HAL
open science

Implémentation en R d'une alternative ("CGEM-EV") au maximum de vraisemblance pour des champs de type Matérn

Didier A. Girard

To cite this version:

Didier A. Girard. Implémentation en R d'une alternative ("CGEM-EV") au maximum de vraisemblance pour des champs de type Matérn. [Rapport Technique] LJK. 2011. hal-00648526v2

HAL Id: hal-00648526

https://hal.science/hal-00648526v2

Submitted on 16 Dec 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Implémentation en R d'une alternative ("CGEM-EV") au maximum de vraisemblance pour des champs de type Matérn

Didier A. Girard
Département Statistique et Probabilités
LJK, Grenoble

Résumé: Après une introduction à l'alternative CGEM-EV qui permet d'estimer les paramètres de champs aléatoires Gaussiens de type Matérn, nous présentons son implémentation en R en commentant l'exécution en R du script ScriptForDemo.R qui exploite 2 fichiers source R annexés. Cette implémentation résout les systèmes linéaires invoqués, par une classique méthode itérative du gradient conjugué associée à un produit de matrice-vecteur par FFT. Elle permet donc de refaire des expérimentations comme celles de [2], mais pour des données de plus grande taille.

I. Introduction

Par exemple en géostatistique, dans les modèles probabilistes communément utilisés pour l'observation, éventuellement bruitée, en des sites multiples connus, d'un champ aléatoire spatial stationnaire, il y a toujours 3 paramètres dont le sens physique est clair: la variance du champ (dite aussi "puissance moyenne"), le niveau de bruit des observations (souvent un bruit de "mesurage" dont on connaît souvent le niveau par calibration préalable), et un vecteur de paramètres qui détermine la fonction d'autocorrélation du champ.
Nous nous sommes intéressés aux cas assez fréquents où <u>le 3ème paramètre est, comme les 2 premiers, réduit à un scalaire</u>, précisément le paramètre d'échelle spatiale (cad une "longueur de décorrélation" ou "support de corrélation" au delà duquel la corrélation est en dessous de, disons, 5%) On parle donc de «*one-range-parameter model*».

Un premier exemple est le classique cas exponentiel :

$$\text{“corrélation entre la valeur du champ en } \mathbf{s} \text{ et celle en } \mathbf{t}\text{”} = \exp(-\ \theta \cdot d)$$

où d est la distance Euclidienne entre les sites **s** et **t**.
Un deuxième exemple important est la modélisation très couramment utilisée par les géostatisticiens (dans le cas dimension 2 ou 3) où la fonction d'autocorrélation est celle dite “d'autocorrélation sphérique” pour laquelle le range-parameter est simplement le diamètre du support (sphérique) de cette fonction. Un autre exemple important est la famille de fonction d'autocorrélation dite de Matern avec son indice de régularité locale ν que l'on peut choisir plus ou moins grande avec $\nu > 0$ (ν=1/2 correspond au cas exponentiel); on peut trouver sa définition par ex. dans [2].
Notons que l'utilisation d'une famille “one-range-parameter” (par ex. Matern avec ν fixé) est fréquente dans le cas où la corrélation est forte car il est connu qu'il faut alors beaucoup de données pour estimer déjà un seul paramètre de forme (travaux de Stein (ou ses collaborateurs) à Chicago et très récemment de Zhang à Purdue) (les tentatives de laisser les seules données choisir l'indice de régularité locale ν sont souvent décevantes dès qu'il y a un bruit même faible) et car il est connu que, même si le vrai modèle n'est pas en fait dans la famille ajustée aux données, la maximisation de la vraisemblance gaussienne (ML) donne malgré tout souvent des résultats utiles.

Pour estimer ces 3 paramètres, la méthode la plus attirante (à cause de ses très bonnes propriétés théoriques) est le célèbre principe ML gaussien (car typiquement asymptotiquement efficient); malheureusement son implémentation est ici trop coûteuse au delà de tailles de quelques centaines de données à moins qu'il n'y ait des astuces rendant possibles les calculs de déterminants nécessaires; un exemple de tel cas bien connu est celui où 1) les observations sont faites aux nœuds d'une grille régulière, 2) le bruit est homogène, 3) l'on “force” de plus implicitement une périodicité du champ spatial ce qui peut fortement dégrader l'efficacité statistique du ML surtout pour les cas de forte corrélation non-périodique. Les matrices de covariance candidates sont alors circulantes ou bloc-circulantes : la transformée de Fourier rapide (FFT) est donc l'outil qui rend possible l'approche ML pour des problèmes de très grande tailles mais sous les 3 restrictions précédentes.

Nous introduisons ici une méthode qui, au vu de résultats théoriques et expérimentaux récents ([1], [2]), semble très bien éliminer la 3ème restriction (si on se limite à des valeurs

de ν "raisonnables", cf [2]). La preuve de son efficacité pour pouvoir éliminer la 2ème restriction reste à faire. Quant à la 1ère restriction, elle pourrait "à peu près" être éliminée elle-aussi si les sites d'observations étaient denses dans un domaine rectangulaire, par un classique "local pooling" (par exemple travaux récents de Paciorek à Harvard) puisque l'inhomogénité du nouveau bruit créée par la possible nonuniformité de la densité des sites pourrait être prise en compte par notre méthode.

Le cadre de ce travail est celui où les variances des bruits d'observation sont connues. Mais nous ne supposons que ces variances sont spatialement homogènes. Un exemple courant est le cas de "pooled gridded data" mentionné au-dessus où le bruit, homogène avant "pooling", est de niveau connu. Soit donc D la matrice diagonale connue, telle que le vecteur des observations $\mathbf{y}$, de taille n, satisfait

$$\mathbf{y} = \mathbf{z} + D^{-1}\varepsilon \quad \text{où} \quad \mathbf{z} \text{ est de loi } \mathrm{N}(0, b_0\, C_{\theta_0}) \text{ et } \varepsilon \text{ de loi } \mathrm{N}(0, I),$$

où I est la matrice identité, et où N(0,V) est la classique loi Gaussienne de matrice de covariance V (le choix d'une variance égale à 1 pour les composantes de ε est sans perte de généralité); les z_i (ou $z_{i,j}$ en dimension 2) étant classiquement ordonnés (ordre lexicographique), et le champ spatial $\mathbf{z}$ discrétisé sur la grille étant stationnaire, la matrice de corrélation C_θ est alors une matrice Toeplitz (ou bloc-Toeplitz avec des blocs eux-mêmes Toeplitz, voir par ex. [2]), et avec tous les termes diagonaux égaux à 1. La variance du champ est donc b_0. Posons

$$A_{b,\theta} := bC_\theta(bC_\theta + D^{-2})^{-1}.$$

Il est bien connu que, si b et θ sont les vraies valeurs, le vecteur $A_{b,\theta}\,\mathbf{y}$ est alors le "meilleur" (en de nombreux sens) estimateur de $\mathbf{z}$; c'est notamment la moyenne conditionnelle de $\mathbf{z}$ sachant $\mathbf{y}$.
Pour n'importe quel couple (b,θ) fixé, calculer le produit $A_{b,\theta}\,\mathbf{y}$ se ramène à la résolution d'un système Toeplitz-Plus-Diagonal : dans les 10 dernières années, de nombreux travaux (e.g. Chan, Nagy) ont montré qu'un tel système se résout relativement aisément par une méthode itérative de type gradient conjugué préconditionné, nécessitant typiquement au plus quelques dizaines de FFTs et une occupation mémoire d'ordre n.

L'alternative au ML pour estimer b et θ que nous étudions ici, nommée "CGEM-EV", est une équation hybride combinant une équation "de vraisemblance" et une variance empirique, et sa version randomized-trace. En prélude à sa définition, notons qu'une simple manipulation algébrique montre que si l'on <u>contraint le paramètre de corrélation à une valeur</u>

fixe θ_1 , chercher le b , disons $\hat{b}_{\mathrm{CML}}(\theta_1)$, qui maximise la vraisemblance implique la "constrained-correlation likelihood equation" en b suivante: $\mathrm{CCLE}(b,\ \theta_1) = 0$ où

$$\mathrm{CCLE}(b,\ \theta) = \langle A_{b,\theta}\mathbf{y}, D^2(I - A_{b,\theta})\mathbf{y}\rangle - \mathrm{Trace}(A_{b,\theta}).$$

La méthode que nous proposons ici dans ce cadre peut maintenant être décrite simplement :

1-estimer b_0 par un simple estimateur de type variance-empirique $\hat{b}_{\mathrm{EV}} = \langle \mathbf{y},\mathbf{y}\rangle / n - \mathrm{Trace}(D^{-2}) / n$.
2-choisir comme estimateur de θ_0 la première valeur θ en partant d'une valeur très petite (cad associée à une très forte corrélation) qui satisfait la randomized-trace version de $\mathrm{CCLE}(\hat{b}_{\mathrm{EV}},\theta) = 0$.

Nous noterons cet estimateur par $\hat{\theta}_{\mathrm{CGEM\text{-}EV}}$ (ou $\hat{\theta}_{\mathrm{CGEM}}(\hat{b}_{\mathrm{EV}})$ puisque $\hat{\theta}_{\mathrm{CGEM}}(b)$ est en fait défini pour un b quelconque qui remplacerait $\hat{b}_{\mathrm{EV}}$ en 2)) où CGEM est un acronyme raccourci pour "Conditional one-range-parametered Gaussian Gibbs Energy Mean", l'apparition ici de la notion d'énergie de Gibbs paramétrée (par θ) pour le champ $\mathbf{z}$ (qui désigne $n^{-1}\langle \mathbf{z}, C_\theta^{-1}\mathbf{z}\rangle$) et de sa moyenne conditionnelle étant expliquée en [1].

En utilisant, par exemple, une classique bissection, la résolution de l'équation dans l'étape 2- (dite équation-estimante) nécessite typiquement en pratique le calcul de la randomized-trace version de $\mathrm{CCLE}(\hat{b}_{\mathrm{EV}},\theta)$ (qui lui même ne nécessite que 2 produits matrice-vecteur où la matrice est du type $A_{b,\theta}$) pour au plus une vingtaine de valeurs candidates pour θ.
Le calcul de $\hat{\theta}_{\mathrm{CGEM}}(\hat{b}_{\mathrm{EV}})$ nécessite donc seulement une place mémoire d'ordre n, et pourrait ne nécessiter que quelques centaines de FFTs, nombre fonction de la vitesse de convergence du gradient conjugué décrit au-dessus, et de la précision souhaitée.

Nous avons pu implémenter en R cette méthode avec relativement peu de lignes de code grâce au package "fields" créé par D. Nychka et ses collaborateurs [3] au National Center for Atmospheric Research, Boulder, CO. Les fichiers source de 2 fonctions, et d'un script qui les utilise sont donnés en pièces jointes. La suite de ce rapport est une "démo" au sens où nous décrivons, par l'exemple, ce que peuvent calculer ces 2 fonctions, en commentant l'exécution de ce script.

Les fonctions R créées peuvent fonctionner en principe pour ν quelconque, mais la théorie asymptotique ([1]) indique que, pour les modèles de type Matérn, la perte d'efficacité statistique (en ce qui concerne l'estimation du range) comparé au ML ne devrait être faible que pour ν "petit". Nous ne les avons testées essentiellement que pour ν=1/2. Par contre, ces fonctions R ne fonctionnent pour l'instant que pour $D=I$.

N.B. Pour être en accord avec le package fields bien documenté, nous utilisons sa terminologie : le "`range`" est utilisé au lieu de la vitesse de décroissance θ des décorrélations de type Matern. Pour le cas exponentiel ci-dessus (i.e. ν =1/2) on a simplement: range=$1/\theta$.

Dans le code nous utilisons "nu" au lieu de ν.

Enfin plutôt que la fonction CCLE nous manipulons dans ce code les valeurs de la fonction CGEM définie par

$$\mathrm{CGEM}(b,\theta) = n^{-1}\, b \cdot \left(\mathrm{CCLE}(b,\theta) + n \right) ,$$

l'équation estimante en θ devenant donc clairement $\mathrm{CGEM}(\hat{b}_{\mathrm{EV}},\theta) = \hat{b}_{\mathrm{EV}}$.

II. Implémentation R et une "démo"

II.a) Début d'analyse d'un cas avec nu=1/2, trueRange=0.2, bTrue=1000

On exécute dans la "console R", tout d'abord, la commande "chargeant" le package fields

```
library(fields)
```

puis les commandes du premier paragraphe du script `"ScriptForDemo.R"` que nous suggérons de copier-coller à partir d'une fenêtre d'édition ouverte sur ce script

(Notons que nous choisissons dans ce script une taille relativement petite pour la grille des observations ici, précisément 27 x 27; mais des tailles beaucoup plus grande pourront être prises en compte, comme nous allons le voir, dès que l'étape "génération des données" (ici en n^3) sera remplacée par une méthode plus rapide):

```
#setting for simulation of a Gaussian Matern field on a grid n1*n1
n1<-27
#each realization of the field will be observed at the sites
#    (x_1,x_2)(i), i=1,..,n1*n1     equispaced on the unit square:
x<- (1./n1)*matrix( c(rep(1: n1, n1),rep(1: n1,each= n1)), ncol=2)
n<- nrow(x)     # n=n1*n1
# true values for the 4 parameters :
bTrue<- 1000.0    #variance (or power) of the field
# noise level =1
trueRange<-0.2
nu<-1./2    #this "smoothness index" will be assumed known
#
```

Cela a initialisé le modèle des champs que l'on va simuler. On peut obtenir une représentation graphique de la fonction d'autocorrélation par le "`plot`" ci-dessous:

```
lags<-(1./n1)*(0:(n1-1))
autocorrelationFunct<- Matern( lags, range= trueRange, smoothness=nu)
plot( lags, autocorrelationFunct, type="l")
```

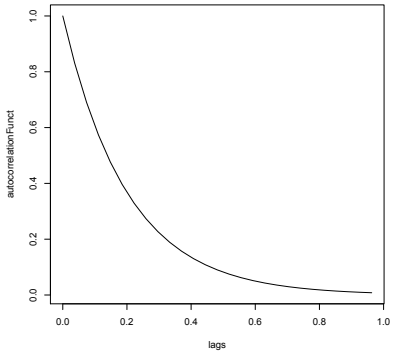

Figure 1: fonction d'autocorrélation avec `trueRange=0.2`

En fait, pour faire appel à la méthode de simulation classique, nous devons d'abord former la matrice $b_0\, C_{\theta_0}$ puis effectuer sa décomposition de Cholesky :

```
ut<-system.time(
{
Cov.mat<-  bTrue* Matern( rdist(x,x), smoothness=nu, range= trueRange))
```

```
A<- chol( Cov.mat)
}
)
```

Nous avons ajouté un appel à la fonction (de R) `system.time` qui fournit l'information suivante en sortie:

```
> ut
   user  system elapsed
  0.893   0.030   0.972
```

Cela donne une idée du temps de calcul demandé, sur notre machine actuelle, par l'évaluation du critère de vraisemblance sur, disons, une grille de 15 valeurs de `range` pour b fixé : ce sera donc environ 15 secondes puisque le déterminant est immédiat après la décomposition de Cholesky.

Une fois cette décomposition calculée, il est assez rapide de simuler plusieurs réalisations du champ aléatoire:

```
######### simulation of 9 realizations  #################
set.seed(987)
Z<- array(NA,c(n1*n1,9))
ut<-system.time(
for(indexReplcitate in 1:9){
#one simulates 9 realizations
gtrue<- t(A) %*% rnorm(n)
Z[,indexReplcitate]<-c(gtrue)
}
)
#
######### plot of these 9 realizations  #################
set.panel( 3,3)
#
x1 <- x2 <- seq(1, n1, 1)
for(indexReplcitate in 1:9){
Ztrue<-Z[,indexReplcitate]
image(x1,x2,matrix(Ztrue,n1,n1))}
```

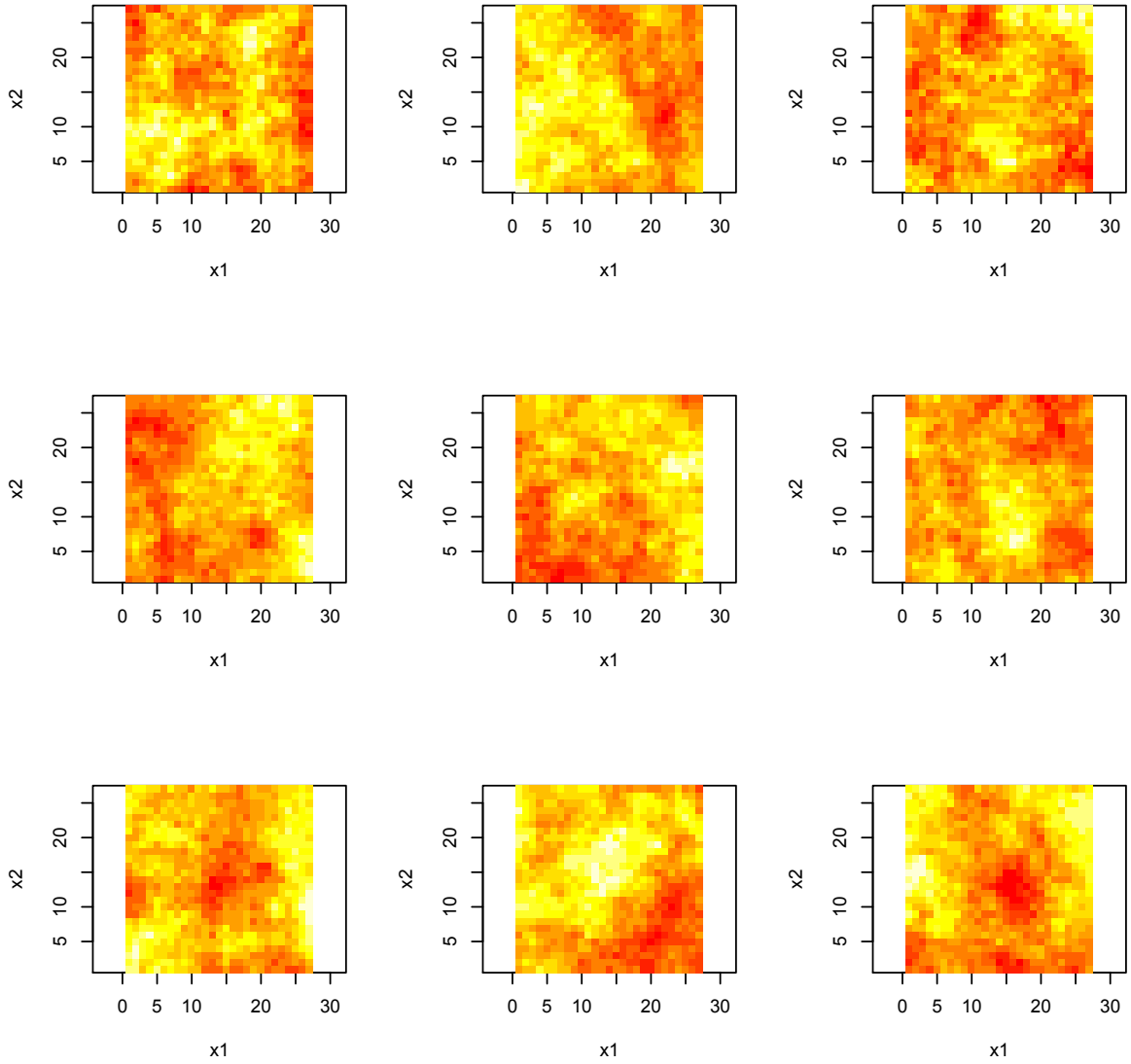

Figure 2: 9 simulations exactes avec `trueRange=0.2`

Créons maintenant un vecteur simulé d'observations $\mathbf{y}$, avec un bruit de variance 1, et calculons `bEV` qui désigne $\hat{b}_{\mathrm{EV}}$

```
set.seed(678)
indexReplcitate <-1
y <- Z[,indexReplcitate]+rnorm(n)
# variance empirique corrigée du biais
bEV<-sum(y**2)/n -1.
bEV
```

Cela nous donne $\hat{b}_{\mathrm{EV}}$:

```
> bEV
[1] 645.4668
```

Nous commenterons cette valeur à la fin de la Section II.c). Notons seulement que si on avait choisi une des 8 autres réalisations, nous aurions eu une valeur de $\hat{b}_{\mathrm{EV}}$ dans l'ensemble:

```
{986.7059, 632.3244, 1046.386, 1334.11, 710.2087, 930.0827, 1322.626, 1433.817}
```

La 1ère réplique n'est donc pas un point "aberrant" de la population.

II.b) La fonction CGEMevalOnGrid, et suite de l'analyse précédente

Puisque nous souhaitons appréhender la puissance et les limites de la méthode d'estimation CGEM-EV, il est souhaitable de d'abord "visualiser" l'équation estimante avant de la résoudre.

Nous avons choisi de créer une fonction R qui, pour un **y** donné, évalue l'énergie conditionnelle `CGEM(range,b)` pour `b=bEV`, sur toute une liste donnée (et ordonnée, nommée ici `candidateRanges.Grid`) de valeurs pour `range` et non pas un seul point car il s'avèrera très utile de faire démarrer l'itération du gradient-conjugué à partir d'une solution "pas trop loin" et pour cela il est naturel d'utiliser le résultat du point voisin sur la grille.

Le fichier "`CGEMevalOnGrid.R`" contient uniquement le code de cette fonction; ce fichier doit être préalablement placé dans le répertoire de travail, et le chargement de cette fonction (sinon il faut ajouter l'information du chemin d'accès au dossier le contenant) sera donc la commande `source("CGEMevalOnGrid.R")` :

```
# use of CGEMevalOnGrid ####################
#génération de l'aléa utilisé dans la "randomized-trace :"
set.seed(345)
w <- rnorm(n)
w<- c( w)
#
candidateRanges.Grid<- trueRange * 10**seq(-1.1,1.1,,15)
#
#ds les 2 lignes suivantes, on "source" la fonction CGEMevalOnGrid,
# on la charge et  on l'execute
source("CGEMevalOnGrid.R")
out <- CGEMevalOnGrid(y,w,candidateRanges.Grid,nu)
out
```

La sortie de la `CGEMevalOnGrid.R` contient les 15 valeurs cherchées (dans `out$values`) mais aussi un historique du gradient-conjugué (précisément `out$niterForY` est la liste, de même longueur que `candidateRanges.Grid` dont chaque valeur est, en sortie, le nombre d'itérations nécessitées pour le calcul du terme quadratique, fonction de **y**, de CGEM, et `out$niterForW` est la liste correspondante pour le terme trace qui, essentiellement, nécessite de répéter les calculs précédents avec **w** à la place de **y**) :

```
> out
$values
          [,1]
 [1,]  484.2787
 [2,]  387.1004
 [3,]  338.3396
 [4,]  348.4035
 [5,]  414.1435
 [6,]  538.3725
 [7,]  733.1031
 [8,] 1020.3133
 [9,] 1432.8247
[10,] 2016.0856
[11,] 2829.6583
[12,] 3947.2540
[13,] 5452.9959
[14,] 7430.2546
[15,] 9940.4523


$niterForY
 [1] 10  7 10 16 23 31 38 44 50 52 57 56 57 59 57

$niterForW
 [1] 10  7 11 17 24 32 41 47 53 55 57 59 59 65 58
```

Faisons un `plot` approximatif (i.e. interpolation linéaire des 15 valeurs) de l'équation estimante `CGEM(range,bEV)=bEV`, et regardons l'impact du choix de **w** pour l'approximation MonteCarlo rapide de la trace (i.e. l'usage d'une "randomized -trace"): on répète 6 fois le calcul précédent en ne changeant que la graine qui a généré **w** :

```
########### impact of w    ##############################
set.panel(1,1)
{plot( candidateRanges.Grid , out$values, type="l", col=1, lty=2,log="x")
    abline(h= bEV)
   }

set.seed(345)
for(indexReplcitateOfW in 1:6){
w <- rnorm(n)
out <- CGEMevalOnGrid(y,w,candidateRanges.Grid,nu)
candidateCGEMvalues <- out$values
lines( candidateRanges.Grid , candidateCGEMvalues, type="l", col=1, lty=2)
}
```

On ne présente pas ici les 7 courbes obtenues car elles sont “très” proches.
On va plutôt présenter les 9 courbes obtenues si l’on envisage d’appliquer l’approche CGEM-EV aux 9 champs simulés au-dessus; cela nous donnera donc une idée de la variabilité de cet estimateur:

```
# use of CGEMevalOnGrid #####################
candidateRanges.Grid<- trueRange * 10**seq(-1.1,1.1,,15)
niterForY <- matrix(NA,length(candidateRanges.Grid),9)
niterForW <- matrix(NA,length(candidateRanges.Grid),9)

set.seed(321)
for(indexReplcitate in 1:9){
y <- Z[,indexReplcitate]+rnorm(n)
bEV<-sum(y**2)/n -1.
#
w <- rnorm(n)
w<- c( w)
#
out <- CGEMevalOnGrid(y,w,candidateRanges.Grid,nu)
candidateCGEMvalues <- out$values
niterForY[, indexReplcitate] <-out$niterForY
niterForW[, indexReplcitate] <-out$niterForW
#
if (indexReplcitate==1)
   {plot( candidateRanges.Grid , candidateCGEMvalues/bEV, type="l", col=1,
lty=2,log="x")
   abline(h= 1, lwd=2)
   }
else lines( candidateRanges.Grid , candidateCGEMvalues/bEV, type="l",
col=1, lty=2)
#
}
niterForY
niterForW
##########################################
```

On obtient la Figure 3 (notons que cette fois nous représentons le ratio `CGEM(range,bEV)/bEV` pour ne pas alourdir le graphique puisque `bEV` change avec **y**) que nous commentons plus loin; et on obtient aussi puisque nous avons stocké les 2 listes `out$niterForY` et `out$niterForW` de chaque réplique:

```
> niterForY
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]   10   10    9   10   10   10   10   10   10
 [2,]    7    6    7    6    6    7    7    6    6
 [3,]   10   10   10   10   10   10   10   10   10
 [4,]   16   16   16   15   15   16   16   15   15
 [5,]   23   22   23   22   22   23   23   22   22
 [6,]   31   30   31   30   30   30   29   30   30
 [7,]   38   38   38   37   37   38   37   37   37
```

```
 [8,]   44   44   44   43   43   43   44   42   43
 [9,]   48   48   49   47   46   48   48   46   45
[10,]   53   50   52   50   48   52   50   51   53
[11,]   55   54   57   54   55   58   54   53   52
[12,]   56   55   56   54   55   56   57   57   55
[13,]   57   56   58   56   58   58   58   58   56
[14,]   56   56   56   56   55   56   59   58   56
[15,]   57   57   57   57   55   57   59   58   57
```

Le tableau `niterForW` a des valeurs très similaires à celles de `niterForY` et n'est donc pas donné ici.

Le temps de calcul de ces 9 x 15 évaluations du critère estimant est, sur notre machine actuelle, d'environ 5 secondes, donc (cf remarque au-dessus sur le temps d'évaluation de 15 valeurs du critère "fonction de Vraisemblance") on peut s'attendre à ce que la mise en oeuvre de la maximisation de la vraisemblance soit au moins une trentaine de fois plus lente, en admettant qu'une quinzaine d'évaluations soient nécessaires. Ce gain sera évidemment encore plus important pour des tailles plus grande que 27*27 à condition que le gradient conjugué fonctionne "bien", comme ci-dessus.

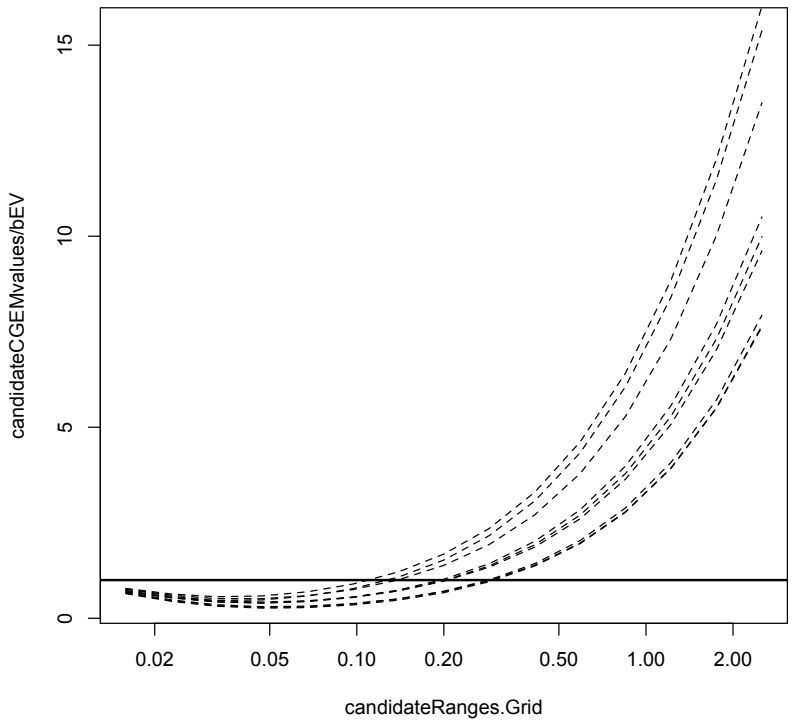

Figure 3: 9 réalisations de l'équation-estimante pour l'estimation de trueRange (=0.2)

Regardons maintenant la qualité de cette méthode d'estimation du `range`. On peut déjà déduire de l'allure des 9 courbes Fig.3 et surtout de la façon dont elles coupent l'axe y=1, que le paramètre range ne pourra être estimée qu'avec une relativement "faible" précision dans de tels contextes d'assez "forte" corrélation. En fait même l'estimateur du maximum de vraisemblance, que nous ne présentons pas ici, aurait eu ici une faible précision : l'expérimentation intensive de [2] (cf sa Table 3, ligne θ^{-1} =0.2) montre que les log en base 10 des estimations ML de θ_0 ont une erreur standard seulement environ 1.12 fois plus petite que les estimations CGEMEV. Ce phénomène général de difficulté d'estimation a été récemment bien étudié et de premiers résultats théoriques ont été établis (cf les références à l'intérieur de [1] et [2]). On sait aussi que le paramètre microergodique est, lui par contre, assez bien estimable. Rappelons que ce paramètre, noté c, est lié à b par la relation :

```
c = b /(range**(2 nu))
```

Pour illustrer ce dernier point, il est utile de représenter chacune des 9 équations-estimantes comme une contrainte sur c en remplaçant l'inconnu `range` par `(bEV/c)**(1/(2 nu))`, où `bEV` est une "constante", chaque fois différente puisque fonction de **y**.

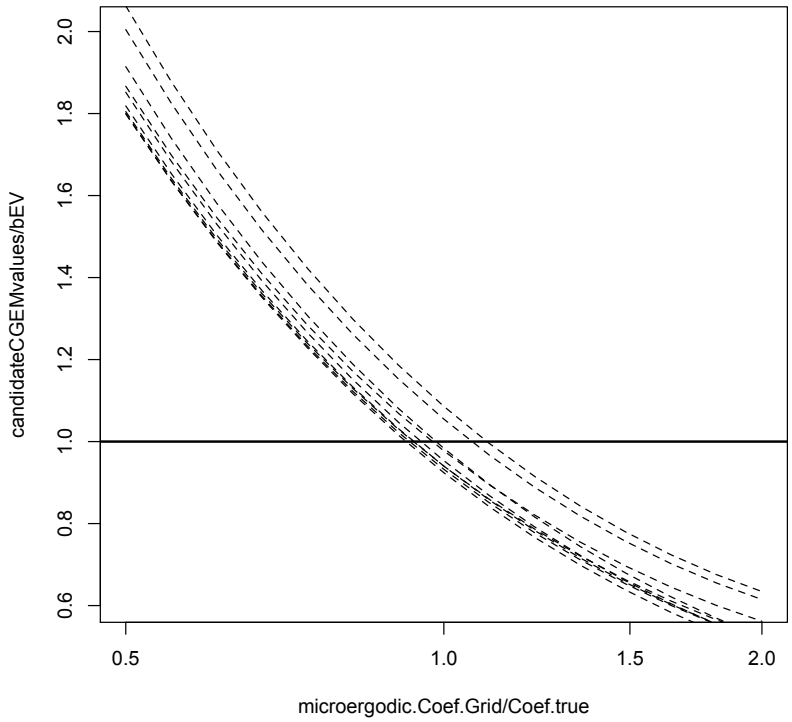

Figure 4: les 9 équations-estimantes fonctions de c/c_0 où c est le coefficient microergodique

Cette Fig.4 est produit par le paragraphe du script commençant par

```
############    another use of CGEMevalOnGrid:
```

Il faut noter que nous n'utilisons pas, en fait, les calculs donnant la Fig.3 car nous faisons maintenant, par rapport à cette Fig.3, un "zoom" sur les abscisses (en échelle logarithmique) assez important: en effet le domaine couvert va d'une valeur seulement 2 fois plus petite que la valeur cherchée, à une valeur seulement 2 fois plus grande, alors qu'il y avait un facteur 10 dans le graphe précédent qui illustrait la variabilité de l'estimateur du `range`. On déduit de cette Fig.4 que CGEM-EV paraît capable d'estimer le coefficient micro-ergodique avec une erreur relative inférieure à 10%. Nous reviendrons sur ce point fin du II.c.

Il est approprié de rediscuter maintenant l'erreur ajouté par l'approximation Monte-Carlo dans chaque courbe CGEM et donc dans l'estimation de c qui en résulte. En effet alors que ce n'était pas le cas pour l'estimation du `range`, cette erreur supplémentaire peut avoir un impact important surtout si on n'utilise pas la version dite "normalisée" de la fonction "randomized-trace". C'est cette version normalisée qui est utilisée dans [2]; et elle est utilisée dans le script produisant la Fig.4, simplement en ajoutant la commande (ligne 186) préalable à l'appel de `CGEMevalOnGrid(y,w,candidateRanges.Grid,nu)`:

```
# normalization of the vector w
w <- w * sqrt( n / sum(w*w) )
#
```

Le fait que la version non-normalisée est à proscrire (elle est de toute manière connue pour être toujours plus variable) est simplement illustré par la simulation réalisée dans le paragraphe du script commençant (ligne 238) par

```
########### impact of w (unnormalized) for the 1rst realiz ###########
```

On y répète 9 fois `CGEMevalOnGrid(y,w,candidateRanges.Grid,nu)` avec **y** toujours égal à la première réalisation du champ de Matern, mais avec un vecteur **w** de différente graine à chaque fois, et sans la normalisation ci-dessus. Ceci produit la Fig. 5 qui montre clairement que la version non-normalisée ajoute une variabilité du même ordre que la variabilité "intrinsèque" (à moins que l'on ne moyenne ces courbes approximantes).

Par des commandes similaires, cad le paragraphe commençant par

```
########### impact of w (normalized) for the 1rst realiz ###########
```

mais avec normalisation, on obtient une figure (non montrée ici) où cette fois les 9 courbes sont quasiment indiscernables: moyenner plusieurs courbes approximantes apparaît clairement ici une perte de temps quand on compare cette erreur à l'erreur intrinsèque de la Fig.4.

Le fait qu'une seule évaluation "randomized-trace" suffit dans ce cadre est confirmé par les expériences décrites dans [2]. Mais rappelons que pour d'autres cas (essentiellement ceux large `Range` et "faible" b, le sens à donner à "faible" dépendant de nu) il s'avère utile de moyenner un petit nombre (n_R=20 est utilisé dans [2] et cela suffit pour tous les cadres) de telles évaluations.

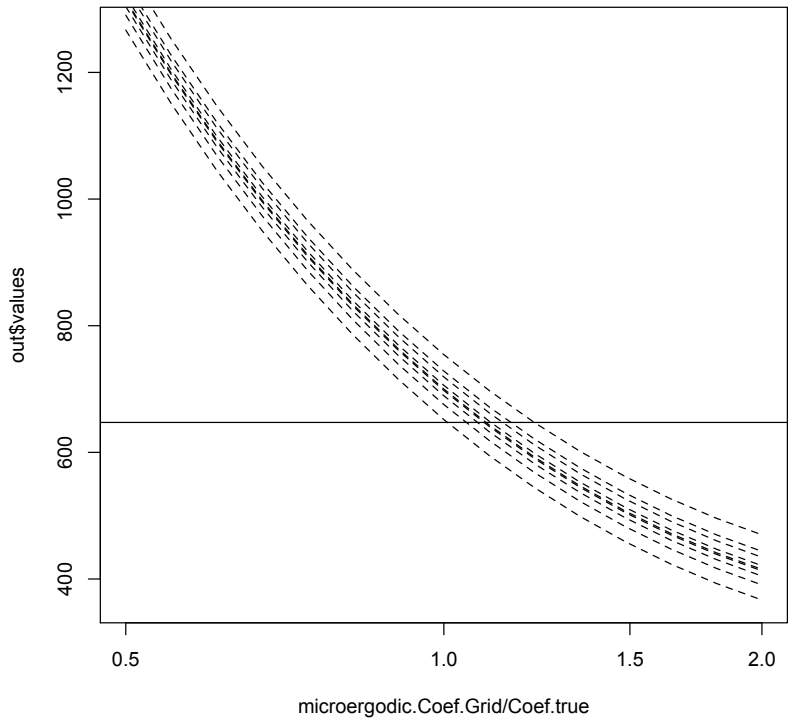

Figure 5: 9 approximations possibles de la 1ère équation-estimante de la Fig.4
si **w** n'est pas normalisé

II.c) La fonction CGEMbisectionLogScaleSearch, et son usage

L'implémentation de CGEM-EV nécessite maintenant le calcul numérique d'une "bonne" racine de l'équation $\mathrm{CGEM}(\hat{b}_{\mathrm{EV}},\theta) = \hat{b}_{\mathrm{EV}}$ en θ (ou `CGEM(range,bEV)=bEV)`. Au vu des graphiques précédents, nous admettons pour l'instant l'unicité de la racine sur le domaine, en

`range`, [0.01, 30.] qui semble suffisamment grand. La classique méthode de la “bissection” a alors une propriété de convergence garantie vers cette racine. Nous y faisons appel, excepté que nous avons implémenté une échelle logarithmique pour la dichotomie et comme dans `CGEMevalOnGrid` nous exploitons les calculs aux points voisins pour accélérer la convergence du gradient conjugué. Le fichier "`CGEMbisectionLogScaleSearch.R`" contenant cette fonction doit être préalablement placé dans le répertoire de travail. Nous allons tout d’abord considérer l’application de cette fonction à la première des 9 réalisations précédentes (pour rappel nous avions $\hat{b}_{\mathrm{EV}}$ =645.4668). Outre les bornes inf et sup pour le domaine de recherche de la racine, il y a un dernier paramètre supplémentaire: la tolérance, ici mise à la valeur 0.0001, ce qui signifiera que la racine est trouvée avec une précision relative de 0.0001.

```
#### use of CGEMbisectionLogScaleSearch #########
w <- rnorm(n)
w<- c( w)
# normalization of the vector w
w <- w * sqrt( n / sum(w*w) )

#ds les 2 lignes suivantes, on "source" la fonction
CGEMbisectionLogScaleSearch,
# on la charge et  on l'execute
source("CGEMbisectionLogScaleSearch.R")
out <- CGEMbisectionLogScaleSearch(y,w,nu,0.01,30.,0.0001)
```

On obtient:

```
> out
$root
[1] 0.1220564

$niterCGiterationsHistory
      [,1] [,2]
 [1,]   86   87
 [2,]   36   37
 [3,]   47   51
 [4,]   40   44
 [5,]   31   31
 [6,]   29   30
 [7,]   26   29
 [8,]   24   26
 [9,]   21   23
[10,]   17   19
[11,]   15   16
[12,]   12   12
[13,]    6   11
[14,]    2    2
[15,]    2    8
[16,]    2    2
```

```
[17,]    6    2
[18,]    0    0
[19,]    0    0
[20,]    0    0
```

Les 2 colonnes ci-dessus contiennent le nombre d'itérations nécessaires au gradient-conjugué pour chacune des 17 évaluations nécessaires à la bissection (ce chiffre 17 dépends de la tolérance choisie).

On observe que le nombre d'itérations nécessaires au gradient-conjugué devient de plus en plus petit quand on s'approche de la convergence de la recherche-bissection (alors qu'environ 2 fois 90 itérations ont été nécessaires au gradient-conjugué lors de la première évaluation de la fonction CGEM, 8 itérations ont suffit au gradient-conjugué lors de la dernière évaluation, cad la 17ème évaluation): l'idée d'utiliser les solutions des systèmes linéaires voisins semble donc efficace en coût calcul.

De la solution root précédente, on déduit l'estimateur du coefficient microergodique, immédiatement par les lignes suivantes:

```
rangeHatCGEMEV <- out$root
cHatCGEMEV<-  bEV*(1/rangeHatCGEMEV)**(2*nu)
cHatCGEMEV
```

qui donne:

```
[1] 5288.266
```

On observe que l'estimation du paramètre microergodique cTrue (=5000) est bien meilleure que celle du range, ou que celle de `bTrue` (=1000): les simulations assez extensives présentées dans [2] (ligne "range=0.2" de la Table 3 de cet article) montrent que c'est bien le cas et que l'on a bien une précision relative sur cTrue presqu'égale à √(2/n) dans ce cadre.

II.d) Le même cas que le précédent , mais avec bTrue=10

On change seulement l'amplitude des 9 champs simulés :

```
bTrueOld<-bTrue
bTrue<- 10.0
Z<- Z*sqrt(bTrue/bTrueOld)
```

On applique à nouveau (cf le script) `CGEMbisectionLogScaleSearch(y,w,nu, 0.01,30.,0.0001),` mais avec **y** construit à partir de la 1ère de ces 9 “nouvelles” images simulées. Pour ce **y** on a:

```
> bEV
[1] 6.186461
```

On obtient en sortie de `CGEMbisectionLogScaleSearch.R`:

```
> out
$root
[1] 0.1340709

$niterCGiterationsHistory
      [,1] [,2]
 [1,]   33   33
 [2,]   25   26
 [3,]   30   31
 [4,]   28   26
 [5,]   26   26
 [6,]   23   24
 [7,]   20   20
 [8,]   17   18
 [9,]   15   15
[10,]   12   13
[11,]   10   10
[12,]    8    9
[13,]    4    4
[14,]    2    8
[15,]    2    2
[16,]    2    2
[17,]    4    2
[18,]    0    0
[19,]    0    0
[20,]    0    0
```

En comparant ces nombres d’itérations nécessaires au gradient-conjugué avec le cas précédent (où `bTrue=1000.)`, on voit ici que la présence d’un bruit d’observation plus important améliore la vitesse du gradient-conjugué d’un facteur 2 ou 3.
Rappelons que la présence d’un bruit impose d’ajouter une diagonale constante (=1/b) à la matrice d’autocorrélation, avant son inversion, et que la vitesse du gradient-conjugué augmente en théorie avec le conditionnement. Donc l’apparition d’un tel effet, croissant avec 1/b, était prévisible.

Les simulations pour ce cas sont aussi analysés dans [2] (ligne “range=0.2” de la Table 6 de cet article).

II.e) Le même cas que le précédent mais avec trueRange=1.0

Cette fois après avoir exécuté

```
trueRange <-  1.
```

les lignes appropriées du script (identiques à celles donnant le plot du II.a) donnent le tracé de la nouvelle autocorrélation:

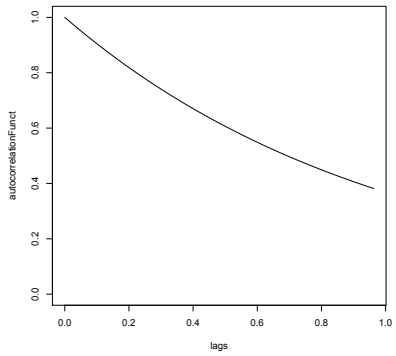

Figure 6: fonction d'autocorrélation avec `trueRange=1.0`

Et il faut recalculer la matrice de correlation et sa décomposition de cholesky pour générer des simulations exactes par le même algorithme: leur représentation est donnée Fig.7.

Commentons ces images en les comparant aux précédentes obtenues avec `trueRange=.2`: on voit "pas mal" de similarités mais à l'échelle spatiale près. Les zones "assez" uniformes sont typiquement plus larges. Un facteur 5 d'agrandissement de ces "zones" était prévisible. Rappelons qu'en fait, plus précisément, si une telle simulation avec `trueRange=1.` avait été étendue sur un domaine 5 fois plus grand, il aurait suffi ensuite de sous-échantilloner, tous les 5 pixels, pour obtenir une simulation du cas `trueRange=.2` sur le domaine initial.

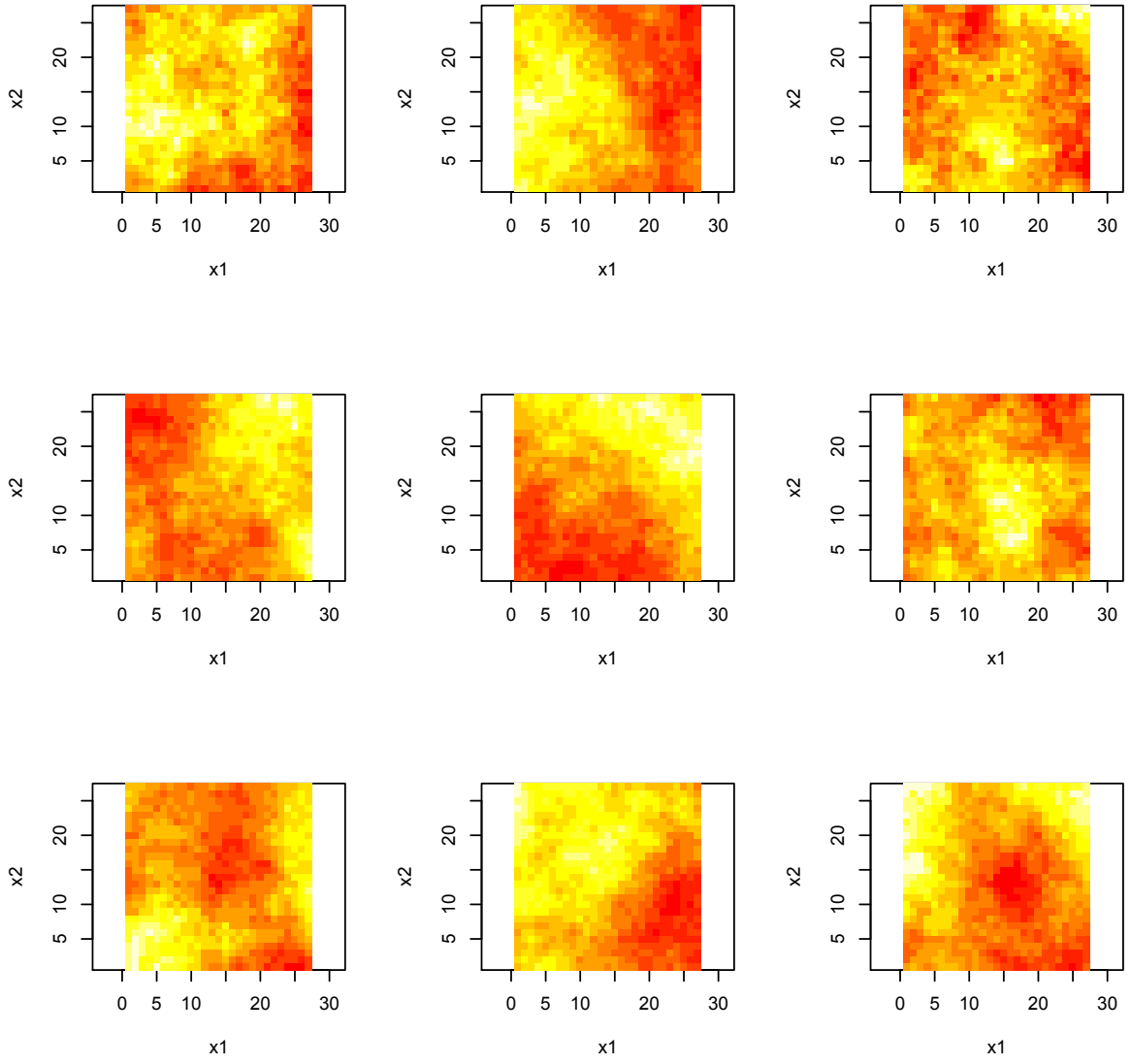

Figure 7: simulations exactes avec `trueRange=1`.

On va maintenant analyser comme dans le cas “moins” corrélé, la 1ère réalisation, par les même lignes de R. Cela donne en sortie:

```
> bEV
[1] 3.025114
```

puis après exécution de `CGEMbisectionLogScaleSearch(y,w,nu, 0.01,30.,0.0001)`:

```
> out
$root
[1] 0.450488

$niterCGiterationsHistory
      [,1] [,2]
 [1,]   23   24
 [2,]   21   21
 [3,]   24   25
 [4,]   22   22
 [5,]   20   20
 [6,]   17   17
 [7,]   16   16
 [8,]   14   15
 [9,]   11   12
[10,]    9   10
[11,]    8   10
[12,]    4    6
[13,]    6    3
[14,]    3    6
[15,]    2    3
[16,]    2    2
[17,]    2    2
[18,]    0    0
[19,]    0    0
[20,]    0    0
```

On observe donc à nouveau un bon comportement du gradient-conjugué.

II.f) Le même cas que le précédent mais avec bTrue=1000

Cette fois, après réassignation de bEV qui vaut `321.1538` et après l'exécution de `CGEMbisectionLogScaleSearch(y,w,nu,0.01,30.,0.0001)`, on obtient:

```
> out$root
[1] 0.3131555
```

Cette estimation du trueRange n'est donc guère amélioré, mais en combinant comme au-dessus :

```
rangeHatCGEMEV <- out$root
cHatCGEMEV<-  bEV*(1/rangeHatCGEMEV)**(2*nu)
cHatCGEMEV
```

on obtient immédiatement:

```
> cHatCGEMEV
[1] 1025.541
```

qui est une bien meilleure estimation que celle du range; ce qui incite à l'expérience suivante.

II.g) Une simulation "extensive" dans le cas précédent

Nous proposons de répéter 500 fois la simulation du type précédent: c'est le premier paragraphe de la partie `''extensive''` simulation dans le script `"ScriptForDemo.R"`.

Les 500 réalisations de l'estimateur CGEM-EV du coefficient microergodique sont stockées dans le vecteur `cHatCGEMEVLogScaleSearch`; on obtient (après environ 300 secondes d'attente sur un intel Core 2 Duo 3 GHz):

```
> ctrue<-bTrue*(1/trueRange)**(2*nu)
> summary(cHatCGEMEVLogScaleSearch/ctrue)
       V1
 Min.   :0.8097
 1st Qu.:0.9639
 Median :1.0019
 Mean   :1.0048
 3rd Qu.:1.0456
 Max.   :1.2265
> sd(cHatCGEMEVLogScaleSearch/ctrue)
[1] 0.0590538
```

Cette mean et cette standard deviation sont tout à fait cohérentes avec la ligne "range=1." de la Table 3 de [2].

L'information nouvelle que fournit ce code R concerne l'efficacité en temps de calcul de la combinaison "gradient conjugué et FFT". La dernière commande du script permet en fait de lister l'historique du nombre d'itération demandées par chaque appel à la méthode du gradient conjugué, et cela pour chacune des 500 répliques. On constate pour ce cas que ce nombre reste toujours inférieur à 84 ce qui est tout à fait rassurant.

Bien sûr on peut s'attendre à des cas où le mauvais conditionnement des systèmes linéaires à résoudre nécessitera de remplacer cette méthode par une version avec préconditionnement. Une extension du code permettant cette possibilité serait donc très utile.

Remerciements

Cette implémentation en R a bénéficié des conseils judicieux de R. Drouilhet, expert R au LJK-imag. L'auteur tient à le remercier pour sa réactivité sans faille.

Références

[0] R Development Core Team (2007). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. www.r-project.org/foundation/

[1] D.A. Girard. *Asymptotic near-efficiency of a "Gibbs-energy" estimating-function approach for fitting Matern covariance models to a dense (noisy) series*. http:// hal.archives-ouvertes.fr/hal-00121174/en/, 2009.

[2] D.A. Girard. *A fast, near efficient, randomized-trace based method for fitting continuous stationary correlation models to large noisy data sets in the case of a single range-parameter.* http://hal.archives-ouvertes.fr/hal-00515832/en/, 2010

[3] R. Furrer, D. Nychka, and S. Sain. *fields: Tools for spatial data,* 2011. R package version 6.6.2.. http://cran.r-project.org/package=fields