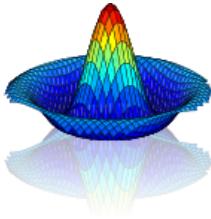


Ajustage de paramètres par ABC - École chercheur·se·s 2018

Nicolas Dumoulin

28 Mars 2018

#ÉquipeAjustage vs #TeamCalibration



MEXICO
WEXICO

Approximate Bayesian Computation

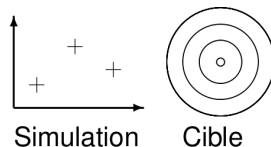
- Tirer $\theta^* \sim \pi(\theta)$
- Simuler $D' \sim f(\theta^*)$
- Si $\rho(D', D) \leq \epsilon$, accepter θ^* , sinon le rejeter
- Répéter jusqu'à ce qu'un échantillon de la taille désirée soit obtenu (Pritchard et al., 1999)



Distribution *a priori*
 $\pi(\theta)$



Distribution *a posteriori*
 $\mathbb{P}(\theta | \rho(D', D) \leq \epsilon)$



Approximate Bayesian Computation

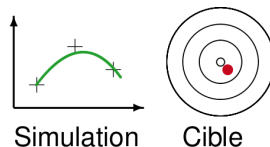
- Tirer $\theta^* \sim \pi(\theta)$
- Simuler $D' \sim f(\theta^*)$
- Si $\rho(D', D) \leq \epsilon$, accepter θ^* , sinon le rejeter
- Répéter jusqu'à ce qu'un échantillon de la taille désirée soit obtenu (Pritchard et al., 1999)



Distribution *a priori*
 $\pi(\theta)$



Distribution *a posteriori*
 $\mathbb{P}(\theta | \rho(D', D) \leq \epsilon)$



Approximate Bayesian Computation

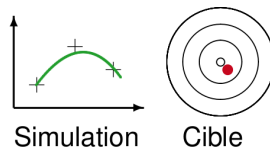
- Tirer $\theta^* \sim \pi(\theta)$
- Simuler $D' \sim f(\theta^*)$
- Si $\rho(D', D) \leq \epsilon$, accepter θ^* , sinon le rejeter
- Répéter jusqu'à ce qu'un échantillon de la taille désirée soit obtenu (Pritchard et al., 1999)



Distribution *a priori*
 $\pi(\theta)$



Distribution *a posteriori*
 $\mathbb{P}(\theta | \rho(D', D) \leq \epsilon)$



Approximate Bayesian Computation

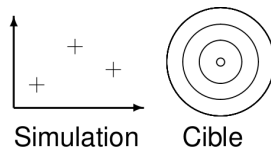
- Tirer $\theta^* \sim \pi(\theta)$
- Simuler $D' \sim f(\theta^*)$
- Si $\rho(D', D) \leq \epsilon$, accepter θ^* , sinon le rejeter
- Répéter jusqu'à ce qu'un échantillon de la taille désirée soit obtenu (Pritchard et al., 1999)



Distribution *a priori*
 $\pi(\theta)$



Distribution *a posteriori*
 $\mathbb{P}(\theta | \rho(D', D) \leq \epsilon)$

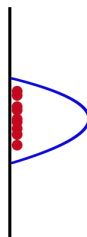


Approximate Bayesian Computation

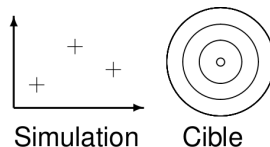
- Tirer $\theta^* \sim \pi(\theta)$
- Simuler $D' \sim f(\theta^*)$
- Si $\rho(D', D) \leq \epsilon$, accepter θ^* , sinon le rejeter
- Répéter jusqu'à ce qu'un échantillon de la taille désirée soit obtenu (Pritchard et al., 1999)



Distribution *a priori*
 $\pi(\theta)$

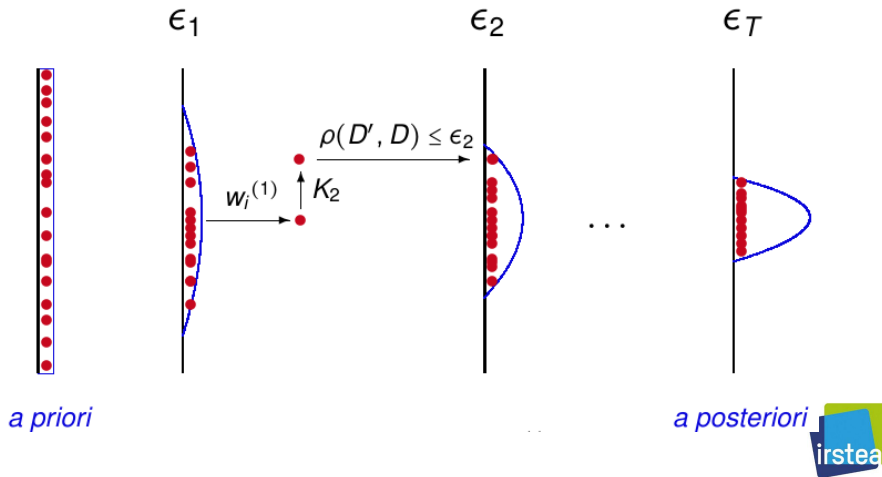


Distribution *a posteriori*
 $\mathbb{P}(\theta | \rho(D', D) \leq \epsilon)$



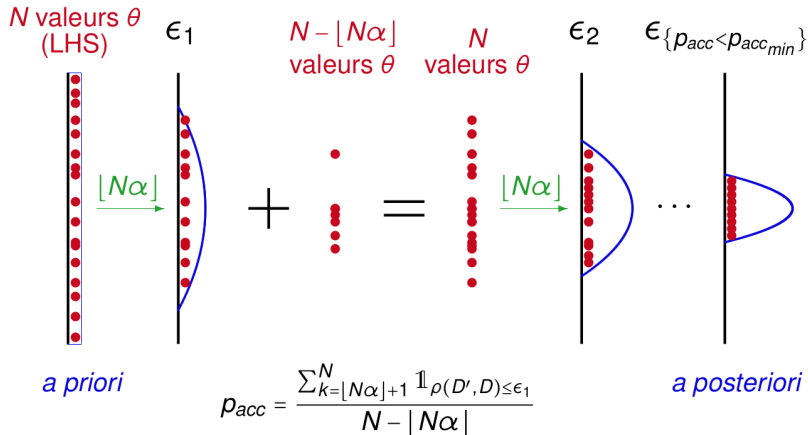
Population Monte Carlo

(Beaumont et al., 2009)



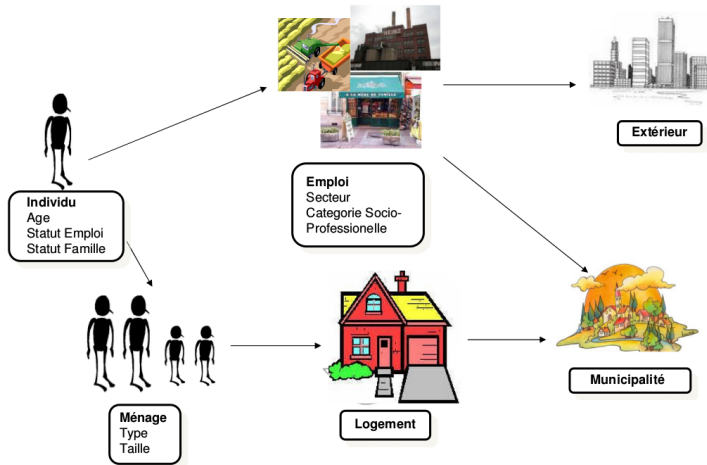
Adaptative Population Monte Carlo

(Lenormand et al., 2012)



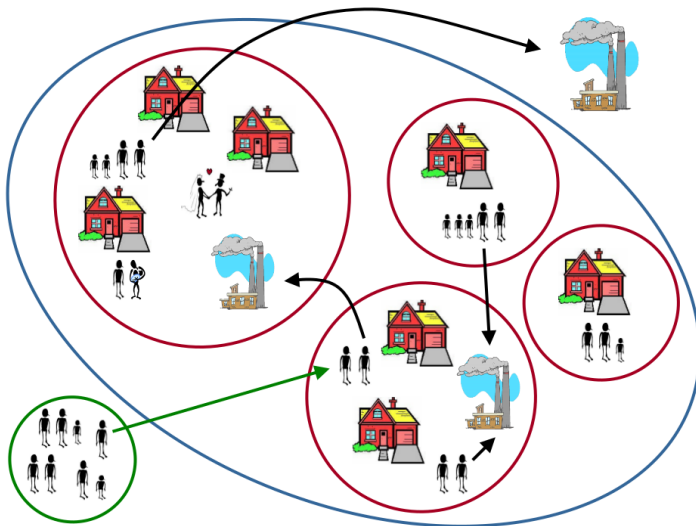
Ajustage d'un "vrai" modèle : SimVillages

- Dynamique de population en zone rurale (Huet et al., 2012)



Ajustage d'un "vrai" modèle : SimVillages

- Dynamique de population en zone rurale (Huet et al., 2012)



Ajustage d'un "vrai" modèle : SimVillages

- Dynamique de population en zone rurale (Huet et al., 2012)

Données

Year	Population (thousands)	Variation of population (%)
1962	1 657	
1968	1 592	-4,10
1975	1 481	-7,50

	1990		1999		2006	
	Number	%	Number	%	Number	%
Farmers	60	10,8	72	15,2	44	8,3
Craftsmen, storekeepers, business owners	124	22,3	50	10,5	65	12,3
Top executive managers, upper intellectual profession	24	4,3	32	6,8	15	2,8
Intermediary professions	104	18,7	58	12,2	98	18,6
Employees	144	25,9	169	35,7	186	35,2
Workers	100	18,0	93	19,6	120	22,7
Total	556	100	474	100	528	100

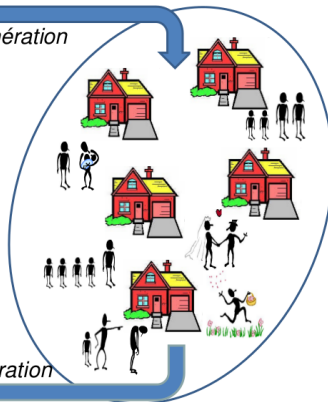
	Effectif	%
Number of households which moved there since less than 10 years	200	100
Less than 2 years	46	23
from 2 to 4 years	83	41,5
from 5 to 9 years	71	35,5
Number of households which moved there since 10 years or more	270	100
from 10 to 19 years	82	30,4
from 20 to 29 years	71	26,3
30 years and more	117	43,3
Total of households	470	

2006	
Number	%
Number of active people working in their municipality of residence	337 77,6
Number of active people working out of their municipality of residence	97 22,4
Total active employed people	434 100

Modèle

Génération

Calibration



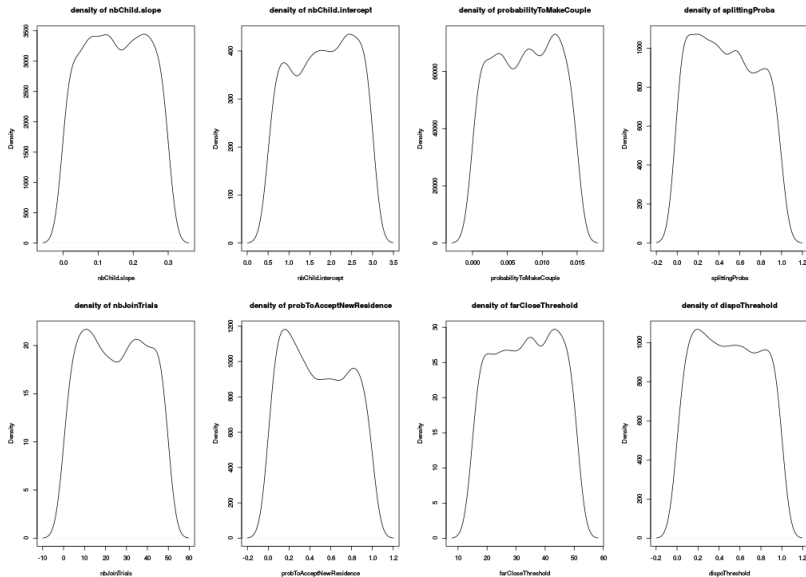
- 260 communes (département du Cantal)
- 8 paramètres d'entrée
- 8 statistiques résumantes (4 en 1999 et 4 en 2006)
- Début de simulation en 1990, ajustage sur 1999 et 2006
- Temps moyen par simulation : 1 minute

Paramètres	Description	Etendue
θ_1	Nombre d'enfants (pente)	$[0, 0.15]$
θ_2	Nombre d'enfants (intersection)	$[0.5, 3]$
θ_3	Indice de satisfaction du logement	$[0, 0.2]$
θ_4	Probabilité de se mettre en couple	$[0, 1]$
θ_5	Nombre d'essais pour se mettre en couple	$[0, 20]$
θ_6	Probabilité de se séparer	$[0, 0.05]$
θ_7	Seuil de proximité	$[0, 50]$
θ_8	Satisfaction de la demande de logement	$[0, 1]$

- Statistiques résumantes :
 - Pyramide des âges
 - Nombre d'habitants par commune
 - Solde migratoire
 - Nombre de migrations internes
- Méthode utilisée : APMC en Java
 - $N = 2000$
 - $\alpha = 0.5$
 - $\text{pacc_min} = 0.01$
- Durée : 3 jours sur cluster de calcul pour 60000 simulations

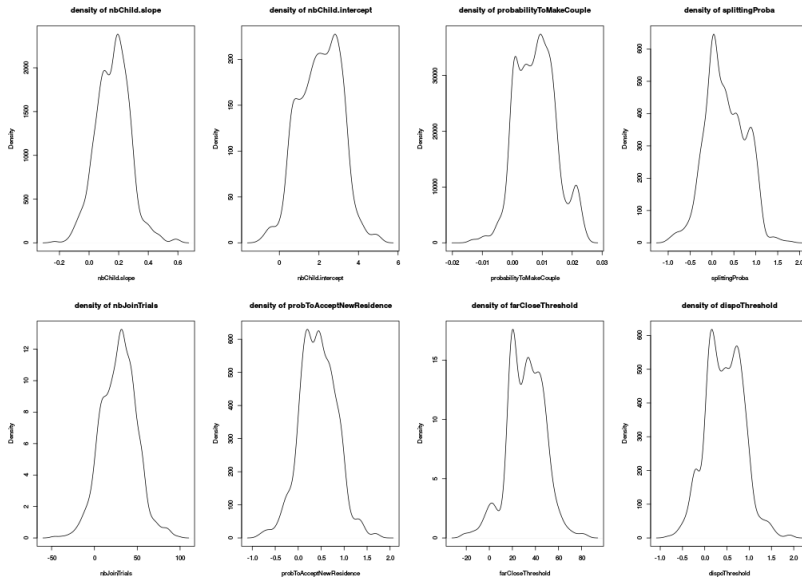
SimVillages APMC : évolution posterior

Posteriors at step 001



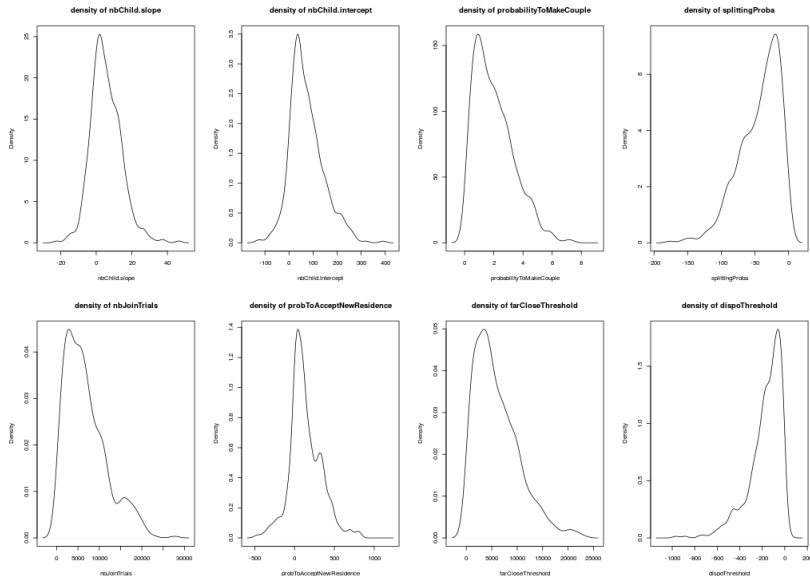
SimVillages APMC : évolution posterior

Posteriors at step 002



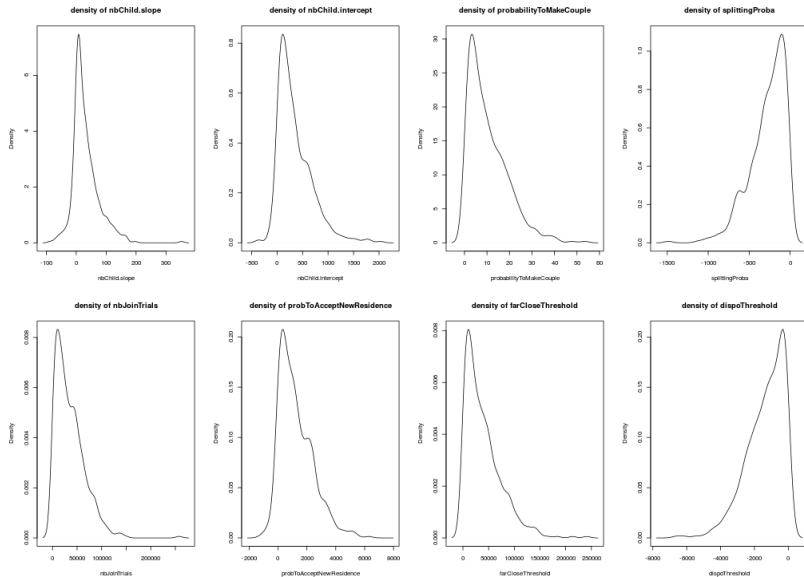
SimVillages APMC : évolution posterior

Posteriors at step 022



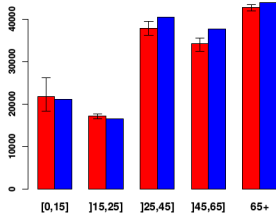
SimVillages APMC : évolution posterior

Posteriors at step 051

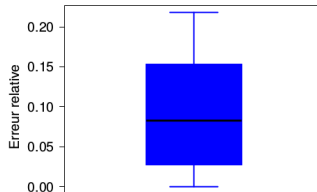


SimVillages APMC : statistiques résumantes 1999

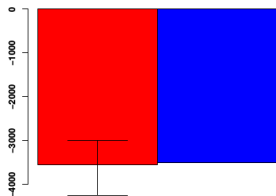
Pyramide des âges



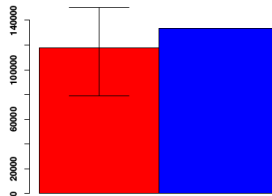
Nombre d'habitants par commune



Solde Migratoire



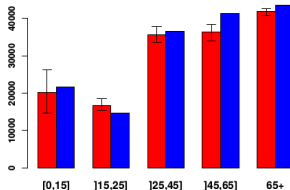
Nombre de migrations internes



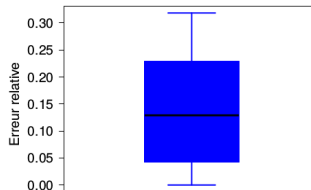
Données simulées D' et **données observées D**

SimVillages APMC : statistiques résumantes 2006

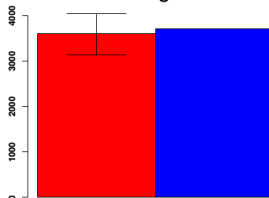
Pyramide des âges



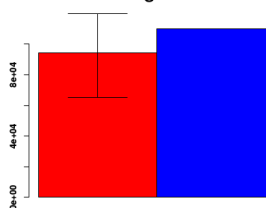
Nombre d'habitants par commune



Solde Migratoire



Nombre de migrations internes



Données simulées D' et **données observées D**

Lequel choisir ?



Hey, je démarre un package pour valoriser les derniers algos ABC !
Banco ?



Hey, je démarre un package pour valoriser les derniers algos ABC !
Banco ?



Une idée, une envie

Hey, je démarre un package pour valoriser les derniers algos ABC !
Banco ?



Ok, je m'occupe de la chorégraphie.

Banco !



Une idée, une envie



- Méthodes implémentées
 - ABC rejection (Pritchard et al., 1999)
 - ABC séquentiel
 - (Beaumont et al., 2009)
 - (Drovandi and Pettitt, 2011)
 - (Del Moral et al., 2012)
 - (Lenormand et al., 2012)
 - Markov Chain Monte Carlo
 - (Marjoram et al., 2003)
 - (Wegmann et al., 2009)
 - Simulated annealing (Albert et al., 2014; Fearnhead et al., 2012)
- Définition de contraintes sur les domaines de définition (" $X1 < X2 + X3$ ")
- Utilisation de fonctions d'échantillonnage arbitraire
- Option multicœurs
- Utilisation modèle binaire (exécutable) ou modèle java

Installation

Easy as 123

```
install.packages("EasyABC")  
library("EasyABC")  
?EasyABC
```

Installation

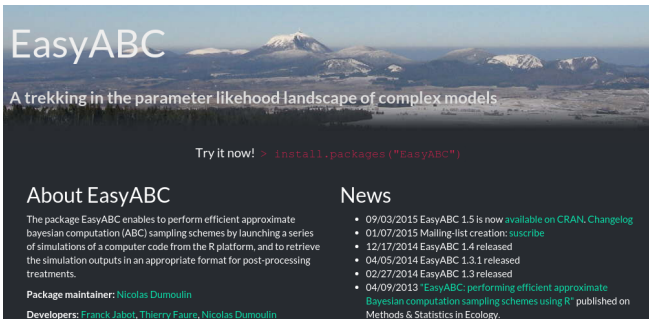
Easy as 123

```
install.packages("EasyABC")  
library("EasyABC")  
?EasyABC
```



- Première version en 2012, publiée sous license libre
- 6 contributions d'utilisateurs, dont un nouvel algorithme (SABC)
- Permet de démarrer rapidement une expérimentation avec ABC

<http://easyabc.r-forge.r-project.org/>



EasyABC

A trekking in the parameter likelihood landscape of complex models

Try it now! `> install.packages("EasyABC")`

About EasyABC

The package EasyABC enables to perform efficient approximate bayesian computation (ABC) sampling schemes by launching a series of simulations of a computer code from the R platform, and to retrieve the simulation outputs in an appropriate format for post-processing treatments.

Package maintainer: **Nicolas Dumoulin**

Developers: **Franck Jabot, Thierry Faure, Nicolas Dumoulin**

News

- 09/03/2015 EasyABC 1.5 is now [available on CRAN](#). [Changelog](#)
- 01/07/2015 Mailing-list creation: [suscribe](#)
- 12/17/2014 EasyABC 1.4 released
- 04/05/2014 EasyABC 1.3.1 released
- 02/27/2014 EasyABC 1.3 released
- 04/09/2013 "EasyABC: performing efficient approximate Bayesian computation sampling schemes using R" published on [Methods & Statistics in Ecology](#).

Premiers pas : le script

```
# Prenons un modèle comme exemple
> toy_model <- function(x){
+   c(x[1]+x[2] + rnorm(1,0,0.1),
+     x[1]*x[2] + rnorm(1,0,0.1))
+ }
# Définissons notre prior
> toy_prior = list(c("unif",0,1),c("normal",1,2))
# la cible
> sum_stat_obs = c(1.5,0.5)
# top départ !
> ABC <- ABC_rejection(model=toy_model, prior=toy_prior,
+   nb_simul=10, summary_stat_target=sum_stat_obs,
+   tol=0.2)
```

Premiers pas : le résultat

```
> ABC
$param
      [,1]      [,2]
param 0.7341818 0.01361074
param 0.6534961 0.80211864
$stats
      [,1]      [,2]
[1,] 0.8332396 0.1414957
[2,] 1.4453496 0.5851046
$weights
[1] 0.5 0.5
$stats_normalization
[1] 1.820294 1.015811
$nsim
[1] 10
$nrec
[1] 2
$comptime
[1] 0.00214529
```

- (Beaumont et al., 2009) l'utilisateur doit fournir les seuils de tolérance successifs (tolerance_tab)

```
> ABC_Beaumont <- ABC_sequential(method="Beaumont",  
+   model=toy_model, prior=toy_prior,  
+   nb_simul=10, summary_stat_target=sum_stat_obs,  
+   tolerance_tab=c(1.25,0.75))
```

- (Lenormand et al., 2012)
 - l'utilisateur doit fournir le nombre de simulations initial (nb_simul)
 - à chaque itération, le nombre de nouvelles simulations est un ratio de nb_simul ($\alpha = 0.5$ par défaut)
 - seuil d'arrêt p_acc_min par défaut à 0.05
 - obligation d'utiliser un LHS uniforme

```
> toy_prior2=list(c("unif",0,1),c("unif",0.5,1.5))
> ABC_Lenormand<-ABC_sequential(method="Lenormand",
+  model=toy_model, prior=toy_prior2,
+  nb_simul=10, summary_stat_target=sum_stat_obs,
+  p_acc_min=0.4)
```


- La fonction du modèle doit accepter comme premier argument l'index du flux de nombre pseudo-aléatoires à utiliser

```
> toy_model_parallel <- function(x) {  
+   set.seed(x[1])  
+   c(x[2]+x[3] + rnorm(1,0,0.1),  
+     x[2]*x[3] + rnorm(1,0,0.1))  
+ }
```

- Chaque fonction d'EasyABC accepte une option `n_cluster` (par défaut 1)

```
> ABC <- ABC_rejection(model=toy_model_parallel,  
+   prior=toy_prior, nb_simul=n, n_cluster=2,  
+   summary_stat_target=sum_stat_obs, tol=0.2,  
+   use_seed=TRUE)
```

Intégration code C++

- Ajout du branchement dans votre code C++

```
extern "C" {  
  void trait_model(double *input, double *stat_to_return){  
    // compute output and fill the array stat_to_return  
  }  
}
```

- Compilation

```
$ R CMD SHLIB trait_model_rc.cpp
```

- Chargement dans R

```
> dyn.load("trait_model_rc.so")
```

- Et voilà

```
trait_model <- function(input=c(1,1,1,1,1,1)) {  
  .C("trait_model", input=input,  
    stat_to_return=array(0,4))$stat_to_return  
}
```



- Voir aussi module python ELFI

```
install.packages("EasyABC")
```

<http://easyabc.r-forge.r-project.org/>

- Voir aussi module python ELFI

```
install.packages("EasyABC")
```

<http://easyabc.r-forge.r-project.org/>

Do you MeOW?

