A modelling framework for assessing lethal and sublethal effects of GM maize pollen on non target Lepidoptera

Emily Walker, Antoine Messéan, Samuel Soubeyrand, Melen Leclerc, Jean–Francois Rey

INRA (France)
BioSP - EcoInnov

April 2016 - Future for Butterflies
Genetically modified crops and Lepidoptera session

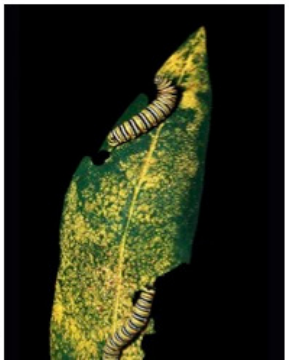
Context

To provide tools for risk assessors and managers: to **assess risk** of **GM pollen** dispersal (from Bt maize) on **non target organisms** (NTO, butterflies) at the **landscape scale**.

- AMIGA project (FP7, European project): Assessing and Monitoring the Impacts of Genetically modified (GM) plants on Agro-ecosystems
- Towards a more quantitative risk assessment of Bt maize cultivation

Objectives:

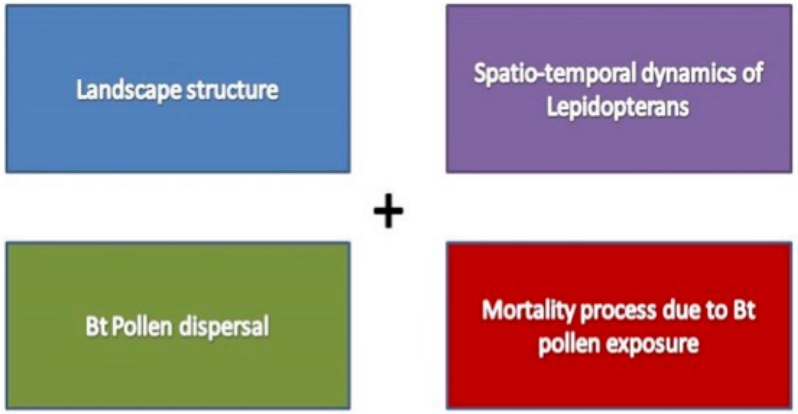
- Design generic **spatially explicit models** for assessing the impacts of GM crops on non–target organisms within agricultural landscapes
- Application to impacts of Bt maize on non–target Lepidoptera larvae (*Inachis io*)



Approach: model structure

Develop a generic landscape model for quantifying and analysing the impacts of Bt maize cultivation on non–target organisms (butterflies).

To assess the impacts of the introduction of Bt maize cultivation depending on receiving environments and management scenarios.



Approach: modelling choices

Spatial scale = regional (landscape)
Temporal scale = day
One larval stage = 1st larval stage

Steps forward compared to existing models (e.g. Perry et al. 2012; Holst et al., 2013):

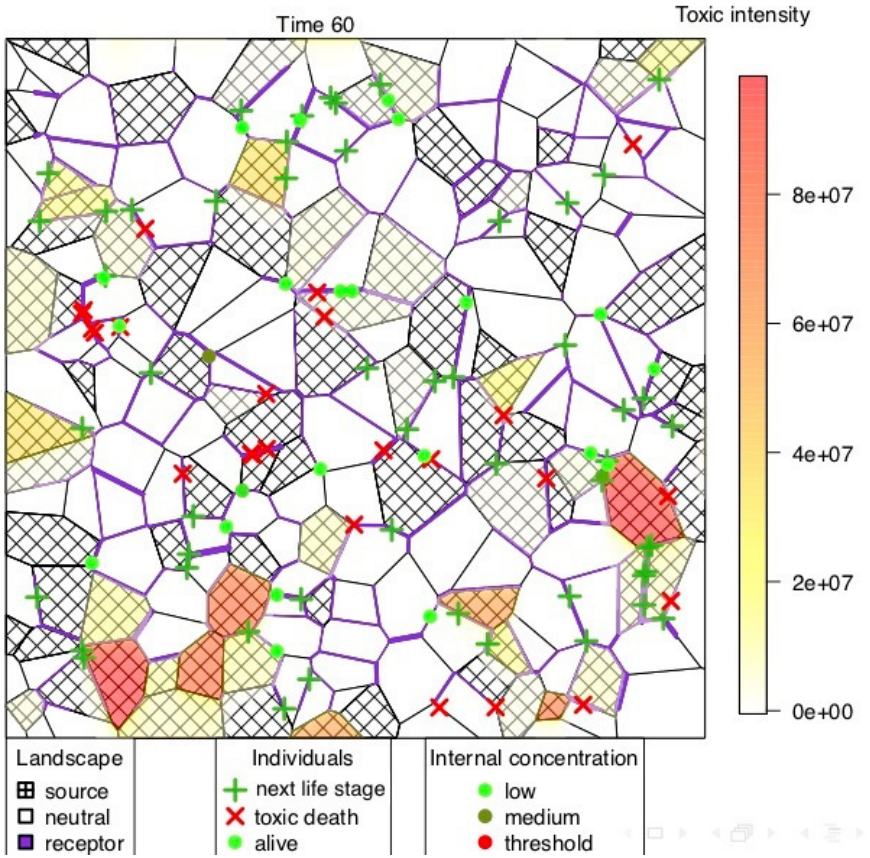
- both spatially and temporally explicit
- different dispersal kernels can be considered
- temporal effects taken into account for both fields (flowering) and butterflies ⇒ phenology
- ecotoxicological model to assess influence of this process on the results

⇒ R package named **briskaR (biological risk assessment with R)**

- Calculating the individual risk,

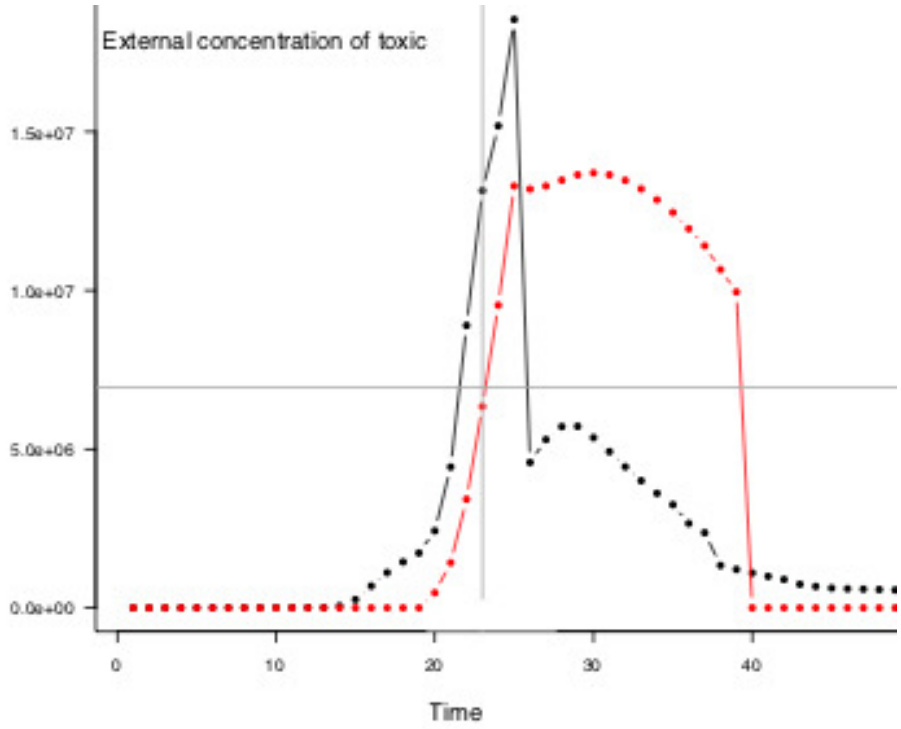
  conditional on the temporal evolution of the external concentration of toxic

End of simulation



Question : Comment modéliser la survenue de la mortalité d'un individu au cours d'une séquence temporelle d'exposition à un toxique ?

Exemples d'expositions au cours du temps subies par 2 individus localisées en 2 endroits différents :



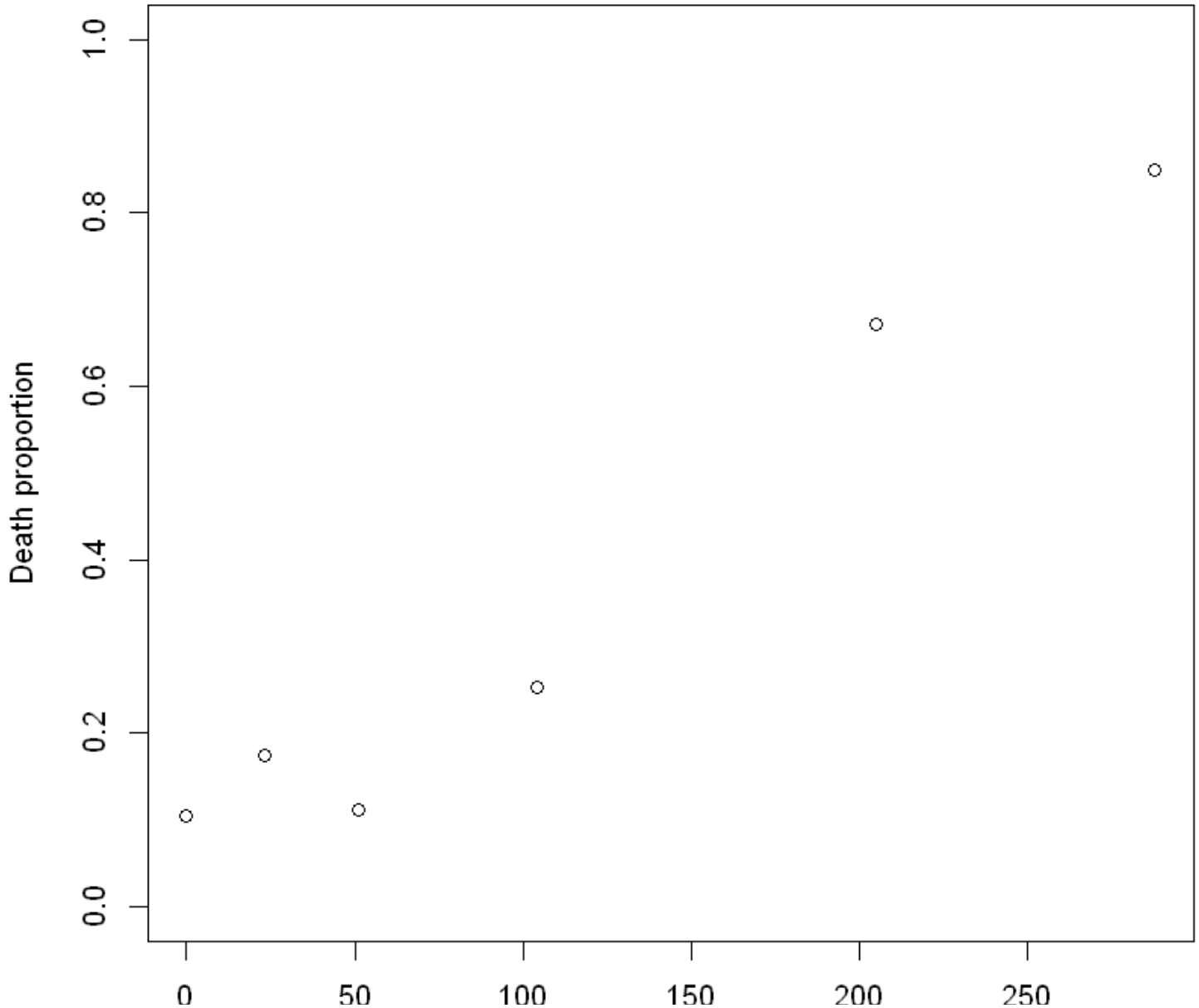
Quand la mort survient, si elle survient, au cours de telles séquences d'exposition ?

Difficulté : pas de données pour des courbes telles que celles présentées ci-dessus mais des données expérimentales pour des processus d'exposition simplifiés

Felke et al. (2010) : larves de paons du jour (*Inachis io*) exposées à du pollen de maïs Bt-176

Données

In [9]:
```r
## (nombre de grains de pollen / cm2) x (2 cm2 ; surface de la portion de feuille) x (40 répétitions)
tox.dose=c(0,23,51,104,205,288)
tox=tox.dose*2*40

## proportion de mortalité
mortality=c(42, 70, 45, 101, 269, 340)/400
names(mortality)=paste("dose",tox.dose,sep="_")

## display data
plot(tox.dose,mortality,ylim=c(0,1),
     xlab="Toxic dose (# toxic pollen grains / cm2)",
     ylab="Death proportion")
```


Death proportion
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250

Modèle de mortalité pour un groupe de n individus

X: dose toxique administrée au groupe de n individus

p_{ij}: proportion de la dose de pollen toxique ingérée par la larve i le jour $j \in \{1, 2\}$ (on partage le *gâteau* en 2n parts éventuellement inégales)

$$(p_{11}, \ldots, p_{n1}, p_{12}, \ldots, p_{n2}) \sim \text{Dirichlet}\left(\frac{1}{\beta}, \ldots, \frac{1}{\beta}\right)$$

$$E(p_{ij}) = \frac{1}{2n} \qquad V(p_{ij}) = \frac{n-1}{n^2(1+n/\beta)} \qquad Cov(p_{ij}, p_{i'j'}) = \frac{-1}{n^2(1+n/\beta)} \qquad (i,j) \neq (i',j')$$

β: paramètre gouvernant la variance des proportions

ρ_{ij}: concentration interne de "toxiques" dans la larve i le jour $j \in \{1, 2, 3, \ldots\}$ (les 2 premiers jours, la larve ingère une dose toxique; à partir du jour 2, la larve élimine une proportion k_{out} de la concentration interne du jour passé)

$$\begin{aligned}
\rho_{i1} &= X p_{i1} \\
\rho_{i2} &= X p_{i1}(1 - k_{\text{out}}) + X p_{i2} \\
\rho_{ij} &= X p_{i1}(1 - k_{\text{out}})^{j-1} + X p_{i2}(1 - k_{\text{out}})^{j-2} \qquad j \geq 2
\end{aligned}$$

Y_{ij}: variable binaire indiquant si la larve i est vivante ($Y_{ij} = 1$) ou morte ($Y_{ij} = 0$) le jour j.

$$\begin{aligned}
Y_{i0} &= 1 \\
Y_{ij} &\underset{\text{indép.}}{\sim} \text{Bernoulli}\Big(\alpha_1 \exp\big(-(\alpha_2 \rho_{ij})^{\alpha_3}\big) Y_{i(j-1)}\Big) \qquad j \geq 2
\end{aligned}$$

$\alpha_1 \in [0, 1]$: probabilité de mortalité journalière "naturelle"

$\alpha_2, \alpha_3 > 0$: paramètres gouvernant l'effet de la concentration toxique interne sur la mortalité journalière

Estimation

Paramètres : $(\beta, k_{\text{out}}, \alpha_1, \alpha_2, \alpha_3)$

Données : taux de mortalité global (i.e. sur 400 individus) au 7ème jour pour chaque dose

Estimation bayésienne classique : nombreuses variables latentes dépendantes (400x2x6 p_{ij} et 400x6x6 Y_{ij})

Recours à l'ABC, en utilisant les taux de mortalité globaux au 7ème jour et des simulations du modèle de mortalité

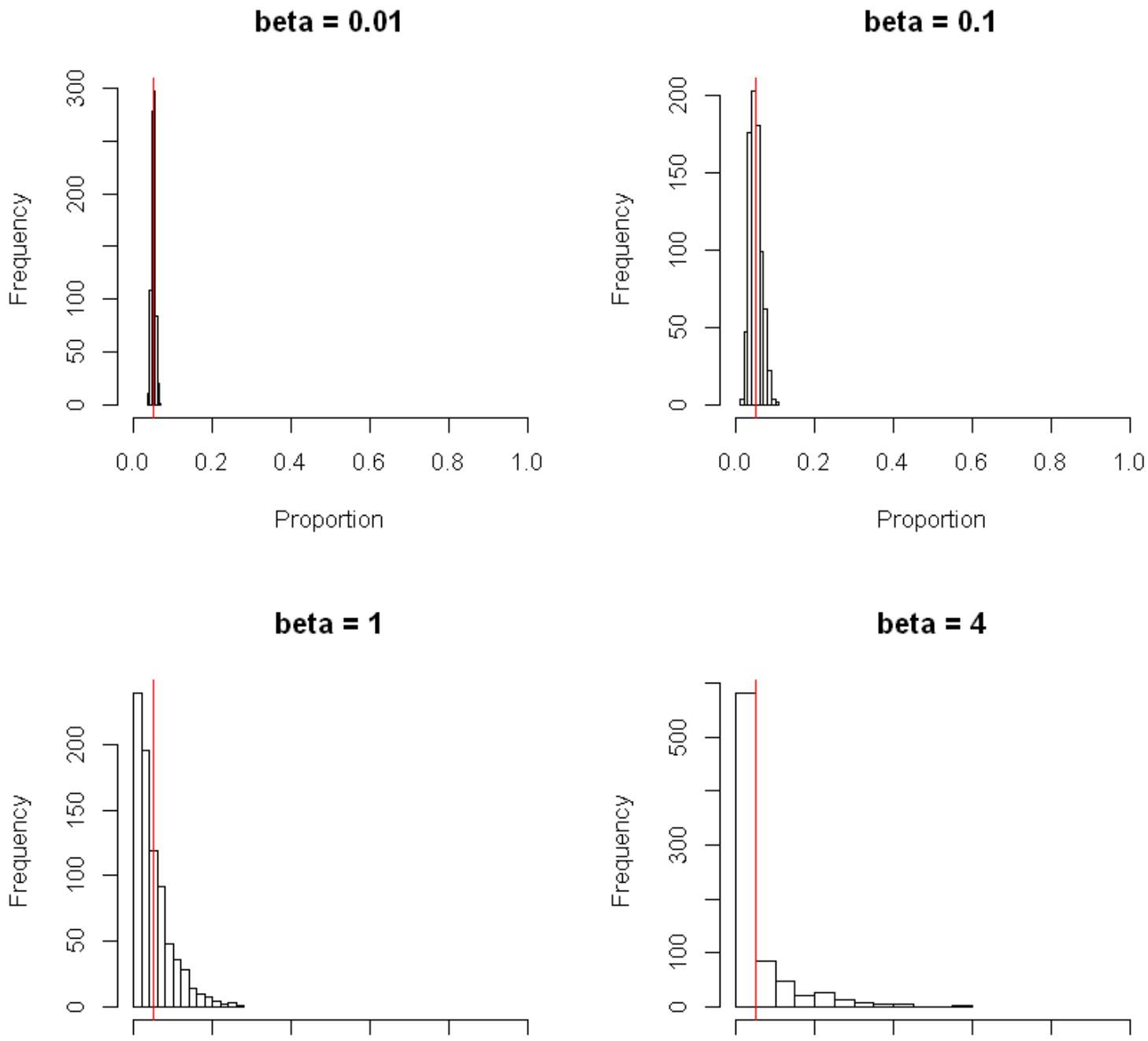
Chargement des packages R utilisés

In [10]:
```
library(MCMCpack) ## load this package for drawing data from the Dirichlet distribution
library(abc) ## load this package for running functions related to ABC
library(psych) ## load this package for generating plots with functions pairs.panels() and violinBy()
```

```
Loading required package: coda
Loading required package: MASS
##
## Markov Chain Monte Carlo Package (MCMCpack)
## Copyright (C) 2003-2019 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
##
## Support provided by the U.S. National Science Foundation
## (Grants SES-0350646 and SES-0350613)
##
Loading required package: abc.data
Loading required package: nnet
Loading required package: quantreg
Loading required package: SparseM

Attaching package: 'SparseM'

The following object is masked from 'package:base':

    backsolve

Loading required package: locfit
locfit 1.5-9.1   2013-03-22
```

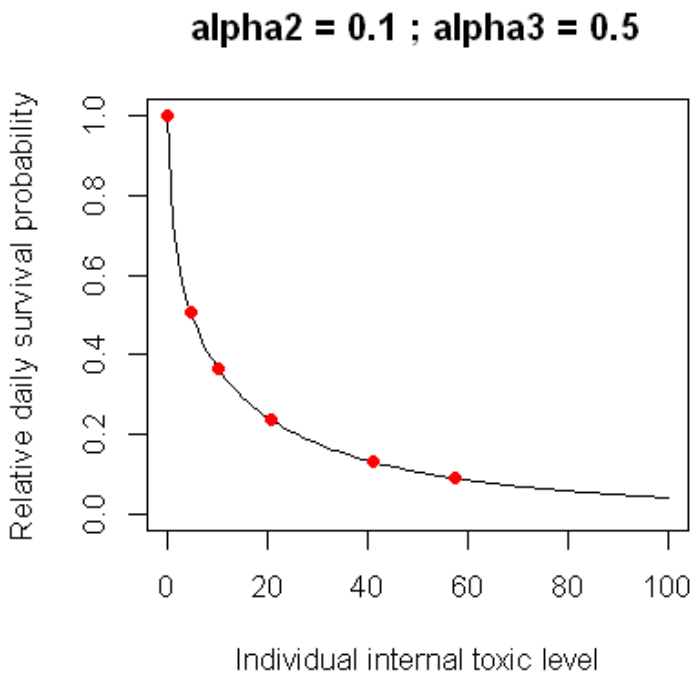
Exploration de la distribution Dirichlet utilisée pour partager le "gâteau"

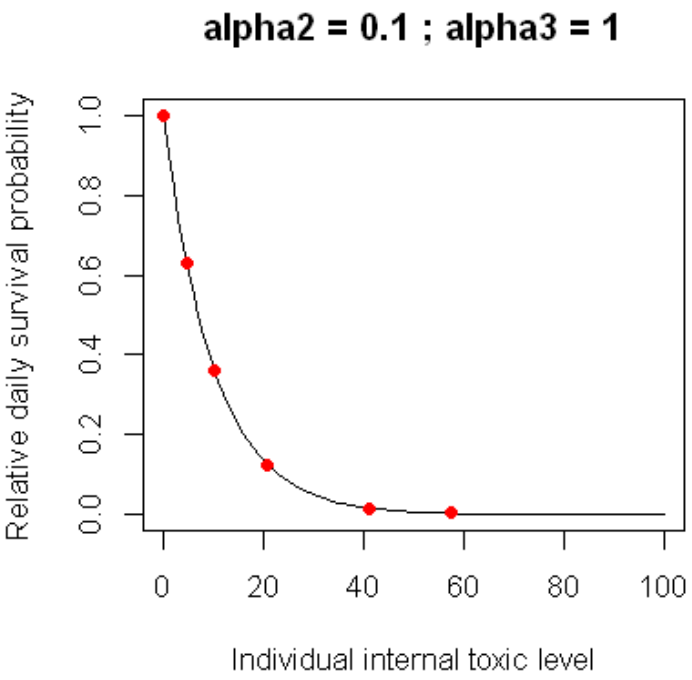
In [11]:
```
par(mfrow=c(2,2))
for(b in c(0.01,0.1,1,4)){
    hist(rdirichlet(n=40,alpha=rep(1/b,20)),xlab="Proportion",xlim=c(0,1),main=paste("beta =",b))
    abline(v=1/20,col=2)
}
```

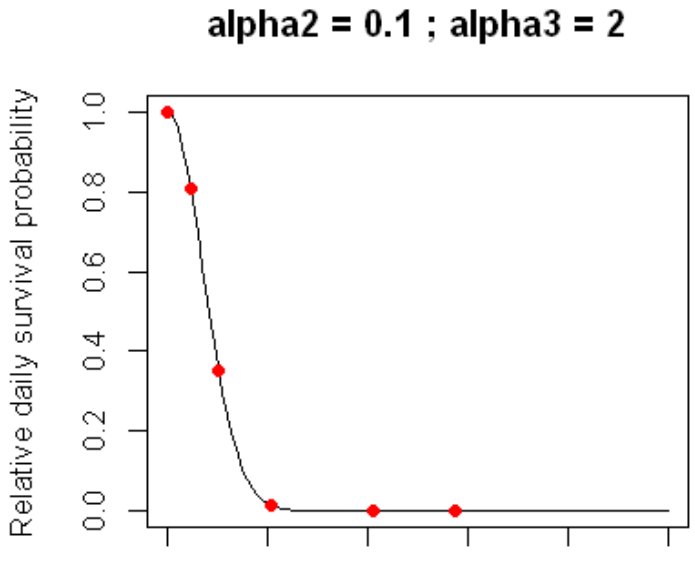

beta = 0.01
Frequency
0
50
100
200
300
0.0
0.2
0.4
0.6
0.8
1.0
Proportion
beta = 0.1
Frequency
0
50
100
150
200
0.0
0.2
0.4
0.6
0.8
1.0
Proportion
beta = 1
Frequency
0
50
100
150
200
beta = 4
Frequency
0
100
300
500

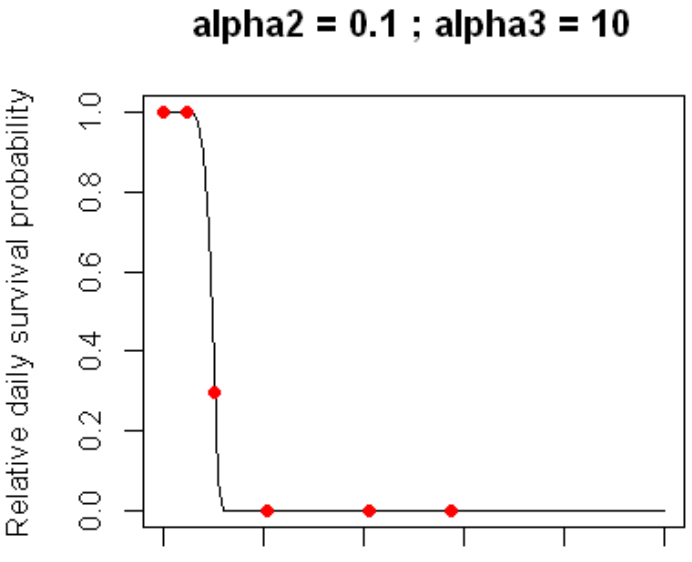
Exploration de la fonction gouvernant la probabilité de mortalité journalière relative (i.e. relative à la mortalité "naturelle")

In [12]:
```
a2=0.1## try both values 0.1 and 0.02
tox.seq=0:100
par(mfrow=c(2,2))
for(a3 in c(0.5,1,2,10)){
    plot(tox.seq,exp(-(a2*tox.seq)^a3),type="l",ylim=c(0,1),
         xlab="Individual internal toxic level",
         ylab="Relative daily survival probability",
         main=paste("alpha2 =",a2,"; alpha3 =",a3))
    points(tox/400,exp(-(a2*tox/400)^a3),pch=19,col=2)
}
```


alpha2 = 0.1 ; alpha3 = 0.5
Relative daily survival probability
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Individual internal toxic level


alpha2 = 0.1 ; alpha3 = 1
Relative daily survival probability
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Individual internal toxic level


alpha2 = 0.1 ; alpha3 = 2
Relative daily survival probability
0.0
0.2
0.4
0.6
0.8
1.0


alpha2 = 0.1 ; alpha3 = 10
Relative daily survival probability
0.0
0.2
0.4
0.6
0.8
1.0

Simulateur du modèle et exemples de sorties

Arguments du simulateur :

n: nombre total d'individus

nj: nombre de groupes

X: vecteur de doses

kout: proportion d'élimination journalière de la concetration toxique interne

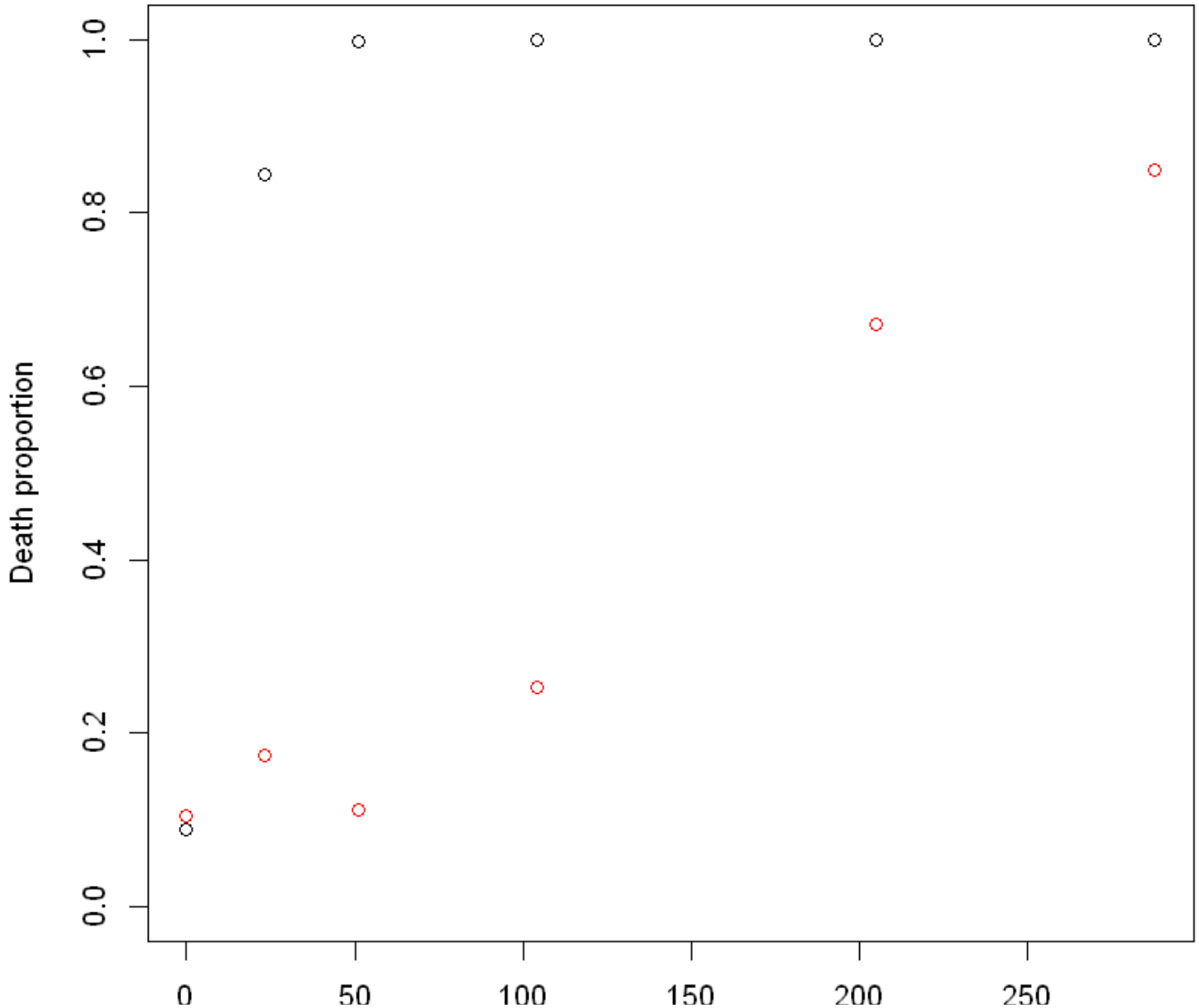
alpha1,alpha2,alpha3: paramètres de mortalité,

beta: paramètre gouvernant la variance des proportions dans le partage de la dose toxique

ngroup: nombre de groupes dans lesquels les individus sont répartis

In [13]:
```
## simulator
mortality.pop=function(n,nj,X,kout,alpha1,alpha2,alpha3,beta, ngroup=1){
    death.prop=rep(0,length(X))
    size=n/ngroup
    for(i in 1:length(X)){
        prop.feed=rdirichlet(n=ngroup,alpha=rep(1/beta,size*2))
        Q=cbind(as.numeric(prop.feed[,1:size]),as.numeric(prop.feed[,size+1:size]))
        rho=cbind(X[i]*Q[,1])%*%((1-kout)^((1:nj)-1)) +
            cbind(X[i]*Q[,2])%*%c(0,(1-kout)^((2:nj)-2))
        event=ceiling(alpha1*exp(-(alpha2*rho)^alpha3)-matrix(runif(nj*n),n,nj))
        surv=apply(event,1,min)
        death.prop[i]=mean(1-surv)
    }
    return(death.prop)
}

## run and show 1 simulation ; play with alpha2 (0.1; 0.01; 0.001)
mortality.sim=mortality.pop(n=400,nj=7,tox, kout=0.5, alpha1= 0.99, alpha2=0.01,
    alpha3=4, beta=0.25, ngroup=40)
cat("Doses toxiques:\n")
print(tox.dose)
cat("Taux de mortalité observés aux différentes doses toxiques:\n")
print(mortality.sim)
plot(tox.dose,mortality.sim,ylim=c(0,1),
     xlab="Toxic dose (# toxic pollen grains / cm2)",
     ylab="Death proportion")
points(tox.dose,mortality,col=2)  ## death proportions observed for Inachis io
```

```
Doses toxiques:
[1]   0  23  51 104 205 288
Taux de mortalité observés aux différentes doses toxiques:
[1] 0.0900 0.8450 0.9975 1.0000 1.0000 1.0000
```


Death proportion
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250

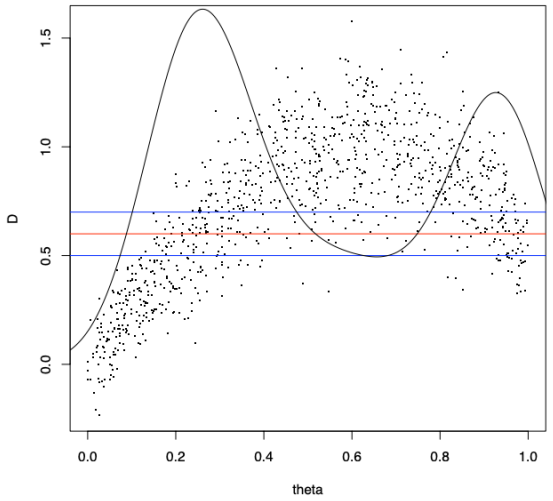
Principe de l'ABC

Notations :

- $\mathcal{D}$: jeu de données observé
- θ : paramètres du modèle
- π : loi a priori des paramètres
- $\mathcal{M}$: Modèle paramétré par θ sensé générer les données $\mathcal{D}$

Procédure :

- Simuler des jeux de paramètres θ_i, $i = 1, \ldots, I$, sous la loi a priori π
- Pour chaque jeu de paramètres θ_i, simuler un jeu de données $\mathcal{D}_i$
- Retenir les $\mathcal{D}_i$ "ressemblant" à $\mathcal{D}$ pour construire la loi a posteriori de θ



Le rôle des statistiques résumées

Dernière étape de la procédure ABC :

- Retenir les $\mathcal{D}_i$ "ressemblant" à $\mathcal{D}$ pour construire la loi a posteriori de θ

Pb quand les $\mathcal{D}_i$ et $\mathcal{D}$ sont de grande dimension et dans des espaces continus

Dans l'ABC, on se sert de statistiques résumées pour comparer les jeux de données $\mathcal{D}_i$ et $\mathcal{D}$

Notations :

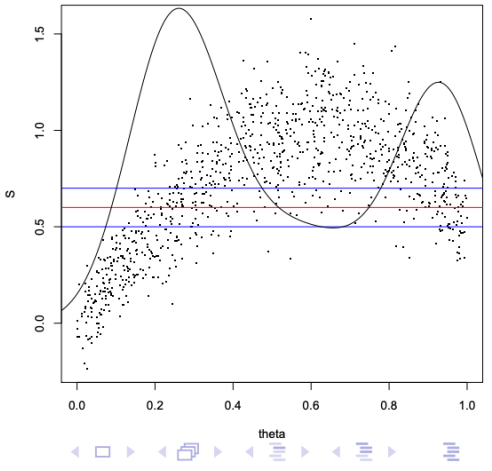
- $S = s(\mathcal{D})$: statistiques résumées observées
- $S_i = s(\mathcal{D}_i)$: statistiques résumées pour les paramètres simulés θ_i

ABC basique : acceptation/rejet avec seuil de tolérance

- S_0 : statistiques résumées observées
- $\theta_i \sim \pi(\cdot)$, $i = 1, \ldots, I$, I très grand
- $S_i = s(\mathcal{M}_{\theta_i}(\omega_i))$ où $\mathcal{M}_{\theta_i}$ est un modèle stochastique implicite ($\mathcal{D}_i = \mathcal{M}_{\theta_i}(\omega_i)$)
- Ensemble des points acceptés pour estimer la loi a posteriori $p(\theta \mid d(S, S_0) \leq \epsilon)$:

$$\{\theta_i : i = 1, \ldots, I \text{ et } d(S_i, S) \leq \epsilon\}$$

$$\hat{p}(\theta \mid d(S, S_0) \leq \epsilon) = \frac{\sum_{i=1}^{I} \delta_{B(S_0,\epsilon)}(S_i)\delta_\theta(\theta_i)}{\sum_{i=1}^{I} \delta_{B(S_0,\epsilon)}(S_i)}$$



Lois a priori des paramètres

A priori, les paramètres sont supposés indépendants

β : paramètre gouvernant la variance des proportions relatives au partage du "gâteau"

$$\beta \sim \text{Uniforme}(0, 4)$$

k_{out} : taux journalier d'élimination de la toxicité interne

$$k_{\text{out}} \sim \text{Uniforme}(0, 1)$$

α_1 : taux journalier de mortalité "naturelle"

$$\alpha_1 \sim \text{Beta}(50, 1)$$

α_2 : premier paramètre gouvernant l'effet de la concentration toxique interne sur la mortalité journalière

$$\alpha_2 \sim \text{Exponential}(1000)$$

α_3 : deuxième paramètre gouvernant l'effet de la concentration toxique interne sur la mortalité journalière

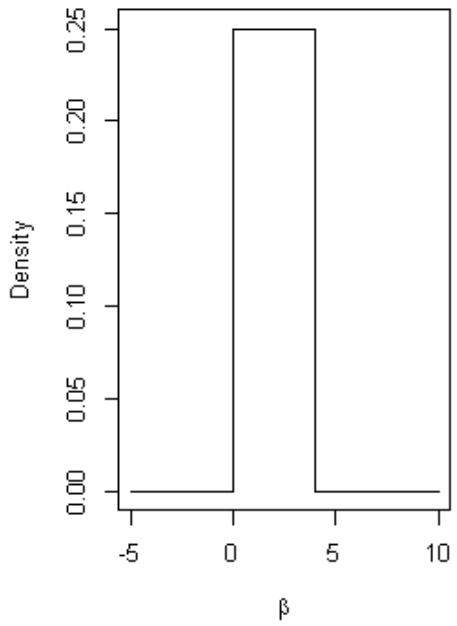
α_3 fixé :

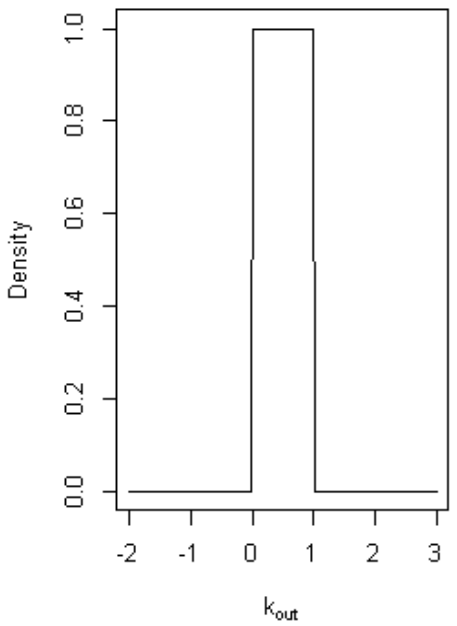
$$\alpha_3 \sim \text{Dirac}(1)$$

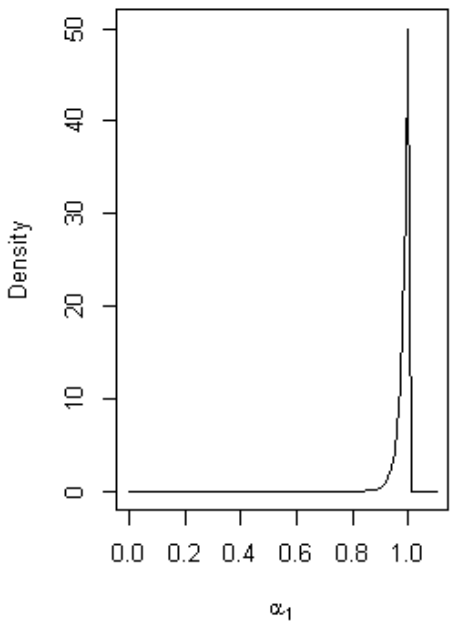
α_3 variables :

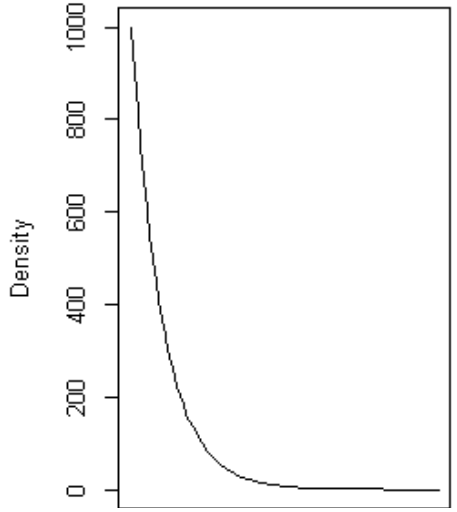
$$\alpha_3 \sim \text{Uniforme}(0, 10)$$

In [14]:
```
## plot priors
par(mfrow=c(2,3))
plot(seq(-5,10,0.01),dunif(seq(-5,10,0.01),0,4),type="l",xlab=expression(beta),ylab="Density")
plot(seq(-2,3,0.01),dunif(seq(-2,3,0.01),0,1),type="l",xlab=expression(k[out]),ylab="Density")
plot(seq(0,1.1,0.01),dbeta(seq(0,1.1,0.01),shape1=50,shape2=1),type="l",xlab=expression(alpha[1]),ylab="Density")
plot(seq(0,0.01,0.0001),dexp(seq(0,0.01,0.0001),rate=1000),type="l",xlab=expression(alpha[2]),ylab="Density")
plot(seq(-5,15,0.01),seq(-5,15,0.01)==1,type="l",xlab=expression(alpha[3]),ylab="Density",
    main="Fixed alpha3")
plot(seq(-5,15,0.01),dunif(seq(-5,15,0.01),0,10),type="l",xlab=expression(alpha[3]),ylab="Density",
    main="Variable alpha3")
```

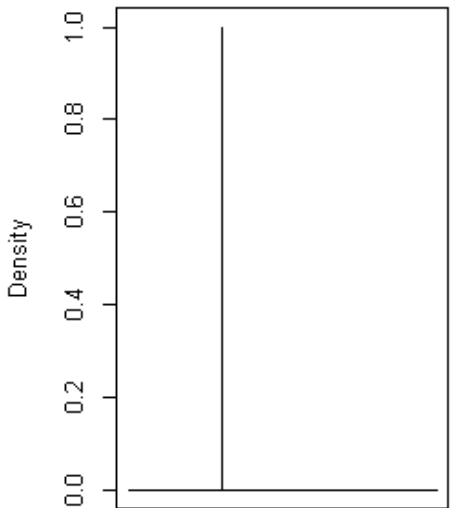




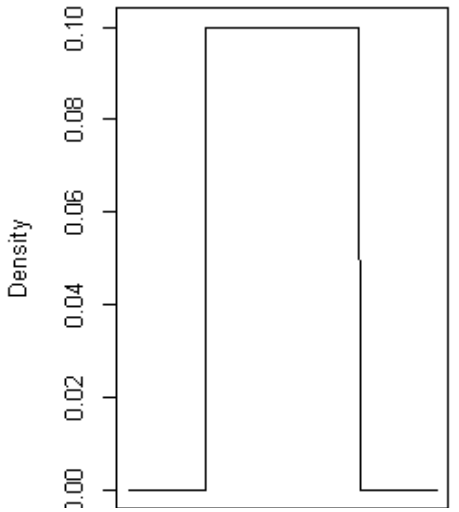




Fixed alpha3



Variable alpha3



Série de simulations avec $\alpha_3 = 1$

In [15]:
```
## run simulations for ABC (try first with alpha3=1)
time0=proc.time()
nsimul=10^4
param=cbind(runif(nsimul,0,1), ## kout
    rbeta(nsimul,shape1=50,shape2=1), ## alpha1
    rexp(nsimul,rate=1000), ## alpha2
    rep(1,nsimul), ## alpha3=1
    runif(nsimul,0,4)) ## beta
colnames(param)=c("kout","alpha1","alpha2","alpha3","beta")
stats=t(apply(param,1,function(u) mortality.pop(n=400, nj=7, tox, u[1],u[2],u[3],u[4],u[5],ngroup=40)))
colnames(stats)=names(mortality)
proc.time()-time0
```

```
   user  system elapsed 
  49.16    0.03   49.32 
```

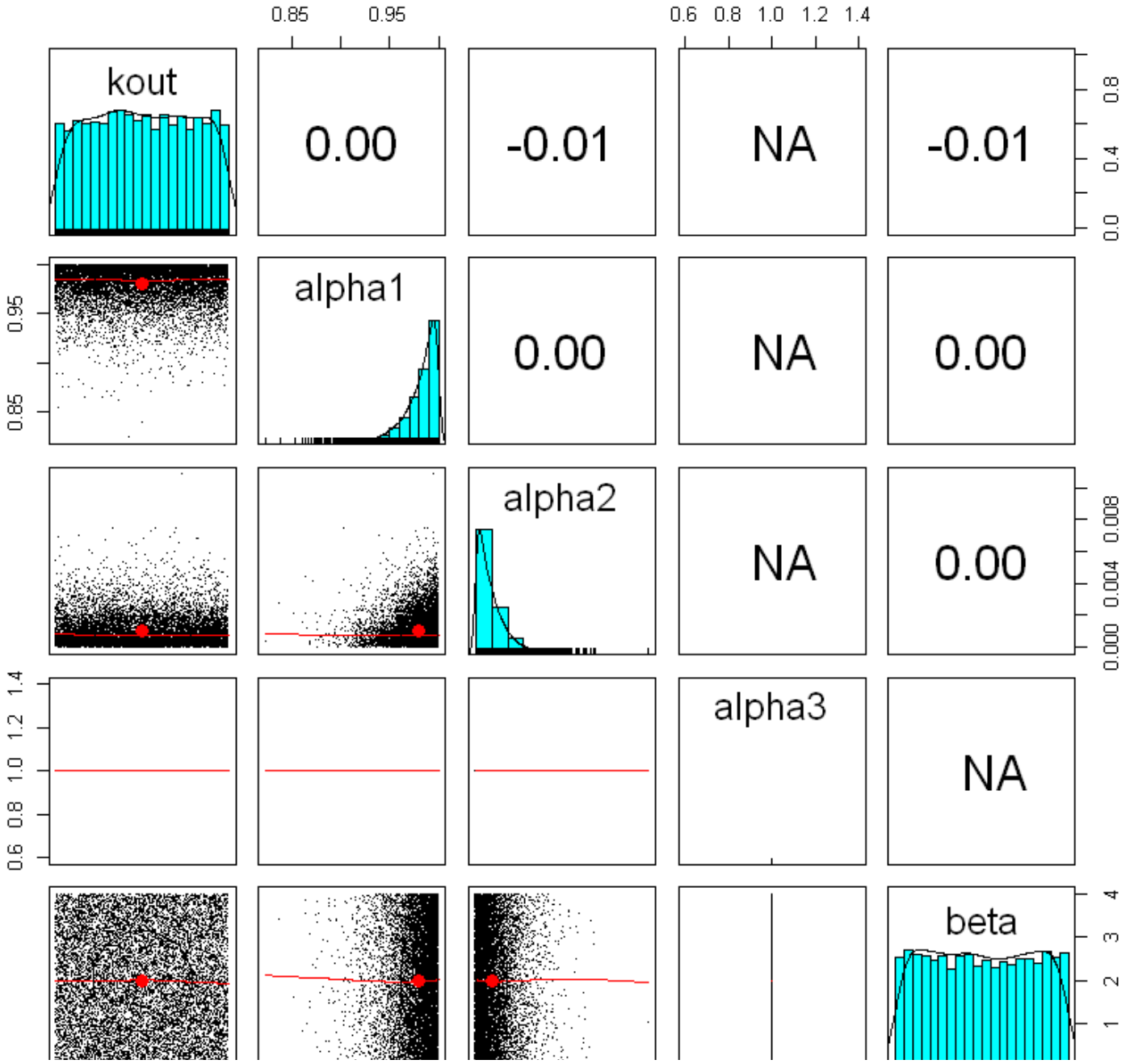
In [16]:
```
param[1:10,]
stats[1:10,]
```

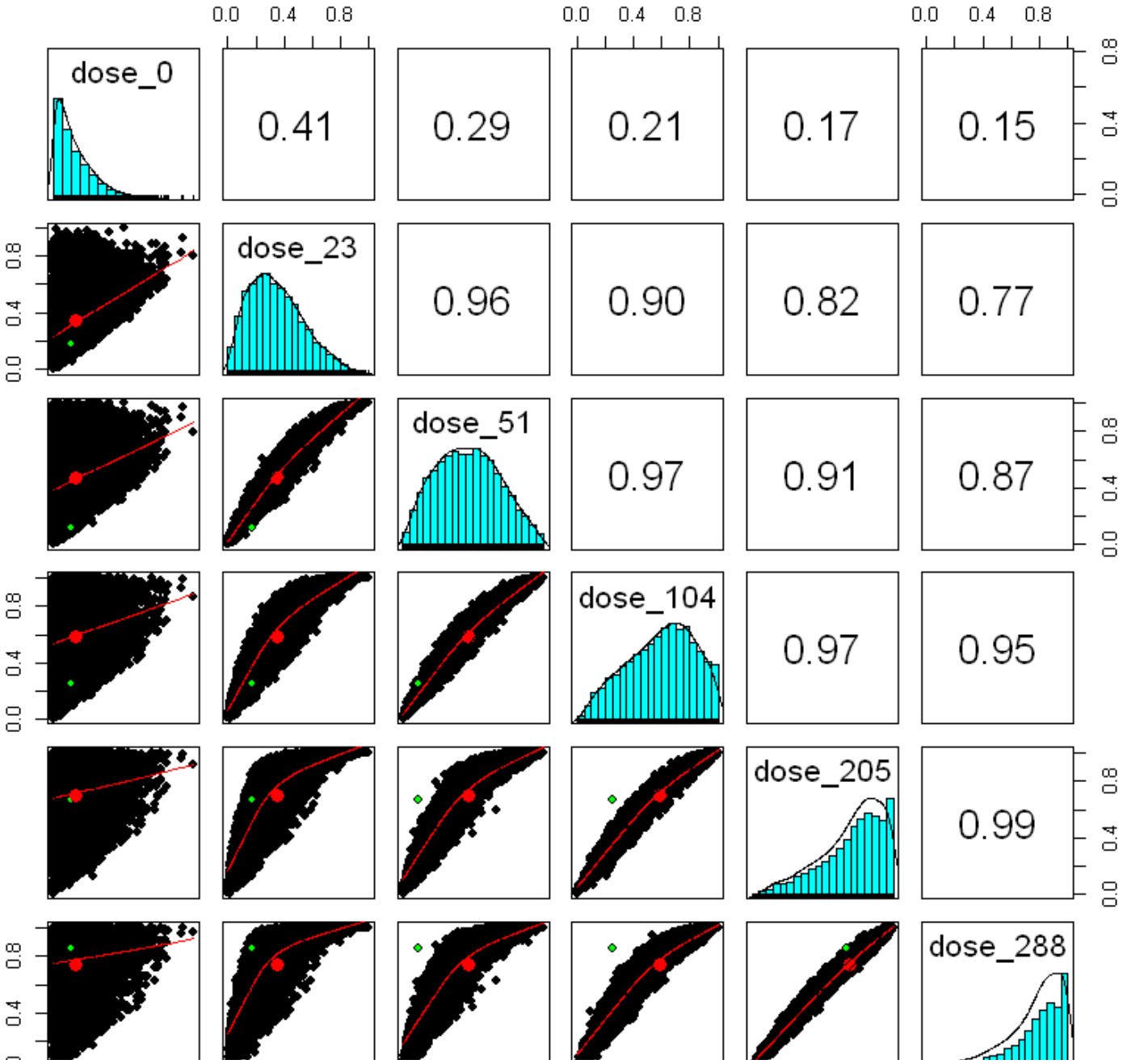
kout	alpha1	alpha2	alpha3	beta
0.08791590	0.9986861	0.0011857282	1	0.5106709
0.59016189	0.9911775	0.0002353319	1	0.2764070
0.11803789	0.9799156	0.0003164854	1	1.9153018
0.94904119	0.9659752	0.0001447648	1	0.1449695
0.82239640	0.9905135	0.0005641909	1	1.3782405
0.51929071	0.9952433	0.0021663594	1	0.4354925
0.23133461	0.9991393	0.0007169982	1	1.9347949
0.94818739	0.9976719	0.0005025948	1	3.0792278
0.02736748	0.9502675	0.0006566872	1	0.7104579
0.15323340	0.9550912	0.0009611865	1	1.6633690

dose_0	dose_23	dose_51	dose_104	dose_205	dose_288
0.0125	0.6400	0.8600	0.9650	0.9950	0.9900
0.0600	0.1825	0.2125	0.3225	0.5025	0.6125
0.1250	0.2950	0.4500	0.5975	0.7850	0.8375
0.1925	0.2375	0.2875	0.3175	0.4275	0.4225
0.0825	0.1975	0.2900	0.4150	0.6175	0.6975
0.0125	0.5400	0.7725	0.9225	0.9950	1.0000
0.0075	0.3250	0.5000	0.7175	0.8375	0.8450
0.0175	0.0975	0.1650	0.2950	0.4400	0.5325
0.2800	0.6225	0.8150	0.9275	0.9800	0.9825
0.2525	0.6250	0.7425	0.8575	0.9275	0.9625

In [17]:
```
## show simulations
nsimul0=min(10^4,nsimul)
pairs.panels(param[1:nsimul0,],pch=".")
pairs.panels(rbind(stats[1:nsimul0,],mortality),pch=c(rep(19,nsimul0),21),bg="green")
```

```
Warning message in cor(x, y, use = "pairwise", method = method):
"l'écart type est nulle"Warning message in cor(x, y, use = "pairwise", method = method):
"l'écart type est nulle"Warning message in cor(x, y, use = "pairwise", method = method):
"l'écart type est nulle"Warning message in cor(x, y, use = "pairwise", method = method):
"l'écart type est nulle"Warning message in cor(x, y, use = "pairwise", method = method):
"l'écart type est nulle"Warning message in min(diff(breaks)):
"aucun argument trouvé pour min ; Inf est renvoyé"Warning message in cor(x, y, use = "pairwise", method = method):
"l'écart type est nulle"Warning message in cor(x, y, use = "pairwise", method = method):
"l'écart type est nulle"
```


kout
alpha1
alpha2
alpha3
beta
0.00
-0.01
NA
0.85
0.95
0.6
0.8
1.0
1.2
1.4
0.0
0.4
0.8
0.000
0.004
0.008
1
2
3
4


dose_0
dose_23
dose_51
dose_104
dose_205
dose_288
0.41
0.29
0.21
0.17
0.15
0.96
0.90
0.82
0.77
0.97
0.91
0.87
0.97
0.95
0.99
0.0
0.4
0.8

Le vecteur des statistiques résumées (mortalités en fct des doses) est clairement "en dehors" de la distribution a priori des statistiques avec $\alpha_3 = 1$

$\rightarrow$ Inutile d'appliquer l'ABC quand α_3 est fixé à 1

Série de simulations avec α_3 distribué a priori dans une loi uniforme

In [ ]:
```
## run simulations for ABC (try with alpha3 a priori uniformly distributed in [0,10])
time0=proc.time()
nsimul=10^5
param=cbind(runif(nsimul,0,1), ## kout
    rbeta(nsimul,shape1=50,shape2=1), ## alpha1
    rexp(nsimul,rate=1000), ## alpha2
    runif(nsimul,0,10), ## alpha3
    runif(nsimul,0,4)) ## beta
colnames(param)=c("kout","alpha1","alpha2","alpha3","beta")
stats=t(apply(param,1,function(u) mortality.pop(n=400, nj=7, tox, u[1],u[2],u[3],u[4],u[5],ngroup=40)))
colnames(stats)=names(mortality)
proc.time()-time0
```

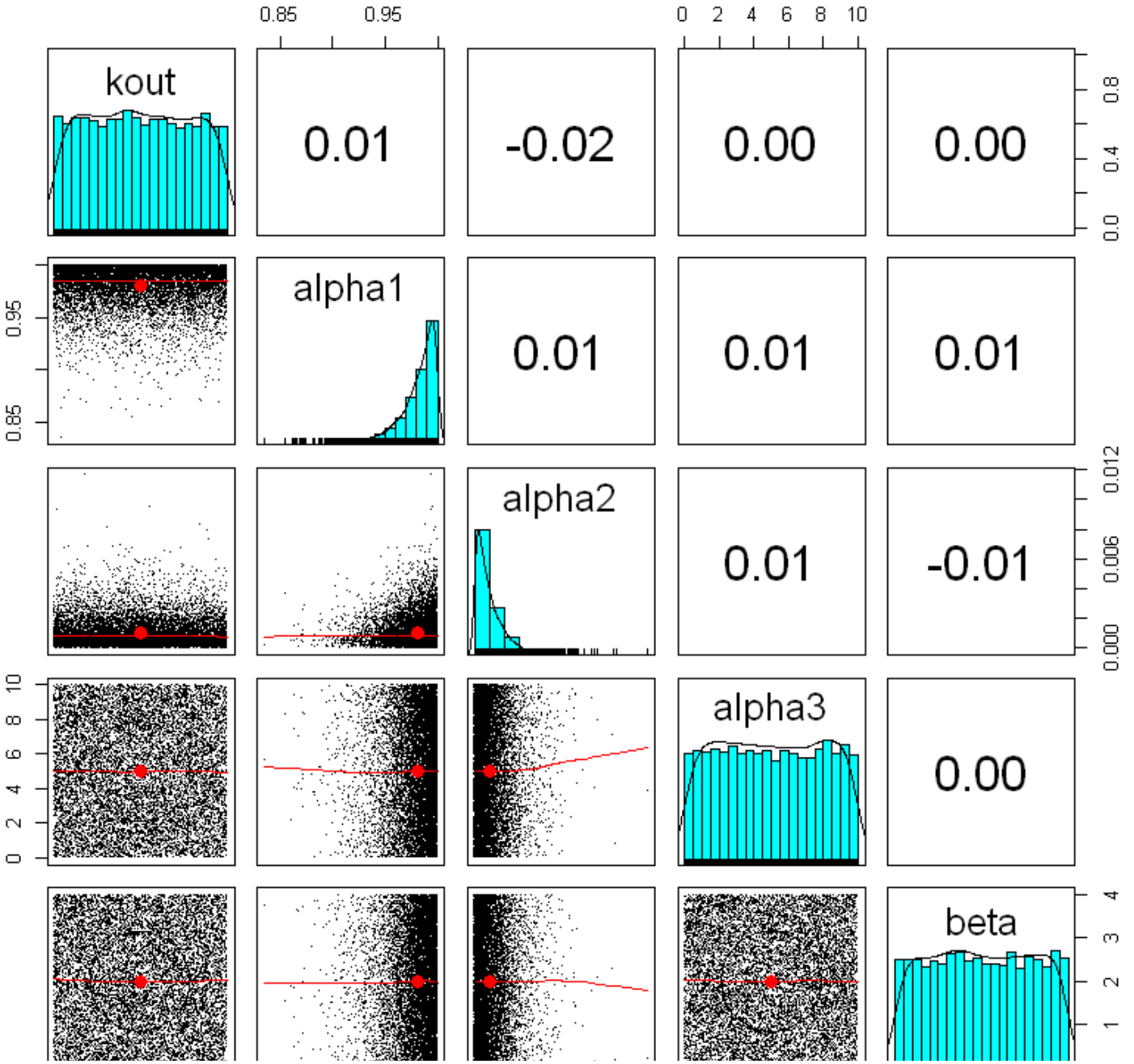
In [18]:
```
## save param and stats as rds files
# saveRDS("save/param.rds")
# saveRDS("save/stats.rds")

## load param and stats from existing files
 param=readRDS("save/param.rds")
 stats=readRDS("save/stats.rds")
 nsimul=nrow(stats)
```

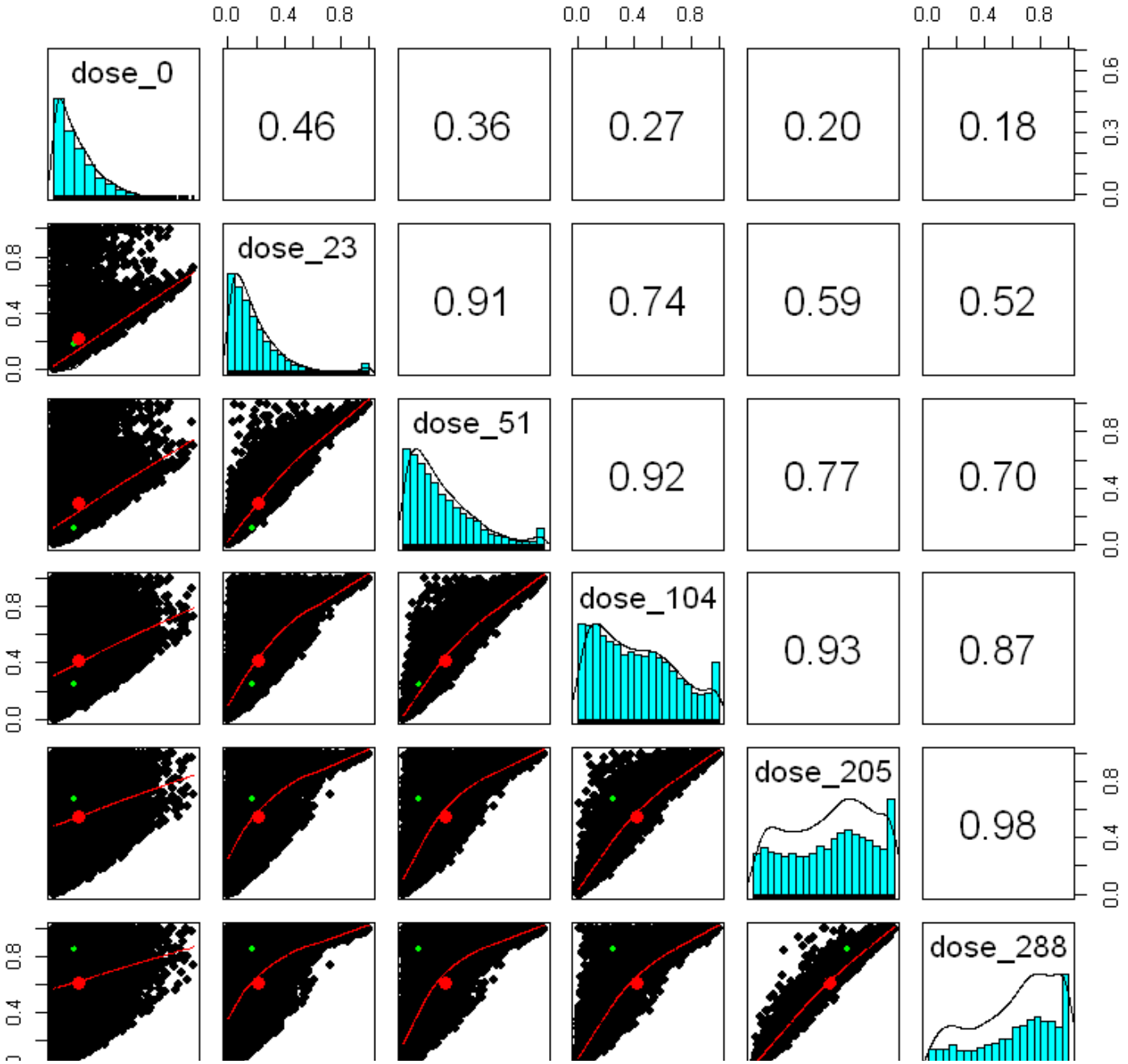
In [19]:
```
param[1:10,]
stats[1:10,]
```

kout	alpha1	alpha2	alpha3	beta
0.55969897	0.9340329	0.0018695354	1.510139	0.66414125
0.63780479	0.9893155	0.0002944948	7.676902	1.06136778
0.09420835	0.9873935	0.0003817791	7.031248	3.97005954
0.51535810	0.9962923	0.0013731660	6.217975	0.31522405
0.30779584	0.9836839	0.0009593332	1.795123	2.73435253
0.36148781	0.9998426	0.0005725886	3.629833	1.00452152
0.60652797	0.9929833	0.0001150793	6.302120	0.05979772
0.66574827	0.9856424	0.0011479206	3.157717	2.13747011
0.04485620	0.9730896	0.0049933861	2.657676	3.33809489
0.85790735	0.9987100	0.0015957442	2.229373	0.68633866

dose_0	dose_23	dose_51	dose_104	dose_205	dose_288
0.3675	0.5475	0.7100	0.8775	0.9700	0.9925
0.0550	0.0675	0.0775	0.0650	0.1350	0.1850
0.1025	0.1200	0.0725	0.2000	0.3400	0.4000
0.0350	0.0300	0.0575	0.4775	0.9000	0.9725
0.0950	0.2075	0.3475	0.5425	0.6850	0.7950
0.0000	0.0050	0.0125	0.1425	0.4350	0.6100
0.0450	0.0450	0.0650	0.0550	0.0375	0.0400
0.1025	0.1250	0.2475	0.4475	0.6050	0.7150
0.1725	0.6400	0.7475	0.8375	0.9100	0.9125
0.0150	0.0800	0.2375	0.6000	0.8700	0.9225

In [20]:
```
## show simulations
nsimul0=min(10^4,nsimul)
pairs.panels(param[1:nsimul0,],pch=".")
pairs.panels(rbind(stats[1:nsimul0,],mortality),pch=c(rep(19,nsimul0),21),bg="green")
```


kout
alpha1
alpha2
alpha3
beta
0.01
-0.02
0.00
0.00
0.01
0.01
0.01
0.01
-0.01
0.00
0.85
0.95
0
2
4
6
8
10
0.0
0.4
0.8
0.000
0.006
0.012
1
2
3
4


dose_0
dose_23
dose_51
dose_104
dose_205
dose_288
0.46
0.36
0.27
0.20
0.18
0.91
0.74
0.59
0.52
0.92
0.77
0.70
0.93
0.87
0.98
0.0
0.4
0.8
0.3
0.6

La fonction *abc()* du package abc de Csilléry et al.

abc: Tools for Approximate Bayesian Computation (ABC)

Lien sur le CRAN

https://cran.r-project.org/web/packages/abc/index.html (https://cran.r-project.org/web/packages/abc/index.html)

Vignette

https://cran.r-project.org/web/packages/abc/vignettes/abcvignette.pdf (https://cran.r-project.org/web/packages/abc/vignettes/abcvignette.pdf)

Description

This function performs multivariate parameter estimation based on summary statistics using an ABC algorithm. The algorithms implemented are rejection sampling, and local linear or non-linear (neural network) regression. A conditional heteroscedastic model is available for the latter two algorithms.

Usage

abc(target, param, sumstat, tol, method, hcorr = TRUE, transf = "none", logit.bounds, subset = NULL, kernel = "epanechnikov", numnet = 10, sizenet = 5, lambda = c(0.0001,0.001,0.01), trace = FALSE, maxit = 500, ...)

Main arguments

target: a vector of the observed summary statistics.

param: a vector, matrix or data frame of the simulated parameter values, i.e. the dependent variable(s) when method is "loclinear", "neuralnet" or "ridge".

sumstat: a vector, matrix or data frame of the simulated summary statistics, i.e. the independent variables when method is "loclinear", "neuralnet" or "ridge".

tol: tolerance, the required proportion of points accepted nearest the target values.

method: a character string indicating the type of ABC algorithm to be applied. Possible values are "rejection", "loclinear", "neuralnet" and "ridge". See also Details.

Application de l'ABC acceptation-rejet avec une tolérance arbitraire

In [21]:
```
## apply ABC (only print output)
rej0=abc(target=mortality, param=param, sumstat=stats, tol=0.001, method="rejection")
rej0
summary(rej0)
## param[rej0$region,] ## table with all retained parameters
```

```
Call:
abc(target = mortality, param = param, sumstat = stats, tol = 0.001,
    method = "rejection")
Method:
Rejection

Parameters:
kout, alpha1, alpha2, alpha3, beta

Statistics:
dose_0, dose_23, dose_51, dose_104, dose_205, dose_288

Total number of simulations 100000

Number of accepted simulations:  100

Call:
abc(target = mortality, param = param, sumstat = stats, tol = 0.001,
    method = "rejection")
Data:
 abc.out$unadj.values (100 posterior samples)

               kout alpha1 alpha2 alpha3   beta
Min.:        0.0272 0.9737 0.0005 1.9972 0.0626
2.5% Perc.:  0.0385 0.9745 0.0006 2.1893 0.1232
Median:      0.5391 0.9820 0.0008 6.7391 0.5083
Mean:        0.5278 0.9811 0.0009 6.3270 0.5252
Mode:        0.6495 0.9827 0.0008 7.6133 0.4960
97.5% Perc.: 0.9925 0.9867 0.0012 9.7252 0.9109
Max.:        0.9987 0.9879 0.0012 9.9615 1.1089
```

Sélection de la tolérance par validation croisée

In [22]:
```
## but what tolerance value?
time0=proc.time()
tol.seq=c(0.0001,0.0005,0.001,0.005,0.01)
rej.cv=cv4abc(param=param, sumstat=stats, abc.out=rej0, nval=200, tols=tol.seq)
proc.time()-time0
summary(rej.cv)
```

```
   user  system elapsed 
 143.11    5.58  148.81 

Prediction error based on a cross-validation sample of 200


            kout     alpha1     alpha2     alpha3       beta
1e-04 1.23814742 0.01555214 0.25248499 0.75504011 0.53913841
5e-04 1.04267018 0.01424883 0.17736708 0.59920630 0.48164205
0.001 1.03033013 0.01399583 0.17565643 0.59072613 0.48406589
0.005 1.01019188 0.01572500 0.21028422 0.62341796 0.54588579
0.01  1.02026505 0.02028228 0.22534595 0.65616566 0.58197754
```

In [23]:
```
## save rej.cv as rds files
# saveRDS(tol.seq,"save/tol.seq.rds")
# saveRDS(rej.cv,"save/rej.cv.rds")

## load param and stats from existing files
 tol.seq=readRDS("save/tol.seq.rds")
 tol.seq
 rej.cv0=readRDS("save/rej.cv.rds")
 summary(rej.cv0)
```

1e-04 5e-04 0.001 0.005 0.01

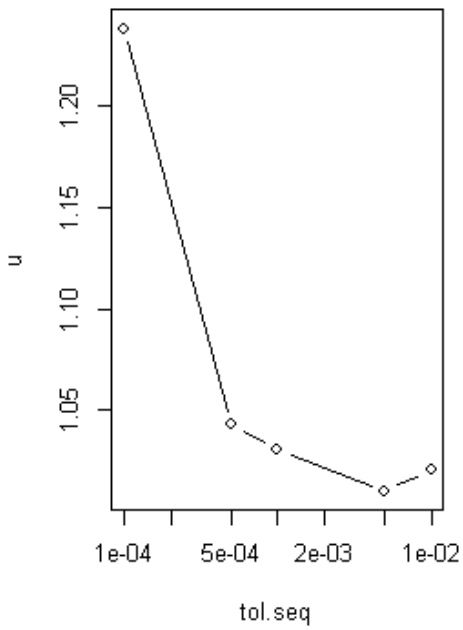
Prediction error based on a cross-validation sample of 200

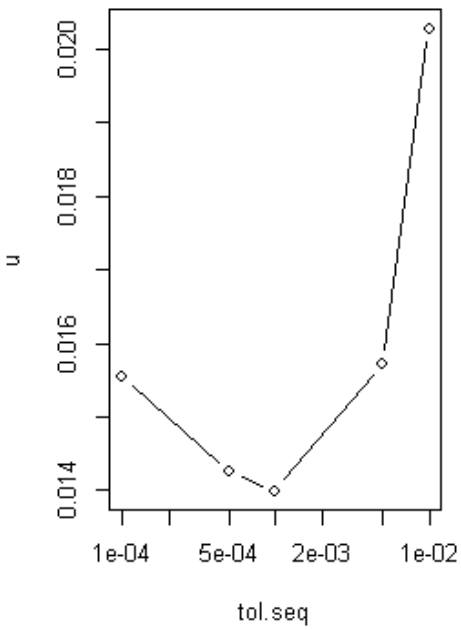
	kout	alpha1	alpha2	alpha3	beta
1e-04	1.10143607	0.01452669	0.25297373	0.74705973	0.49459478
5e-04	0.99326266	0.01383862	0.29495168	0.63282850	0.48427586
0.001	0.96564110	0.01340302	0.31463558	0.61102225	0.47952613
0.005	0.96398685	0.02232215	0.32293695	0.61243902	0.54430178
0.01	0.97565385	0.03133070	0.34031939	0.62631977	0.58209551

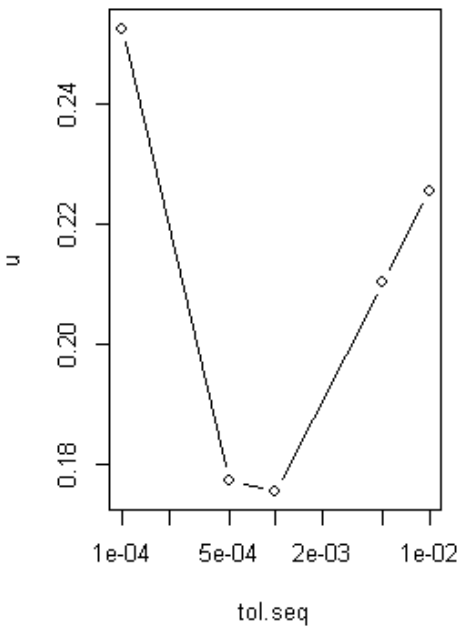
In [24]:
```
## display mean error assessed by cross-validation over each parameter
summ.rej.cv=summary(rej.cv)
par(mfrow=c(2,3))
apply(summ.rej.cv,2,function(u) plot(tol.seq,u,type="b",log="x"))
mean.error=rowMeans(summ.rej.cv)
plot(tol.seq,mean.error,type="b",log="x")
tol.opt=tol.seq[mean.error==min(mean.error)]

cat("Tolérance sélectionnée par validation croisée:\n")
as.numeric(tol.opt)

plot(rej.cv)
```

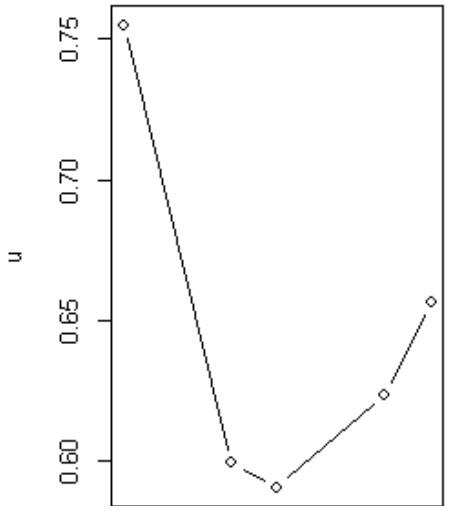
```
Prediction error based on a cross-validation sample of 200

NULL

Tolérance sélectionnée par validation croisée:
```

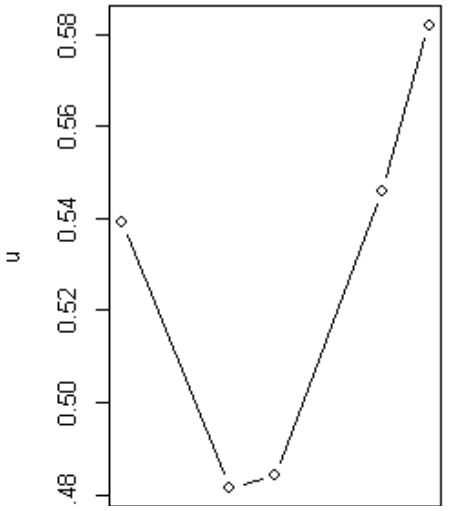
0.001

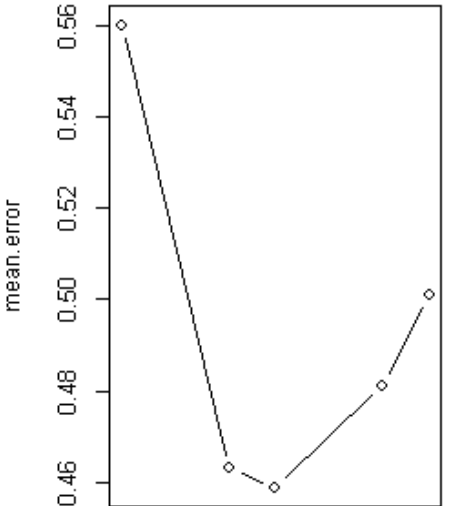

u
1.05
1.10
1.15
1.20
1e-04
5e-04
2e-03
1e-02
tol.seq

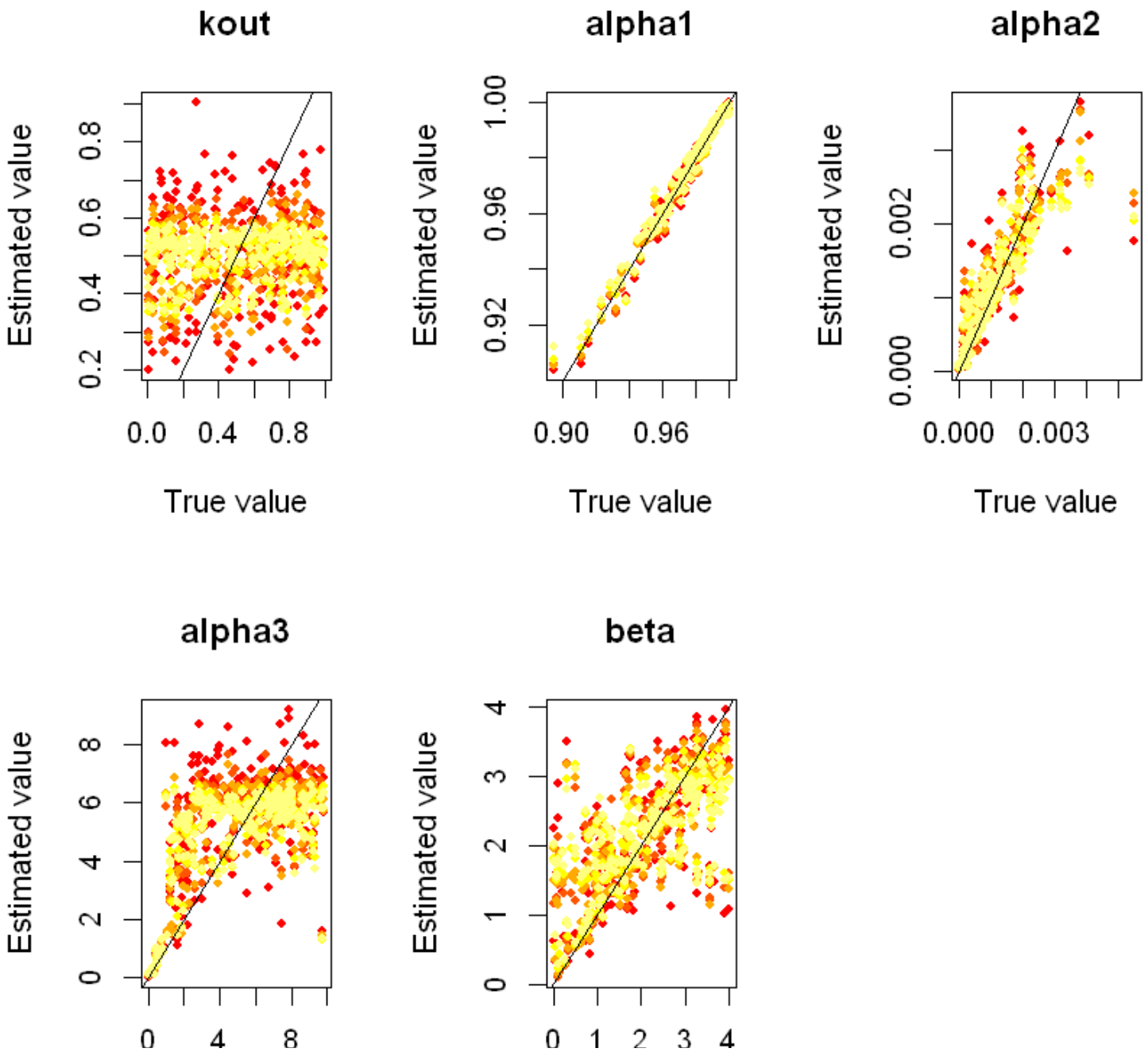

u
0.014
0.016
0.018
0.020
1e-04
5e-04
2e-03
1e-02
tol.seq


u
0.18
0.20
0.22
0.24
1e-04
5e-04
2e-03
1e-02
tol.seq


u
0.60
0.65
0.70
0.75


u
.48
0.50
0.52
0.54
0.56
0.58

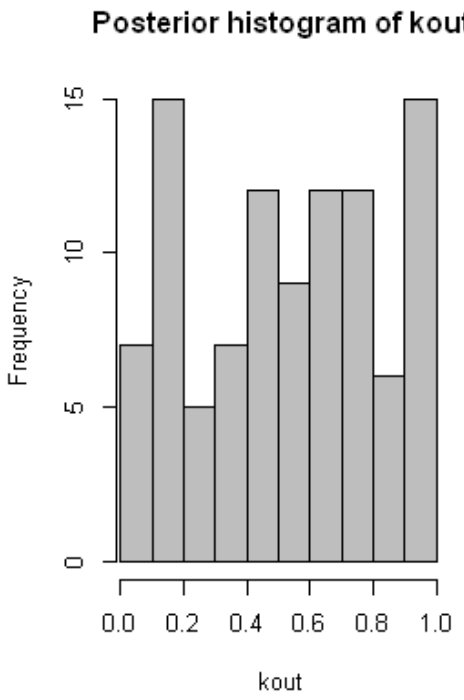

mean.error
0.46
0.48
0.50
0.52
0.54
0.56

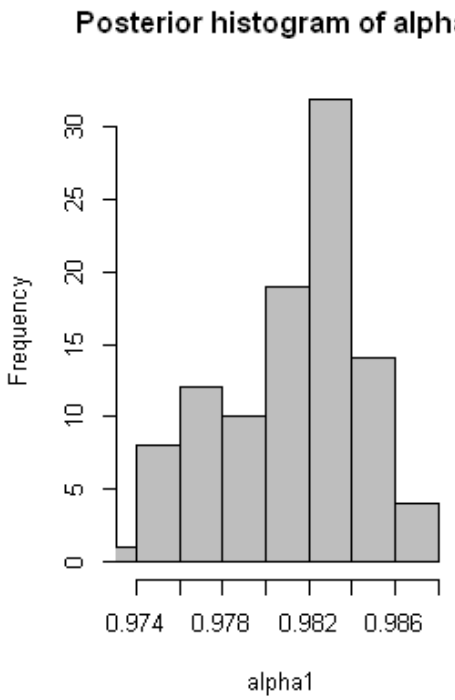

kout
Estimated value
0.2
0.4
0.6
0.8
0.0
0.4
0.8
True value
alpha1
Estimated value
0.92
0.96
1.00
0.90
0.96
True value
alpha2
Estimated value
0.000
0.002
0.000
0.003
True value
alpha3
Estimated value
0
2
4
6
8
0
4
8
beta
Estimated value
0
1
2
3
4
0
1
2
3
4

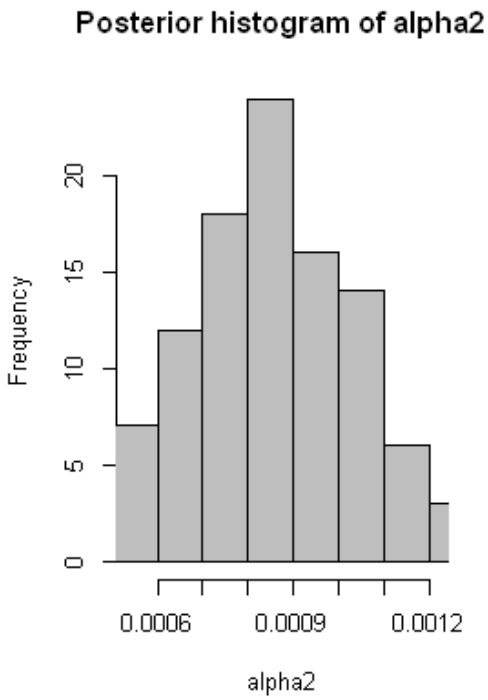
Application de l'ABC acceptation-rejet avec le seuil de tolérance sélectionné par validation croisée

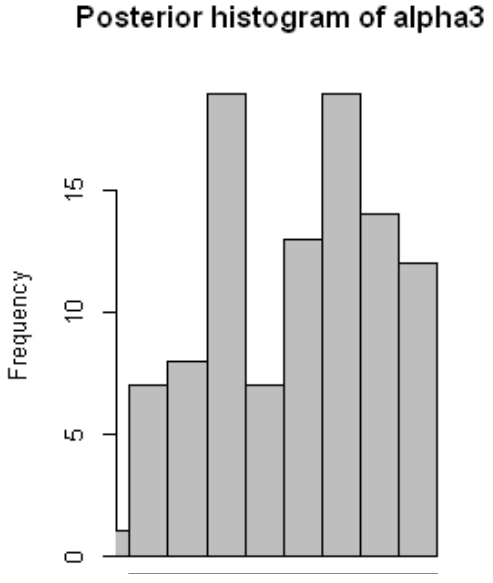
In [25]:
```
## apply ABC
rej=abc(target=mortality, param=param, sumstat=stats, tol=tol.opt, method="rejection")
rej
summary(rej)
par(mfrow=c(2,3))
hist(rej)
```

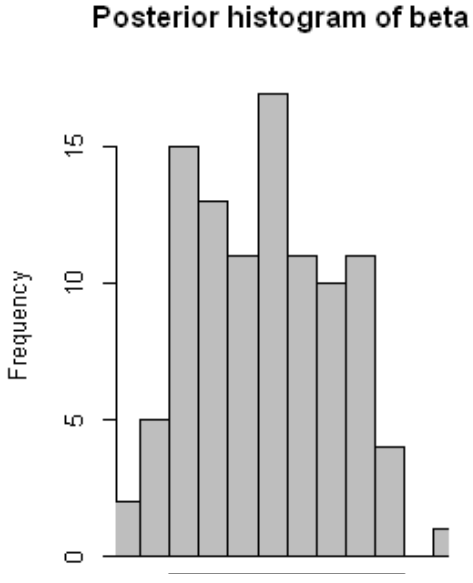
```
Call:
abc(target = mortality, param = param, sumstat = stats, tol = tol.opt,
    method = "rejection")
Method:
Rejection

Parameters:
kout, alpha1, alpha2, alpha3, beta

Statistics:
dose_0, dose_23, dose_51, dose_104, dose_205, dose_288

Total number of simulations 100000

Number of accepted simulations:  100

Call:
abc(target = mortality, param = param, sumstat = stats, tol = tol.opt,
    method = "rejection")
Data:
 abc.out$unadj.values (100 posterior samples)

               kout alpha1 alpha2 alpha3   beta
Min.:        0.0272 0.9737 0.0005 1.9972 0.0626
2.5% Perc.:  0.0385 0.9745 0.0006 2.1893 0.1232
Median:      0.5391 0.9820 0.0008 6.7391 0.5083
Mean:        0.5278 0.9811 0.0009 6.3270 0.5252
Mode:        0.6495 0.9827 0.0008 7.6133 0.4960
97.5% Perc.: 0.9925 0.9867 0.0012 9.7252 0.9109
Max.:        0.9987 0.9879 0.0012 9.9615 1.1089
```


Posterior histogram of kout
Frequency
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
kout


Posterior histogram of alpha1
Frequency
0
5
10
15
20
25
30
0.974
0.978
0.982
0.986
alpha1

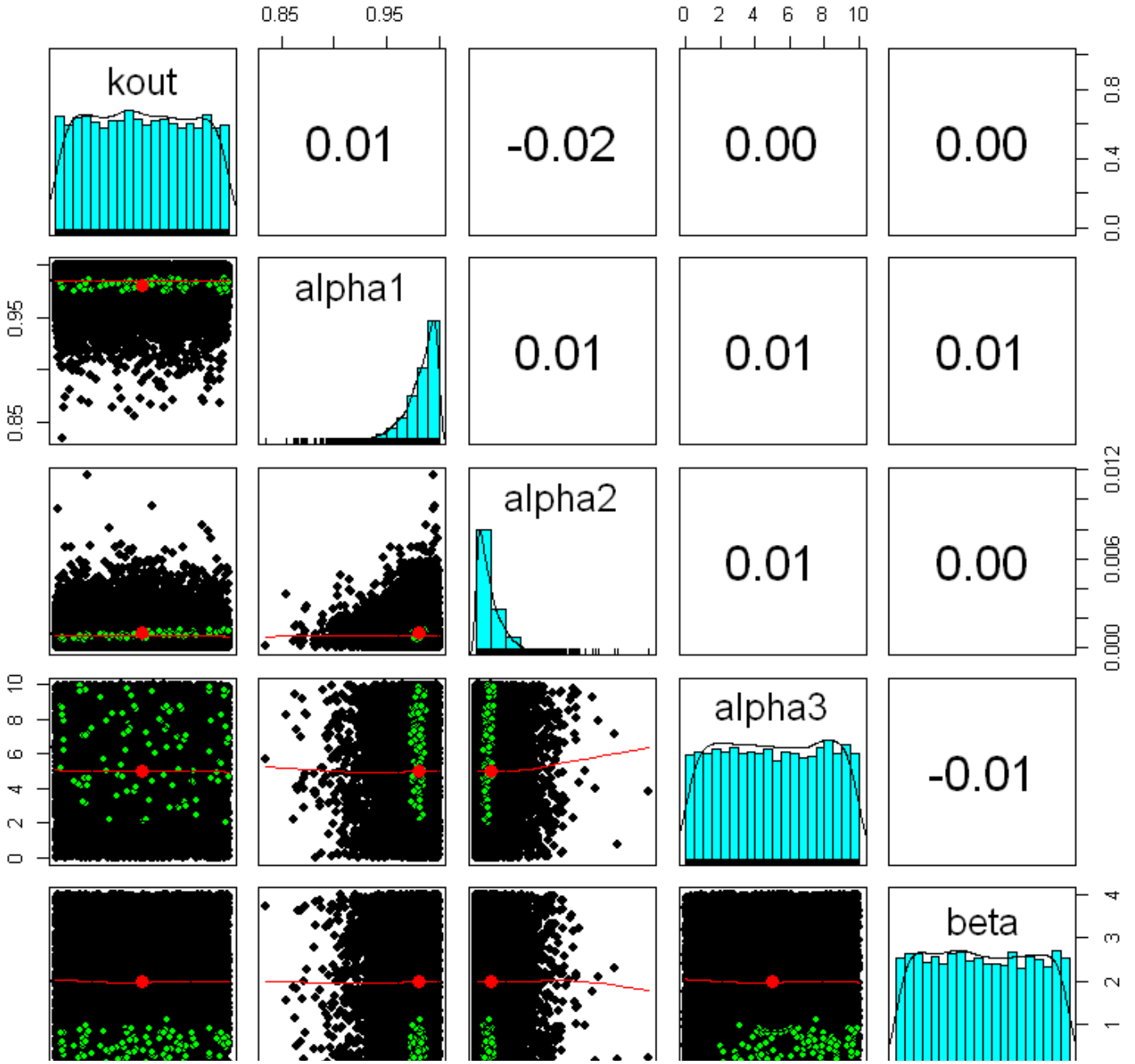

Posterior histogram of alpha2
Frequency
0
5
10
15
20
0.0006
0.0009
0.0012
alpha2

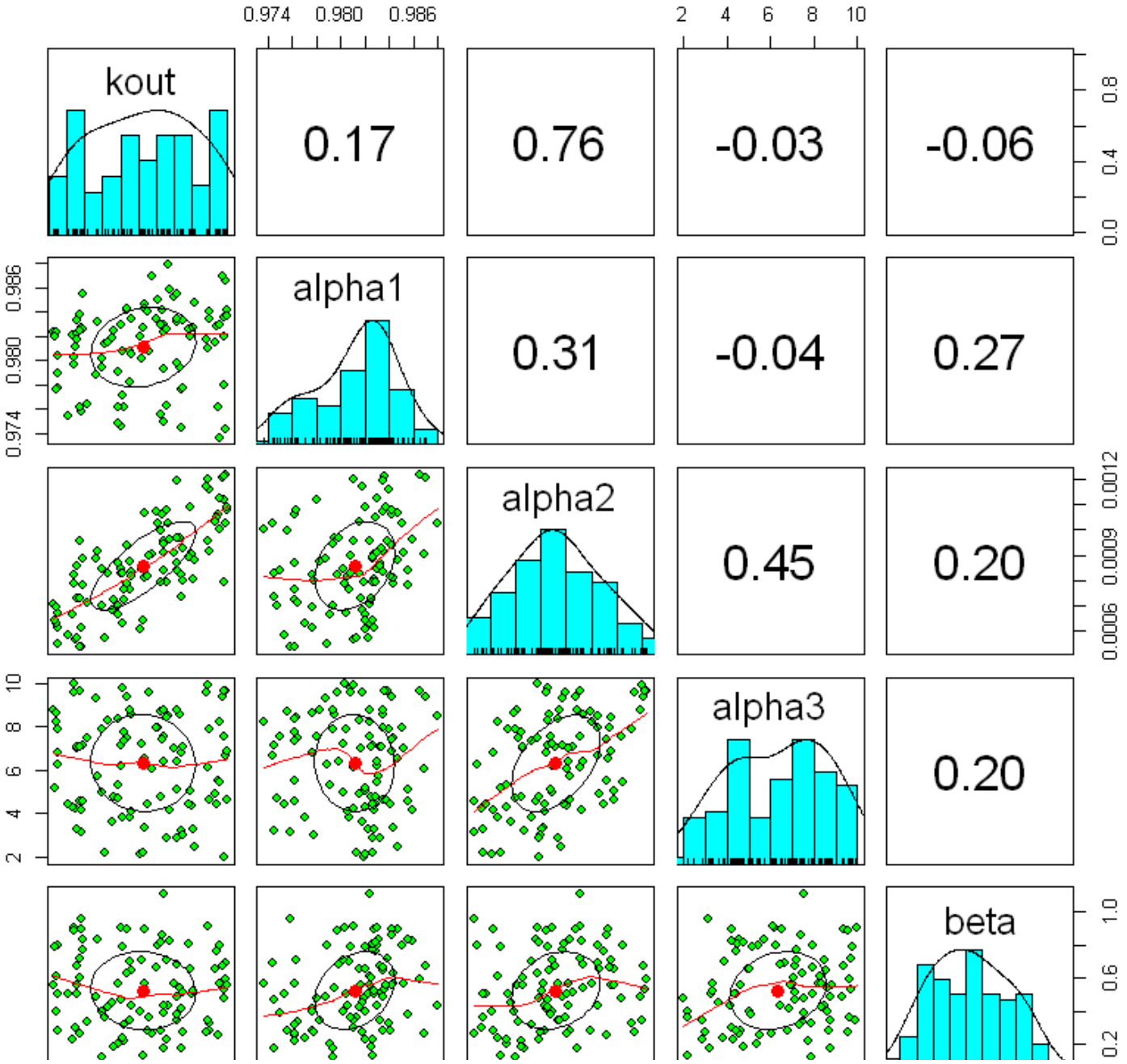

Posterior histogram of alpha3
Frequency
0
5
10
15


Posterior histogram of beta
Frequency
0
5
10
15

Echantillons a priori et a posteriori des paramètres

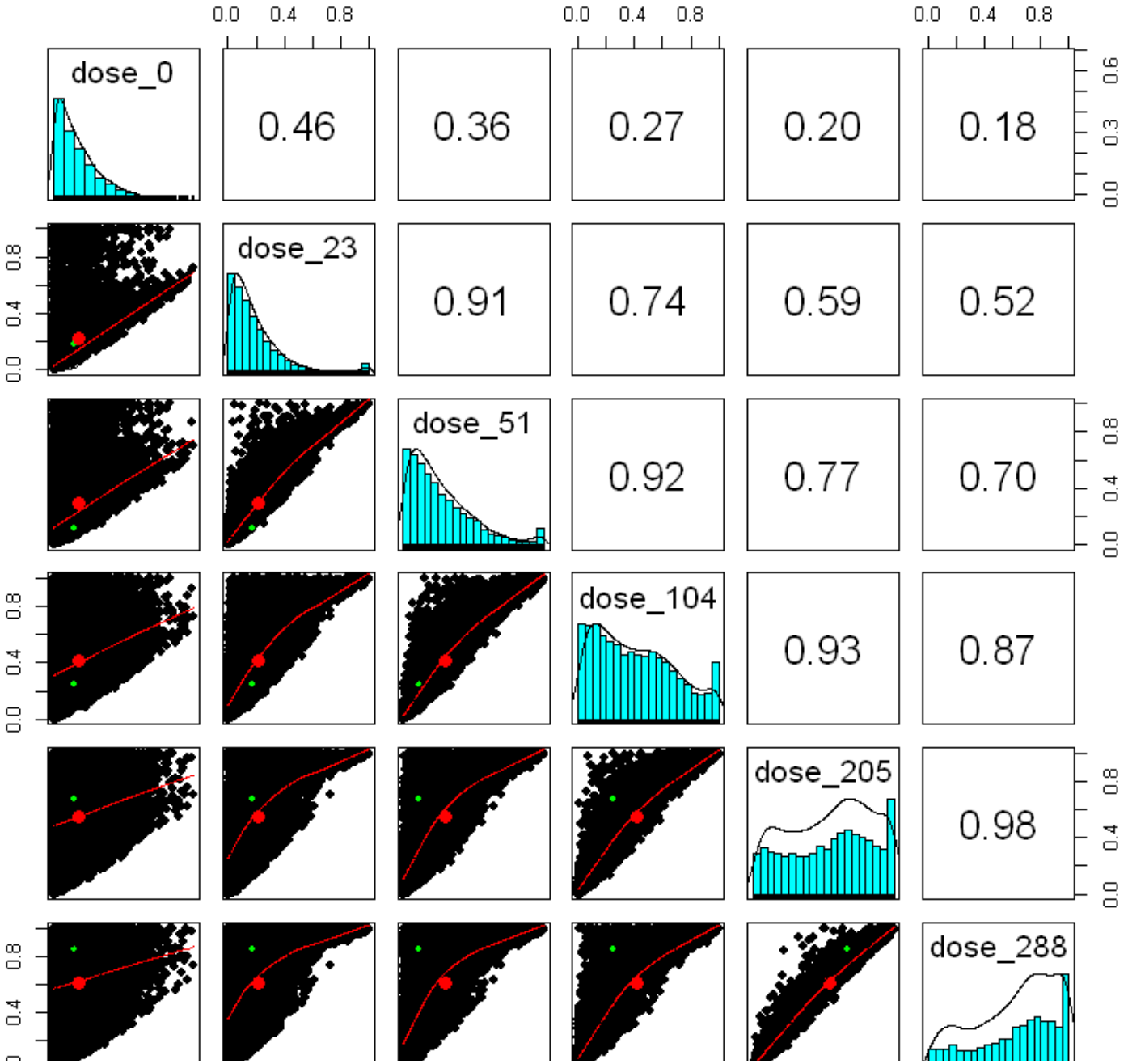
In [26]:
```
pairs.panels(rbind(param[1:nsimul0,],param[rej$region,]),pch=c(rep(19,nsimul0),rep(21,sum(rej$region))),
             bg="green")

pairs.panels(param[rej$region,],pch=21,bg="green")
```

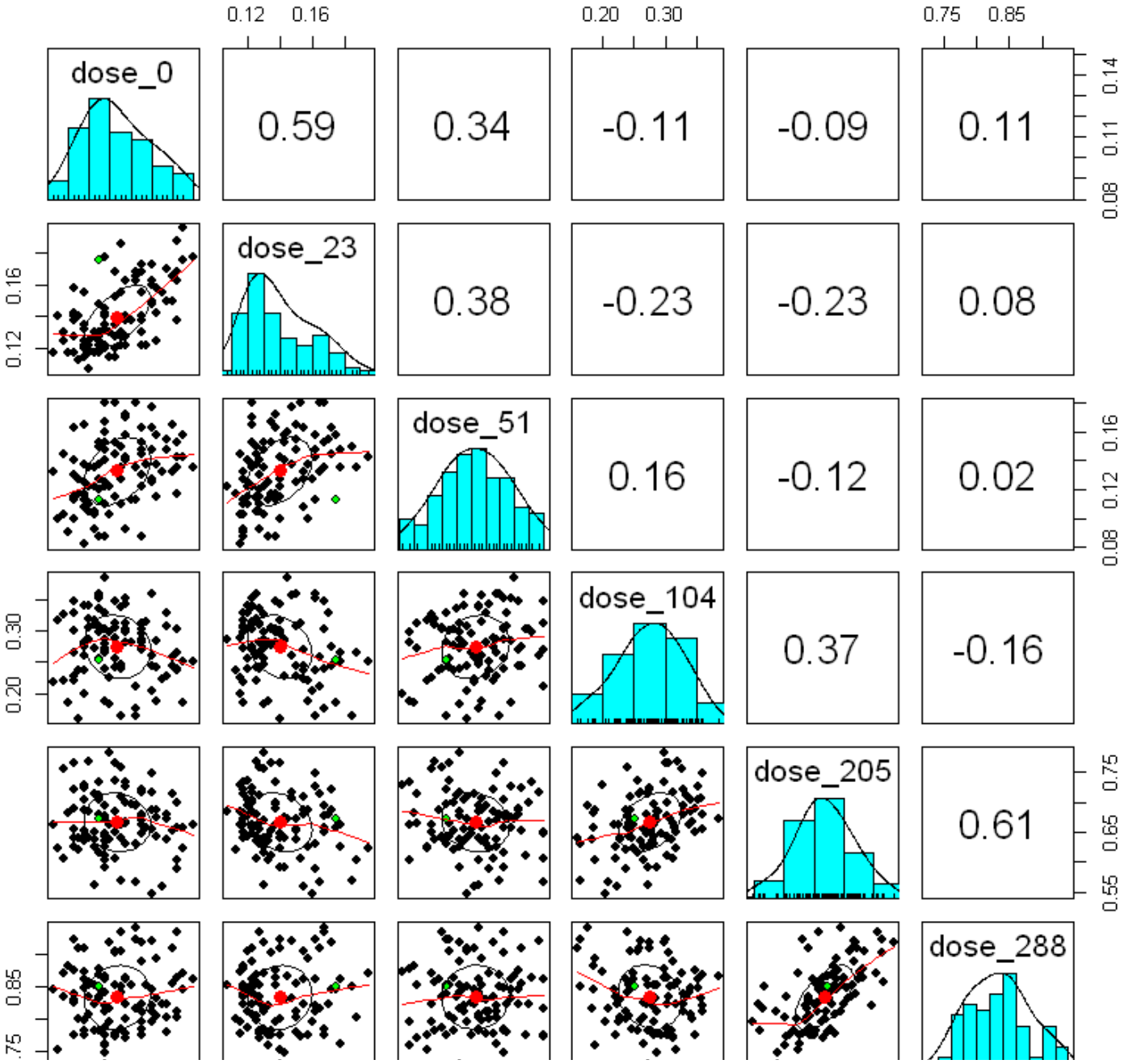

kout
alpha1
alpha2
alpha3
beta
0.01
-0.02
0.00
0.00
0.01
0.01
0.01
0.01
0.00
-0.01
0.85
0.95
0
2
4
6
8
10
0.0
0.4
0.8
0.000
0.006
0.012
1
2
3
4


kout
alpha1
alpha2
alpha3
beta
0.17
0.76
-0.03
-0.06
0.31
-0.04
0.27
0.45
0.20
0.20
0.974
0.980
0.986
2
4
6
8
10
0.0006
0.0009
0.0012
0.0
0.4
0.8
0.2
0.6
1.0

Echantillons a priori et a posteriori des statistiques

In [27]:
```
pairs.panels(rbind(stats[1:nsimul0,],mortality),pch=c(rep(19,nsimul0),21),
             bg="green")

pairs.panels(rbind(stats[rej$region,],mortality),pch=c(rep(19,sum(rej$region)),21),
             bg="green")
```

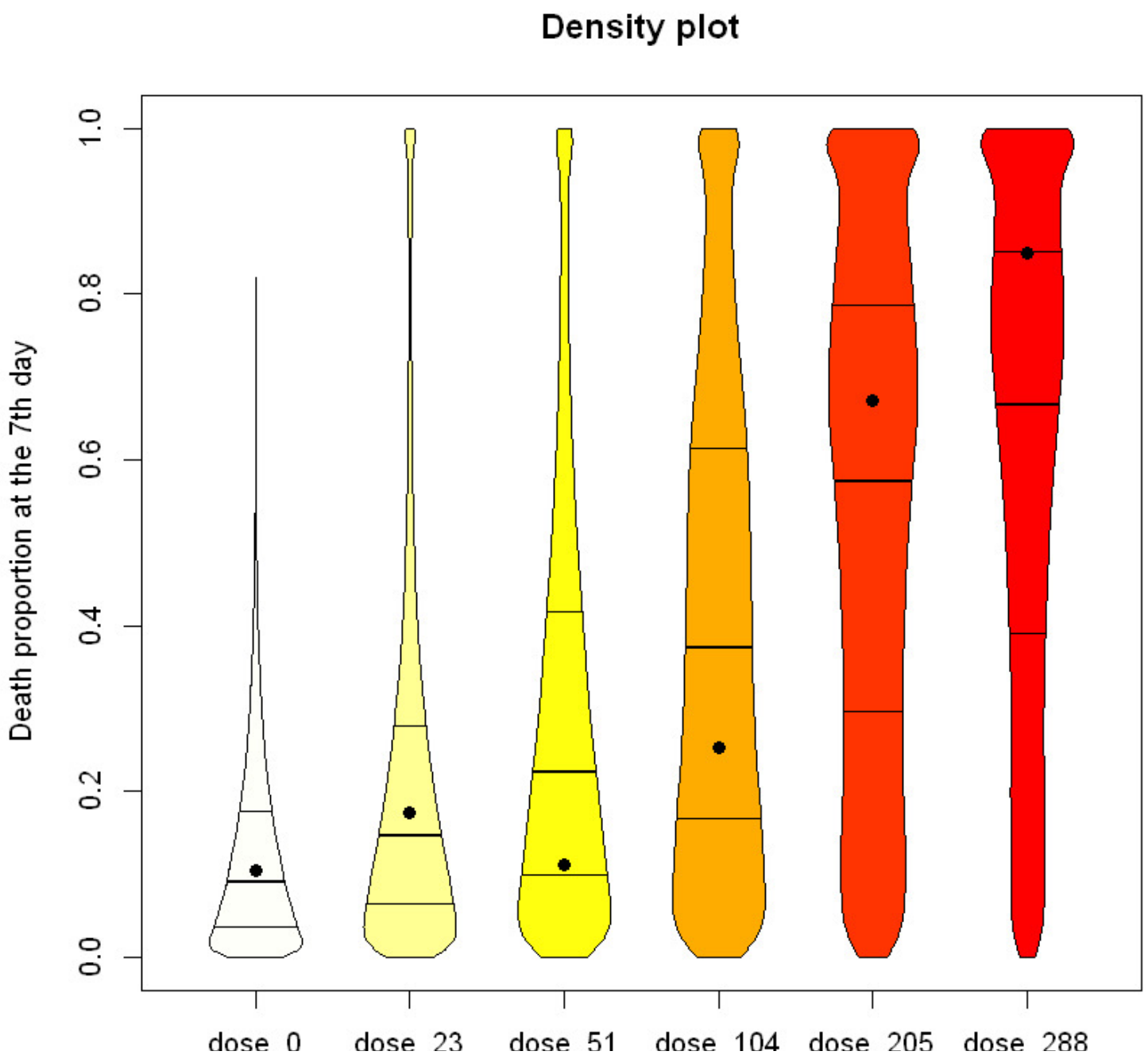

dose_0
dose_23
dose_51
dose_104
dose_205
dose_288
0.46
0.36
0.27
0.20
0.18
0.91
0.74
0.59
0.52
0.92
0.77
0.70
0.93
0.87
0.98
0.0
0.4
0.8
0.3
0.6


dose_0
dose_23
dose_51
dose_104
dose_205
dose_288
0.59
0.34
-0.11
-0.09
0.11
0.38
-0.23
-0.23
0.08
0.16
-0.12
0.02
0.37
-0.16
0.61
0.12
0.16
0.20
0.30
0.75
0.85
0.08
0.11
0.14
0.55
0.65
0.75

Distributions a posteriori des prédictions des proportions de mortalité aux différentes doses toxiques

In [28]:
```
## Marginal posterior distributions of mortality proportions
median.mortality=apply(stats,2,median)
median.mortality
cols.index=round((max(median.mortality)-median.mortality)/(max(median.mortality)-min(median.mortality))*100)
violinBy(stats,col=heat.colors(100)[pmax(1,cols.index)],
    xlab="Toxic dose (# toxic pollen grains / cm2)",
         ylab="Death proportion at the 7th day")
points(mortality,pch=19)
```

dose_0	0.0925
dose_23	0.1475
dose_51	0.225
dose_104	0.375
dose_205	0.575
dose_288	0.6675


Density plot
Death proportion at the 7th day
0.0
0.2
0.4
0.6
0.8
1.0
dose_0
dose_23
dose_51
dose_104
dose_205
dose_288

Pour aller plus loin

Dans le package abc

Nous avons utilisé l'ABC acceptation-rejet : abc(target=mortality, param=param, sumstat=stats, tol=tol.opt, method="rejection")

Autres méthodes implémentées dans le package abc :

method="loclinear" : a local linear regression method corrects for the imperfect match between S(y) and S(y0)

method="ridge" : method that performs local-linear ridge regression and deals with the collinearity issue

method="neuralnet" : non-linear regression to minimize the departure from non-linearity using the function nnet

Analyses complémentaires disponibles dans le package abc :

Model selection, Goodness-of-fit, Posterior predictive checks.

Autres packages R

abctools: Tuning ABC analyses https://journal.r-project.org/archive/2015-2/nunes-prangle.pdf (https://journal.r-project.org/archive/2015-2/nunes-prangle.pdf)

abcrf: ABC via random forests

EasyABC: Several algorithms for performing efficient ABC sampling schemes, including 4 sequential sampling schemes and 3 MCMC schemes

Logiciels

DIY-ABC: to perform parameter estimation and model selection for population genetics models

ABC-SysBio python package: parameter inference and model selection for dynamical systems

ABCtoolbox programs: various ABC algorithms including rejection sampling, MCMC without likelihood, a particle-based sampler, and ABC-GLM

PopABC: software package for inference of the pattern of demographic divergence, coalescent simulation, Bayesian model choice

In [ ]: