Statistiques et Traitement de Données : interpolation spatiale

Josselin Fatah-Roux

Université de Toulon
Master 2 Physique Science de l'Ingénieur
Spécialité Physique Surveillance de l'Environnement
vufic@outlook.com

22 décembre 2017

Résumé

Un champ de température de surface (SST pour Sea Surface Temperature) extrait d'un modèle est donné dans la zone du Golfe du Lion. Dans l'hypothèse où seules N mesures ponctuelles ou échantillons, sur une grille irrégulière, sont disponibles, on peut alors reconstruire un champ de température $\hat{T}(\vec{r})$ sur une grille régulière $\vec{r} = (x, y)$ par interpolation linéaire :

$$\hat{T}(\vec{r}) = \sum_{i=1}^{N} w_i(\vec{r}) T(\vec{r_i})$$

où $T(\vec{r_i})$ est l'ième mesure disponible et $w_i(\vec{r})$ le poids correspondant.

Après avoir définie, en fonction de la localisation des échantillons, la zone sur laquelle interpolées, on utilisera, pour le calcul des poids, les méthodes suivantes :

-Pondération par inverse du carrée de la distance et par gaussienne
-Krigeage

Mots-clés :

Interpolation spatiale ; Température de surface SST ; Pondération par inverse du carré de la distance ; Pondération par gaussienne ; Krigeage ; Golfe du Lion

1 Introduction

L'interpolation spatiale permet grâce à des points dont la valeur est connue d'estimer celles qui sont inconnues en d'autres points plus ou moins proche. Dit différemment, c'est la traduction d'une information disponible pour un nombre de lieux limités vers une information disponible pour tout l'espace.

Il existe plusieurs méthodes d'interpolation dont seulement trois seront présenté dans ce rapport, à savoir : pondération par inverse du carrée de la distance, pondération par gaussienne et Krigeage.

2 Matériels et Méthodes

2.1 Pondération par inverse du carrée de la distance

À notre disposition un fichier "load_GLAZUR.m" qui permet de générer des stations réparties aléatoirement à partir des données du modèle GLAZUR64. Ces stations, placées sur le champ de température de surface simulé, récupère la température se trouvant en leur point. L'objectif est de réaliser avec ces données, par interpolation spatiale, un autre champ de température de surface plus ou moins équivalent à celui du modèle.

La méthode utilisée dans cette partie est celle de la pondération par inverse du carré de la distance. Elle consiste à ce que l'influence d'un point dont la valeur est connue par rapport au point créer décline en fonction de la distance. Cette influence est régit par le coefficient de pondération $w_i(\vec{r})$ qui diminue lorsque la distance est grande et inversement. Mathématiquement il s'exprime de la manière suivante :

$$w_i(\vec{r}) = \frac{w_i^{'}(\vec{r})}{\sum_{i=1}^{N} w_i^{'}(\vec{r})} \text{ avec } \boxed{w_i^{'}(\vec{r}) = \frac{1}{\mid \vec{r} - \vec{r_i} \mid} = \frac{1}{\text{Distance}^2}}$$

Ainsi il est question de calculer pour chaque station les distances qui les sépare de chaque point de la grille. Or il est à noter une chose très importante qui est que ces variables sont en coordonnées GPS alors que les expressions mathématiques précédentes s'adapte pour des coordonnées cartésiennes. Par conséquent le passage de l'un à l'autre se fait par les formules ci-après :

$$\Delta Lat = (Y_{\text{station}} - Y_{\text{point grille}}) * \frac{\pi}{180}$$

$$\Delta Lon = (X_{\text{station}} - X_{\text{point grille}}) * \frac{\pi}{180}$$

$$\Delta Latmoy = \frac{(Y_{\text{station}} + Y_{\text{point grille}})}{2} * \frac{\pi}{180}$$

$$\boxed{D^2 = R^2(\Delta Lat^2 + (cos(\Delta Latmoy * \Delta Lon^2)}$$

avec R le rayon de la terre soit environ 6371.10^3m (6321km)

Après toutes ces explications il est enfin possible de concevoir le code sur MATLAB. Toutefois il est primordial d'avoir la correspondance entre variable théorique et expérimentale :

Théorique	X_{station}	$X_{\text{point grille}}$	Y_{station}	$Y_{\text{point grille}}$	$T(\vec{r_i})$	ΔLat	ΔLon	$\Delta Latmoy$	$Distance^2$	$\hat{T}(\vec{r})$
Expérimentale	X_st	X	Y_st	Y	T_st	$dlat$	$dlon$	$latm$	$D2$	T

La première étape est l'intégration des variables en exécutant le fichier désigné au paragraphe 1, puis l'initialisation des paramètres :

```
%Execution du script load_GLAZUR :
run load_GLAZUR
```

```
%Initialisation des parametres :
R=6371000;
rad=pi/180;
T=zeros(length(X(1,:)),length(Y(1,:)));
T_st=T_st(~isnan(T_st));
n_st=length(T_st);
```

La ligne T_st=T_st(∼isnan(T_st)) est indispensable car elle permet de supprimer toutes les températures de T_st qui sont des NaN et qui sont produites lorsqu'une ou plusieurs stations sont en dehors du champs de température de surface du modèle (sur les côtes et non sur l'eau). De ce fait il est essentiel de récupérer pour une des boucles qui va suivre le nombre de station restante. D'où : n_st=length(T_st).

La deuxième et dernière étape est la construction de trois boucles effectuant le calcul des distances pour chaque station à chaque point de la grille impliquant l'obtention de w' puis w et donc de T :

```
%Pour chaque point de la grille :
for i=1:length(T(:,1))
for j=1:length(T(1,:))

%Variable intermediaire :
T_int=zeros(n_st,1);

%Calcul des distances stations-points grille :
for k=1:n_st
dlat=(Y_st(k)-Y(j))*rad;
dlon=(X_st(k)-X(i))*rad;
latm=((Y_st(k)+Y(j))/2)*rad;
D2(k)=R^2*(dlat^2+(cos(latm)*(dlon))^2);

%Si point de la grille modele sur cotes alors T_int=
 ↪ NaN :
if isnan(glazur_T(i,j))==1
T_int(k)=NaN;
%Sinon si D2 different de 0 alors T_int(k)=T_st(k)/D2(
 ↪ k) sinon T_int(k) reste a 0 pour ne pas faire de
 ↪ division par 0 :
else
if D2(k)~=0
T_int(k)=T_st(k)/D2(k);
end
end

end

%Calcul SST :
w_prime=1./D2;
w=sum(w_prime);
T(i,j)=sum(T_int)./w;

end
end
```

```
figure(2)
pcolor(T'),colorbar,shading flat
title('Champ de temperature de surface (ponderation
  ↪ par inverse du carree de la distance)')
```

NB : dans ce programme on passe par une variable intermédiaire "T_int" qui est la partie suivante du calcul :

$$\hat{T}(\vec{r}) = \sum_{i=1}^{N} w_i(\vec{r}) T(\vec{r_i}) \Leftrightarrow \hat{T}(\vec{r}) = \sum_{i=1}^{N} \frac{w_i^{'}(\vec{r})}{\sum_{i=1}^{N} w_i^{'}(\vec{r})} T(\vec{r_i}) \Leftrightarrow$$

$$\hat{T}(\vec{r}) = \sum_{t=1}^{N} \frac{\frac{1}{D^2}}{\sum_{i=1}^{N} \frac{1}{D^2}} T(\vec{r_i}) \Leftrightarrow \hat{T}(\vec{r}) = \sum_{t=1}^{N} \left(\frac{1}{\sum_{i=1}^{N} D^2} \boxed{\frac{T(\vec{r_i})}{D^2}} \right)$$

En conséquence le w du code MATLAB n'est pas le w théorique mais une autre partie non entourée :

$$w_{\text{matlab}} = \frac{1}{\sum_{i=1}^{N} D^2}$$

2.2 Pondération par gaussienne

La méthode par pondération gaussienne diffère par rapport à ce qui a été dit juste avant au niveau de son coefficient de pondération $w_i(\vec{r})$ qui s'écrit :

$$w_i(\vec{r}) = \frac{w_i^{'}(\vec{r})}{\sum_{i=1}^{N} w_i^{'}(\vec{r})} \text{ avec } \boxed{w_i^{'}(\vec{r}) = \frac{1}{e^{\left(\frac{(x-x_i)^2}{x_{\text{scale}}^2} + \frac{(y-y_i)^2}{y_{\text{scale}}^2}\right)}}}$$

où x_{scale} et y_{scale} sont deux facteurs d'échelle à "optimiser". La formule étant réalisée pour des coordonnées GPS.

Dès lors il suffit de réutiliser le script précédent en utilisant cette fois-ci la nouvelle expression :

```
%Execution du script load_GLAZUR :
run load_GLAZUR

%Pour calculer les distances en cartesien :
R=6371000;
rad=pi/180;
T=zeros(377,170);
T_st=T_st(~isnan(T_st));
n_st=length(T_st);
xscale=0.4/3;
yscale=0.5/3;

for i=1:377
for j=1:170
T_int=zeros(n_st,1);
```

```
for k=1:n_st
dlat=(Y_st(k)-Y(j))*rad;
dlon=(X_st(k)-X(i))*rad;
latm=((Y_st(k)+Y(j))/2)*rad;
D2(k)=R^2*(dlat^2+(cos(latm)*(dlon))^2);

w_prime(k)=1/exp(((X(i)-X_st(k))^2)/(xscale^2)+((Y(j)-
  ↪ Y_st(k))^2)/(yscale^2));

if isnan(glazur_T(i,j)) == 1
T_int(k)=NaN;
else
if D2(k)~=0
T_int(k)=T_st(k).*w_prime(k);
end
end

end

w=sum(w_prime);
T(i,j)=sum(T_int)./w;

end
end

figure(2)
pcolor(T'),colorbar,shading flat
title('Champ de temperature de surface (ponderation
  ↪ par gaussienne)')
```

NB : la variable intermédiaire prend ce coup-ci la partie suivante du calcul :

$$\hat{T}(\vec{r}) = \sum_{i=1}^{N} w_i(\vec{r}) T(\vec{r_i}) \Leftrightarrow \hat{T}(\vec{r}) = \sum_{i=1}^{N} \frac{w_i^{'}(\vec{r})}{\sum_{i=1}^{N} w_i^{'}(\vec{r})} T(\vec{r_i}) \Leftrightarrow \hat{T}(\vec{r}) = \sum_{i=1}^{N} \frac{1}{\sum_{i=1}^{N} w_i^{'}(\vec{r})} \boxed{w_i^{'}(\vec{r}).T(\vec{r_i})}$$

Le w de MATLAB vaut quant à lui :

$$w_{matlab} = \sum_{i=1}^{N} w_i^{'}(\vec{r})$$

2.3 Krigeage

2.3.1 Calcul du semi-variogramme

Le semi-variogramme s'exprime comme :

$$\boxed{\gamma(h) = \frac{1}{2n(h)} \sum_{i,j=1}^{n(h)} [T(\vec{r_i}) - T(\vec{r_j})]^2}$$

où la somme est faite sur les n(h) échantillons tels que $\mid \vec{r_i} - \vec{r_j} \mid < h$ avec h la distance.

Pour l'appliquer sur MATLAB il faut tout d'abord calculer les distances de chaque station entre chaque station. Cela nous amène à une matrice 64×64 valeurs qu'on ré-arrange en une matrice 1×4096 via la fonction reshape. Puis on applique la même technique pour la différence de température de chaque station entre chaque station. Si la manipulation a bien été réalisé alors la matrice finale des distances doit correspondre à la matrice finale des différences de températures de telle sorte que la première distance par exemple est bien celle des stations de la première valeur de différence de température :

```
%Execution du script load_GLAZUR :
run load_GLAZUR

%Calcul des distances :
R=6371000;
rad=pi/180;
T=zeros(377,170);
T_st=T_st(~isnan(T_st));
n_st=length(T_st);

for l=1:n_st
for k=1:n_st

dlat=(Y_st(l)-Y_st(k))*rad;
dlon=(X_st(l)-X_st(k))*rad;
latm=((Y_st(l)+Y_st(k))/2)*rad;
D(l,k)=R*sqrt((dlat^2+(cos(latm)*(dlon))^2));
dT2(l,k)=(T_st(l)-T_st(k))^2;

end
end

figure(2)
h_vect=reshape(D,[n_st*n_st],1);
dT2_vect=reshape(dT2,[n_st*n_st],1);
```

Ensuite grâce à la fonction histogramme on peut trier les distances sur n cases équidistantes créées entre les valeurs minimum et maximum de la matrice finale des distances (h_vect) :

```
classe=20;
hist(h_vect,classe)
title('Histogramme')
```

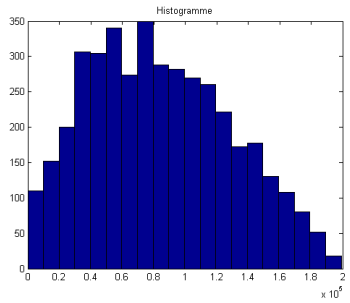

FIGURE 1 – Histogramme

Avec la commande ci-après il est alors possible de récupérer le milieu de chaque case ainsi que le nombre de distance qui s'y trouve à l'intérieur de chacune d'elle :

```
[nc,m]=hist(h_vect,classe);
```

Or néanmoins ce que l'on veut ce sont les intervalles et non les milieux. Par conséquent il est primordial de créer un vecteur intervalle en utilisant ces milieux. Ce qui est somme toute basique et qui s'effectue de la manière suivante :

```
intervalle=zeros(1,classe+1);
Gamma=zeros(1,classe);

for p=2:length(intervalle)
if p==length(intervalle)
intervalle(p)=intervalle(p-1)+(intervalle(2)-
  ↪ intervalle(1));
else
intervalle(p)=(m(p-1)+m(p))/2;
end
end
```

Enfin la dernière étape consiste à trouver par la fonction find tous les indices matricielle des distances pour chaque intervalle qui permettra par la suite de classer les différences de température afin de calculer $\gamma(h)$:

```
for n=1:length(intervalle)-1

index=find(h_vect<intervalle(n+1));
selec=dT2_vect(index);
index2=find(h_vect(index)>=intervalle(n));
selec2=selec(index2);
Gamma(n)=(1/(2*nc(n)))*sum(selec2);

end

figure(3)
plot(m,Gamma)
```

2.3.2 Interpolation du variogramme par une fonction analytique

Si l'on souhaite avoir le variogramme pour des valeurs continues de h il faut alors interpoler $\gamma(h)$ - qui est en discret - par une fonction analytique. On prendra ici un polynôme d'ordre 3 :

$$\boxed{\tilde{\gamma}(h) = p3(h) = a_0 + a_1 h + a_2 h^2 + a_3 h^3}$$

Les coefficients de ce polynôme s'obtiennent via la fonction polyfit de MATLAB. Il ne reste alors plus qu'à créer un vecteur h pour faire le calcul de $\tilde{\gamma}(h)$:

```
P=polyfit(m,Gamma,3);

h=[0:1000:intervalle(end)];
Gammat=P(4)+P(3).*h+P(2).*(h.^2)+P(1).*(h.^3);

figure(4)
plot(h,Gammat)
```

2.3.3 Détermination des poids

La détermination des poids se fait en résolvant le système matricielle suivant :

$$\underbrace{\begin{pmatrix} \tilde{\gamma}(h_{1,1}) & \tilde{\gamma}(h_{1,2}) & \dots & \tilde{\gamma}(h_{1,N}) & 1 \\ (h_{2,1}) & \tilde{\gamma}(h_{2,2}) & \dots & \tilde{\gamma}(h_{2,N}) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \tilde{\gamma}(h_{N,1}) & \tilde{\gamma}(h_{N,2}) & \dots & \tilde{\gamma}(h_{N,N}) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix}}_{\Gamma} \underbrace{\begin{pmatrix} w_1(\vec{r}_p) \\ w_2(\vec{r}_p) \\ \vdots \\ w_N(\vec{r}_p) \\ \lambda \end{pmatrix}}_{w_p} = \underbrace{\begin{pmatrix} \tilde{\gamma}(h_{1,p}) \\ \tilde{\gamma}(h_{2,p}) \\ \vdots \\ \tilde{\gamma}(h_{N,p}) \\ 1 \end{pmatrix}}_{\gamma_p} \Leftrightarrow$$

$$\begin{pmatrix} \tilde{\gamma}(h_{1,1}) & \tilde{\gamma}(h_{1,2}) & \dots & \tilde{\gamma}(h_{1,N}) & 1 \\ (h_{2,1}) & \tilde{\gamma}(h_{2,2}) & \dots & \tilde{\gamma}(h_{2,N}) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \tilde{\gamma}(h_{N,1}) & \tilde{\gamma}(h_{N,2}) & \dots & \tilde{\gamma}(h_{N,N}) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \tilde{\gamma}(h_{1,1}) & \tilde{\gamma}(h_{1,2}) & \dots & \tilde{\gamma}(h_{1,N}) & 1 \\ (h_{2,1}) & \tilde{\gamma}(h_{2,2}) & \dots & \tilde{\gamma}(h_{2,N}) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \tilde{\gamma}(h_{N,1}) & \tilde{\gamma}(h_{N,2}) & \dots & \tilde{\gamma}(h_{N,N}) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix} \begin{pmatrix} w_1(\vec{r}_p) \\ w_2(\vec{r}_p) \\ \vdots \\ w_N(\vec{r}_p) \\ \lambda \end{pmatrix} =$$

$$\begin{pmatrix} \tilde{\gamma}(h_{1,1}) & \tilde{\gamma}(h_{1,2}) & \dots & \tilde{\gamma}(h_{1,N}) & 1 \\ (h_{2,1}) & \tilde{\gamma}(h_{2,2}) & \dots & \tilde{\gamma}(h_{2,N}) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \tilde{\gamma}(h_{N,1}) & \tilde{\gamma}(h_{N,2}) & \dots & \tilde{\gamma}(h_{N,N}) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \tilde{\gamma}(h_{1,p}) \\ \tilde{\gamma}(h_{2,p}) \\ \vdots \\ \tilde{\gamma}(h_{N,p}) \\ 1 \end{pmatrix} \Leftrightarrow$$

$$\boxed{\underbrace{\begin{pmatrix} w_1(\vec{r}_p) \\ w_2(\vec{r}_p) \\ \vdots \\ w_N(\vec{r}_p) \\ \lambda \end{pmatrix}}_{w_p} = \underbrace{\begin{pmatrix} \tilde{\gamma}(h_{1,1}) & \tilde{\gamma}(h_{1,2}) & \dots & \tilde{\gamma}(h_{1,N}) & 1 \\ (h_{2,1}) & \tilde{\gamma}(h_{2,2}) & \dots & \tilde{\gamma}(h_{2,N}) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \tilde{\gamma}(h_{N,1}) & \tilde{\gamma}(h_{N,2}) & \dots & \tilde{\gamma}(h_{N,N}) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix}^{-1}}_{\Gamma^{-1}} \underbrace{\begin{pmatrix} \tilde{\gamma}(h_{1,p}) \\ \tilde{\gamma}(h_{2,p}) \\ \vdots \\ \tilde{\gamma}(h_{N,p}) \\ 1 \end{pmatrix}}_{\gamma_p}}$$

La différence entre Γ et γ_p réside sur le h. Dans le premier cas c'est la distance entre les échantillons (stations), dans le deuxième cas c'est la distance entre échantillons et points de la grille.

L'application sur MATLAB est la suivante :

```
clear l k p n

for i=1:n_st
for j=1:n_st

Gamma(i,j)=P(4)+P(3)*D(i,j)+P(2)*(D(i,j)^2)+P(1)*(D(i,
  ↪ j)^3);

end
end

Gamma(n_st+1,:)=ones(1,n_st);
Gamma(1:n_st,n_st+1)=ones(n_st,1);
Gamma(n_st+1,n_st+1)=0;

clear i j
```

```
for i=1:377
for j=1:170

T_int=zeros(n_st,1);

if isnan(glazur_T(i,j)) == 1
T(i,j)=NaN;
else
for k=1:n_st
dlat=(Y_st(k)-Y(j))*rad;
dlon=(X_st(k)-X(i))*rad;
latm=((Y_st(k)-Y(j))/2)*rad;
D(k)=R*sqrt((dlat^2+(cos(latm)*(dlon))^2));

if D(k)~=0
Gamma_p(k)=P(4)+P(3)*D(k)+P(2)*(D(k)^2)+P(1)*(D(k)^3);
end
end

Gamma_p(k+1)=1;
w_p=Gamma_p*pinv(Gamma);
T_int=T_st.*w_p(1:end-1)';
T(i,j)=sum(T_int)/sum(w_p);

end
end
end

figure(5)
pcolor(T'),colorbar,shading flat
```

3 Résultats

3.0.1 Pondération par inverse du carrée de la distance

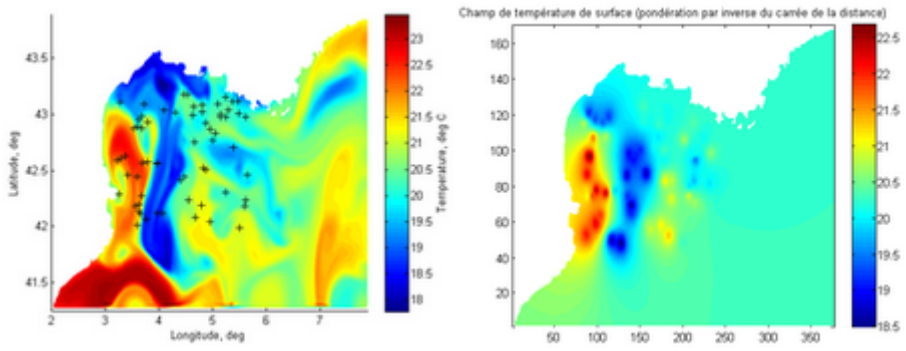

FIGURE 2 – À gauche le SST du modèle et à droite le SST interpolé

La similitude est bien visible entre le champ de température de surface du modèle et celui interpolé. Néanmoins cela reste relativement approximatif.

3.0.2 Pondération par gaussienne

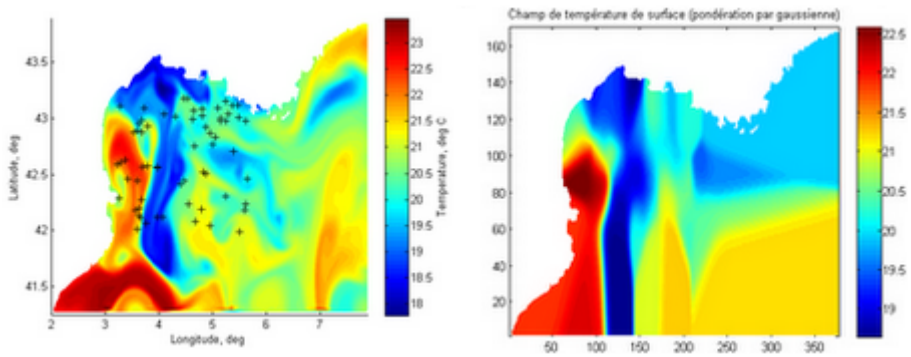

FIGURE 3 – À gauche SST du modèle et à droite SST interpolé

La remarque est la même que précédemment à ceci près que la méthode par pondération gaussienne semble un peu plus précise (à condition que x_{scale} et y_{scale} soient correctement choisie).

3.0.3 Visualisation du semi-variogramme interpolé par une fonction analytique

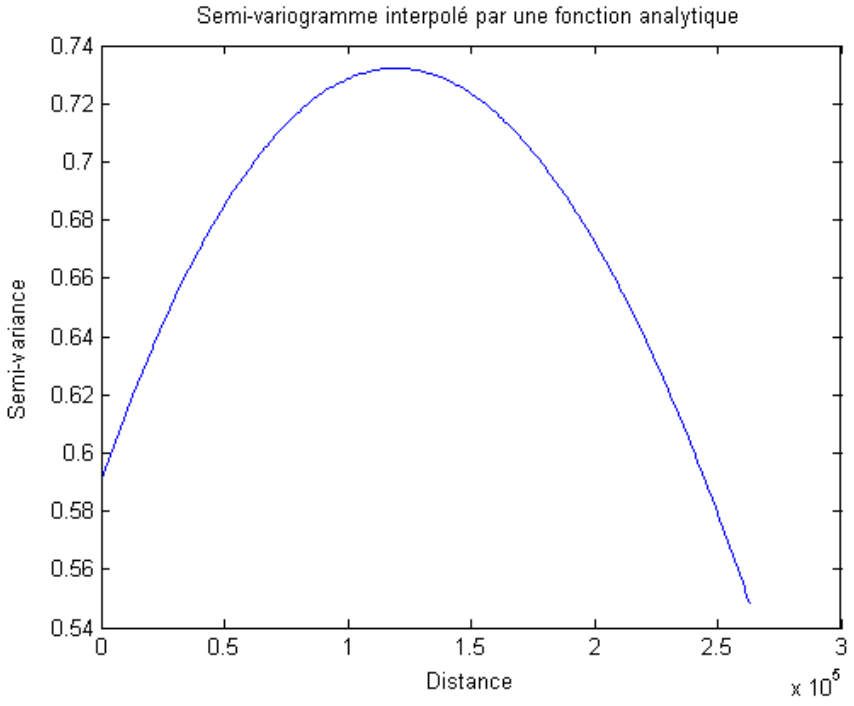

FIGURE 4 – Semi-variogramme interpolé

Ci-dessus le semi-variogramme interpolé par une fonction analytique.

3.0.4 Krigeage

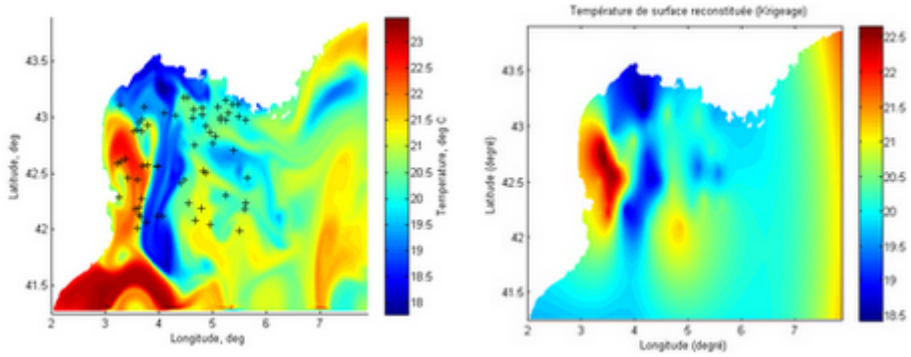

FIGURE 5 – À gauche SST du modèle et à droite SST interpolé

On constate que le Krigeage va jusqu'à reproduire minutieusement les formes que l'on trouve dans le champ de température de surface du modèle. Ainsi parmi les trois techniques elle est la plus puissante.

4 Discussion

Pondération par inverse du carrée de la distance

Cette méthode présente l'avantage d'être efficace pour des distances courtes et un nombre de station conséquent. Toutefois pour des distances longues l'information se perd et la qualité de l'interpolation décline proportionnellement à la distance.

Pondération par gaussienne

Dans le cas d'une pondération par gaussienne la précision apparaît meilleure. Les points qui sont proches des stations ont de très bonnes valeurs. Lorsqu'il y a éloignement il s'effectue une moyenne permettant de limiter la perte de l'information. Toutefois cela à pour désavantage d'extrapoler sur des zones trop distante amenant à une uniformisation de la température en ces lieux qui n'est pas juste .

Krigeage

Parmi les trois procédés le Krigeage reste la technique la plus satisfaisante parce qu'étant la plus optimale. En conséquence elle doit être utilisée préférentiellement aux deux autres. Cependant sa mise en œuvre s'avère laborieuse car bien plus complexe mathématiquement.

5 Conclusion

L'intérêt de l'interpolation spatiale est qu'elle permet d'obtenir des estimations plus ou moins fiable - en fonction de la méthode utilisée - sur un large spectre à partir d'une faible quantité d'information disponible. De plus elle est très utile dans la plupart des domaines scientifiques. Enfin bien que dans ce rapport ne figure que 3 méthodes il en existe en réalité une multitude qui sont peu ou prou performante.

Références

[1] GRATTON, Yves. 2002. *Le Krigeage : la méthode optimale d'interpolation spatiale.*