

Appendix C: R Code and tutorial
for
On using Integral Projection Models to generate demographically
driven predictions of species distributions: development and
validation using sparse data

Cory Merow, Andrew Latimer, Adam Wilson, Sean McMahon, Tony Rebelo, John Silander

April 3, 2014

Contents

1	Setup	2
2	Data	2
3	Regression	5
3.1	Growth	5
3.2	Fecundity	5
3.3	Recruitment	6
3.4	Survival	7
3.4.1	Seedlings	7
3.4.2	Adults	10
4	Kernels	12
4.1	Set Up Inputs	12
4.2	Vital Rate Functions	13
4.3	Build growth/survival kernel	14
4.4	Build fecundity kernel	15
5	Analyses	15
5.1	Population Growth Rate (λ)	15
5.2	Uncertainty Analysis	18
5.3	Parameter Elasticity	19
5.4	Environmental Sensitivity	21
6	Functions	23
6.1	Miscellaneous functions	23

DDDMs can require substantial programming effort and computing power. This appendix illustrates a number of solutions. Computation time can be large when modeling landscapes with a large number of cells or when propagating uncertainty, which is further exacerbated when using stochastic models. Of course, these challenges can be substantially minimized by ignoring uncertainty, using coarse resolution landscapes, or building deterministic models. While the fastest run-time option would be to write the model in a compiled language such as C, the accessibility, wide-use, and existing tools available in R may be preferable. In this study, we built the discretized IPM subkernels (100x100 matrices) for both growth and fecundity for each of 20,000 cells (1' x 1') across the Cape floristic Region. The uncertainty analysis added another dimension (1,000 posterior samples), leading to challenges of both output storage and computation time. To deal with these, we followed general protocols from Visser et al. (in review) that including use of R's profiling function to identify and remove bottlenecks by 1) using matrices rather than data frames, 2) vectorizing operations (rather than using loops) wherever possible, 3) breaking the workflow into separate modules to facilitate running various stages of analysis separately (e.g. write the growth and fecundity kernels to disk, then run simulations separately), and 4) parallelizing over the landscape.

1 Setup

Note that a number of functions needed throughout the analysis can be found in section 6.

```
##== Bayesian models
library(MCMCpack); library(MCMCglmm); library(coda); library(rjags)
##== spatial manipulation
library(rasterVis); library(rgdal); library(raster); library(maptools); library(sp)
##== parallelization
library(foreach); library(doSNOW)
##== misc
library(Matrix); library(ROCR); library(fields)
```

Constructing the IPM kernels for a large number of cells is facilitated by parallel computations, which we perform with the help of the doSNOW library. We set this up below, where 7 is the number of available cores on our machine (on your own machine, try choosing n-1).

```
registerDoSNOW(makeCluster(7, type = "SOCK"))
```

2 Data

The data are downloadable as supplementary materials to the paper. The data have been jittered, hence the results below will be similar, but not exactly the same as those from the main text.

```
(load('Protea_IPM_Shared_Data.Rdata'))
[1] "d"      "m2"      "d2"      "se"      "psr"
[6] "cfr.env"
##== data for growth and seedhead production models
head(d,3)
      indiv      site year sample.x replicate sizeNext
1 GROWTH10110C GROWTH101 2007      10          C 3.113000
2 GROWTH10111C GROWTH101 2007       1          C 1.920833
3 GROWTH1012C  GROWTH101 2007       2          C 2.952500
      size fire.yr date.est tsf survey.yr avg.growth lon.1
1 2.971773  1994   known  13    2007  0.2484108 19.025
2 1.920613  1994   known  13    2007  0.1703728 19.025
3 2.894268  1994   known  13    2007  0.2113176 19.025
      lat.1    min07      map rainconc    smdsum
1 -31.35833 -0.747739 -0.3521319 1.345609 -1.265015
```

```

2 -31.35833 -0.747739 -0.3521319 1.345609 -1.265015
3 -31.35833 -0.747739 -0.3521319 1.345609 -1.265015
      smdwin      fert4      text1      ph1 diameter flowers
1 0.6359487 -0.8752157 -0.4135085 1.364749      2940      NA
2 0.6359487 -0.8752157 -0.4135085 1.364749      3100      0
3 0.6359487 -0.8752157 -0.4135085 1.364749      2570      0
seedheads
1      NA
2      NA
3      NA
#== data for adult survival model
head(m2,3)
      Sample Site      south      east      Plot minage DeadPlants
1      2      1 -33.67472 19.06197 10 x 4      12      0
2      5      4 -33.67567 19.06103 10 x 4      2      9
3      9      5 -33.67463 19.06258 10 x 4      3      0
      LivePlants HalfDeadPlants Skeletons      max01      min07
1      3      0      0 -0.8079089 0.2097063
2      14      0      0 -0.8079089 0.2097063
3      17      0      0 -0.8079089 0.2097063
      map rainconc      smdsum      smdwin      fert4
1 2.293553 0.8939931 0.4018806 2.467686 -0.8752157
2 2.293553 0.8939931 0.4018806 2.467686 -0.8752157
3 2.293553 0.8939931 0.4018806 2.467686 -0.8752157
      text1      ph1 meanage
1 -0.4135085 0.3853808      13.5
2 -0.4135085 0.3853808      2.5
3 -0.4135085 0.3853808      3.5
#== data for juvenile survival and recruitment models
head(psr,3)
      site recruityear samples longitude latitude elevation
1 PSR1      2007      5 19.13595 -33.59244      736.7
2 PSR10      2008      5 19.08551 -32.57023      1025.1
3 PSR12      2008      5 19.50528 -33.16791      918.4
      area DeadParent DeadSeedling LiveParent LiveSeedling age
1 1000      1      0      0      76 2
2 1000      27      0      0      33 2
3 1000      38      0      0      22 2
      seedlingmort minburnage maxburnage minage maxage      lon
1      0      14      14      2      2 19.14167
2      0      7      7      2      2 19.09167
3      0      10      10      2      2 19.50833
      lat      max01      min07      map rainconc
1 -33.59167 -1.3545595 -0.2690163 3.14984898 0.793634
2 -32.57500 -0.7623546 -0.3753992 0.13937421 1.245250
3 -33.17500 -1.2178969 -1.5456101 -0.09485915 1.144891
      smdsum      smdwin      fert4      text1      ph1
1 1.0596213 2.557620 -0.8752157 -0.4135085 1.364749
2 -0.9263643 1.117504 -0.8752157 -0.4135085 1.364749
3 -1.0655151 0.811473 -0.8752157 -0.4135085 1.364749
#== data for seed/seedhead model
head(se,3)
      Location PlantID No.of.plant SeedAge Fertile Sterile Duds
1      131      5      1      New      0      0      0

```

```

2      131      5      2      New      10      4      13
3      131      5      1      Medium      2      14      3
total.seeds                                     Notes X
1      0 Only two cones and 1 predated upon
2      27
3      19 Predated
#== data for offspring size model
head(d2,3)
      site year sample replicate      indiv      length
1 GROWTH101 1992      3      C GROWTH1013C 0.05752555
2 GROWTH101 1992     10      C GROWTH10110C 0.13919263
3 GROWTH101 1992      6      C GROWTH1016C 0.11861729
      tsf lon.1 lat.1 min07      map rainconc
1  0 19.025 -31.35833 -0.747739 -0.3521319 1.345609
2  0 19.025 -31.35833 -0.747739 -0.3521319 1.345609
3  0 19.025 -31.35833 -0.747739 -0.3521319 1.345609
      smdsum smdwin      fert4      text1      ph1
1 -1.265015 0.6359487 -0.8752157 -0.4135085 1.364749
2 -1.265015 0.6359487 -0.8752157 -0.4135085 1.364749
3 -1.265015 0.6359487 -0.8752157 -0.4135085 1.364749
#== data for the environment in the cape floristic region
head(cfr.env,3)
      lon lat apan pptcv      frost
1955 18.94167 -31.125 1.959873 1.2785529 -0.2809171
1956 18.95833 -31.125 1.211830 0.8938266 0.0755387
1957 18.97500 -31.125 1.626674 1.0477172 0.1646527
      htunt max01 min07      map rainconc
1955 1.7090237 1.6975734 -0.1626335 -1.1469894 1.295429
1956 -0.1341986 0.6953805 -0.8009304 -0.8052390 1.295429
1957 0.9795783 1.2875854 -0.4817820 -0.9511549 1.295429
      smdsum smdwin      trans      fert1
1955 -1.460909 -0.06686385 -0.7491434 -0.523890579
1956 -1.423866 0.08249786 -0.7491434 0.838742963
1957 -1.423866 0.08249786 -0.7491434 -0.007029581
      fert2 fert3 fert4      text1      text2
1955 -0.4041251 2.570409 -0.8752157 -0.4135085 1.1057710
1956 -0.4041251 0.336416 -0.8752157 -0.4135085 -0.2819675
1957 -0.4041251 1.723032 -0.8752157 -0.4135085 0.6431915
      text3 text4      ph1      ph2
1955 -0.3093487 -0.5356563 -0.54621355 0.6964511
1956 -0.8504879 1.3092570 0.83923439 -0.6077145
1957 -0.8504879 0.3738080 -0.02069882 0.2617292
      ph3 tsf.weib MaxAbun Presence Presence_3x3
1955 -0.1992523 19.82434      NA      NA      NA
1956 -0.3997661 19.67636      0      0      0
1957 -0.3997661 19.43095      NA      NA      NA
      Presence_5x5 MaxAbunClass RepensCount AtlasCount pres
1955      NA      NA      NA      NA      NA
1956      0      1      0      1      0
1957      NA      NA      NA      NA      NA
      mean.tsf
1955      18
1956      17
1957      17

```

3 Regression

In all of the regressions below, the candidate predictors in the full formula (full.form) were chosen based on having a significant linear or quadratic coefficient in a univariate model of the form: vital rate \sim predictor + I(predictor \wedge 2). For the sake of illustration and quick computation, we use short MCMC chains below, and these can readily be adjusted with the 'burnin', 'nitt', and 'thin' arguments.

3.1 Growth

```
form.full=avg.growth~fert4+I(fert4^2)+smdwin+I(smdwin^2)+ph1+I(ph1^2)+
  min07+I(min07^2) +smdsum+I(smdsum^2)
gr.reg.DIC=stepDIC(form.full,data=d,burnin=3000,nitt=8000,thin=5)
gr.reg=gr.reg.DIC$model.best
save(gr.reg,file='PR_growth.post')
```

```
summary(gr.reg)
```

3.2 Fecundity

Seedheads per plant:

```
form.full=seedheads~size+min07+I(min07^2)+smdsum+I(smdsum^2)+ph1+I(ph1^2)
fec1.reg.DIC=stepDIC(form.full,data=d[d$seedheads>0 & !is.na(d$seedheads>0)],,
  DIC.diff=3,burnin=3000,nitt=5000,thin=2,family='poisson')
fec1.reg=fec1.reg.DIC$model.best
save(fec1.reg,file='PR_seedhead.post')
```

```
summary(fec1.reg)
```

Offspring size:

```
form.full=length~ph1+I(ph1^2)+fert4+I(fert4^2)+min07+I(min07^2)+smdsum+I(smdsum^2)
offspr.reg.DIC=stepDIC(form.full,data=d2,burnin=3000,nitt=13000,thin=10)
offspr.reg=offspr.reg.DIC$model.best
save(offspr.reg,file='PR_offspring.post')
```

```
summary(offspr.reg)
```

Flowering probability:

```
d$repro=ifelse(d$seedheads>0,1,0)
form.full=repro~size
fl.reg=MCMClogit(form.full, data=d[!is.na(d$seedheads>0)],,b0=0, B0=.001)
save(fl.reg,file='PR_flowering.post')
```

```
summary(fl.reg)
```

Seeds per seedhead:

Note that we estimated the number of seeds per seedhead only from new seedheads that had not been subject to seed predators. Ultimately, this decision had little consequence on our predictions because this value trades off with the estimated recruitment rate. That is, had we estimated a lower value of seeds/seedhead, we would have estimated a higher value for recruitment rate. Observations of seed production and recruitment at the same site would be needed to properly identify these parameters.

```

se.reg=MCMCglmm(total.seeds~1,
  data=se[se$SeedAge=='New' & se$total.seeds>10,],
  burnin=3000,nitt=5000,thin=2,
  family='poisson', verbose=FALSE)
save(se.reg,file='PR_seed.post')

```

```

summary(se.reg)
predict(se.reg)[1,1]

```

3.3 Recruitment

The full set of steps for building the recruitment model are described verbally in Appendix A.

```

#== predict size from age
psr.growth=predict.MCMCglmm.cm(gr.reg,newdata=psr,return.post.pred=TRUE)
names(psr.growth)=psr$site
apply(psr.growth,2,mean)
#== to get parent size, add up the avg growth for the right number of years,
#== according to the burn age of the parents
psr.size=0*psr.growth
for(i in 1:ncol(psr.growth)){ psr.size[,i]=psr.growth[,i]*psr$minburnage[i]
}
apply(psr.size,2,mean)
#== predict fecundity from size
#== combine the full uncertainty of the 1000 size samples with 1000
#== posterior samples from the fecundity model, then grab just 1000
psr.fec=matrix(NA,1000,ncol(psr.growth))
for(i in 1:ncol(psr.fec)){ # for each posterior sample of size
  psr.fec[,i]=c(predict.MCMCglmm.cm(fec1.reg, newdata=data.frame(
    size=psr.size[,i],psr[i,]), return.post.pred=TRUE))[sample(1:1e6,1000)]
}
#== multiply by avg # seeds per flower
seed.num=exp(se.reg$Sol+.5*se.reg$VCV)
psr.nseed.indiv=apply(psr.fec,2,function(x) x*seed.num)
#== predict total number of seeds by the parents using
#== avg fecundity per plant * number of plants in the psr transect
psr.nseed.pop=0*psr.nseed.indiv
for(i in 1:ncol(psr.growth)){
  psr.nseed.pop[,i]=psr.nseed.indiv[,i]*(psr$DeadParent[i])
}
#== predict recruitment from number of seeds and # seedlings
psr.germ=0*psr.nseed.pop
for(i in 1:ncol(psr.growth)){
  psr.germ[,i]=(psr$LiveSeedling[i])/psr.nseed.pop[,i]
}
cbind(psr$minburnage,psr$minage,apply(psr.germ,2,mean))
#== make a posterior sample of recruitment by taking the mean of each sample of 1 yr old sites
psr.germ.1yr=psr.germ[,psr$minage==1]
germ.post=apply(psr.germ.1yr,1,mean) #take mean across sites
save(germ.post,file='PR_germ.post')

```

3.4 Survival

We illustrate the survival model last because it depends on the output of the previous models to impute size and reproductive output.

3.4.1 Seedlings

We begin by writing out a JAGS model file, to be called below. We found this to be the simplest way to incorporate samples from the posteriors of other vital rate models to obtain a posterior sample in the survival model.

```
cat('
model{

# Impute the number and size of one-year-old plants at each site
for (i in 1:n.sites) {

    # impute size of parents at time of fire
    parentgrowth[i] <- beta.growth[1] + beta.growth[2]*fert4[i] +
        beta.growth[3]*fert4.2[i] + beta.growth[4]*smdwin[i] +
        beta.growth[5]*smdwin.2[i] + beta.growth[6]*ph1[i] +
        beta.growth[7]*ph1.2[i] + beta.growth[8]*min07[i] +
        beta.growth[9]*smdsum[i] + beta.growth[10]*smdsum.2[i]
    parentsizes[i] <- parentgrowth[i] * parent.age[i]
    parentsizes.2[i] <- pow(parentsizes[i], 2)

    # impute parental fecundity
    # seedheads for year of fire
    seedheads.per.parent.0[i] <- beta.fec[1] + beta.fec[2]*parentsizes[i] +
        beta.fec[3]*min07[i] + beta.fec[4]*ph1[i] + beta.fec[5]*ph1.2[i]

    # add seedheads for previous 2 years
    parentsizesminus1[i] <- parentsizes[i]-parentgrowth[i]
    parentsizesminus1.2[i] <- pow(parentsizesminus1[i],2)
    seedheads.per.parent.minus1[i] <- beta.fec[1] + beta.fec[2]*parentsizesminus1[i] +
        beta.fec[3]*min07[i] + beta.fec[4]*ph1[i] + beta.fec[5]*ph1.2[i]
    parentsizesminus2[i] <- parentsizes[i]-2*parentgrowth[i]
    parentsizesminus2.2[i] <- pow(parentsizesminus1[i],2)
    seedheads.per.parent.minus2[i] <- beta.fec[1] + beta.fec[2]*parentsizesminus2[i] +
        beta.fec[3]*min07[i] + beta.fec[4]*ph1[i] + beta.fec[5]*ph1.2[i]
    seedheads.per.parent[i] <- seedheads.per.parent.0[i] + seedheads.per.parent.minus1[i] +
        seedheads.per.parent.minus2[i]

    seedlings.per.parent[i] <- seedheads.per.parent[i] * seeds.per.seedhead *
        germ.rate
    initial.seedlings[i] <- max(round(seedlings.per.parent[i] * n.parents[i]),
        round(n.seedlings.max[i]*1.05)) # force seedling count to be bigger than
        # observed number of seedlings.

    # impute seedling initial size
    seedling.initial.size.pred[i] <- beta.initial.size[1] +
        beta.initial.size[2]*ph1[i] + beta.initial.size[3]*fert4[i] +
        beta.initial.size[4]*fert4.2[i] + beta.initial.size[5]*min07[i] +
        beta.initial.size[6]*smdsum[i] + beta.initial.size[7]*smdsum.2[i]

    # force this to give seedlings at least 2 cm tall
```

```

seedling.initial.size[i] <- max(seedling.initial.size.pred[i], 0.02)
seedling.final.size[i] <- seedling.initial.size[i] +
  parentgrowth[i]*seedling.age[i] # calculate this to check it

# Likelihood for survival model
#       given initial number of seedlings, seedling age, and seedling sizes

n.seedlings[i] ~ dbin(p.surv[i], initial.seedlings[i])
p.surv[i] <- min(p.total[i], 1)
for (j in 1:7) { # set upper age limit here to match max observed age
  logit(p.annual[i,j]) <- beta.surv[1] + beta.surv[2]*(seedling.initial.size[i]
    + parentgrowth[i]*j) + beta.surv[3]*min07[i] + beta.surv[4]*map[i]
}
p.total[i] <- exp(sum(log(p.annual[i,2:seedling.age[i]])))
}

# Priors
for (j in 1:4) {
  beta.surv[j] ~ dnorm(0, 0.1)
}

} # End model
',file='seedlingsurv.jags.txt')

```

Run the JAGS model specified above:

```

# Growth model coefficients
beta.growth = apply(gr.reg$Sol, 2, mean)
# Seedhead model coefficients
beta.fec = apply(fec1.reg$Sol, 2, mean)
# Seeds per seedhead
# Note currently a fixed number for all seedheads
seeds.per.seedhead = mean(germ.post)
# recruitment rate (1-year old seedlings per seed)
# Note currently also just a fixed number for all sites
germ.rate = predict(se.reg)[1,1]
# Offspring initial size coefficients
beta.initial.size = apply(offspr.reg$Sol, 2, mean)
#####
# Assemble data and fitted coefficients for hierarchical survival model
#####

#### Remove site 19 because no parents observed there (!)
psr = psr[-19,]
#### Remove sites with age = 1 because those inform only "recruitment" rate
psr = psr[psr$age>1,]
# Assemble PSR data: TSF, Parent count, seedling count, environment
n.parents = psr$LiveParent + psr$DeadParent
n.seedlings = psr$LiveSeedling
n.seedlings.max = psr$LiveSeedling
parent.age = psr$maxburnage
seedling.age = as.integer(psr$age)
ph1 = psr$ph1
ph1.2 = psr$ph1^2
map = psr$map

```



```

map.2 = psr$map^2
max01 = psr$max01
max01.2 = psr$max01^2
smdwin = psr$smdwin
smdwin.2 = psr$smdwin^2
smdsum = psr$smdsum
smdsum.2 = psr$smdsum^2
fert1 = psr$fert1
fert1.2 = psr$fert1^2
fert4 = psr$fert4
fert4.2 = psr$fert4^2
rainconc = psr$rainconc
rainconc.2 = psr$rainconc^2
min07 = psr$min07
min07.2 = psr$min07^2
##### Run JAGS model j times for i samples each time.
### For each j, draw growth model, seedlings size model, and fecundity model coefficients from
### the posteriors of fitted vital rate models.
#== for the actual analysis we used n_coef_samples = 100 and n_MCMC_samples = 10
n_coef_samples = 100 # number of times to sample from posteriors of other vital rate regressions
n_MCMC_samples = 10 # number of MCMC samples to save per batch of coefficients
seedlingsurv.out = list()
for (j in 1:n_coef_samples) {

  jagsdata = list(n.sites=length(n.parents), n.parents=n.parents, n.seedlings=n.seedlings,
    n.seedlings.max = n.seedlings.max, parent.age=parent.age,
    seedling.age=seedling.age, ph1=ph1, ph1.2=ph1.2, map=map, map.2=map.2,
    max01=max01, max01.2=max01.2, smdsum=smdsum, smdsum.2=smdsum.2,
    smdwin=smdwin, smdwin.2=smdwin.2, fert1=fert1, fert1.2=fert1.2,
    fert4=fert4, fert4.2=fert4.2, rainconc=rainconc, rainconc.2=rainconc.2,
    min07=min07, min07.2=min07.2, beta.growth=gr.reg$Sol[sample(1:1000,1),],
    beta.fec=fec1.reg$Sol[sample(1:1000,1),],
    beta.initial.size = offspr.reg$Sol[sample(1:1000,1),],
    germ.rate=germ.rate, seeds.per.seedhead=seeds.per.seedhead)

  #jagsinits = list(beta.surv = c(0,0,0,0,0,0))
  # note that the model code referenced in the .txt file is is above and must
  # be stored in a file to be called by 'jags.model'
  seedlingsurv.model = jags.model("seedlingsurv.jags.txt", data=jagsdata,
    n.chains=1, n.adapt=5000)

  seedlingsurv.samp = coda.samples(seedlingsurv.model, variable.names=c("beta.surv"),
    n.iter=2000*n_MCMC_samples, thin=2000, burnin=5000)

  seedlingsurv.out = c(seedlingsurv.out, seedlingsurv.samp)

}
# Assemble the output
PR_seedlingsurv.post = seedlingsurv.out[[1]][1:10,]
for (i in 2:100) PR_seedlingsurv.post = rbind(PR_seedlingsurv.post, seedlingsurv.out[[i]][1:10,])
# Save the output
save(PR_seedlingsurv.post, file="PR_seedlingsurv.post")
(load('PR_seedlingsurv.post'))

```

```
s.sv.post=data.frame(PR_seedlingsurv.post)
names(s.sv.post)=c('Intercept','size','min07','map')
```

3.4.2 Adults

We begin by writing out a JAGS model file, to be called below.

```
# Survival model code -- for adult mortality sites

cat('
model{

for (i in 1:n.sites.adult) {
  # impute size of parents at time of fire
  adultgrowth[i] <- beta.growth[1] + beta.growth[2]*fert4.m[i] +
    beta.growth[3]*fert4.2.m[i] + beta.growth[4]*smdwin.m[i] +
    beta.growth[5]*smdwin.2.m[i] + beta.growth[6]*ph1.m[i] +
    beta.growth[7]*ph1.2.m[i] + beta.growth[8]*min07.m[i] +
    beta.growth[9]*smdsum.m[i] + beta.growth[10]*smdsum.2.m[i]

  # impute seedling initial size
  adult.initial.size.pred[i] <- beta.initial.size[1] +
    beta.initial.size[2]*ph1.m[i] + beta.initial.size[3]*fert4.m[i] +
    beta.initial.size[4]*fert4.2.m[i] + beta.initial.size[5]*min07.m[i] +
    beta.initial.size[6]*smdsum.m[i] + beta.initial.size[7]*smdsum.2.m[i]

  # force this to give seedlings at least 2 cm tall
  adult.initial.size[i] <- max(adult.initial.size.pred[i], 0.02)
  adultsize[i] <- adult.initial.size[i] + adultgrowth[i]*(adult.age[i]-2) # subtract 2
  #because first year of growth is already in initial size, and we want size one
  # year ago to predict mortality this year
  adultsize.2[i] <- adultsize[i]^2

  # Likelihood for adult survival model
  # given initial number of seedlings, seedling age, and seedling sizes

  n.live[i] ~ dbin(p.surv.adult[i], n.adults[i])
  logit(p.surv.adult[i]) <- beta.surv[1] + beta.surv[2]*adultsize[i] +
    beta.surv[3]*min07.m[i] + beta.surv[4]*map.m[i] }

  # Priors
  for (j in 1:4) {
    beta.surv[j] ~ dnorm(0, 0.1)
  }
} # End model
',file='adultsurv.jags.txt')
```

Run the JAGS model specified above:

```
# Growth model coefficients
beta.growth = apply(gr.reg$Sol, 2, mean)
# Offspring initial size coefficients
beta.fec=apply(fec1.reg$Sol, 2, mean)
beta.initial.size = apply(offspr.reg$Sol, 2, mean)
```

```
#####
# Run adult survival model in JAGS
# As in seedling survival model, draw a number of samples from the posteriors of
# the coefficients from the other vital rate regression models, then use these
# to get a few MCMC samples each.

# Assemble data for JAGS
n.adults = m2$LivePlants + m2$DeadPlants + m2$HalfDeadPlants
n.live = m2$LivePlants + m2$HalfDeadPlants
adult.age = as.integer(m2$meanage)
ph1.m = m2$ph1
ph1.2.m = m2$ph1^2
map.m = m2$map
map.2.m = m2$map^2
max01.m = m2$max01
max01.2.m = m2$max01^2
smdsum.m = m2$smdsum
smdsum.2.m = m2$smdsum^2
smdwin.m = m2$smdwin
smdwin.2.m = m2$smdwin^2
fert4.m = m2$fert4
fert4.2.m = m2$fert4^2
rainconc.m = m2$rainconc
rainconc.2.m = m2$rainconc^2
min07.m = m2$min07
min07.2.m = m2$min07^2
##### Run JAGS model

n_coef_samples = 100 # number of times to sample from posteriors of other vital rate regressions
n_MCMC_samples = 10 # number of MCMC samples to save per batch of coefficients
adultsurv.out = list()
for (j in 1:n_coef_samples) {

  jagsdata = list(n.sites.adult=length(n.adults), n.adults=n.adults, n.live=n.live,
    adult.age=adult.age, ph1.m=ph1.m, ph1.2.m=ph1.2.m, map.m=map.m,
    map.2.m=map.2.m, max01.m=max01.m, max01.2.m=max01.2.m, smdwin.m=smdwin.m,
    smdwin.2.m=smdwin.2.m, smdsum.m=smdsum.m, smdsum.2.m=smdsum.2.m,
    fert4.m=fert4.m, fert4.2.m=fert4.2.m, rainconc.m=rainconc.m,
    rainconc.2.m=rainconc.2.m, min07.m=min07.m, min07.2.m=min07.2.m,
    beta.growth=gr.reg$Sol[sample(1:1000,1),],
    beta.initial.size = offspr.reg$Sol[sample(1:1000,1),])

  adultsurv.model = jags.model("adultsurv.jags.txt", data=jagsdata, n.chains=1,
    n.adapt=5000)

  adultsurv.samp = coda.samples(adultsurv.model, variable.names=c("beta.surv"),
    n.iter=1000*n_MCMC_samples, thin=1000, burnin=5000)

  adultsurv.out = c(adultsurv.out, adultsurv.samp)

}
# Put the output together
PR_adultsurv.post = adultsurv.out[[1]][1:10,]
```

```

for (i in 2:100) PR_adultsurv.post = rbind(PR_adultsurv.post, adultsurv.out[[i]][1:10,])
# Save the model
save(PR_adultsurv.post, file="PR_adultsurv.post")
(load('PR_adultsurv.post'))
a.sv.post=data.frame(PR_adultsurv.post)
names(a.sv.post)=c('Intercept','size','min07','map')

```

4 Kernels

In the following example, we build a set of kernels for the posterior mean parameter values and calculate the value of λ in each grid cell on the landscape. This can be repeated for each posterior parameter sample to obtain a posterior sample for λ , however we do not illustrate that here, for brevity.

4.1 Set Up Inputs

```

#== bounds for the IPM
min.size=0
max.size=5
#== number of IPM cells
n.matrix = 100
b=min.size+c(0:n.matrix)*(max.size-min.size)/n.matrix
#== mesh points (midpoints of the cells)
y=0.5*(b[1:n.matrix]+b[2:(n.matrix+1)])
#== width of the cells
h=y[2]-y[1]

#== set mean parameters
gr.params.mean=apply(gr.reg$Sol,2,mean) # growth
gr.params.sd=mean(gr.reg$VCV[, 'units']) # growth variance
sv.s.params=apply(s.sv.post,2,mean) # seedling survival
sv.a.params=apply(a.sv.post,2,mean) # adult survival
sv.a.params[1]=sv.a.params[1]
nseeds=exp(mean(se.reg$Sol)+0.5 * mean(se.reg$VCV)) # # seeds/seedhead
germ=mean(germ.post)
flower.params=apply(fl.reg,2,mean)
offspr.params.mean=apply(offspr.reg$Sol,2,mean) # recruit size
offspr.param.sd=mean(offspr.reg$VCV[, 'units']) # recruit size variance
fec1.params=apply(fec1.reg$Sol,2,mean)
#
#== specify formulas for use in vital rate functions (defined below)
#== although the functions below can build these automatically, we found that
#== computation was much faster if we supplied them
offspring.form=as.formula(paste('~',paste(names(offspr.params.mean)[-1], collapse='+'))))
offspring.form=as.formula(paste('~',paste(names(offspr.params.mean)[-1], collapse='+'))))
seedhead.form=as.formula(paste('~',paste(names(fec1.params)[-1], collapse='+'))))
flower.form=as.formula(paste('~',paste(names(flower.params)[-1], collapse='+'))))
#== use the whole landscape (or subset during exploration)
cfr.env.subset=as.matrix(cfr.env)
#
#== set up indices for cells being used
(ncells=nrow(cfr.env.subset))
use=1:nrow(cfr.env.subset)

```

```
#
#== set up splitting for parallelizing
ntasks=7
chop.tasks=function(x,n) split(x, cut(seq_along(x), n, labels = FALSE))
task.split=chop.tasks(use,ntasks)
```

4.2 Vital Rate Functions

In this section, we define the vital rate functions that efficiently predict based on the regression models fit above. With many regression packages `predict()` functions are available in which the user can specify new values of the data (other than those used during fitting). Unfortunately, the `predict()` function for `MCMCglmm` does not support prediction for new data, so we constructed our own functions for each vital rate model. This can also be a useful strategy if the predict functions are slow. Building kernels across a landscape can require many thousands of calls to these functions, hence optimizing them to include only the critical functionality (and not overhead that makes them very general) can substantially decrease computation time.

```
#== growth function used to build IPM below
gr=function(size,size.next,mean.params,sd.param,covariates){
  temp.form=as.formula(paste('~',paste(names(mean.params)[-1], collapse='+'))
  design=model.matrix(temp.form, data=covariates)
  dnorm(size.next,mean=size+design%*%mean.params,sd=sqrt(sd.param))
}
#== define survival function
sv=function(size,seedling.params,adult.params,covariates,min.s=.182,max.s=.728){
  #== the default min (max) sizes are chosen to be the mean predicted size of 1
  #== (4) year olds across the region
  #== seedlings are size<max.s
  if(any(size<max.s)){
    design=as.matrix(data.frame(Intercept=1,size=size,
    covariates[,na.omit(match(names(seedling.params), names(covariates)))]))
    u=exp(design%*%seedling.params)
    sv.seed=u/(1+u)
  } else {
    sv.seed=NULL
  }
  #== adults are size>= min.s
  if(any(size>=min.s)){
    temp.form=as.formula(paste('~',paste(names(adult.params)[-1], collapse='+'))
    design=model.matrix(temp.form, data=data.frame(size=size,covariates))
    u=exp(design%*%adult.params)
    sv.ad=u/(1+u)
  } else {
    sv.ad=NULL
  }

  # smooth between 2 arbitrary sizes (min.s and max.s)
  int=size>=min.s & size<=max.s
  smoothed= sv.seed[int] * (max.s/(max.s-min.s)-(1/(max.s-min.s))*size[int])+
    sv.ad[int] * (1-(max.s/(max.s-min.s)-(1/(max.s-min.s))*size[int]))

  return(c(sv.seed[size<min.s],smoothed,sv.ad[size>max.s]))
}
#== seedhead function used to build IPM below
seedhead=function(size,fec1.params,covariates,seedhead.form=NULL){
  if(is.null(seedhead.form))
```

```

    seedhead.form=as.formula(paste('~',paste(names(fec1.params)[-1], collapse='+')))
    design=model.matrix(seedhead.form, data=data.frame(size,covariates))
    exp(design%*%fec1.params)
  }
#== offspring size function used to build IPM below
offspring=function(size,next,mean.params,sd.param,covariates,offspring.form=NULL){
  if(is.null(offspring.form))
    offspring.form=as.formula(paste('~',paste(names(mean.params)[-1], collapse='+')))
  design=model.matrix(offspring.form, data=covariates)
  dnorm(size,next,mean=design%*%mean.params,sd=sqrt(sd.param))
}
# for plotting
offspring.mean=function(reg,covariates){
  temp.form=as.formula(paste('~',as.character(reg$Fixed$formula)[3]))
  design=model.matrix(temp.form, data=covariates)
  design%*%apply(reg$Sol,2,mean)
}
#== flowering function, used for building IPM below
flower=function(size,params,covariates,flower.form=NULL) {
  if(is.null(flower.form))
    flower.form=as.formula(paste('~',paste(names(params)[-1], collapse='+')))
  design=model.matrix(flower.form, data=data.frame(size,covariates))
  plogis(design%*%params)
}
#== fecundity function combining all components
f.yx=function(xp,x,params) {
  params$establishment.prob*
  dnorm(xp,mean=params$recruit.size.mean,sd=params$recruit.size.sd)*
  exp(params$seed.int+params$seed.slope*x)
}

```

4.3 Build growth/survival kernel

```

ptm=proc.time() #often useful to time computations, at least initially
p.post.mean=foreach(i = 1:ntasks,.packages="Matrix") %dopar% {
  post.temp=list(p=lapply(1:length(task.split[[i]]), function(i) 1))
  for(k in 1:length(task.split[[i]])){
    ind=task.split[[i]][k]
    tenv=data.frame(t(cfr.env.subset[ind,]))
    # growth kernel
    P=h*outer(y,y,gr,gr.params.mean,gr.params.sd,tenv)
    # survival
    S=sv(y,sv.s.params,sv.a.params,tenv)
    # growth/survival kernel
    Ps=matrix(rep(apply(P,2,sum),n.matrix),byrow=T,nrow=n.matrix)
    Ss=matrix(rep(S,n.matrix),byrow=T,nrow=n.matrix)
    P=(Ss*P)/Ps
    post.temp[[k]]=shrink.matrix(P)
  }
  post.temp
}
ptm2=proc.time()-ptm
writeLines(paste("Wait time: ",round(ptm2[3],2),"s, ",
  round((ptm2)[3]/ncells,3),"s Per Cell",sep=""))

```

```

p.post.mean=unlist(p.post.mean,recursive=F)
#== save for later use, since this can be slow to compute (~1 min with 3 cores our laptop)
#== note that this file is around 150Mb.
save(p.post.mean,file= 'mean_growth_kernel.post')

```

4.4 Build fecundity kernel

```

ptm=proc.time()
f.post.mean=foreach(i = 1:ntasks,.packages="Matrix") %dopar% {
  post.temp=list(F=lapply(1:length(task.split[[i]]), function(x) 1))
  for(k in 1:length(task.split[[i]])){
    ind=task.split[[i]][k]
    tenv=data.frame(t(cfr.env.subset[ind,]))
    offspr.size=offspring(size.next=y,mean.params=offspr.params.mean,
sd.param=offspr.param.sd, tenv,offspring.form)
    tf=matrix(rep(flower(y,flower.params,tenv,flower.form),n.matrix),
byrow=T,nrow=n.matrix)
    tsh=matrix(rep(seedhead(y,fec1.params,tenv,seedhead.form),n.matrix),
byrow=T,nrow=n.matrix)
    F=h*offspr.size*nseeds*germ*tf*tsh
    post.temp[[k]]=shrink.matrix(F,1e-3)
  }
  post.temp
}
ptm2=proc.time()-ptm
writeLines(paste("Wall time: ",round(ptm2[3],2),"s, ",
round((ptm2)[3]/ncells,3),"s Per Cell",sep=""))
f.post.mean=unlist(f.post.mean,recursive=F)
#== save for later use, since this can be slow to compute (~1 min with 3 cores our laptop)
#== note that this file is around 200Mb.
save(f.post.mean,file= 'mean_fec_kernel.post')

```

5 Analyses

5.1 Population Growth Rate (λ)

```

hetero.fire=TRUE # alterantive is to specify a common fire interval for all sites
sim=foreach(i1 = 1:ntasks) %dopar% { # do for sequential
  lam.temp=list(lapply(1:length(task.split[[i1]]), function(i) 1))
  if(hetero.fire){fire.interval=cfr.env.subset[, 'mean.tsf']}
  if(!hetero.fire){fire.interval=rep(fire.return.time, nrow(cfr.env.subset))}
  for(i in 1:length(task.split[[i1]])){
    ind=task.split[[i1]][i]
    P=t(p.post.mean[[ind]])
    temp=f.post.mean[[ind]]%%P
    for(ii in 1:(fire.interval[ind]-2)){ temp=temp%%P }
    lam.temp[[i]]= Re(eigen(temp)$values[1])^(1/fire.interval[ind])
  }
  lam.temp
}

```

```

lam.temp=unlist(sim,recursive=F) # use this to unlist the parallel version
lam=unlist(lam.temp)
lam[is.nan(lam)]=0 #turn NaNs (from /0) to 0s
#== save for later use
save(lam, file='lambda.pred')

```

We found the following to be a useful diagnostic plot during model building.

```

#== make a raster from the dataframe with cfr environmetnal data
cfr.r=cfr.env
coordinates(cfr.r)=c('lon','lat')
cfr.r=SpatialPixelsDataFrame(cfr.r,tol=0.0001,data=data.frame(cfr.r))
fullgrid(cfr.r)=TRUE
cfr.r=stack(cfr.r)
#== set up object for
lam.cfr=cfr.r[[1]] #placeholder with right spatial structure
lam.cfr[!is.na(values(lam.cfr))]=lam
#== evalute predcition accuracy against persence/absence data
threshold=1 # value of lambda used to determin a correct prediction
rocr.data=list( predictions=values(lam.cfr)[!is.na(values(cfr.r[['Presence_5x5']]))],
  labels=values(cfr.r[['Presence_5x5']])[!is.na(values(cfr.r[['Presence_5x5']]))])
pred1 <- prediction( rocr.data$predictions, rocr.data$labels)
perf1 <- performance(pred1,"tpr","tnr")
p.cor.a=perf1@x.values[[1]][tail(which(perf1@alpha.values[[1]]>1),1)]
p.cor.p=perf1@y.values[[1]][tail(which(perf1@alpha.values[[1]]>1),1)]
auc= performance(pred1,"auc")@y.values[[1]]
#== it's useful to write the plot to a pdf to allow a high resolution display
pdf('mean_lam_map_temp.pdf',h=5,w=5)
par(mfrow=c(2,1),mar=c(.2,0,3,1))
#== panel a
image.plot(lam.cfr,
  col=c(grey(seq(.7,.1,length=100)),colorRampPalette(c('steelblue4','steelblue1',
  'gold','red1','red4'))(35)),
  xaxt='n', yaxt='n', bty='n',main=' ',
  at=c(seq(0,threshold-.0001,len=100),seq(threshold,1.35,len=10)),zlim=c(0,1.35))
text(23.5,-31.4,paste(100*round(p.cor.p,2),'% presences correct'),cex=1)
text(23.5,-31.9,paste(100*round(p.cor.a,2),'% absences correct'),cex=1)
text(23.5,-32.4,paste('AUC = ',round(auc,2)),cex=1)
#== panel b
image.plot(lam.cfr,col='grey70',xaxt='n', yaxt='n', bty='n',main=' ',
  at=c(seq(0,.9999,len=20),seq(1,1.5,len=10)),zlim=c(0,1.3),legend=F)
p.correct= coordinates(cfr.r[['Presence_5x5']])[values(cfr.r[['Presence_5x5']])==1
  & !is.na(values(cfr.r[['Presence_5x5']])) & values(lam.cfr[[1]])>=threshold,]
p.incorrect= coordinates(cfr.r[['Presence_5x5']])[values(cfr.r[['Presence_5x5']])==1
  & !is.na(values(cfr.r[['Presence_5x5']])) & values(lam.cfr[[1]])<threshold,]
a.correct= coordinates(cfr.r[['Presence_5x5']])[values(cfr.r[['Presence_5x5']])==0
  & !is.na(values(cfr.r[['Presence_5x5']])) & values(lam.cfr[[1]])<threshold,]
a.incorrect= coordinates(cfr.r[['Presence_5x5']])[values(cfr.r[['Presence_5x5']])==0
  & !is.na(values(cfr.r[['Presence_5x5']])) & values(lam.cfr[[1]])>=threshold,]
points(p.incorrect[,1],p.incorrect[,2],cex=.1,pch=15,col='red')
points(a.incorrect[,1],a.incorrect[,2],cex=.15,pch=17,col='orange')
points(p.correct[,1],p.correct[,2],cex=.1,pch=15)
points(a.correct[,1],a.correct[,2],cex=.15,pch=17,col='grey20')
legend('topright',c('correct presence','incorrect presence','correct absence','incorrect absence'),
  col=c('black','red','grey20','orange'), pch=c(15,15,17,17),cex=.8,bty='n')

```



```
dev.off()  
system("open mean_lam_map_temp.pdf") #to open a pdf in the working directory (on a *nix system)
```

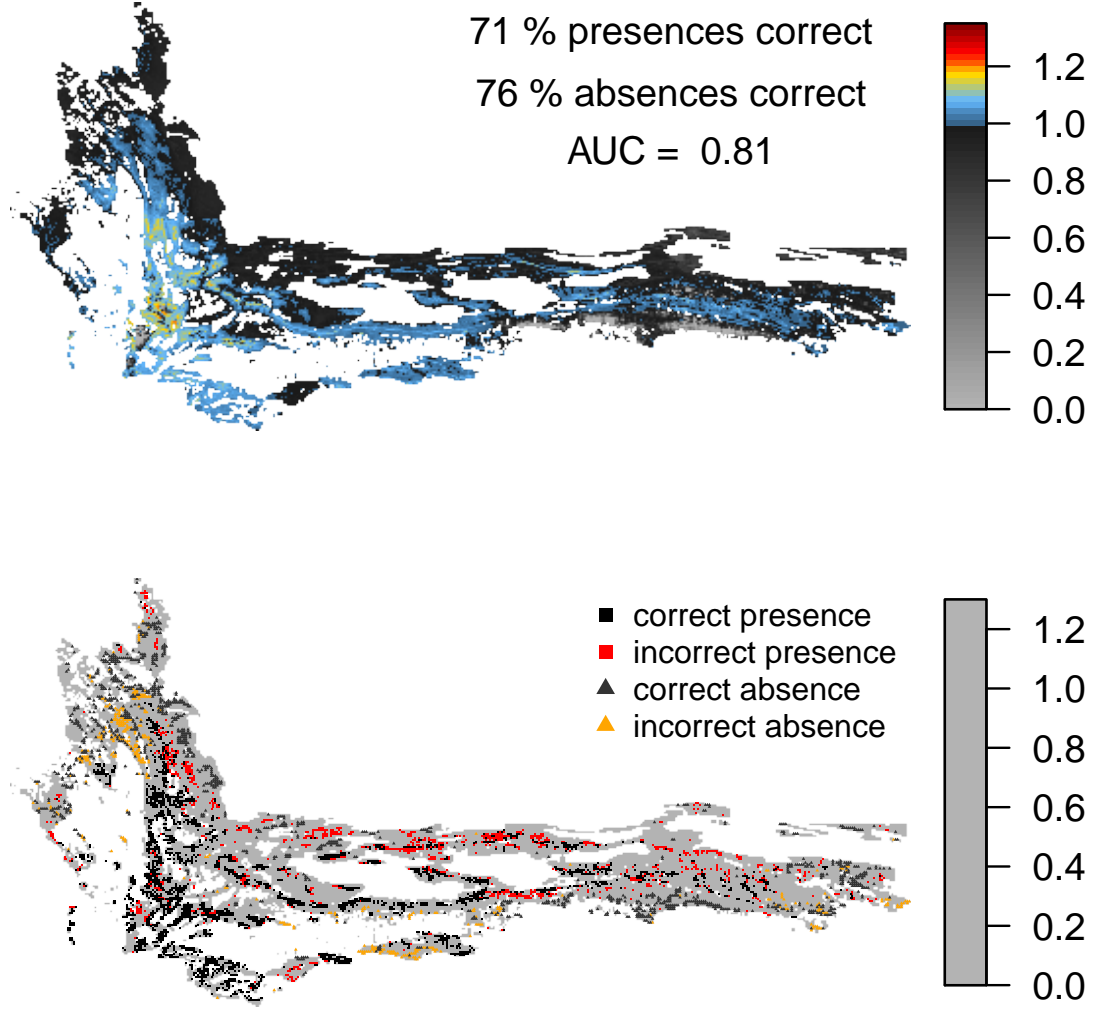


Figure 1: A sample diagnostic plot, useful during model exploration.

5.2 Uncertainty Analysis

For brevity, we simply explain how the code for analyses above can be modified to calculate the posterior distribution of λ . In Section 4.1, rather than using the posterior mean parameter values, we can use a sample of parameters from the posterior. We can loop over the construction of the growth/survival kernel, fecundity kernel, and calculation of λ for number of desired posterior samples. Because the number of samples is often large, it can become impractical to store the large kernel files. However, it is worthwhile to determine the amount of space necessary to store all the kernels to avoid recalculating them if possible (which will depend on the number of cells on the landscape, the resolution of the matrix and the number of posterior samples). For our model, calculations for 1000 posterior samples took approximately 1.5 days when parallelizing computations over 12 nodes.

5.3 Parameter Elasticity

For brevity, we show the calculations for intercepts and size-slope parameters (shown in Fig. 5 of the main text). Analogous calculations can be performed for any regression parameter

```
#== increase each of the size parameters by .01 and recalculate
delta=1e-2
offspr.form=as.formula(paste('~',paste(names(offspr.params.mean)[-1], collapse='+'))))
seedhead.form=as.formula(paste('~',paste(names(fec1.params)[-1], collapse='+'))))
flower.form=as.formula(paste('~',paste(names(flower.params)[-1], collapse='+'))))
cfr.env.subset=as.matrix(cfr.env)
#== for size predictors
predictor.names1=c('growth_int','growth_var','sv_s_int','sv_s_sl',
  'sv_a_int','sv_a_sl','seeds_int','germ_int','fl_int','fl_sl','off_int',
  'off_sd','fec1_int','fec1_sl')
for(ii in 1:length(predictor.names1)){
  gr.params.mean=apply(gr.reg$Sol,2,mean) # growth
  gr.params.sd=mean(gr.reg$VCV['units']) # growth variance
  sv.s.params=apply(s.sv.post,2,mean) # seedling survival
  sv.a.params=apply(a.sv.post,2,mean) # adult survival
  nseeds=exp(mean(se.reg$Sol)+0.5 * mean(se.reg$VCV)) # seeds/seedhead
  germ=mean(germ.post)
  flower.params=apply(fl.reg,2,mean)
  offspr.params.mean=apply(offspr.reg$Sol,2,mean) # recruit size
  offspr.param.sd=mean(offspr.reg$VCV['units']) # recruit size variance
  fec1.params=apply(fec1.reg$Sol,2,mean)

  #== perturb, in a clunky way
  if(ii==1){gr.params.mean[1]=gr.params.mean[1]*(1+delta)}
  if(ii==2){gr.params.sd=gr.params.sd*(1+delta)}
  if(ii==3){sv.s.params[1]=sv.s.params[1]+abs(sv.s.params[1]*delta)}
  if(ii==4){sv.s.params[2]=sv.s.params[2]*(1+delta)}
  if(ii==5){sv.a.params[1]=sv.a.params[1]+abs(sv.a.params[1]*delta)}
  if(ii==6){sv.a.params[2]=sv.a.params[2]+abs(sv.a.params[2]*delta)}
  if(ii==7){nseeds=nseeds*(1+delta)}
  if(ii==8){germ=germ*(1+delta)}
  if(ii==9){flower.params[1]=flower.params[1]+abs(flower.params[1]*delta)}
  if(ii==10){flower.params[2]=flower.params[2]+abs(flower.params[2]*delta)}
  if(ii==11){offspr.params.mean[1]=offspr.params.mean[1]*(1+delta)}
  if(ii==12){offspr.param.sd=offspr.param.sd*(1+delta)}
  if(ii==13){fec1.params[1]=fec1.params[1]+abs(fec1.params[1]*delta)}
  if(ii==14){fec1.params[2]=fec1.params[2]+abs(fec1.params[2]*delta)}

  #== calculation of growth/survival kernel, fecundity kernel, and lambda are
  #== exactly the same as in the sections above.
  #== GROWTH KERNEL =====
  p.post.mean=foreach(i = 1:ntasks,.packages="Matrix") %dopar% {
    post.temp=list(p=lapply(1:length(task.split[[i]]), function(i) 1))
    for(k in 1:length(task.split[[i]])){
      ind=task.split[[i]][k]
      tenv=data.frame(t(cfr.env.subset[ind,]))
      P=h*outer(y,y,gr,gr.params.mean,gr.params.sd,tenv)
      S=sv(y,sv.s.params,sv.a.params,tenv)
      Ps=matrix(rep(apply(P,2,sum),n.matrix),byrow=T,nrow=n.matrix)
      Ss=matrix(rep(S,n.matrix),byrow=T,nrow=n.matrix)
      P=(Ss*P)/Ps
    }
  }
```

```

        post.temp[[k]]=shrink.matrix(P)
    }
    post.temp
}
p.post.mean=unlist(p.post.mean,recursive=F)

#== FECUNDITY KERNEL =====
f.post.mean=foreach(i = 1:ntasks,.packages="Matrix") %dopar% {
    post.temp=list(F=lapply(1:length(task.split[[i]]), function(x) 1))
    for(k in 1:length(task.split[[i]])){
        ind=task.split[[i]][k]
        tenv=data.frame(t(cfr.env.subset[ind,]))
        offspr.size=offspring(size.next=y,mean.params=offspr.params.mean,
sd.param=offspr.param.sd, tenv,offspr.form)
        tf=matrix(rep(flower(y,flower.params,tenv,flower.form),n.matrix),
byrow=T,nrow=n.matrix)
        tsh=matrix(rep(seedhead(y,fec1.params,tenv,seedhead.form),n.matrix),
byrow=T,nrow=n.matrix)
        F=h*offspr.size*nseeds*germ*tf*tsh
        post.temp[[k]]=shrink.matrix(F,1e-3)
    }
    post.temp
}
f.post.mean=unlist(f.post.mean,recursive=F)

#== Simulate Dynamics =====
hetero.fire=TRUE
sim=foreach(i1 = 1:ntasks) %dopar% {      # do for sequential
    lam.temp=list(lapply(1:length(task.split[[i1]]), function(i) 1))
    if(hetero.fire){fire.interval=cfr.env.subset[, 'mean.tsf']}
    if(!hetero.fire){fire.interval=rep(fire.return.time, nrow(cfr.env.subset))}
    for(i in 1:length(task.split[[i1]])){
        ind=task.split[[i1]][i]
        P=t(p.post.mean[[ind]])
        temp=f.post.mean[[ind]]%*%P
        for(ii in 1:(fire.interval[ind]-2)){ temp=temp%*%P }
        lam.temp[[i]]= Re(eigen(temp)$values[1])^(1/fire.interval[ind])
    }
    lam.temp
}
lam.temp=unlist(sim,recursive=F)
lam=unlist(lam.temp)
lam[is.nan(lam)]=0
#== save for later use
save(lam, file=paste('lam_sens_p.01_',predictor.names1[ii],'.pred',sep=''))
print(ii)
}

```

Using the values of λ from the perturbed parameter values above, we next calculate the parameter elasticity.

```

#== Load for calculating sensitivities
params=list( apply(gr.reg$Sol,2,mean)[1], mean(gr.reg$VCV[, 'units']),
    apply(s.sv.post,2,mean)[1], apply(s.sv.post,2,mean)[2],

```

```

        apply(a.sv.post,2,mean)[1], apply(a.sv.post,2,mean)[2],
        exp(mean(se.reg$Sol)+0.5 * mean(se.reg$VCV)),
        mean(germ.post),
        apply(fl.reg,2,mean)[1], apply(fl.reg,2,mean)[2],
        apply(offspr.reg$Sol,2,mean)[1], mean(offspr.reg$VCV[, 'units']),
        apply(fec1.reg$Sol,2,mean)[1], apply(fec1.reg$Sol,2,mean)[2]
    )
names(params)=predictor.names1
(load('lambda.pred'))
true.lam=lam
lam.sens=lam.elas=list()
for(i in 1:length(params)){
    load(paste('lam_sens_p.01_', predictor.names1[i], '.pred', sep=''))
    lam.sens[[i]]=(lam-true.lam)/(params[[i]]*delta)
    lam.elas[[i]]=(lam-true.lam)/(true.lam*delta)
}
names(lam.sens)=names(lam.elas)=predictor.names1
sens.stack=stack(lapply(1:length(params),
    function(i){tr=cfr.r[[1]]
        tr[!is.na(values(tr))]=lam.sens[[i]]
        return(tr)
    })))
elas.stack=stack(lapply(1:length(params),
    function(i){tr=cfr.r[[1]]
        tr[!is.na(values(tr))]=lam.elas[[i]]
        return(tr)
    })))
names(sens.stack)=names(elas.stack)=predictor.names1

```

The results can be plotted using `image()` to obtain plots analogous to Fig. 5 in the main text.

5.4 Environmental Sensitivity

```

env.names=c("map", "min07", "smdwin", "smdsum", "ph1", "fert4")
for(ii in 1:length(env.names)){
    cfr.env.subset=as.matrix(cfr.env)
    #== this is the only line that differs from code for kernel construction
    #== and calculation of lambda from the unperturbed analyses above
    cfr.env.subset[, env.names[ii]]=cfr.env.subset[, env.names[ii]]+.1
    #== GROWTH KERNEL =====
    p.post.mean=foreach(i = 1:ntasks, .packages="Matrix") %dopar% {
        post.temp=list(p=lapply(1:length(task.split[[i]]), function(i) 1))
        for(k in 1:length(task.split[[i]])){
            ind=task.split[[i]][k]
            tenv=data.frame(t(cfr.env.subset[ind,]))
            P=h*outer(y,y,gr,gr.params.mean,gr.params.sd,tenv)
            S=sv(y,sv.s.params,sv.a.params,tenv)
            Ps=matrix(rep(apply(P,2,sum),n.matrix),byrow=T,nrow=n.matrix)
            Ss=matrix(rep(S,n.matrix),byrow=T,nrow=n.matrix)
            P=(Ss*P)/Ps
            post.temp[[k]]=shrink.matrix(P)
        }
        post.temp
    }
    p.post.mean=unlist(p.post.mean,recursive=F)
}

```

```

#== FECUNDITY KERNEL =====
f.post.mean=foreach(i = 1:ntasks,.packages="Matrix") %dopar% {
  post.temp=list(F=lapply(1:length(task.split[[i]]), function(x) 1))
  for(k in 1:length(task.split[[i]])){
    ind=task.split[[i]][k]
    tenv=data.frame(t(cfr.env.subset[ind,]))
    offspr.size=offspring(size.next=y,mean.params=offspr.params.mean,
sd.param=offspr.param.sd, tenv,offspring.form)
    tf=matrix(rep(flower(y,flower.params,tenv,flower.form),n.matrix),
byrow=T,nrow=n.matrix)
    tsh=matrix(rep(seedhead(y,fecl.params,tenv,seedhead.form),n.matrix),
byrow=T,nrow=n.matrix)
    F=h*offspr.size*nseeds*germ*tf*tsh
    post.temp[[k]]=shrink.matrix(F,1e-3)
  }
  post.temp
}
f.post.mean=unlist(f.post.mean,recursive=F)

```

```

#== Simulate Dynamics =====
hetero.fire=TRUE
sim=foreach(i1 = 1:ntasks) %dopar% {      # do for sequential
  lam.temp=list(lapply(1:length(task.split[[i1]]), function(i) 1))
  if(hetero.fire){fire.interval=cfr.env.subset[, 'mean.tsf']}
  if(!hetero.fire){fire.interval=rep(fire.return.time, nrow(cfr.env.subset))}
  for(i in 1:length(task.split[[i1]])){
    ind=task.split[[i1]][i]
    P=t(p.post.mean[[ind]])
    temp=f.post.mean[[ind]]*%P
    for(ii in 1:(fire.interval[ind]-2)){ temp=temp*%P }
    lam.temp[[i]]= Re(eigen(temp)$values[1])^(1/fire.interval[ind])
  }
  lam.temp
}
lam.temp=unlist(sim,recursive=F)
lam=unlist(lam.temp)
lam[is.nan(lam)]=0 #turn NaNs (from /0) to 0s

#== save for later use
save(lam, file=paste('lam_sens_p.1_',env.names[ii],'.pred',sep=''))
}

```

From the predictions under perturbation of the environmental conditions calculated above, we can use the unperturbed predictions to calculate the sensitivity to the environmental conditions below.

```

#== true model
(load('lambda.pred'))
true.lam=lam
lam.sens.env=list()
for(i in 1:length(env.names)){
  load(paste('lam_sens_p.1_', env.names[i],'.pred',sep=''))
  lam.sens.env[[i]]=(lam-true.lam)/(.1)
}
names(lam.sens.env)=env.names

```

```

sens.stack.env=stack(lapply(1:length(env.names),
                           function(i){
                             tr=cfr.r[[1]]
                             tr[!is.na(values(tr))]=lam.sens.env[[i]]
                             return(tr) }) )
names(sens.stack.env)=env.names

```

The results can be plotted using `image()` to obtain plots analogous to Fig. 6 in the main text.

6 Functions

6.1 Miscellaneous functions

A function for backward stepwise DIC selection with `MCMCglmm`. Note that the model is constrained to keep all lower order transformations of a predictor if the higher order term is retained. That is, if a quadratic transformation of a predictor is selected, the linear term must also be retained. However a linear term can be selected even if the quadratic term is not.

```

stepDIC=function(form.best,data,keep=NULL,DIC.diff=3,...){
  # keep specifies some terms to keep for sure to speed it up
  cand=attr(terms(form.best),'term.labels')
  model.best=MCMCglmm( form.best, data=data, verbose=FALSE,...)
  DIC.best=model.best$DIC
  cat('full model dic: ',DIC.best[1],'#####\n')
  print(summary(model.best))
  delta.DIC=-Inf
  step.counter=1
  resp=as.character(model.best$Fixed$formula)[2]
  stored.summaries=list()
  # list the linear terms to help formulas below
  linear=cand[-grep('I',cand)]
  while(all(delta.DIC<DIC.diff)){
    cat('-----step   ',step.counter,' -----\n')
    cat('best DIC so far: ', DIC.best , ' \n')
    cand=attr(terms(form.best),'term.labels')
    DICs.cand=vector()
    models=list()
    if(is.null(keep)) which.to.test=(1:length(cand))
    if(!is.null(keep)) which.to.test=(1:length(cand))[-match(keep,cand)]
    stored.summaries[[step.counter]]=list()
    # ensure higher order terms are only in if the linear predictor is

    temp.form=list()
    for(i in 1:length(which.to.test)){
      toss=grep(cand[which.to.test[i]],cand,fixed=T)
      temp.form[[i]]=as.formula(paste(resp,'~', paste(cand[-toss], collapse='+')))
    }

    for(i in 1:length(temp.form)){
      models[[i]]=MCMCglmm( temp.form[[i]], data=data, verbose=FALSE,...)
      DICs.cand[i]=models[[i]]$DIC
      cat('#####', 'model',rep(i,20))
      print(summary(models[[i]]))
      stored.summaries[[step.counter]][[i]]=summary(models[[i]])
    }
  }
}

```

```

    }
    delta.DIC=apply((outer(DICs.cand,DIC.best,'-')),2,min)
    if(all(delta.DIC<DIC.diff)){
      which.best=which.min(DICs.cand- DIC.best[step.counter])
      form.best=models[[which.best]]$Fixed$formula
      DIC.best[step.counter+1]=DICs.cand[which.best]
      model.best=models[[which.best]]
    }
    step.counter=step.counter+1
  }
  cat('+++++ best model +++++')
  print(summary(model.best))

  return(list(model.best=model.best,stored.summaries=stored.summaries))
}

```

A function to reduce the size of large matrices to make storage more practical.

```

shrink.matrix=function(m,min=1e-5){
  m[m<min]=0
  m=as(m,'sparseMatrix')
}

```

Below is a general predict function for MCMCglmm model objects because the existing predict function in package MCMCglmm does not predict for new data (it only predicts at the data values used for fitting.) This function is a modification of the predict.MCMCglmm() function from package MCMCglmm (Hadfield (2010). MCMC Methods for Multi-Response Generalized Linear Mixed Models: The MCMCglmm R Package. Journal of Statistical Software, 33(2), 1-22). This function has been tested primarily for the specific applications performed here (used in the recruitment model above) and we highly recommend against using this for any other purposes without extensive testing first. Some caveats are listed in the function heading.

```

predict.MCMCglmm.cm=function (object, newdata = NULL, marginal = object$Random$formula,
                              type = "response", interval = "none", level = 0.95,
                              return.post.pred=FALSE,...)
{
  #== doesn't work well with factors. only binary factors coded as 0,1 work.
  #== note that random effects in new data need to be named as their columnname.#
  #== if you want to predict with Random effects and use new data, you've got to
  #== specify the values of the Random effects in the newdata, which is a little annoying.
  #== currently only works for 1 Random effect
  #== unfortunately, if you want to do transformations of variables, they need
  #== separate columns, as this function does not make them for you like most predict functions do.

  if (is.null(object$Random$nf1) == FALSE) {
    rcomponents <- (as.character(object$Random$formula)[2])
    mcomponents <- (as.character(marginal)[2])
    marginal <- rep(rep(as.numeric(rcomponents %in% mcomponents),
                        object$Random$nr1), object$Random$nf1)

    st <- c(1, cumsum(rep(object$Random$nr1, object$Random$nf1)) + 1)
    st <- st[-length(st)]
    end <- cumsum(rep(object$Random$nr1, object$Random$nf1))
    comp <- rep(1:length(object$Random$nf1), object$Random$nf1)
    keep <- unlist(mapply(st[which(marginal == 0)], end[which(marginal ==
0)], FUN = ":"))
  } else {

```



```

        keep <- NULL
        rand = NULL
        marginal=-1
    }

    object$Sol <- object$Sol[, c(1:object$Fixed$nf1, object$Fixed$nf1 + keep), drop = FALSE]

    if ( is.null(newdata) ){
        W <- cBind(object$X, object$Z)
        W <- W[, c(1:object$Fixed$nf1, object$Fixed$nf1 + keep), drop = FALSE]
    } else {
        X=matrix(NA,nrow(newdata),object$Fixed$nf1)
        temp.form=as.formula(paste('~',as.character(object$Fixed$formula)[3]))
        for(i in 1:nrow(newdata)){
            X[i,]=model.matrix(temp.form, data=newdata[i,])
        }
        if(marginal==0) {
            rand= paste(labels(terms(object$Random$formula)),
1:object$Random$nr1,sep='.')
        } else {
            rand=NULL
        }
        W=cbind(X,as.matrix(newdata[,rand]))

        W=as(W,"dgCMatrix")
    }

# only for prediction interval, and probably grabbing variance
    if ((type == "response" & any(object$family != "gaussian" &
        object$family != "cengaussian")) | interval == "prediction") {
        if(is.null(newdata)){
            vpred <- matrix(0, dim(object$X)[1],
sum(object$Random$nf1[which(marginal ==
            1)]^2, na.rm = T) + sum(object$Residual$nf1^2))
            cnt <- 0
            if (any(marginal == 1)) {
                st <- st[which(marginal == 1)]
                end <- end[which(marginal == 1)]
                comp <- comp[which(marginal == 1)]
                for (i in 1:length(st)) {
                    for (j in 1:length(st)) {
                        if (comp[i] == comp[j]) {
                            cnt <- cnt + 1
                            vpred[, cnt] <- diag(object$Z[, st[i]:end[i]] %*%
                                t(object$Z[, st[j]:end[j]]))
                        }
                    }
                }
            }
            comp <- rep(1:length(object$Residual$nf1), object$Residual$nf1)
            for (i in 1:length(comp)) {
                for (j in 1:length(comp)) {
                    if (comp[i] == comp[j]) {
                        cnt <- cnt + 1

```

```

                                vpred[, cnt][which(object$error.term == i &
                                object$error.term == j)] <- 1
                                }
                                }
                                }
} else {
    vpred=matrix(1,nrow(newdata),1)
}
if (is.null(object$Random$nf1) == FALSE) {
    keep <- which(rep(rep(as.numeric(rcomponents %in% mcomponents),
object$Random$nrt), object$Random$nf1^2) == 1)
} else {
    keep=NULL
}
keep <- c(keep, which(rep(rep(rep(1, length(object$Residual$nrt)),
    object$Residual$nrt), object$Residual$nf1^2) == 1))
postvar <- t(apply(object$VCV[, keep, drop = FALSE],
    1, function(x) { (vpred %*% x)}))
}

# predictions made here
post.pred <- t(apply(object$Sol, 1, function(x) { (W %*% x)@x })))

# probably samples post pred with right variance to make prediction interval
if (interval == "prediction") {
    post.pred <- matrix(rnorm(prod(dim(post.pred))), post.pred,
        sqrt(postvar)), dim(post.pred)[1], dim(post.pred)[2])
}

# changes prediction to response scale
if (type == "response") {
    if (any(object$family %in% c("poisson", "cenpoisson",
        "multinomial", "categorical", "gaussian", "cengaussian",
        "ordinal") == FALSE)) {
        stop("sorry - prediction on data scale not implemented for this family")
    }
    if (any(object$family %in% c("poisson", "cenpoisson"))) {
        if(is.null(newdata)){
            keep <- which(object$family %in% c("poisson", "cenpoisson"))

        } else {
            keep <- 1:nrow(newdata)
        }
        if (interval == "prediction") {
            post.pred[, keep] <- exp(post.pred[, keep])
        } else {
            post.pred[, keep] <- exp(post.pred[, keep] +
                0.5 * postvar[,keep])
        }
    }
    if (any(object$family %in% c("multinomial", "categorical"))) {
        c2 <- (16 * sqrt(3)/(15 * pi))^2
        if(is.null(newdata)){
            keep <- which(object$family %in% c("multinomial", "categorical"))

```

```

    } else {
      keep <- 1:nrow(newdata)
    }
    if (interval == "prediction") {
      post.pred[, keep] <- plogis(post.pred[, keep])
    } else {
      post.pred[, keep] <- plogis(post.pred[, keep]/sqrt(1 +
        c2 * postvar[, keep]))
    }
  }
  if (any(object$family %in% c("ordinal"))) {
    if(is.null(newdata)){
      keep <- which(object$family %in% c("ordinal"))
    } else {
      keep <- 1:nrow(newdata)
    }
    CP <- cbind(-Inf, 0, object$CP, Inf)
    q <- matrix(0, dim(post.pred)[1], length(keep))
    if (interval == "prediction") {
      for (i in 2:(dim(CP)[2] - 1)) {
        q <- q + (pnorm(CP[, i + 1] - post.pred[, keep]) -
          pnorm(CP[, i] - post.pred[, keep])) * (i -
            1)
      }
    } else {
      for (i in 2:(dim(CP)[2] - 1)) {
        q <- q + (pnorm(CP[, i + 1] - post.pred[, keep],
          0, sqrt(postvar[, keep] + 1)) - pnorm(CP[,
            i] - post.pred[, keep], 0, sqrt(postvar[,
              keep] + 1))) * (i - 1)
      }
    }
    post.pred[, keep] <- q
    rm(q)
  }
}

# process for final output
pred <- matrix(colMeans(post.pred), dim(post.pred)[2], 1)
if (!interval == "none") {
  pred <- cbind(pred, coda::HPDinterval(mcmc(post.pred), prob = level))
  colnames(pred) <- c("fit", "lwr", "upr")
}
rownames(pred) <- 1:dim(pred)[1]
if(return.post.pred) return(post.pred)
if(!return.post.pred) return(pred)
}

```