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

Didier A. Girard

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 résoud 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 le 3ème paramètre est,

comme les 2 premiers, réduit à un scalaire, 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 :

$$\text{"corrélation entre la valeur en } \mathbf{s} \text{ et celle en } \mathbf{t}\text{"} = \exp(-\theta\, d)$$

où d est la distance euclidienne entre **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 $\nu > 0$ ($\nu=1/2$ 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 bien connu que, 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 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 3ième restriction (si on se limite à des valeurs de ν raisonables). La preuve de son efficacité pour pouvoir éliminer la 2ième 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 pourra être prise en compte par notre méthode.

Le cadre de ce travail est celui où les variances du bruit sont connues à un facteur près (voir cependant **Rem. 1**). C'est toujours le cas quand les données sont des "pooled gridded data" obtenue à partir de données initialement irrégulièrement espacées mais à bruit homogène de niveau connu. Soit donc c la matrice diagonale, normalisée par $\mathrm{Trace}(D^2)/n = 1$, telle que le vecteur des observations $\mathbf{y}$, de taille n, satisfait

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

où N(0,V) est la classique loi Gaussienne de matrice de covariance V, et où pour simplifier la présentation ici, la variance des erreurs normalisées ε est aussi supposée connue (égale à 1 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 $\mathbf{z}$.
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 contraint le paramètre de corrélation à une valeur 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

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

Rem. 1: Notons que dans le cas d'un niveau de bruit inconnu σ_ε^2 cette équation CCLE se modifie simplement: $\mathbf{y}$ doit être normalisé par $\mathbf{y}\Big/\left(\langle \mathbf{y}, (I - A_{b,\theta})\mathbf{y}\rangle / n\right)^{1/2}$ le terme normalisant étant en fait l'estimateur de σ_ε à la convergence (cad avec la solution de la nouvelle équation en b substitué à b).

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 $\hat{\theta}_{\text{CGEM-EV}}$ (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 pour le champ $\mathbf{z}$ (qui désigne $\langle \mathbf{z}, C_\theta^{-1}\mathbf{z}\rangle$) et de sa moyenne conditionnelle étant expliquée en [1].

En utilisant, par exemple, une classique bissection, la résolution de l'équation en 2) (dite équation-estimante) nécessite typiquement en pratique le calcul de la randomized-trace version de 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 ne pourrait 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]. 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 elle commente l'exécution de ce script.

Les fonctions R créées peuvent fonctionner en théorie pour ν quelconque, mais nous ne les avons testées pour l'instant que pour ν=1/2. Par contre, elles 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: 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

$$\mathrm{CGEM}(b,\theta) = b\big(\mathrm{CCLE}(b,\theta) + n\big)\big/n,$$

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

II. Implémentation R et une “démo”

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

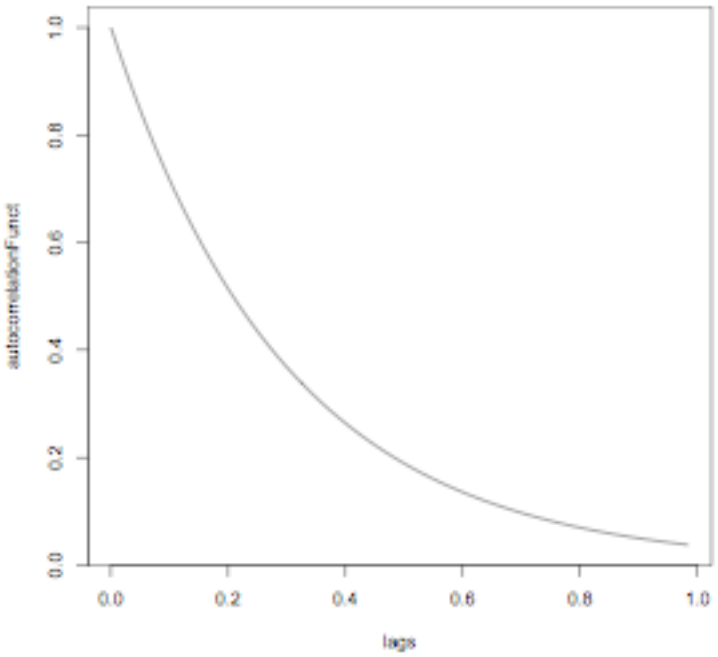
On excécute dans la “console R”, les commandes suivantes (premier paragraphe du script `“ScriptForDemo.R”)`, en commençant bien sûr, par charger le package fields (Notons que nous choisissons dans ce script une taille relativement petite pour la grille des observations ici: 27 x 27; mais des tailles beaucoup plus grande pourront être prises en compte dès que l’étape “génération des données” (ici en n^3) sera remplacée par une méthode plus rapide):

```
library(fields)

#simulation of a Gaussian Matern field on a grid n1*n1
n1<-27
#abscissae of discrete observations :
x<- (1./n1)*matrix( c(rep(1: n1, n1),rep(1: n1,each= n1)),
ncol=2)
n<- nrow(x)
# 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

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

cela a initialisé le modéle des champs que l’on va simuler et donne la représentation graphique suivante:



Ensuite, pour faire appel à la méthode de simulation classique, nous devons d'abord effctuer la décompostion de Cholesky de $b_0 C_{\theta_0}$:

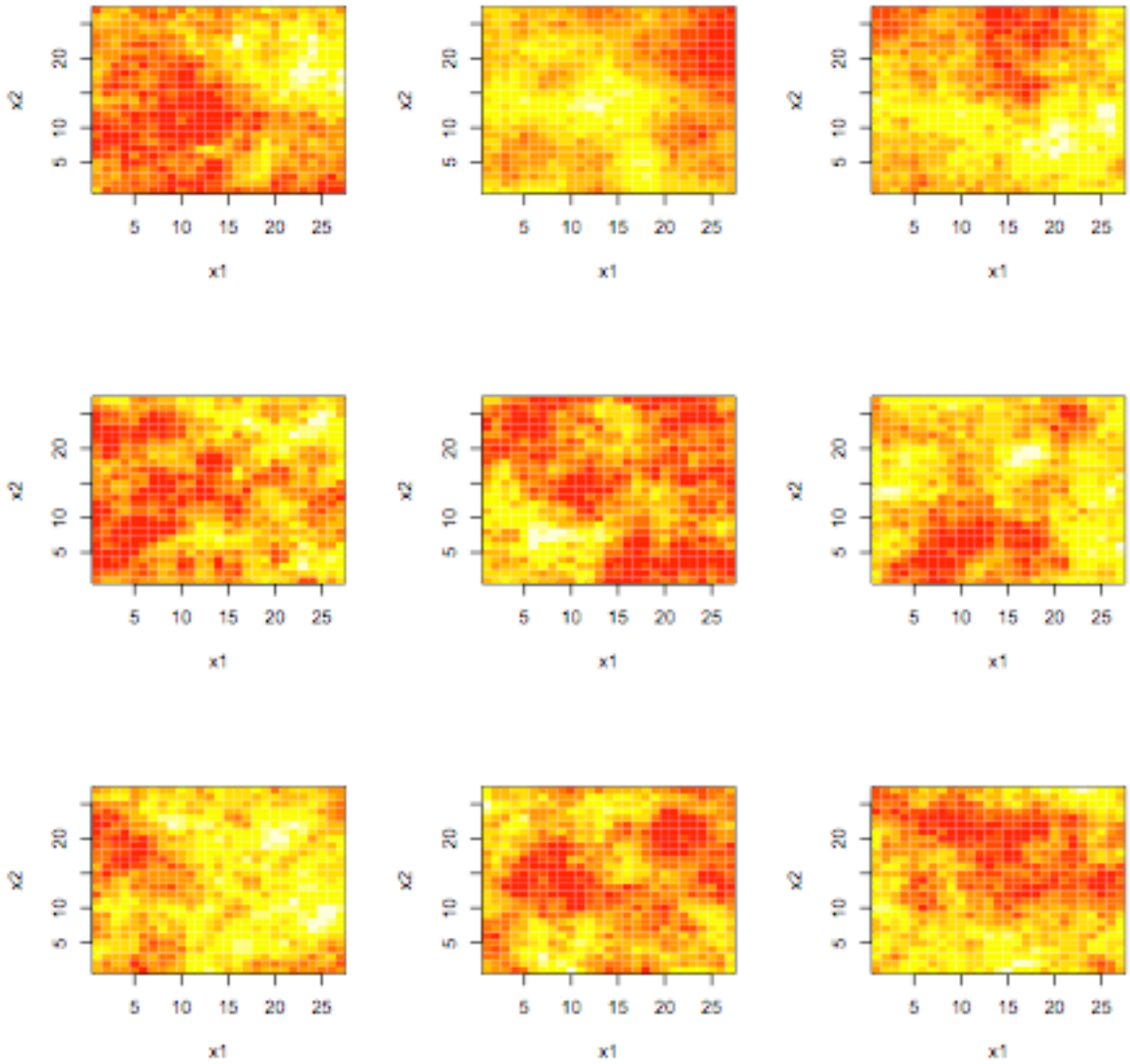
```
ut<-system.time(Cov.mat<-  bTrue* Matern( rdist(x,x),
smoothness=nu, range= trueRange))
ut
   user  system elapsed
  0.760   0.023   0.783

ut<-system.time(A<- chol( Cov.mat))
ut
   user  system elapsed
  0.082   0.000   0.106
```

Nous avons ajouté l'information donnée par la fonction `system.time` pour avoir 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 pour à fixé : ce sera donc environ 15 secondes puisque le determinant est immédiat après la décompostion de Cholesky.

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

```
######### simulation of 9 realizations  #################
set.seed(789)
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))}
```



Créons maintenant un vecteur simulé d'observations $\mathbf{y}$, et calculons bEV

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

```
> bEV
[1] 952.256
```

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 évalue CGEM(range) sur toute une grille donnée et non pas un seul point car il s'avèrera très utile de faire démarrer l'itération du gradient solution 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) est donc

source("CGEMevalOnGrid.R") # Source la fonction CGEMevalOnGrid

```
# 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
$values
           [,1]
 [1,]   662.2320
 [2,]   486.6372
 [3,]   381.5612
 [4,]   358.1515
 [5,]   403.9172
 [6,]   514.4640
 [7,]   697.7908
```

```
 [8,]   974.2708
 [9,]  1377.5502
[10,]  1957.2356
[11,]  2783.5958
[12,]  3953.3940
[13,]  5596.7599
[14,]  7883.0768
[15,] 11023.6395

$niterForY
 [1] 10  6 10 15 22 30 37 43 49 53 56 55 56 56 56

$niterForW
 [1] 10  7 11 17 24 32 41 47 52 55 59 65 63 60 62

>
```

Faisons un plot de cette équation estimante, 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

```
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 la figure obtenue car les 6 courbes sont quasiment indiscernables.

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)
```

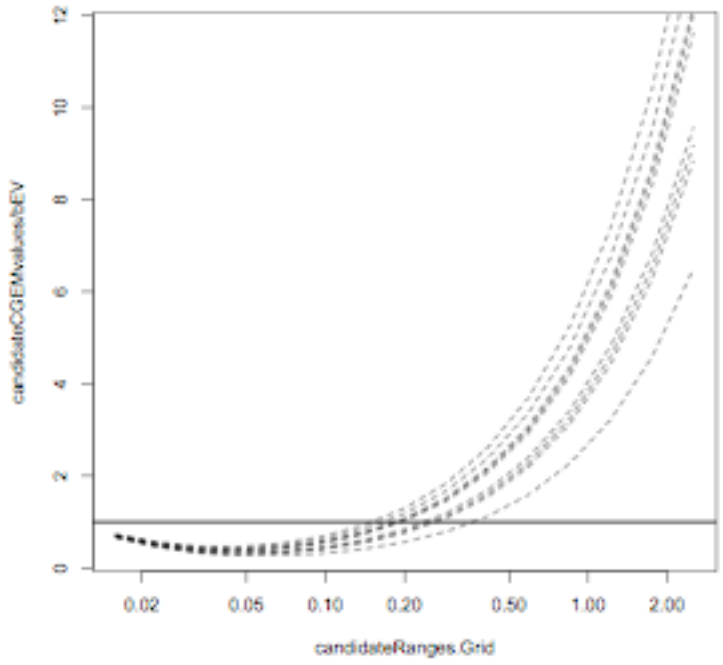
```
set.seed(321)
for(indexReplcitate in 1:9){
y <- Z[,indexReplcitate]+rnorm(n)
bEV<-sum(y**2)/n -1.
#
w <- rnorm(n)
w<- c( w)
evEqualTo1.w <- c( w)*sqrt((n/sum( w *w)))
#
out <- CGEMevalOnGrid(y,evEqualTo1.w,candidateRanges.Grid,nu)
candidateCGEMvalues <- out$values
niterForY[, indexReplcitate] <-out$niterForY
niterForW[, indexReplcitate] <-out$niterForW
#
if (indexReplcitate==1)
   {plot( candidateRanges.Grid , candidateCGEMvalues/bEV,
type="l", col=1, lty=2,log="x")
   abline(h= bEV, lwd=2)
   }
else lines( candidateRanges.Grid , candidateCGEMvalues/bEV,
type="l", col=1, lty=2)
#
}
niterForY
niterForW
##########################################
```

```
> niterForY
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    10   10   10   10   10   10   10   10   10
[2,]     6    6    6    6    6    6    6    6    6
[3,]    10   10   10   10   10   10   10   10   10
[4,]    15   15   15   16   16   16   15   16   15
[5,]    22   22   22   23   22   22   22   22   22
[6,]    30   28   30   30   30   30   30   31   30
[7,]    38   37   37   37   37   38   37   37   37
[8,]    44   44   42   44   44   44   43   44   43
[9,]    49   49   49   49   49   50   49   50   47
[10,]   52   49   53   53   49   53   51   49   50
[11,]   56   53   55   55   55   56   53   54   55
[12,]   55   55   55   56   56   56   55   56   56
[13,]   56   56   56   57   56   56   56   58   58
[14,]   56   56   58   56   56   56   56   56   56
[15,]   55   55   58   57   57   56   56   57   59
```

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'évaluation suffise. 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.

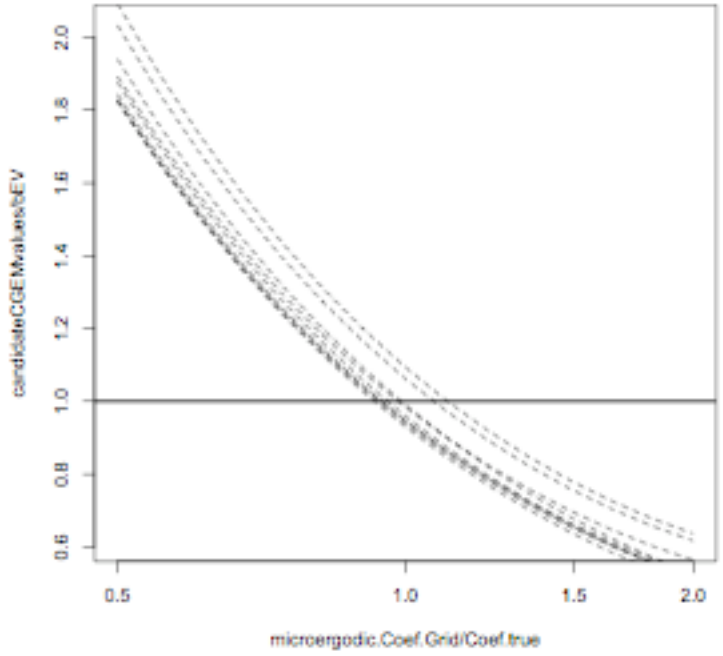
Le tableau `niterForW` a des valeurs très similaires et n'est donc pas donné ici.



On observe sur ces simulations 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. Ce phénomème général a été récemment bien étudié et de premiers résultats théoriques ont été établis (cf [1] et les références à l'intérieur). On sait aussi que le paramètre microergodique est, lui, assez bien estimable. Rappelons que ce paramètre, noté `c`, est lié à `b` par la relation :

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

Pour commenter ce dernier point, il est utile de représenter chacune des 6 equations-estimantes comme une contrainte sur `c` en remplaçant l'inconnu `range` par `(bEV / c )**(1/(2 nu))`, où `bEV` est une "constante", chaque fois diférente puisque fonction de **y**.



Noter sur ce graphe qu'il y a un zoom (en échelle logarithmique) par rapport au graphe précédent, zoom autour de la valeur cherchée: le domaine couvert va en effet d'une valeur 2 fois plus petite à une valeur 2 fois plus grande, alors qu'il y avait un facteur 10 dans le graphe précédent.

II.c) La fonction CGEMbisectionLogScaleSearch, et son usage

```
####################################
set.seed(678)
indexReplcitate <-9
y <- Z[,indexReplcitate]+rnorm(n)
bEV<-sum(y**2)/n -1.
#

> bEV
[1] 1303.433

#### use of CGEMbisectionLogScaleSearch #########
w <- rnorm(n)
w<- c( w)
# evEqualTo1.w <- c( 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)


> out
$root
[1] 0.2683632

$niterCGiterationsHistory
      [,1] [,2]
 [1,]   86   93
 [2,]   34   37
 [3,]   44   51
 [4,]   49   54
 [5,]   43   51
 [6,]   41   46
 [7,]   39   41
 [8,]   32   40
```

```
 [9,]   25   32
[10,]   22   28
[11,]   17   22
[12,]   10   14
[13,]   12   14
[14,]    3    9
[15,]    3   12
[16,]    3    6
[17,]    2    2
[18,]    0    0
[19,]    0    0
[20,]    0    0
```

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: l'idée d'utiliser les solutions des systèmes linéaires voisins est donc efficace en coût calcul.

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

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

```
[1] 4856.974
```

On observe que l'estimation du paramètre microergodique (cTrue=5000) est bien meilleure que celle du range, ou celle de `bTrue` (=1000): les simulations assez extensives présentées dans [2] montrent que c'est bien le cas et que l'on a bien une précision relative d'environ $\sqrt{(2/n)}$.

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

On change 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** consruit à partir de la 9ème de ces 9 “nouvelles” images simulées. On obtient

```
> out
$root
[1] 0.1783508

$niterCGiterationsHistory
      [,1] [,2]
 [1,]   40   41
 [2,]   28   31
 [3,]   35   40
 [4,]   30   32
 [5,]   28   31
 [6,]   26   29
 [7,]   24   26
 [8,]   20   24
 [9,]   19   21
[10,]   14   17
[11,]   13   15
[12,]    8   13
[13,]    4    8
[14,]    2    2
[15,]    4    9
[16,]    2    2
[17,]    2    2
[18,]    0    0
[19,]    0    0
[20,]    0    0
```

En comparant ces nombres d’itérations nécessaaires au gradient-conjugué avec le cas précédent (où `bTrue=1000.`), on voit ici que la présence d’un bruit d’observation plus important amméliore la vitesse du gradient-conjugué d’une manière notable.
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 conditionnment. Donc l’apparition d’un tel effet, croissant avec 1/b, était prévisible.

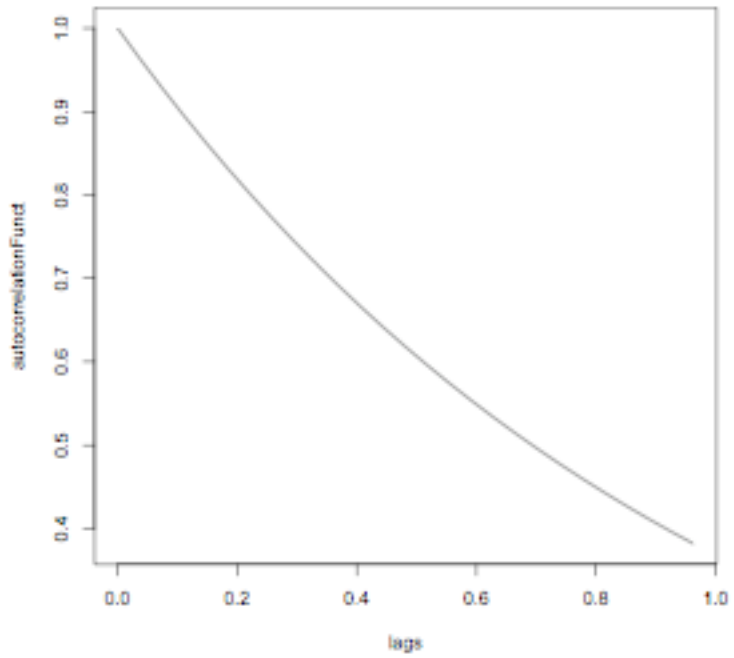
Les simulations pour ce cas sont aussi analysés dans [2] (ligne “range=0.2” de la Table 7 de cet article).

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

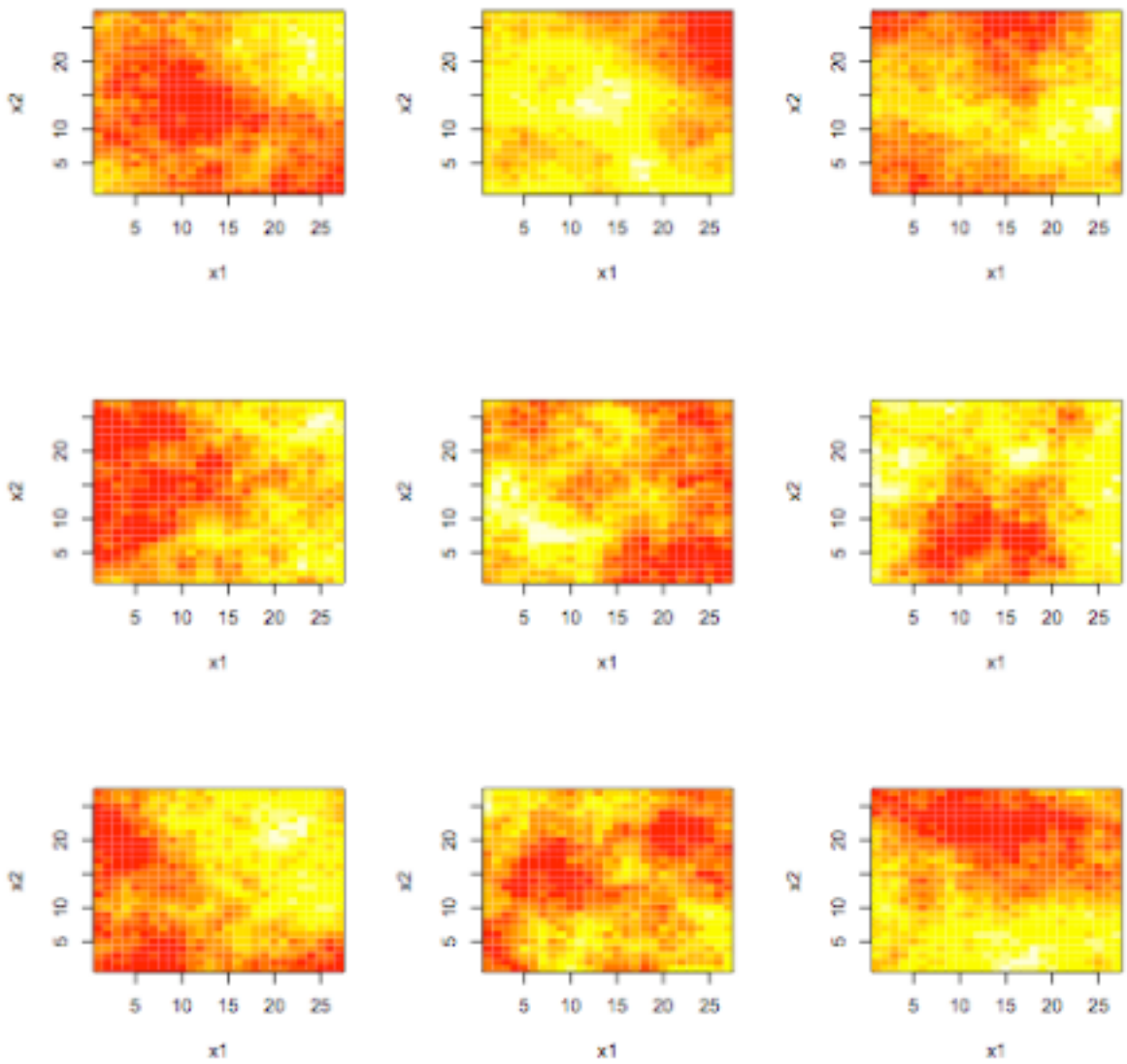
Cette fois après avoir exécuté

```
trueRange <-  1.
```

les lignes appropriées du script donne le tracé de la nouvelle autocorrélation:



Et il faut recalculer la matrice de correlation et sa décomposition de cholesky pour générer des simulations exactes par le même algorithme:



simulations exactes avec `trueRange=1.`

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” unifomes sont typiquement plus larges. Un facteur 5 d’agrandissement de ces “zones” était prévisible. Notons en effet que 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 intial.

On va maintenant analyser comme dans le cas “moins” corrélé, la 9ème réalisation, par les même lignes de R. Cela donne en sortie:

```
> bEV
[1] 13.3387
```

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

```
> out
$root
[1] 0.4206733

$niterCGiterationsHistory
      [,1] [,2]
 [1,]   38   41
 [2,]   27   31
 [3,]   34   40
 [4,]   31   37
 [5,]   27   33
 [6,]   25   30
 [7,]   23   27
 [8,]   20   24
 [9,]   16   21
[10,]   14   17
[11,]    9   12
[12,]    3   12
[13,]   10    5
[14,]    3    3
[15,]    3    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 `1322.002` (ce n'est pas exactement 100 fois le bEV précédent car le bruit d'observation a été généré avec une autre graine dans le script `"ScriptForDemo.R"` et exécution de `CGEMbisectionLogScaleSearch(y,w,nu,0.01,30.,0.0001)`, on obtient:

```
> out$root
[1] 1.316079
```

Cette estimation du trueRange n'est donc guère amélioré, mais on obtient

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

```
> cHatCGEMEV
[1] 1004.5
```

qui est une bien meilleure estimation; 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 d`ans 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.   :0.8292
 1st Qu.:0.9668
 Median :1.0026
 Mean   :1.0053
 3rd Qu.:1.0422
 Max.   :1.1821
```

```
> sd(cHatCGEMEVLogScaleSearch/ctrue)
[1] 0.05866452
```

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 90 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 sa 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

[1] D.A. Girard *Asymptotic near-efficiency of a "Gibbs-energy" estimating-function approach for fitting Matern covariance models to a dense (noisy) series*.
URL: 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*.
URL: 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