0.0271838    0.1649898
## -4.040404    0.0271803    0.1649075
## -3.030303    0.02717894    0.1648762
## -2.020202    0.02717843    0.1648646
## -1.010101    0.02717824    0.1648604
## 0    0.02717817    0.1648588
```

```
## -50    1.48195e-08    0.9385119
## -49.49495    2.455693e-08    0.9385119
## -48.9899    4.069254e-08    0.9385119
## -48.48485    6.743035e-08    0.9385119
## -47.9798    1.117368e-07    0.938512
## -47.47475    1.851555e-07    0.938512
## -46.9697    3.068155e-07    0.938512
## -46.46465    5.084146e-07    0.9385121
## -45.9596    8.424781e-07    0.9385121
## -45.45455    1.396044e-06    0.9385123
## -44.94949    2.313342e-06    0.9385125
## -44.44444    3.833367e-06    0.9385129
## -43.93939    6.352151e-06    0.9385136
## -43.43434    1.052595e-05    0.9385147
## -42.92929    1.74422e-05    0.9385165
## -42.42424    2.890289e-05    0.9385196
## -41.91919    4.789394e-05    0.9385246
```

```
## -41.41414    7.936319e-05    0.9385329
## -40.90909    0.0001315092    0.9385467
## -40.40404    0.0002179169    0.9385695
## -39.89899    0.0003610954    0.9386072
## -39.39394    0.0005983378    0.9386696
## -38.88889    0.0009914262    0.9387724
## -38.38384    0.001642694    0.9389415
## -37.87879    0.002721599    0.9392182
## -37.37374    0.004508618    0.9396674
## -36.86869    0.007467635    0.9403878
## -36.36364    0.01236491    0.9415222
## -35.85859    0.02046359    0.9432617
## -35.35354    0.03383874    0.9458367
## -34.84848    0.05588025    0.9494867
## -34.34343    0.09207507    0.9544165
## -33.83838    0.1511721    0.9607581
## -33.33333    0.2467858    0.9685545
## -32.82828    0.3992911    0.9777649
## -32.32323    0.6373335    0.9882781
## -31.81818    0.9973315    0.9999254
## -31.31313    1.518324    1.01249
## -30.80808    2.229837    1.025722
## -30.30303    3.134172    1.039348
## -29.79798    4.191259    1.0531
## -29.29293    5.317838    1.066723
## -28.78788    6.406001    1.079997
## -28.28283    7.352502    1.092734
## -27.77778    8.083237    1.104783
## -27.27273    8.563191    1.116022
## -26.76768    8.792736    1.126354
## -26.26263    8.796913    1.1357
## -25.75758    8.613935    1.143993
## -25.25253    8.286141    1.151173
## -24.74747    7.85416    1.157188
## -24.24242    7.353798    1.161987
## -23.73737    6.814799    1.16552
## -23.23232    6.260753    1.167739
## -22.72727    5.709613    1.168595
## -22.22222    5.174476    1.168041
## -21.71717    4.664429    1.166029
## -21.21212    4.185353    1.16251
## -20.70707    3.740635    1.157438
## -20.20202    3.331777    1.150769
## -19.69697    2.958885    1.142459
## -19.19192    2.621074    1.132466
## -18.68687    2.316779    1.120754
## -18.18182    2.043993    1.107287
## -17.67677    1.800454    1.092032
## -17.17172    1.583781    1.074961
## -16.66667    1.391574    1.056047
## -16.16162    1.221489    1.035267
## -15.65657    1.07129    1.012601
## -15.15152    0.9388959    0.9880354
## -14.64646    0.8224087    0.9615654
```

```
## -14.14141    0.7201338    0.9332053
## -13.63636    0.6305901    0.9029966
## -13.13131    0.5525058    0.8710213
## -12.62626    0.4847991    0.8374158
## -12.12121    0.4265443    0.8023829
## -11.61616    0.376928    0.7662007
## -11.11111    0.3352022    0.729227
## -10.60606    0.3006418    0.691897
## -10.10101    0.272514    0.6547205
## -9.59596    0.2500649    0.6182834
## -9.090909    0.232524    0.5832633
## -8.585859    0.2191217    0.5504623
## -8.080808    0.2091155    0.520831
## -7.575758    0.2018178    0.4953999
## -7.070707    0.1966176    0.4750001
## -6.565657    0.1929948    0.4598438
## -6.060606    0.190525    0.4493551
## -5.555556    0.188875    0.4424759
## -5.050505    0.1877929    0.4381176
## -4.545455    0.187095    0.4354125
## -4.040404    0.1866515    0.4337531
## -3.535354    0.186373    0.4327424
## -3.030303    0.1862    0.4321293
## -2.525253    0.1860935    0.4317583
## -2.020202    0.1860282    0.4315342
## -1.515152    0.1859884    0.4313989
## -1.010101    0.1859643    0.4313172
## -0.5050505    0.1859497    0.4312679
## 0    0.1859408    0.4312381
```

Constructing the component matrices and their transposes

```
# Put all component matrices into 3-dimensional arrays
P <- array(NA,dim=c(n.big.matrix,n.big.matrix)) #P[j,i,a] will be h*P_{a-1}(x_j,x_i)
B <- array(NA,dim=c(n.big.matrix,n.big.matrix)) #B[j,i,a] will be h*F_{a-1}(x_j,x_i)

stable.dist=array(NA,dim=c(n,n.age,2))
lam.stable.age=rep(NA,2);

for(i in 1:2){
    p.vec=store.p.vec[,i]
    if(i==1) p.est=est.p.est[1] else p.est=est.p.est[2]
    P<-h*t(outer(y,y,pxy,params=p.vec))
    B<-h*t(outer(y,y,fxy,params=p.vec))

#============================================================================#
#   Model iteration functions
#============================================================================#

    # population now and next year
    Nt=matrix(0,n.big.matrix,n.age);
    Nt1=Nt
    Nt2=Nt
```

```r
    iteration=function(Nt1,Nt){
        for(age in 2:n.age){
            Nt2[,age]=P%*%Nt1[,age-1]
            }
        Nt2[,1]=0;

        for(age in 1:n.age){
            Nt2[,1]=Nt2[,1]+B%*%Nt[,age]
            }
        return(Nt2)
        }

  #==============================================================================================#
  # Start using the model
  #==============================================================================================#

  # Estimate lambda and w by iterating unperturbed matrix
    Nt1=matrix(1,n.big.matrix,n.age);
    Nt=Nt1
    qmax=1000;
    lam=1;
    tol=1.e-8;
    while(qmax>tol) {
        Nt2=iteration(Nt1,Nt);
        qmax=sum(abs(Nt2-lam*Nt1));
        lam=sum(Nt2)/sum(Nt1);

        Nt=Nt1
        Nt1=Nt2

        tot=sum(Nt1+Nt2)

        Nt=Nt/tot
        Nt1=Nt1/tot

        cat(lam,qmax,"\n");
    }

    stable.dist[,,i]=Nt/sum(Nt); lam.stable.age[i]=lam;

}
```

```
## 13.10921 65390.92
## 1.782144 6.443682
## 0.8532779 0.853326
## 0.9019406 0.4638502
## 0.9322872 0.4690996
## 0.947794 0.4705897
## 0.9632477 0.4740592
## 1.003417 0.4810223
## 1.070745 0.4702969
## 1.126989 0.4238015
## 1.139101 0.316531
## 1.114777 0.237085
```

```
## 1.07756 0.1932876
## 1.043481 0.1638186
## 1.019821 0.1393329
## 1.009401 0.1149309
## 1.012472 0.09301079
## 1.026331 0.08650453
## 1.045044 0.08517669
## 1.061324 0.08074008
## 1.069947 0.07018763
## 1.069803 0.05377798
## 1.063193 0.04651208
## 1.053809 0.04348423
## 1.045093 0.04060368
## 1.039412 0.0359911
## 1.037784 0.02940216
## 1.039884 0.02478934
## 1.044328 0.02328394
## 1.049235 0.0223247
## 1.052936 0.02035471
## 1.054513 0.01692887
## 1.053949 0.01344377
## 1.051912 0.01211758
## 1.049365 0.0114507
## 1.047198 0.01062042
## 1.045986 0.009294319
## 1.045895 0.007333126
## 1.046713 0.006404357
## 1.047992 0.006082099
## 1.049235 0.005764814
## 1.05006 0.005139516
## 1.050301 0.004170206
## 1.050014 0.003470244
## 1.049404 0.003218434
## 1.048731 0.003058625
## 1.048216 0.002781568
## 1.047984 0.002326089
## 1.048044 0.001838944
## 1.048315 0.001658408
## 1.048666 0.0015765
## 1.048972 0.001481695
## 1.049146 0.001301804
## 1.049165 0.001029269
## 1.049056 0.0008915237
## 1.048881 0.0008422209
## 1.048708 0.000797148
## 1.048591 0.0007123052
## 1.048554 0.0005809545
## 1.04859 0.0004801131
## 1.048673 0.0004444454
## 1.048766 0.000422889
## 1.048839 0.0003857617
## 1.048873 0.0003236903
## 1.048866 0.0002553406
## 1.04883 0.0002285865
```

```
## 1.048782 0.0002164064
## 1.048739 0.0002050192
## 1.048714 0.0001808462
## 1.04871 0.0001437799
## 1.048725 0.0001237995
## 1.048749 0.0001166703
## 1.048773 0.0001105277
## 1.048789 9.904714e-05
## 1.048795 8.109299e-05
## 1.04879 6.654436e-05
## 1.048779 6.134535e-05
## 1.048766 5.834867e-05
## 1.048756 5.334498e-05
## 1.048751 4.499008e-05
## 1.048752 3.548039e-05
## 1.048756 3.163463e-05
## 1.048763 2.975346e-05
## 1.048769 2.837511e-05
## 1.048773 2.512998e-05
## 1.048773 2.007998e-05
## 1.048771 1.718272e-05
## 1.048768 1.61476e-05
## 1.048765 1.530626e-05
## 1.048762 1.375426e-05
## 1.048762 1.130576e-05
## 1.048762 9.21359e-06
## 1.048764 8.460083e-06
## 1.048765 8.045673e-06
## 1.048767 7.373232e-06
## 1.048768 6.282462e-06
## 1.048768 4.927107e-06
## 1.048767 4.374882e-06
## 1.048766 4.13828e-06
## 1.048765 3.923612e-06
## 1.048765 3.488803e-06
## 1.048765 2.801797e-06
## 1.048765 2.382936e-06
## 1.048765 2.233192e-06
## 1.048766 2.118165e-06
## 1.048766 1.908677e-06
## 1.048766 1.575056e-06
## 1.048766 1.274764e-06
## 1.048766 1.165805e-06
## 1.048766 1.108504e-06
## 1.048765 1.01828e-06
## 1.048765 8.765463e-07
## 1.048765 6.836776e-07
## 1.048765 6.045458e-07
## 1.048766 5.751355e-07
## 1.048766 5.433435e-07
## 1.048766 4.839897e-07
## 1.048766 3.906306e-07
## 1.048766 3.302194e-07
## 1.048766 3.086081e-07
```

```
## 1.048766 2.928951e-07
## 1.048766 2.646608e-07
## 1.048766 2.193023e-07
## 1.048766 1.762856e-07
## 1.048766 1.605412e-07
## 1.048766 1.526048e-07
## 1.048766 1.405199e-07
## 1.048766 1.222005e-07
## 1.048766 9.544158e-08
## 1.048766 8.395716e-08
## 1.048766 7.987022e-08
## 1.048766 7.550258e-08
## 1.048766 6.709055e-08
## 1.048766 5.441889e-08
## 1.048766 4.572556e-08
## 1.048766 4.261378e-08
## 1.048766 4.04695e-08
## 1.048766 3.669203e-08
## 1.048766 3.052352e-08
## 1.048766 2.436389e-08
## 1.048766 2.209321e-08
## 1.048766 2.099475e-08
## 1.048766 1.948496e-08
## 1.048766 1.702282e-08
## 1.048766 1.336593e-08
## 1.048766 1.168279e-08
## 1.048766 1.108323e-08
## 1.048766 1.048373e-08
## 1.048766 9.332066e-09
## 37.13518 184984.8
## 1.943714 18.53056
## 0.9649965 0.9545099
## 0.9581129 0.4774575
## 0.9646732 0.4797136
## 1.010824 0.4943921
## 1.110175 0.5098155
## 1.21606 0.4996874
## 1.264506 0.4013
## 1.252516 0.2868261
## 1.214239 0.2185538
## 1.176268 0.1692032
## 1.150734 0.1300536
## 1.140786 0.09714775
## 1.144134 0.07317022
## 1.155008 0.06083462
## 1.166672 0.05115002
## 1.174303 0.040338
## 1.17646 0.02917502
## 1.174442 0.02208808
## 1.170636 0.01821929
## 1.167134 0.01500802
## 1.165099 0.01169302
## 1.164718 0.008323485
## 1.16551 0.006481437
```

```
## 1.166739 0.005499711
## 1.167784 0.004589373
## 1.168325 0.003541853
## 1.168351 0.002530865
## 1.168049 0.002022336
## 1.167656 0.001686244
## 1.167351 0.001393991
## 1.167215 0.001059406
## 1.167234 0.0007745915
## 1.167341 0.0006290405
## 1.167464 0.0005233023
## 1.167552 0.0004194298
## 1.167584 0.0003137264
## 1.16757 0.0002345326
## 1.167533 0.0001931044
## 1.167495 0.0001605005
## 1.167471 0.0001258364
## 1.167464 9.124331e-05
## 1.16747 6.980583e-05
## 1.167483 5.828009e-05
## 1.167494 4.850466e-05
## 1.167501 3.796426e-05
## 1.167502 2.701244e-05
## 1.167499 2.107237e-05
## 1.167495 1.788132e-05
## 1.167492 1.490396e-05
## 1.16749 1.149088e-05
## 1.16749 8.227082e-06
## 1.167491 6.588804e-06
## 1.167492 5.493176e-06
## 1.167493 4.542759e-06
## 1.167494 3.449993e-06
## 1.167494 2.526405e-06
## 1.167493 2.05367e-06
## 1.167493 1.708335e-06
## 1.167493 1.365972e-06
## 1.167493 1.020659e-06
## 1.167493 7.642516e-07
## 1.167493 6.299431e-07
## 1.167493 5.236405e-07
## 1.167493 4.103052e-07
## 1.167493 2.966409e-07
## 1.167493 2.273163e-07
## 1.167493 1.899714e-07
## 1.167493 1.584221e-07
## 1.167493 1.23891e-07
## 1.167493 8.806228e-08
## 1.167493 6.881345e-08
## 1.167493 5.842725e-08
## 1.167493 4.867722e-08
## 1.167493 3.749347e-08
## 1.167493 2.688771e-08
## 1.167493 2.155742e-08
## 1.167493 1.793847e-08
```

```
## 1.167493 1.482559e-08
## 1.167493 1.124708e-08
## 1.167493 8.248694e-09
```

Calculation: Stable distribution and size-dependent total elasticity

```
stable.dist.flow=stable.dist

p.surv.flow=sx(y,p.vec)*fx(y,p.vec)

for(i in 1:2){
    for(age in 1:n.age){
        stable.dist.flow[,age,i]=stable.dist[,age,i]*p.surv.flow
    }
}

for(i in 1:2){
    stable.dist.flow[,,i]=stable.dist.flow[,,i]/(sum(stable.dist.flow[,,i]))
}

dataf=data.frame(read.table("ipm.saf.data.txt",header=T))
attach(dataf)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     n
```

```
flow.age=age.05[flo.05==1]
flow.site=as.numeric(site[flo.05==1])
age=age.05
site=as.numeric(site)

stable.dist.age=array(NA,dim=c(n.age,2))
stable.dist.age.flow=array(NA,dim=c(n.age,2))
```

Plot: Stable age distribution (see Fig. 2)

```
dev.new()
par(mfrow=c(2,2), mar=c(3,3,1,2)+0.1, bty="l",pty="s", cex.main=1, cex.axis=1, cex.lab=1, tck=0.02, mgp=

for(i in 1:2){
    hist(age[site==i],breaks=seq(0,50,3),freq=F,col="grey",main="",xlab="Age (years)", ylim=c(0,0.2))
    stable.dist.age[,i]=apply(stable.dist[,,i],2,sum)
    points(0:(n.age-1),stable.dist.age[,i],type="l")
    # if (i==1) text(locator(1),"a") else text(locator(1),"c")

    hist(flow.age[flow.site==i],breaks=seq(0,50,3),freq=F,col="grey",main="",xlab="Age (years)",ylim=c(
    stable.dist.age.flow[,i]=apply(stable.dist.flow[,,i],2,sum)
    points(0:(n.age-1),stable.dist.age.flow[,i],type="l")
    # if (i==1) text(locator(1),"b") else text(locator(1),"d")
}

mean.age.f=rep(NA,2)
mean.size.f=rep(NA,2)
```

```r
for(i in 1:2){
    mean.age.f[i] <-sum((1:(n.age))*apply(stable.dist.flow[,,i],2,sum))-1;
    cat("Mean flowering age",mean.age.f[i],"\n")
    mean.size.f[i]<-sum(exp(y)*apply(stable.dist.flow[,,i],1,sum));
    cat("Mean flowering size",mean.size.f[i],"\n")
}
```

```
## Mean flowering age 8.016105
## Mean flowering size 2990.466
## Mean flowering age 6.957204
## Mean flowering size 2849.351
```

```r
stable.dist.size=array(NA,dim=c(n.big.matrix,2))
stable.dist.size.flow=array(NA,dim=c(n.big.matrix,2))
```

Stable size distribution (see Fig. 3)

```r
dev.new()
par(mfrow=c(2,2), mar=c(3,3,1,2)+0.1, bty="l",pty="s", cex.main=1, cex.axis=1, cex.lab=1, tck=0.02, mgp

for(i in 1:2){
    hist(size.all.plus.seedlings[site.all.plus.seedlings==i],freq=T,col="grey",main="",xlab="Plant size
    stable.dist.size[,i]=sum(!is.na(size.all.plus.seedlings[site.all.plus.seedlings==i]))*apply(stable.d
    points(y,stable.dist.size[,i],type="l")
    # if (i==1) text(locator(1),"a)") else text(locator(1),"c)")

    hist(store.size.flow[store.site.flow==site.code.l[i]],freq=T,col="grey",main="",xlab="Plant size",b
    stable.dist.size.flow[,i]=sum(!is.na(store.size.flow[store.site.flow==site.code.l[i]]))*apply(stabl
    points(y,stable.dist.size.flow[,i],type="l")
    # if (i==1) text(locator(1),"b)") else text(locator(1),"d)")
}
```

Iterate model with time lag

```r
stable.dist.tl=array(0,dim=c(n,2))
lam.stable.tl=rep(NA,2)
b = L+c(0:n)*(U-L)/n; y = 0.5*(b[1:n]+b[2:(n+1)]);
h = y[2]-y[1]

for(i in 1:2){
    p.vec=store.p.vec[,i]
    if(i==1) p.est= est.p.est[1] else p.est=est.p.est[2]
    P<-h*t(outer(y,y,pxy,params=p.vec))
    B<-h*t(outer(y,y,fxy,params=p.vec))

    qmax=1000

    # population now, next year and the one after
    Nt=matrix(1/n,n);
    Nt1=Nt/2;

    while(qmax>1e-10) {
        Nt2=P%*%Nt1+B%*%Nt
        qmax=sum(abs(Nt2-lam*Nt1));
        lam=sum(Nt2)/sum(Nt1);
```

```
        Nt=Nt1;
        Nt1=Nt2;
        tot=sum(Nt+Nt1)
        Nt=Nt/tot
        Nt1=Nt1/tot
    }
    stable.dist.tl[,i]=Nt/sum(Nt); lam.stable.tl[i]=lam;
}

lam.stable.age
```

```
## [1] 1.048766 1.167493
```

```
lam.stable.tl
```

```
## [1] 1.048766 1.167493
```

Calculation: sensitivity and elasticity by perturbation P matrix

```
sen.big.P<-array(NA,dim=c(n,n))     #array to store the results
elas.big.P<-array(NA,dim=c(n,n,2))

for(i in 1:2){
    p.vec=store.p.vec[,i]
    if(i==1) p.est= est.p.est[1] else p.est=est.p.est[2]               #actual

    P<-h*t(outer(y,y,pxy,params=p.vec))
    B<-h*t(outer(y,y,fxy,params=p.vec))


    for(row in 1:n) {       # loop over y values
        # choose x* to maximize e(y,x) for this y value, by scanning across the row
        big.one=which(P[row,]*stable.dist.tl[,i]==max(P[row,]*stable.dist.tl[,i]));

        # perturb the kernel up and down near (y,x*)
        delta=0.1*h*P[row,big.one];
        Pup=P; Pup[row,big.one] = P[row,big.one]+delta/h;
        Pdown=P; Pdown[row,big.one] = P[row,big.one]-delta/h;

        qmax=1; lamup=1; lamdown=1;
        Nt.up<-stable.dist.tl[,i]; Nt1.up<-stable.dist.tl[,i]
        Nt.down<-stable.dist.tl[,i]; Nt1.down<-stable.dist.tl[,i]

        while(qmax>1e-10) {
            Nt2.up=Pup%*%Nt1.up+B%*%Nt.up
            qmax=sum(abs(Nt2.up-lamup*Nt1.up));
            lamup=sum(Nt2.up)/sum(Nt1.up);

            Nt.up=Nt1.up;
            Nt1.up=Nt2.up;
            tot=sum(Nt.up+Nt1.up)
            Nt.up=Nt.up/tot
            Nt1.up=Nt1.up/tot

            Nt2.down=Pdown%*%Nt1.down+B%*%Nt.down
```

```
            qmax=qmax+sum(abs(Nt2.down-lamdown*Nt1.down));
            lamdown=sum(Nt2.down)/sum(Nt1.down);

            Nt.down=Nt1.down;
            Nt1.down=Nt2.down;
            tot=sum(Nt.down+Nt1.down)
            Nt.down=Nt.down/tot
            Nt1.down=Nt1.down/tot

            #cat(lamup,lamdown,qmax,"\n");
        }


        sen.big.row<-(lamup-lamdown)/(2*delta) #sensitivity for perturbation at (y,x*)
        sen.big.P[row,]<-(stable.dist.tl[,i]/stable.dist.tl[,i][big.one])*sen.big.row #sensitivity at o
        cat(row,big.one,lamup,lamdown," sens=",sen.big.row, "\n")
    }

    elas.big.P[,,i]=(P/h)*sen.big.P/lam.stable.tl[i];

}
```

```
## 1 10 1.048766 1.048765  sens= 0.005454862
## 2 11 1.048766 1.048765  sens= 0.007311486
## 3 11 1.048766 1.048765  sens= 0.007888722
## 4 11 1.048766 1.048765  sens= 0.008491295
## 5 12 1.048767 1.048765  sens= 0.01112822
## 6 12 1.048767 1.048764  sens= 0.01191879
## 7 13 1.048767 1.048764  sens= 0.01529594
## 8 13 1.048768 1.048764  sens= 0.01629973
## 9 14 1.048768 1.048763  sens= 0.02047714
## 10 14 1.048769 1.048763  sens= 0.02171342
## 11 15 1.048769 1.048762  sens= 0.02670417
## 12 15 1.04877 1.048761  sens= 0.02818576
## 13 16 1.048771 1.04876  sens= 0.03394624
## 14 16 1.048773 1.048758  sens= 0.03567942
## 15 16 1.048774 1.048757  sens= 0.03742866
## 16 17 1.048776 1.048755  sens= 0.04408845
## 17 17 1.048778 1.048753  sens= 0.04608596
## 18 18 1.048781 1.048751  sens= 0.0532379
## 19 18 1.048783 1.048748  sens= 0.05547129
## 20 19 1.048786 1.048745  sens= 0.06288889
## 21 19 1.048789 1.048742  sens= 0.06533647
## 22 20 1.048793 1.048738  sens= 0.07275844
## 23 21 1.048797 1.048735  sens= 0.07976483
## 24 21 1.048801 1.048731  sens= 0.08254343
## 25 22 1.048805 1.048726  sens= 0.08906918
## 26 22 1.048809 1.048722  sens= 0.09195496
## 27 23 1.048814 1.048717  sens= 0.09779822
## 28 24 1.048818 1.048713  sens= 0.1027802
## 29 24 1.048823 1.048709  sens= 0.1057775
## 30 25 1.048827 1.048704  sens= 0.1099203
## 31 26 1.048831 1.0487  sens= 0.1132568
```

```
## 32 27 1.048835 1.048696  sens= 0.115921
## 33 28 1.048839 1.048692  sens= 0.1180858
## 34 29 1.048843 1.048688  sens= 0.1199414
## 35 30 1.048847 1.048685  sens= 0.1216757
## 36 31 1.04885 1.048681  sens= 0.1234559
## 37 32 1.048855 1.048676  sens= 0.125416
## 38 33 1.048859 1.048672  sens= 0.1276506
## 39 34 1.048864 1.048667  sens= 0.1302141
## 40 35 1.04887 1.048661  sens= 0.133125
## 41 37 1.048876 1.048655  sens= 0.137126
## 42 38 1.048883 1.048649  sens= 0.1409344
## 43 39 1.04889 1.048641  sens= 0.1449776
## 44 40 1.048898 1.048633  sens= 0.1492285
## 45 41 1.048906 1.048625  sens= 0.1536704
## 46 42 1.048915 1.048616  sens= 0.158298
## 47 43 1.048924 1.048607  sens= 0.1631169
## 48 44 1.048934 1.048597  sens= 0.1681417
## 49 45 1.048944 1.048587  sens= 0.1733938
## 50 46 1.048955 1.048575  sens= 0.1788989
## 51 48 1.048968 1.048563  sens= 0.1872507
## 52 49 1.048981 1.048549  sens= 0.1935597
## 53 50 1.048996 1.048535  sens= 0.2002243
## 54 51 1.049012 1.048519  sens= 0.2072716
## 55 52 1.049029 1.048502  sens= 0.2147246
## 56 53 1.049046 1.048484  sens= 0.2225939
## 57 54 1.049065 1.048465  sens= 0.2308616
## 58 55 1.049085 1.048446  sens= 0.239447
## 59 56 1.049105 1.048426  sens= 0.2481475
## 60 57 1.049123 1.048407  sens= 0.2565592
## 61 58 1.049139 1.048391  sens= 0.2640269
## 62 59 1.049149 1.048381  sens= 0.2697606
## 63 60 1.049149 1.048381  sens= 0.2732565
## 64 61 1.049135 1.048395  sens= 0.2747103
## 65 61 1.049107 1.048423  sens= 0.287907
## 66 62 1.049077 1.048453  sens= 0.2935719
## 67 63 1.049026 1.048505  sens= 0.2956162
## 68 63 1.048975 1.048556  sens= 0.325758
## 69 64 1.048916 1.048615  sens= 0.3216117
## 70 64 1.048868 1.048664  sens= 0.3581379
## 71 65 1.048826 1.048705  sens= 0.339029
## 72 65 1.048799 1.048732  sens= 0.3785209
## 73 65 1.048782 1.04875  sens= 0.4227772
## 74 66 1.048773 1.048758  sens= 0.3789262
## 75 67 1.048768 1.048763  sens= 0.3184554
## 76 67 1.048767 1.048765  sens= 0.3557802
## 77 68 1.048766 1.048765  sens= 0.2781786
## 78 68 1.048766 1.048766  sens= 0.3107784
## 79 69 1.048766 1.048766  sens= 0.2244587
## 80 70 1.048766 1.048766  sens= 0.1487973
## 81 71 1.048766 1.048766  sens= 0.09002201
## 82 71 1.048766 1.048766  sens= 0.1005655
## 83 72 1.048766 1.048766  sens= 0.05524331
## 84 73 1.048766 1.048766  sens= 0.02742904
## 85 74 1.048766 1.048766  sens= 0.01226074
```

```
## 86 74 1.048766 1.048766  sens= 0.0136971
## 87 1 1.048766 1.048766  sens= 0.0455951
## 88 1 1.048766 1.048766  sens= 0.05062695
## 89 1 1.048766 1.048766  sens= 0.0573334
## 90 1 1.048766 1.048766  sens= 0.06607761
## 91 1 1.048766 1.048766  sens= 0.07956223
## 92 1 1.048766 1.048766  sens= 0.07238231
## 93 1 1.048766 1.048766  sens= 0.04422606
## 94 1 1.048766 1.048766  sens= 0.1633379
## 95 1 1.048766 1.048766  sens= 0
## 96 1 1.048766 1.048766  sens= 0
## 97 1 1.048766 1.048766  sens= 0.5376048
## 98 1 1.048766 1.048766  sens= 0
## 99 1 1.048766 1.048766  sens= -1.95941
## 100 1 1.048766 1.048766  sens= -3.782457
## 1 13 1.167493 1.167493  sens= 0.006573187
## 2 13 1.167494 1.167492  sens= 0.006846797
## 3 14 1.167494 1.167492  sens= 0.008826239
## 4 14 1.167494 1.167492  sens= 0.00916991
## 5 15 1.167494 1.167492  sens= 0.01171274
## 6 15 1.167494 1.167492  sens= 0.01213367
## 7 16 1.167495 1.167491  sens= 0.01532707
## 8 16 1.167495 1.167491  sens= 0.01583015
## 9 17 1.167496 1.16749  sens= 0.01974538
## 10 17 1.167496 1.16749  sens= 0.02033422
## 11 18 1.167497 1.167489  sens= 0.025018
## 12 18 1.167498 1.167488  sens= 0.02569749
## 13 19 1.167499 1.167487  sens= 0.03116606
## 14 19 1.1675 1.167486  sens= 0.03194612
## 15 20 1.167501 1.167485  sens= 0.03818235
## 16 20 1.167502 1.167484  sens= 0.03908129
## 17 21 1.167504 1.167482  sens= 0.046034
## 18 21 1.167506 1.16748  sens= 0.04707996
## 19 22 1.167508 1.167478  sens= 0.05466423
## 20 22 1.167511 1.167475  sens= 0.05589391
## 21 23 1.167514 1.167472  sens= 0.06399206
## 22 23 1.167517 1.167469  sens= 0.06544761
## 23 24 1.16752 1.167466  sens= 0.07391227
## 24 25 1.167524 1.167462  sens= 0.08234352
## 25 25 1.167528 1.167458  sens= 0.08430022
## 26 26 1.167532 1.167454  sens= 0.09272677
## 27 26 1.167537 1.167449  sens= 0.09501746
## 28 27 1.167542 1.167444  sens= 0.1032659
## 29 27 1.167546 1.16744  sens= 0.1059355
## 30 28 1.167552 1.167434  sens= 0.1138546
## 31 29 1.167557 1.167429  sens= 0.1210511
## 32 29 1.167562 1.167424  sens= 0.1244212
## 33 30 1.167568 1.167418  sens= 0.1311011
## 34 31 1.167573 1.167413  sens= 0.13698
## 35 31 1.167578 1.167408  sens= 0.14111
## 36 32 1.167584 1.167402  sens= 0.1465196
## 37 33 1.167589 1.167397  sens= 0.1512891
## 38 34 1.167593 1.167393  sens= 0.1555717
## 39 35 1.167598 1.167388  sens= 0.1595448
```

```
## 40 36 1.167603 1.167383  sens= 0.1633925
## 41 37 1.167608 1.167378  sens= 0.1672895
## 42 38 1.167613 1.167372  sens= 0.1713885
## 43 39 1.167619 1.167367  sens= 0.1758114
## 44 40 1.167625 1.167361  sens= 0.1806459
## 45 41 1.167632 1.167354  sens= 0.1859461
## 46 43 1.167639 1.167347  sens= 0.1897571
## 47 44 1.167647 1.167338  sens= 0.1959401
## 48 45 1.167656 1.16733  sens= 0.2025128
## 49 46 1.167665 1.167321  sens= 0.2094363
## 50 47 1.167675 1.167311  sens= 0.2166669
## 51 48 1.167684 1.167301  sens= 0.2241552
## 52 49 1.167695 1.167291  sens= 0.2318434
## 53 50 1.167705 1.167281  sens= 0.2396604
## 54 51 1.167715 1.167271  sens= 0.2475161
## 55 52 1.167725 1.167261  sens= 0.2552942
## 56 53 1.167734 1.167252  sens= 0.2628453
## 57 54 1.167742 1.167244  sens= 0.2699808
## 58 55 1.167748 1.167238  sens= 0.2764667
## 59 56 1.167752 1.167233  sens= 0.2820181
## 60 57 1.167753 1.167233  sens= 0.2862932
## 61 57 1.16775 1.167235  sens= 0.3102621
## 62 58 1.167746 1.16724  sens= 0.3146851
## 63 59 1.167737 1.167249  sens= 0.3168267
## 64 60 1.167722 1.167263  sens= 0.3159649
## 65 61 1.167703 1.167283  sens= 0.3113107
## 66 61 1.167681 1.167304  sens= 0.3456706
## 67 62 1.167657 1.167329  sens= 0.3365092
## 68 63 1.167629 1.167356  sens= 0.3213071
## 69 63 1.167603 1.167383  sens= 0.3591803
## 70 64 1.167578 1.167408  sens= 0.3355005
## 71 65 1.167556 1.16743  sens= 0.3052066
## 72 65 1.167537 1.167449  sens= 0.3415181
## 73 66 1.167523 1.167463  sens= 0.301627
## 74 67 1.167512 1.167474  sens= 0.2579342
## 75 67 1.167504 1.167482  sens= 0.2883908
## 76 68 1.167499 1.167487  sens= 0.2380745
## 77 69 1.167496 1.16749  sens= 0.1892971
## 78 69 1.167495 1.167491  sens= 0.2114774
## 79 70 1.167494 1.167492  sens= 0.1615087
## 80 71 1.167493 1.167493  sens= 0.1182056
## 81 71 1.167493 1.167493  sens= 0.1319881
## 82 72 1.167493 1.167493  sens= 0.09233366
## 83 73 1.167493 1.167493  sens= 0.06159784
## 84 74 1.167493 1.167493  sens= 0.03909314
## 85 74 1.167493 1.167493  sens= 0.04363611
## 86 75 1.167493 1.167493  sens= 0.02628137
## 87 76 1.167493 1.167493  sens= 0.01498673
## 88 77 1.167493 1.167493  sens= 0.008072109
## 89 78 1.167493 1.167493  sens= 0.004096732
## 90 79 1.167493 1.167493  sens= 0.001954348
## 91 79 1.167493 1.167493  sens= 0.00218006
## 92 80 1.167493 1.167493  sens= 0.0009729938
## 93 1 1.167493 1.167493  sens= 0.1481825
```

```
## 94 1 1.167493 1.167493  sens= 0.1651511
## 95 1 1.167493 1.167493  sens= 0.1840778
## 96 1 1.167493 1.167493  sens= 0.2067409
## 97 1 1.167493 1.167493  sens= 0.2314724
## 98 1 1.167493 1.167493  sens= 0.2561141
## 99 1 1.167493 1.167493  sens= 0.284823
## 100 1 1.167493 1.167493  sens= 0.3210859
```

```
sum(elas.big.P[,,1]*h*h)
```

```
## [1] 0.81223
```

```
sum(elas.big.P[,,2]*h*h)
```

```
## [1] 0.7876917
```

Calculation: sensitivity and elasticity by perturbation B matrix

```
sen.big.B<-array(NA,dim=c(n,n))      #array to store the results
elas.big.B<-array(NA,dim=c(n,n,2))

for(i in 1:2){
    p.vec=store.p.vec[,i]
    if(i==1) p.est= est.p.est[1] else p.est=est.p.est[2]                #actual

    P<-h*t(outer(y,y,pxy,params=p.vec))
    B<-h*t(outer(y,y,fxy,params=p.vec))

    for(row in 1:n) {       # loop over y values
        # choose x* to maximize e(y,x) for this y value, by scanning across the row
        big.one=which(B[row,]*stable.dist.tl[,i]==max(B[row,]*stable.dist.tl[,i]));

        # perturb the kernel up and down near (y,x*)
        delta=0.1*h*B[row,big.one];
        Bup=B; Bup[row,big.one] = B[row,big.one]+delta/h;
        Bdown=B; Bdown[row,big.one] = B[row,big.one]-delta/h;

        qmax=1;
        lamup=1;
        lamdown=1;

        Nt.up<-stable.dist.tl[,i];
        Nt1.up<-stable.dist.tl[,i]

        Nt.down<-stable.dist.tl[,i];
        Nt1.down<-stable.dist.tl[,i]

        while(qmax>1e-10) {
            Nt2.up=P%*%Nt1.up+Bup%*%Nt.up
            qmax=sum(abs(Nt2.up-lamup*Nt1.up));
            lamup=sum(Nt2.up)/sum(Nt1.up);

            Nt.up=Nt1.up;
            Nt1.up=Nt2.up;
            tot=sum(Nt.up+Nt1.up)
            Nt.up=Nt.up/tot
```

```
            Nt1.up=Nt1.up/tot

            Nt2.down=P%*%Nt1.down+Bdown%*%Nt.down

            qmax=qmax+sum(abs(Nt2.down-lamdown*Nt1.down));
            lamdown=sum(Nt2.down)/sum(Nt1.down);

            Nt.down=Nt1.down;
            Nt1.down=Nt2.down;
            tot=sum(Nt.down+Nt1.down)
            Nt.down=Nt.down/tot
            Nt1.down=Nt1.down/tot

            #cat(lamup,lamdown,qmax,"\n");
        }

        sen.big.row<-(lamup-lamdown)/(2*delta) #sensitivity for perturbation at (y,x*)
        sen.big.B[row,]<-(stable.dist.tl[,i]/stable.dist.tl[,i][big.one])*sen.big.row #sensitivity at o
        cat(row,big.one," sens=",sen.big.row, "\n")
    }

    elas.big.B[,,i]=2*(B/h)*sen.big.B/lam.stable.tl[i];

}
```

```
## 1 66  sens= 0.01658482
## 2 66  sens= 0.01793492
## 3 66  sens= 0.01935087
## 4 66  sens= 0.02082897
## 5 66  sens= 0.02236486
## 6 66  sens= 0.02395371
## 7 66  sens= 0.0255904
## 8 66  sens= 0.02726976
## 9 66  sens= 0.02898679
## 10 66  sens= 0.03073683
## 11 66  sens= 0.03251568
## 12 66  sens= 0.0343197
## 13 66  sens= 0.03614579
## 14 66  sens= 0.03799128
## 15 66  sens= 0.03985386
## 16 66  sens= 0.04173141
## 17 66  sens= 0.04362189
## 18 66  sens= 0.04552322
## 19 66  sens= 0.04743324
## 20 66  sens= 0.04934972
## 21 66  sens= 0.05127037
## 22 66  sens= 0.05319292
## 23 66  sens= 0.05511518
## 24 66  sens= 0.05703512
## 25 66  sens= 0.05895087
## 26 66  sens= 0.06086085
## 27 66  sens= 0.06276368
## 28 66  sens= 0.0646583
## 29 66  sens= 0.0665439
```

```
## 30 66  sens= 0.06841996
## 31 66  sens= 0.07028623
## 32 66  sens= 0.07214274
## 33 66  sens= 0.07398977
## 34 66  sens= 0.07582786
## 35 66  sens= 0.07765781
## 36 66  sens= 0.07948064
## 37 66  sens= 0.08129759
## 38 66  sens= 0.08311015
## 39 66  sens= 0.08491998
## 40 66  sens= 0.086729
## 41 66  sens= 0.08853929
## 42 66  sens= 0.09035316
## 43 66  sens= 0.09217311
## 44 66  sens= 0.09400186
## 45 66  sens= 0.09584234
## 46 66  sens= 0.09769771
## 47 66  sens= 0.09957133
## 48 66  sens= 0.1014668
## 49 66  sens= 0.1033881
## 50 66  sens= 0.1053391
## 51 66  sens= 0.1073245
## 52 66  sens= 0.1093491
## 53 66  sens= 0.1114186
## 54 66  sens= 0.1135398
## 55 66  sens= 0.115721
## 56 66  sens= 0.1179723
## 57 66  sens= 0.1203033
## 58 66  sens= 0.1227189
## 59 66  sens= 0.1252085
## 60 66  sens= 0.1277365
## 61 66  sens= 0.1302546
## 62 66  sens= 0.1327995
## 63 66  sens= 0.1357094
## 64 66  sens= 0.139912
## 65 66  sens= 0.1465924
## 66 66  sens= 0.1568553
## 67 66  sens= 0.1706509
## 68 66  sens= 0.1885469
## 69 66  sens= 0.2096132
## 70 66  sens= 0.2378871
## 71 66  sens= 0.2688554
## 72 66  sens= 0.2806507
## 73 66  sens= 0.3708832
## 74 66  sens= 0.220231
## 75 66  sens= 0.6677404
## 76 66  sens= -2.067543
## 77 66  sens= 6.53761
## 78 66  sens= 0
## 79 66  sens= 69.61474
## 80 66  sens= 0
## 81 66  sens= 806.2213
## 82 66  sens= 0
## 83 66  sens= 0
```

```
## 84 66  sens= 0
## 85 66  sens= 0
## 86 66  sens= 0
## 87 66  sens= 0
## 88 66  sens= 0
## 89 66  sens= 0
## 90 66  sens= 0
## 91 66  sens= 0
## 92 66  sens= 0
## 93 66  sens= 0
## 94 66  sens= 0
## 95 66  sens= 0
## 96 66  sens= 0
## 97 66  sens= 0
## 98 66  sens= 0
## 99 66  sens= 0
## 100 66  sens= 0
## 1 66  sens= 0.008190646
## 2 66  sens= 0.008531584
## 3 66  sens= 0.008875671
## 4 66  sens= 0.009221268
## 5 66  sens= 0.009566746
## 6 66  sens= 0.009910555
## 7 66  sens= 0.01025129
## 8 66  sens= 0.01058777
## 9 66  sens= 0.01091911
## 10 66  sens= 0.01124473
## 11 66  sens= 0.01156447
## 12 66  sens= 0.01187856
## 13 66  sens= 0.01218763
## 14 66  sens= 0.01249267
## 15 66  sens= 0.01279502
## 16 66  sens= 0.01309626
## 17 66  sens= 0.01339815
## 18 66  sens= 0.01370257
## 19 66  sens= 0.01401144
## 20 66  sens= 0.01432663
## 21 66  sens= 0.01464996
## 22 66  sens= 0.01498319
## 23 66  sens= 0.01532796
## 24 66  sens= 0.01568586
## 25 66  sens= 0.01605843
## 26 66  sens= 0.0164472
## 27 66  sens= 0.01685368
## 28 66  sens= 0.01727946
## 29 66  sens= 0.01772616
## 30 66  sens= 0.01819552
## 31 66  sens= 0.01868937
## 32 66  sens= 0.01920967
## 33 66  sens= 0.01975857
## 34 66  sens= 0.02033836
## 35 66  sens= 0.02095158
## 36 66  sens= 0.02160099
## 37 66  sens= 0.02228963
```

```
## 38 66  sens= 0.02302086
## 39 66  sens= 0.02379839
## 40 66  sens= 0.02462638
## 41 66  sens= 0.02550941
## 42 66  sens= 0.02645265
## 43 66  sens= 0.02746188
## 44 66  sens= 0.0285436
## 45 66  sens= 0.02970517
## 46 66  sens= 0.03095495
## 47 66  sens= 0.03230243
## 48 66  sens= 0.03375854
## 49 66  sens= 0.03533584
## 50 66  sens= 0.03704893
## 51 66  sens= 0.03891489
## 52 66  sens= 0.04095388
## 53 66  sens= 0.04318996
## 54 66  sens= 0.04565213
## 55 66  sens= 0.04837577
## 56 66  sens= 0.05140451
## 57 66  sens= 0.05479259
## 58 66  sens= 0.05860773
## 59 66  sens= 0.06293434
## 60 66  sens= 0.06787636
## 61 66  sens= 0.07355907
## 62 66  sens= 0.08012839
## 63 66  sens= 0.08774687
## 64 66  sens= 0.09658622
## 65 66  sens= 0.1068183
## 66 66  sens= 0.1186081
## 67 66  sens= 0.1321109
## 68 66  sens= 0.1474769
## 69 66  sens= 0.1648602
## 70 66  sens= 0.1844254
## 71 66  sens= 0.2063692
## 72 66  sens= 0.2309338
## 73 66  sens= 0.2583767
## 74 66  sens= 0.2890939
## 75 66  sens= 0.3234114
## 76 66  sens= 0.3596084
## 77 66  sens= 0.4040586
## 78 66  sens= 0.4472531
## 79 66  sens= 0.4792865
## 80 66  sens= 0.5009174
## 81 66  sens= 0.7573948
## 82 66  sens= 1.559319
## 83 66  sens= 0
## 84 66  sens= 0
## 85 66  sens= 0
## 86 66  sens= 0
## 87 66  sens= 0
## 88 66  sens= 0
## 89 66  sens= 0
## 90 66  sens= 0
## 91 66  sens= 0
```

```
## 92 66  sens= 0
## 93 66  sens= 0
## 94 66  sens= 0
## 95 66  sens= 0
## 96 66  sens= 0
## 97 66  sens= 0
## 98 66  sens= 0
## 99 66  sens= 0
## 100 66  sens= 0
```

```
sum(elas.big.B[,,1]*h*h)+sum(elas.big.P[,,1]*h*h)
```

```
## [1] 1.000001
```

```
sum(elas.big.B[,,2]*h*h)+sum(elas.big.P[,,2]*h*h)
```

```
## [1] 1
```

Plot: elasticity (see Fig 5)

```
dev.new()
par(mfrow=c(2,2), mar=c(3,3,1,2)+0.1, bty="l",pty="s", cex.main=1, cex.axis=1, cex.lab=1, tck=0.02, mgp=

zmax=max(elas.big.P,elas.big.B)

image(y,y,t(elas.big.P[,,1]),xlab="Plant size year t",ylab="Plant size year t+1",col=grey(1:300/300),gam
```

```
## Warning in plot.window(...): graphical parameter "gamma" is obsolete

## Warning in plot.xy(xy, type, ...): graphical parameter "gamma" is obsolete

## Warning in axis(side = side, at = at, labels = labels, ...): graphical
## parameter "gamma" is obsolete

## Warning in axis(side = side, at = at, labels = labels, ...): graphical
## parameter "gamma" is obsolete

## Warning in box(...): graphical parameter "gamma" is obsolete

## Warning in title(...): graphical parameter "gamma" is obsolete
```

```
    contour(y,y,t(elas.big.P[,,1]),add=T,cex=3,levels = c(0.01,0.05,0.1,0.15,0.2,0.25,0.3));
    # text(locator(1),"a)",col="white")

image(y,y,t(elas.big.B[,,1]),xlab="Plant size year t-1",ylab="Plant size year t+1",col=grey(1:300/300),g
```

```
## Warning in plot.window(...): graphical parameter "gamma" is obsolete

## Warning in plot.xy(xy, type, ...): graphical parameter "gamma" is obsolete

## Warning in axis(side = side, at = at, labels = labels, ...): graphical
## parameter "gamma" is obsolete

## Warning in axis(side = side, at = at, labels = labels, ...): graphical
## parameter "gamma" is obsolete

## Warning in box(...): graphical parameter "gamma" is obsolete

## Warning in title(...): graphical parameter "gamma" is obsolete
```

```
    contour(y,y,t(elas.big.B[,,1]),add=T,cex=3,levels = c(0.01,0.05,0.1,0.15,0.2,0.25,0.3));
    # text(locator(1),"b)",col="white")

image(y,y,t(elas.big.P[,,2]),xlab="Plant size year t",ylab="Plant size year t+1",col=grey(1:300/300),gan
```

```
## Warning in plot.window(...): graphical parameter "gamma" is obsolete

## Warning in plot.xy(xy, type, ...): graphical parameter "gamma" is obsolete

## Warning in axis(side = side, at = at, labels = labels, ...): graphical
## parameter "gamma" is obsolete

## Warning in axis(side = side, at = at, labels = labels, ...): graphical
## parameter "gamma" is obsolete

## Warning in box(...): graphical parameter "gamma" is obsolete

## Warning in title(...): graphical parameter "gamma" is obsolete
```

```
    contour(y,y,t(elas.big.P[,,2]),add=T,cex=3,levels = c(0.01,0.05,0.1,0.15,0.2,0.25,0.3));
    # text(locator(1),"c)",col="white")

image(y,y,t(elas.big.B[,,2]),xlab="Plant size year t-1",ylab="Plant size year t+1",col=grey(1:300/300),g
```

```
## Warning in plot.window(...): graphical parameter "gamma" is obsolete

## Warning in plot.xy(xy, type, ...): graphical parameter "gamma" is obsolete

## Warning in axis(side = side, at = at, labels = labels, ...): graphical
## parameter "gamma" is obsolete

## Warning in axis(side = side, at = at, labels = labels, ...): graphical
## parameter "gamma" is obsolete

## Warning in box(...): graphical parameter "gamma" is obsolete

## Warning in title(...): graphical parameter "gamma" is obsolete
```

```
    contour(y,y,t(elas.big.B[,,2]),add=T,cex=3,levels = c(0.01,0.05,0.1,0.15,0.2,0.25,0.3));
    # text(locator(1),"d)",col="white")
```

Calculation: new big matrix approximation

```
M.tl=array(0,dim=c(2*n,2*n))
lam.stable.bm=rep(NA,2)
R0.stable.bm=rep(NA,2)

for(i in 1:2){
    p.vec=store.p.vec[,i]
    if(i==1) p.est= est.p.est[1] else p.est=est.p.est[2]
    if(i==1) p.vec[3]=-58.67228 else p.vec[3]= -26.25266

    P<-h*t(outer(y,y,pxy,params=p.vec))
    B<-h*t(outer(y,y,fxy,params=p.vec))

    M.tl[1:n,1:n]=P
    M.tl[1:n,(n+1):(2*n)]=B
    M.tl[(n+1):(2*n),1:n]=diag(n)
```

```
    lam.stable.bm[i]=Re(eigen(M.tl)$values[1]);

    #R0
    M.P=array(0,dim=c(2*n,2*n))
    M.B=array(0,dim=c(2*n,2*n))

    M.P[1:n,1:n]=P
    M.P[(n+1):(2*n),1:n]=diag(n)

    M.B[1:n,(n+1):(2*n)]=B


    N<-solve(diag(2*n)-M.P);
    R<- M.B %*% N
    R0.stable.bm[i]<-Re(eigen(R)$values[1]);

    # Generation time
    T.stable.mb <-log(R0.stable.bm)/log(lam.stable.bm)
}


lam.stable.bm   # Lambda
```

```
## [1] 1.099069 1.135874
```

```
R0.stable.bm    # Net reproductive rate
```

```
## [1] 8.171492 8.794975
```

```
T.stable.mb     # Generation time
```

```
## [1] 22.23775 17.06543
```

Simulate Lambda, R0 and T with varying p.est

```
seq.start <- 0.00001
seq.end <- 0.001
seq.by <- 0.00001
p.est.seq <- seq(seq.start,seq.end,by=seq.by)
n.p.est <- length(p.est.seq)

p.est.FU <- 0.00016
p.est.SP <- 0.00078

Res.p.est <- matrix(0,n.p.est,8)
dimnames(Res.p.est) <- list(1:n.p.est,c("p.est","p.est","l.FU","l.SP","Ro.FU","Ro.SP","T.FU","T.SP"))

#new big matrix approximation
M.tl=array(0,dim=c(2*n,2*n))
lam.stable.bm=rep(NA,2)
R0.stable.bm=rep(NA,2)

for (k in 1:n.p.est){
    for(i in 1:2){
        p.est= p.est.seq[k]
        p.vec=store.p.vec[,i]
```

```r
        P<-h*t(outer(y,y,pxy,params=p.vec))
        B<-h*t(outer(y,y,fxy,params=p.vec))
        M.tl[1:n,1:n]=P
        M.tl[1:n,(n+1):(2*n)]=B
        M.tl[(n+1):(2*n),1:n]=diag(n)
        lam.stable.bm=Re(eigen(M.tl)$values[1]);

    # R0
        M.P=array(0,dim=c(2*n,2*n))
        M.B=array(0,dim=c(2*n,2*n))
        M.P[1:n,1:n]=P
        M.P[(n+1):(2*n),1:n]=diag(n)
        M.B[1:n,(n+1):(2*n)]=B
        N<-solve(diag(2*n)-M.P);
        R<- M.B %*% N
        R0.stable.bm<-Re(eigen(R)$values[1]);

    # Generation time
        T.stable.bm <-log(R0.stable.bm)/log(lam.stable.bm)

    # Filling in of result matrix
        Res.p.est[k,i] <- p.est
        Res.p.est[k,i+2] <- lam.stable.bm
        Res.p.est[k,i+4] <- R0.stable.bm
        Res.p.est[k,i+6] <- T.stable.bm
    }
}

Res.p.est
```

```
##      p.est   p.est      l.FU      l.SP     Ro.FU      Ro.SP     T.FU
## 1  0.00001 0.00001 0.8263343 0.8244452 0.1018270 0.06432277 11.97593
## 2  0.00002 0.00002 0.8725686 0.8584002 0.2036540 0.12864554 11.67403
## 3  0.00003 0.00003 0.9020529 0.8812231 0.3054810 0.19296832 11.50411
## 4  0.00004 0.00004 0.9241485 0.8988413 0.4073080 0.25729109 11.38638
## 5  0.00005 0.00005 0.9419942 0.9133644 0.5091351 0.32161386 11.29661
## 6  0.00006 0.00006 0.9570517 0.9258096 0.6109621 0.38593663 11.22424
## 7  0.00007 0.00007 0.9701276 0.9367518 0.7127891 0.45025941 11.16373
## 8  0.00008 0.00008 0.9817169 0.9465506 0.8146161 0.51458218 11.11179
## 9  0.00009 0.00009 0.9921463 0.9554467 0.9164431 0.57890495 11.06636
## 10 0.00010 0.00010 1.0016434 0.9636101 1.0182701 0.64322772 11.02600
## 11 0.00011 0.00011 1.0103736 0.9711654 1.1200971 0.70755049 10.98973
## 12 0.00012 0.00012 1.0184608 0.9782070 1.2219241 0.77187327 10.95681
## 13 0.00013 0.00013 1.0260006 0.9848083 1.3237511 0.83619604 10.92668
## 14 0.00014 0.00014 1.0330683 0.9910274 1.4255782 0.90051881 10.89892
## 15 0.00015 0.00015 1.0397242 0.9969114 1.5274052 0.96484158 10.87319
## 16 0.00016 0.00016 1.0460176 1.0024988 1.6292322 1.02916436 10.84923
## 17 0.00017 0.00017 1.0519892 1.0078217 1.7310592 1.09348713 10.82680
## 18 0.00018 0.00018 1.0576731 1.0129070 1.8328862 1.15780990 10.80574
## 19 0.00019 0.00019 1.0630980 1.0177775 1.9347132 1.22213267 10.78588
## 20 0.00020 0.00020 1.0682886 1.0224531 2.0365402 1.28645544 10.76710
## 21 0.00021 0.00021 1.0732659 1.0269505 2.1383672 1.35077822 10.74929
## 22 0.00022 0.00022 1.0780484 1.0312845 2.2401943 1.41510099 10.73236
## 23 0.00023 0.00023 1.0826520 1.0354682 2.3420213 1.47942376 10.71623
```

```
## 24  0.00024 0.00024 1.0870908 1.0395128  2.4438483 1.54374653 10.70083
## 25  0.00025 0.00025 1.0913772 1.0434286  2.5456753 1.60806931 10.68609
## 26  0.00026 0.00026 1.0955223 1.0472245  2.6475023 1.67239208 10.67197
## 27  0.00027 0.00027 1.0995359 1.0509085  2.7493293 1.73671485 10.65841
## 28  0.00028 0.00028 1.1034269 1.0544880  2.8511563 1.80103762 10.64537
## 29  0.00029 0.00029 1.1072032 1.0579694  2.9529833 1.86536039 10.63282
## 30  0.00030 0.00030 1.1108719 1.0613588  3.0548103 1.92968317 10.62072
## 31  0.00031 0.00031 1.1144397 1.0646613  3.1566374 1.99400594 10.60903
## 32  0.00032 0.00032 1.1179123 1.0678821  3.2584644 2.05832871 10.59775
## 33  0.00033 0.00033 1.1212953 1.0710255  3.3602914 2.12265148 10.58682
## 34  0.00034 0.00034 1.1245935 1.0740957  3.4621184 2.18697425 10.57625
## 35  0.00035 0.00035 1.1278115 1.0770965  3.5639454 2.25129703 10.56600
## 36  0.00036 0.00036 1.1309535 1.0800313  3.6657724 2.31561980 10.55605
## 37  0.00037 0.00037 1.1340232 1.0829033  3.7675994 2.37994257 10.54640
## 38  0.00038 0.00038 1.1370243 1.0857156  3.8694264 2.44426534 10.53701
## 39  0.00039 0.00039 1.1399600 1.0884710  3.9712534 2.50858812 10.52789
## 40  0.00040 0.00040 1.1428335 1.0911718  4.0730805 2.57291089 10.51901
## 41  0.00041 0.00041 1.1456474 1.0938207  4.1749075 2.63723366 10.51036
## 42  0.00042 0.00042 1.1484045 1.0964198  4.2767345 2.70155643 10.50193
## 43  0.00043 0.00043 1.1511073 1.0989712  4.3785615 2.76587920 10.49371
## 44  0.00044 0.00044 1.1537579 1.1014768  4.4803885 2.83020198 10.48569
## 45  0.00045 0.00045 1.1563587 1.1039385  4.5822155 2.89452475 10.47786
## 46  0.00046 0.00046 1.1589116 1.1063580  4.6840425 2.95884752 10.47022
## 47  0.00047 0.00047 1.1614185 1.1087370  4.7858695 3.02317029 10.46275
## 48  0.00048 0.00048 1.1638812 1.1110768  4.8876965 3.08749307 10.45544
## 49  0.00049 0.00049 1.1663014 1.1133791  4.9895236 3.15181584 10.44830
## 50  0.00050 0.00050 1.1686807 1.1156451  5.0913506 3.21613861 10.44130
## 51  0.00051 0.00051 1.1710205 1.1178761  5.1931776 3.28046138 10.43446
## 52  0.00052 0.00052 1.1733222 1.1200733  5.2950046 3.34478415 10.42775
## 53  0.00053 0.00053 1.1755873 1.1222379  5.3968316 3.40910693 10.42118
## 54  0.00054 0.00054 1.1778169 1.1243710  5.4986586 3.47342970 10.41474
## 55  0.00055 0.00055 1.1800123 1.1264735  5.6004856 3.53775247 10.40843
## 56  0.00056 0.00056 1.1821746 1.1285465  5.7023126 3.60207524 10.40223
## 57  0.00057 0.00057 1.1843049 1.1305910  5.8041397 3.66639802 10.39615
## 58  0.00058 0.00058 1.1864042 1.1326077  5.9059667 3.73072079 10.39018
## 59  0.00059 0.00059 1.1884734 1.1345975  6.0077937 3.79504356 10.38432
## 60  0.00060 0.00060 1.1905136 1.1365612  6.1096207 3.85936633 10.37857
## 61  0.00061 0.00061 1.1925256 1.1384997  6.2114477 3.92368910 10.37291
## 62  0.00062 0.00062 1.1945102 1.1404136  6.3132747 3.98801188 10.36736
## 63  0.00063 0.00063 1.1964682 1.1423037  6.4151017 4.05233465 10.36189
## 64  0.00064 0.00064 1.1984005 1.1441705  6.5169287 4.11665742 10.35652
## 65  0.00065 0.00065 1.2003077 1.1460148  6.6187557 4.18098019 10.35124
## 66  0.00066 0.00066 1.2021906 1.1478372  6.7205828 4.24530297 10.34604
## 67  0.00067 0.00067 1.2040498 1.1496382  6.8224098 4.30962574 10.34092
## 68  0.00068 0.00068 1.2058859 1.1514184  6.9242368 4.37394851 10.33589
## 69  0.00069 0.00069 1.2076997 1.1531784  7.0260638 4.43827128 10.33093
## 70  0.00070 0.00070 1.2094917 1.1549186  7.1278908 4.50259405 10.32604
## 71  0.00071 0.00071 1.2112624 1.1566396  7.2297178 4.56691683 10.32123
## 72  0.00072 0.00072 1.2130124 1.1583419  7.3315448 4.63123960 10.31650
## 73  0.00073 0.00073 1.2147423 1.1600259  7.4333718 4.69556237 10.31183
## 74  0.00074 0.00074 1.2164525 1.1616920  7.5351988 4.75988514 10.30722
## 75  0.00075 0.00075 1.2181435 1.1633407  7.6370259 4.82420792 10.30269
## 76  0.00076 0.00076 1.2198158 1.1649724  7.7388529 4.88853069 10.29821
## 77  0.00077 0.00077 1.2214698 1.1665875  7.8406799 4.95285346 10.29380
```

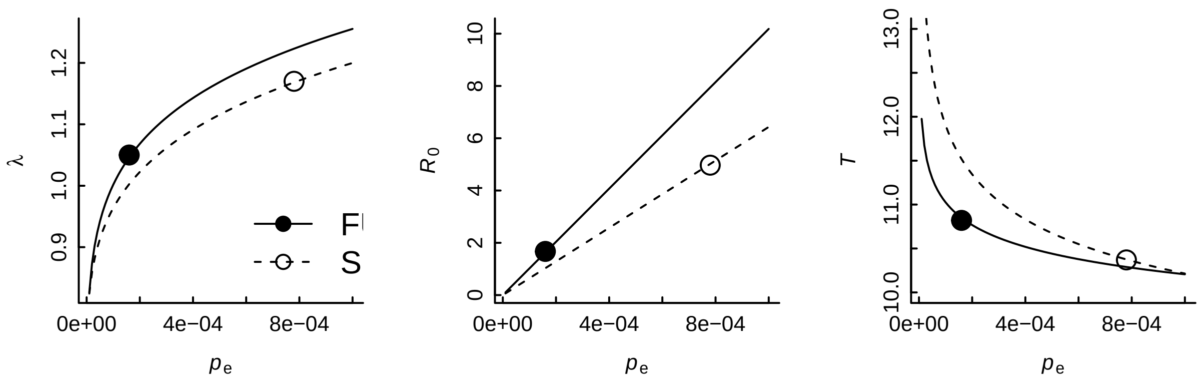
```
## 78  0.00078 0.00078 1.2231060 1.1681864  7.9425069 5.01717623 10.28945
## 79  0.00079 0.00079 1.2247248 1.1697694  8.0443339 5.08149900 10.28516
## 80  0.00080 0.00080 1.2263266 1.1713369  8.1461609 5.14582178 10.28092
## 81  0.00081 0.00081 1.2279118 1.1728893  8.2479879 5.21014455 10.27674
## 82  0.00082 0.00082 1.2294807 1.1744268  8.3498149 5.27446732 10.27262
## 83  0.00083 0.00083 1.2310338 1.1759498  8.4516419 5.33879009 10.26854
## 84  0.00084 0.00084 1.2325713 1.1774586  8.5534690 5.40311286 10.26452
## 85  0.00085 0.00085 1.2340936 1.1789535  8.6552960 5.46743564 10.26055
## 86  0.00086 0.00086 1.2356011 1.1804348  8.7571230 5.53175841 10.25663
## 87  0.00087 0.00087 1.2370940 1.1819028  8.8589500 5.59608118 10.25275
## 88  0.00088 0.00088 1.2385727 1.1833577  8.9607770 5.66040395 10.24893
## 89  0.00089 0.00089 1.2400374 1.1847998  9.0626040 5.72472673 10.24515
## 90  0.00090 0.00090 1.2414885 1.1862294  9.1644310 5.78904950 10.24141
## 91  0.00091 0.00091 1.2429262 1.1876466  9.2662580 5.85337227 10.23772
## 92  0.00092 0.00092 1.2443507 1.1890519  9.3680851 5.91769504 10.23406
## 93  0.00093 0.00093 1.2457625 1.1904453  9.4699121 5.98201781 10.23046
## 94  0.00094 0.00094 1.2471616 1.1918271  9.5717391 6.04634059 10.22689
## 95  0.00095 0.00095 1.2485483 1.1931975  9.6735661 6.11066336 10.22336
## 96  0.00096 0.00096 1.2499230 1.1945568  9.7753931 6.17498613 10.21987
## 97  0.00097 0.00097 1.2512858 1.1959051  9.8772201 6.23930890 10.21642
## 98  0.00098 0.00098 1.2526369 1.1972426  9.9790471 6.30363168 10.21300
## 99  0.00099 0.00099 1.2539765 1.1985696 10.0808741 6.36795445 10.20963
## 100 0.00100 0.00100 1.2553049 1.1998862 10.1827011 6.43227722 10.20629
##         T.SP
## 1   14.21351
## 2   13.43090
## 3   13.01148
## 4   12.72913
## 5   12.51819
## 6   12.35079
## 7   12.21257
## 8   12.09521
## 9   11.99345
## 10  11.90380
## 11  11.82378
## 12  11.75161
## 13  11.68595
## 14  11.62577
## 15  11.57026
## 16  11.51878
## 17  11.47081
## 18  11.42592
## 19  11.38376
## 20  11.34402
## 21  11.30647
## 22  11.27087
## 23  11.23705
## 24  11.20485
## 25  11.17412
## 26  11.14474
## 27  11.11660
## 28  11.08961
## 29  11.06368
## 30  11.03873
```

```
## 31  11.01470
## 32  10.99153
## 33  10.96915
## 34  10.94752
## 35  10.92659
## 36  10.90632
## 37  10.88667
## 38  10.86760
## 39  10.84909
## 40  10.83111
## 41  10.81361
## 42  10.79659
## 43  10.78002
## 44  10.76387
## 45  10.74813
## 46  10.73278
## 47  10.71779
## 48  10.70315
## 49  10.68885
## 50  10.67488
## 51  10.66121
## 52  10.64783
## 53  10.63475
## 54  10.62193
## 55  10.60938
## 56  10.59707
## 57  10.58502
## 58  10.57319
## 59  10.56159
## 60  10.55021
## 61  10.53903
## 62  10.52806
## 63  10.51729
## 64  10.50670
## 65  10.49629
## 66  10.48607
## 67  10.47601
## 68  10.46612
## 69  10.45639
## 70  10.44682
## 71  10.43739
## 72  10.42812
## 73  10.41898
## 74  10.40999
## 75  10.40112
## 76  10.39239
## 77  10.38379
## 78  10.37531
## 79  10.36695
## 80  10.35870
## 81  10.35057
## 82  10.34255
## 83  10.33464
## 84  10.32684
```

```
## 85  10.31913
## 86  10.31153
## 87  10.30402
## 88  10.29661
## 89  10.28929
## 90  10.28207
## 91  10.27493
## 92  10.26788
## 93  10.26091
## 94  10.25403
## 95  10.24722
## 96  10.24050
## 97  10.23385
## 98  10.22728
## 99  10.22078
## 100 10.21436
```

Plot: Simulated Lambda, R0 and T with varying p.est (see Figure 4)

```
# dev.new()
par(mfrow=c(1,3), mar=c(3,3,1,2)+0.1, bty="l",pty="s", cex.main=1, cex.axis=1, cex.lab=1, tck=0.02, mgp=

plot(Res.p.est[,1],Res.p.est[,3],type="n",xlab=expression(paste(italic(p)[e])),ylab=expression(lambda))
    lines(Res.p.est[,1],Res.p.est[,3],lty=1)
    lines(Res.p.est[,1],Res.p.est[,4],lty=2)
    points(p.est.FU,1.05, pch=19, cex=2.0); points(p.est.SP,1.17, pch=1, cex=2.0);
    legend(0.0006,1,c("FU", "SP"),pch=c(19,1),lty=c(1,2),bty="n",xjust=0,cex=1.5)

plot(Res.p.est[,1],Res.p.est[,5],type="n",xlab=expression(paste(italic(p)[e])),ylab=expression(paste(ita
    lines(Res.p.est[,1],Res.p.est[,5],lty=1)
    lines(Res.p.est[,1],Res.p.est[,6],lty=2)
    points(p.est.FU,1.67, pch=19, cex=2.0); points(p.est.SP,4.97, pch=1, cex=2.0)

plot(Res.p.est[,1],Res.p.est[,7],type="n",xlab=expression(paste(italic(p)[e])),ylab=expression(italic(T)
    lines(Res.p.est[,1],Res.p.est[,7],lty=1)
    lines(Res.p.est[,1],Res.p.est[,8],lty=2)
    points(p.est.FU,10.82, pch=19, cex=2.0); points(p.est.SP,10.37, pch=1, cex=2.0)
```




```
## Comment out bootstrap because takes multiple days to run -- way to convert to parallel processing?

# #==========================================================================#
# #  Bootstrap lambda, R0 and generation time (see Appendix S4)
```

```
# #=============================================================================#
#
#
# n.boot=5000
# dem.stats=array(NA,dim=c(n.boot,3,2))
# boot.ESS=array(NA,dim=c(n.boot,2))
#
# M.tl=array(0,dim=c(2*n,2*n))
#
# for(b.samp in 1:n.boot){
#
#   #growth
#
#   size.t=size.t.all[flow.all==0]
#   size.t1=size.t1.all[flow.all==0]
#   site.s=site.all[flow.all==0]
#   test=complete.cases(size.t,size.t1,site.s)
#   size.t=size.t[test]
#   size.t1=size.t1[test]
#   site.s=site.s[test]
#   sample.boot=c(sample(1:735,replace=T),735+sample(1:357,replace=T))
#   size.t.boot=size.t[sample.boot]
#   size.t1.boot=size.t1[sample.boot]
#   fit.grow.gls.boot<-gls(size.t1.boot~site.s+size.t.boot-1,na.action=na.omit,weight=varExp(form=~fitt
#   g.intercepts.boot=fit.grow.gls.boot$coef[1:2]
#   g.slopes.boot=rep(fit.grow.gls.boot$coef[3],2)
#   var.exp.coef.boot=fit.grow.gls.boot$modelStruct$varStruct
#   sigma.g.boot=fit.grow.gls.boot$sigma
#
#   #survival and flowering
#
#   sample.boot=c(sample(1:827,replace=T),827+sample(1:746,replace=T))
#   size.t.boot=size.t.all[sample.boot]
#   flow.all.boot=flow.all[sample.boot]
#   surv.all.boot=surv.all[sample.boot]
#   fit.flow.boot=glm(flow.all.boot~site.all*size.t.boot-1,family=binomial)
#   f.intercepts.boot=fit.flow.boot$coef[1:2]
#   f.slopes.boot=c(fit.flow.boot$coef[3],fit.flow.boot$coef[3]+fit.flow.boot$coef[4])
#   fit.surv.boot=glm(surv.all.boot~site.all*size.t.boot-1,family=binomial)
#   s.intercepts.boot=fit.surv.boot$coef[1:2]
#   s.slopes.boot=c(fit.surv.boot$coef[3],fit.surv.boot$coef[3]+fit.surv.boot$coef[4])
#
#   #fecundity
#
#   sample.boot=c(sample(1:25,replace=T),25+sample(1:28,replace=T))
#       size.t.f.boot=size.t.f[sample.boot]
#       si.t1.f.boot=si.t1.f[sample.boot]
#       fit.fec.boot=lm(log(si.t1.f.boot)~site.t.f+size.t.f.boot-1)
#
#   #seedlings
#
#   sample.boot=c(sample(1:573,replace=T),573+sample(1:123,replace=T))
#   seedlings.size.t.boot=seedlings.size.t[sample.boot]
```

```
#   fit.seedlings.boot=lm(seedlings.size.t.boot~seedlings.site-1)
#
#   #p.est
#
#   sample.boot=c(sample(1:70,replace=T),70+sample(1:42,replace=T))
#   p.est.seeds.t.boot=p.est.seeds.t[sample.boot]
#   p.est.seedlings.boot=p.est.seedlings[sample.boot]
#   est.p.est.boot= sapply(split(p.est.seedlings.boot,p.est.site),sum)/sapply(split(p.est.seeds.t.boot,p
#
#
#   for(i in 1:2){
#       p.vec[1]<- s.intercepts.boot[i]
#       p.vec[2]<- s.slopes.boot[i]
#       p.vec[3]<- f.intercepts.boot[i]
#       p.vec[4]<- f.slopes.boot[i]
#       p.vec[5]<- g.intercepts.boot[i]
#       p.vec[6]<- g.slopes.boot[i]
#       p.vec[7]<- sigma.g.boot^2
#       p.vec[8]<- fit.fec.boot$coef[i]
#       p.vec[9]<- fit.fec.boot$coef[3]
#       p.vec[10]<- fit.seedlings.boot$coef[i]
#       p.vec[11]<- summary(fit.seedlings.boot)$sigma^2
#       p.vec[12]<- var.exp.coef.boot[i]
#
#       if(i==1) p.est= est.p.est.boot[1] else p.est=est.p.est.boot[2]
#
#       P<-h*t(outer(y,y,pxy,params=p.vec))
#       B<-h*t(outer(y,y,fxy,params=p.vec))
#
#       M.tl=array(0,dim=c(2*n,2*n))
#       M.tl[1:n,1:n]=P
#       M.tl[1:n,(n+1):(2*n)]=B
#       M.tl[(n+1):(2*n),1:n]=diag(n)
#
#       lam=Re(eigen(M.tl)$values[1]);
#
#   #R0
#
#       M.P=array(0,dim=c(2*n,2*n))
#       M.B=array(0,dim=c(2*n,2*n))
#
#       M.P[1:n,1:n]=P
#       M.P[(n+1):(2*n),1:n]=diag(n)
#
#       M.B[1:n,(n+1):(2*n)]=B
#
#       N<-solve(diag(2*n)-M.P);
#       R<- M.B %*% N
#       R0<-Re(eigen(R)$values[1]);
#
#   #generation time
#
#       T=log(R0)/log(lam)
```

```
#        boot.data=c(lam,R0,T)
#
#        dem.stats[b.samp,,i]=boot.data
#
#        if(any(!is.na(boot.data))){
#            boot.ESS[b.samp,i]<-optimize(R0.betas,c(-150,10),maximum=T,tol=0.01)$maximum
#        } else boot.ESS[b.samp,i]=NA
#    }
#    cat("sample ",b.samp,"\n")
# }
#
# getStats=function(x){
#    ci.normal.app=c(mean(x)-1.96*sd(x),mean(x)+1.96*sd(x))
#    res=c(mean(x,na.rm=T),quantile(x,p=c(0.025,0.5,0.975),na.rm=T),ci.normal.app)
#    return(res)
#    }
#
# for(i in 1:2){
#    cat("site ",site.code.l[i],"\n")
#    print(apply(dem.stats[,,i],2,getStats))
#    print(getStats(boot.ESS[,i]))
# }
#
#
# #==============================================================================#
# #  Plot: Bootstrap lambda, R0 and generation time (see Appendix S4)
# #==============================================================================#
#
#
# dev.new()
# par(mfrow=c(2,4), mar=c(3,3,1,2)+0.1, bty="l",pty="s", cex.main=1, cex.axis=1, cex.lab=1, tck=0.02, m
#
# main.titles=c(expression(italic(lambda)),expression(italic(R[0])),"Generation time, T",expression("ES
#
# for(i in 1:2){
#    for(j in 1:3){
#        hist(dem.stats[,j,i][dem.stats[,j,i]<30],xlab=main.titles[j],col="grey",main="")
#        abline(v=mean(dem.stats[,j,i],na.rm=T))
#        abline(v=median(dem.stats[,j,i],na.rm=T),col="blue")
#        abline(v=quantile(dem.stats[,j,i],p=0.025,na.rm=T),col="red")
#        abline(v=quantile(dem.stats[,j,i],p=0.975,na.rm=T),col="red")
#    }
#
#    hist(boot.ESS[,i],col="grey",xlab=main.titles[4],main="")
#    abline(v=mean(boot.ESS[,i],na.rm=T))
#    abline(v=median(boot.ESS[,i],na.rm=T),col="blue")
#    abline(v=quantile(boot.ESS[,i],p=0.025,na.rm=T),col="red")
#    abline(v=quantile(boot.ESS[,i],p=0.975,na.rm=T),col="red")
#    abline(v=f.intercepts[i],col="green",lwd=2)
# }
#
# lam.diff=dem.stats[,1,1]-dem.stats[,1,2]
# mean(lam.diff,na.rm=T)
```

```r
# quantile(lam.diff,p=c(0.025,0.5,0.975),na.rm=T)
# R0.diff=dem.stats[,2,1]-dem.stats[,2,2]
# mean(R0.diff,na.rm=T)
# quantile(R0.diff,p=c(0.025,0.5,0.975),na.rm=T)
# T.diff=dem.stats[,3,1]-dem.stats[,3,2]
# mean(T.diff,na.rm=T)
# quantile(T.diff,p=c(0.025,0.5,0.975),na.rm=T)
#
# #save(dem.stats,boot.ESS,file="c:\\temp\\ipm sheffield\\bootstrap samples.Rdata")
```