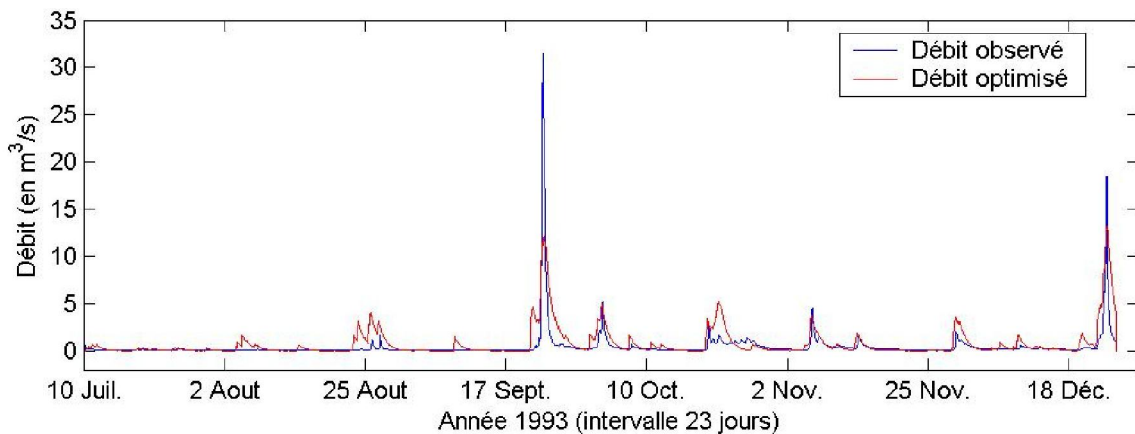


OPTIMISATION EN HYDRAULIQUE

Régression Non Linéaire Et Calage de Modèles Réservoirs



Synthèse du travail

Par

**Quentin ARAUD
Renaud CHAMPREDONDE**

SOMMAIRE

SOMMAIRE	2
TABLE DES ILLUSTRATIONS	3
1. INTRODUCTION	4
2. REGRESSION NON LINEAIRE	5
2.1. Première approche de « Isqcurvefit »	5
2.1.1. L'opérateur « backslash »	5
2.1.2. Optimisation avec la fonction « Isqcurvefit »	6
2.1.3. Contraintes Upper Bound et Lower Bound.....	8
2.1.4. Contraintes indirectes.....	10
2.2. Analyse de la méthode.....	10
2.3. Propagation du souffle de l'explosion AZF	10
2.3.1. Utilisation d'une fonction sinusoïdale	11
2.3.2. Utilisation d'une fonction polynomiale	17
3. CALAGE DE MODELES RESERVOIRS PLUIES-DEBITS	21
3.1. Méthode de résolution.....	21
3.1.1. Formulation et intégration analytique du problème différentiel	21
3.1.2. Discrétisation du modèle	23
3.1.3. Méthode de programmation	24
3.1.4. Résultats de l'optimisation.....	26
3.2. Commentaires et critiques des résultats	27
3.2.1. Différences entre Qmodel et Qobs.....	27
3.2.2. Perspective du modèle.....	28
4. CONCLUSION.....	31

TABLE DES ILLUSTRATIONS

Figure 1 : Optimisation avec la commande "backslash"	6
Figure 2 : Optimisation avec la fonction "lsqcurvefit"	8
Figure 3 : Optimisation sous contrainte "Lower" et "Upper" bounds	9
Figure 4 : Champ de pression AZF - fonction sinus	13
Figure 5 : Champ de pression AZF fonction sinus - données étendues	15
Figure 6 : Champ de pression AZF - fonction sinus - Contraintes sur les paramètres.....	16
Figure 7 : Champ de pression - Fonction polynome	18
Figure 8 : Champ de pression - Fonction polynome - données étendues	19
Figure 9 : Champ de surpression à l'intérieur de la sphère de choc	20
Figure 10 : Champ de surpression à l'intérieur de la sphère de choc - 3D	21
Figure 11 : Optimisation du débit du réservoir	26
Figure 12 : Différence entre débit observé et débit modélisé	26
Figure 13 : Perspective du modèle – Réservoir en parallèle.....	29
Figure 14 : Perspective du modèle – Réservoir en série.....	29

1. INTRODUCTION

L'optimisation est une science active avec le développement des méthodes numériques de résolution de problèmes ou de modélisation des phénomènes physiques. L'objectif de ce rapport est de présenter les travaux réalisés tout au long de ce module d'optimisation.

Nous avons utilisé trois méthodes différentes au cours de cette introduction. Dans une première partie, nous présenterons les résultats d'une optimisation effectuée à l'aide de l'opérateur « backslash » de Matlab. Ensuite nous détaillerons les démarches d'une optimisation à l'aide de la fonction « lsqcurvefit » qui s'intéresse à la résolution des problèmes non linéaires de moindres carrés. Nous testerons les réactions de cette méthode aux conditions initiales. Des contraintes supplémentaires seront imposées au système pour en connaître leurs spécificités. Enfin nous appliquerons l'ensemble de cette programmation au problème de la modélisation de l'explosion du souffle de l'usine AZF de Toulouse.

Dans une troisième partie, nous nous intéresserons à un cas particulier en hydrogéologie que nous résoudrons avec la fonction « lsqnonlin ». Nous proposerons alors un modèle de réservoir pluies-débits pour une source karstique après s'être intéressé à l'écriture équationnelle du problème.

2. REGRESSION NON LINEAIRE

2.1. Première approche de « Isqcurvefit »

Nous nous intéressons dans cette partie aux deux premières applications effectuées en travaux dirigés, à savoir l'opérateur « backslash » et le premier programme d'optimisation dont le script utilise « Isqcurvefit ».

2.1.1. L'opérateur « backslash »

L'optimisation de la fonction « yobs » a été effectuée à l'aide de l'opérateur Matlab « backslash ».

Le script suivant montre le détail du programme permettant d'arriver aux premiers résultats de l'optimisation :

```
clear all; close all;

t= [0 0.3 0.8 1.1 1.6 2.3];
yobs=[0.82 0.72 0.63 0.60 0.55 0.50];

E1 = [exp(-t')];
A = ones(6,1);

E=[E1 A];

a=E\yobs'

t1=0:0.1:2.5;
yopt=a(2,1)+a(1,1)*exp(-t1);
plot(t,yobs,'or ')
hold on
plot(t1,yopt,'-b')
```

Dans ce programme, nous commençons par définir un vecteur temps et nous insérons dans « yobs » les valeurs observées. Nous définissons ensuite une matrice composée de valeurs d'une fonction du type $\exp(-t)$. L'opérateur « backslash » permet d'ajuster les valeurs de E à « yobs » pour ainsi fournir à l'utilisateur les deux paramètres qui correspondent à cet ajustement. Nous obtenons deux valeurs pour la matrice « a » :

$$a = \begin{matrix} 0.3413 \\ 0.4760 \end{matrix}$$

Cet ajustement conduit à la visualisation suivante :

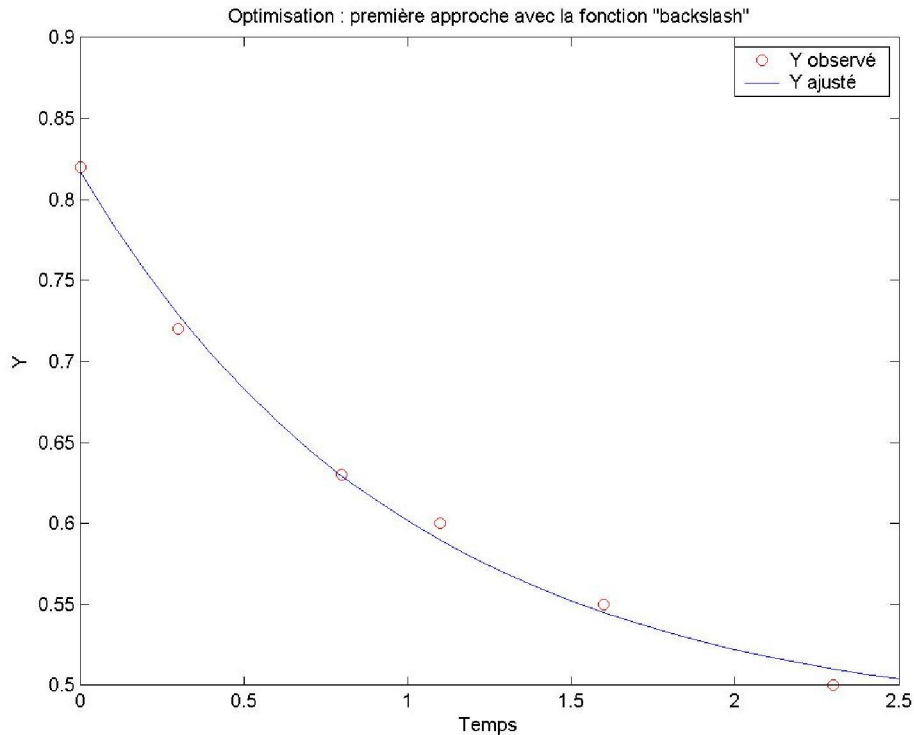


Figure 1 : Optimisation avec la commande "backslash"

Nous constatons que les valeurs pour Y ajusté forme une fonction continue qui, moyennant un écart aux valeurs relativement faible, passe par l'ensemble des points observés.

L'opérateur qui s'intéresse au pseudo inverse « backslash » a permis un ajustement correct entre les valeurs observées et les valeurs optimisées.

2.1.2. Optimisation avec la fonction « Isqcurvefit »

Dans ce paragraphe nous allons utiliser la fonction Matlab « Isqcurvefit » qui résout les problèmes non-linéaires de moindres carrés en minimisant la somme de la différence élevée au carré. Nous présenterons les résultats de l'utilisation de cette fonction sur le cas précédent puis sur l'optimisation du problème de l'explosion de l'usine AZF.

Nous reprenons les deux séries en entrée du paragraphe précédent et nous appliquons la fonction Isqcurvefit à ceux-ci.

Le script de la programmation ainsi que la fonction « mafonction » sont présentés ci-dessous :

```
function yfun = mafonction(xparam,xdata)
yfun = xparam(2) + xparam(1)*exp(-xdata);
```

```
clear all;close all;
xdata = [0 0.3 0.8 1.1 1.6 2.3]; % t
ydata = [0.82 0.72 0.63 0.60 0.55 0.50]; % yobs
xparam(1)=1;
xparam(2)=1;
x_opt = lsqcurvefit(@mafonction, [xparam(1) xparam(2)], xdata, ydata)
```

Nous avons choisi d'optimiser cette fonction avec deux paramètres initialement pris tous deux à la valeur unité. Ce choix de deux paramètres consiste à tester les différences avec l'utilisation de l'opérateur « backslash ».

Nous pouvons également visualiser la norme du résidu par la ligne de commande « resnorm » :

```
[xparam,resnorm]=lsqcurvefit(@mafonction,[xparam(1) xparam(2)],xdata,ydata)
```

Cette requête nous permet ensuite de calculer la racine carrée de la norme du résidu qui correspond à la valeur d'epsilon :

```
epsilon = 0.0180
```

Cette valeur est cohérente car elle est de l'ordre de l'erreur sur une valeur de « ydata ».

Nous pouvons ensuite tracer les valeurs observées et optimisées à l'aide des commandes ci-après :

```
grid on
plot(xdata,ydata,'or ')
x=0:0.001:1;
yfun = xparam(2) + xparam(1)*exp(-xdata);
hold on
plot(xdata,yfun,'-b ')
legend('Y observé','Y optimisé')
```

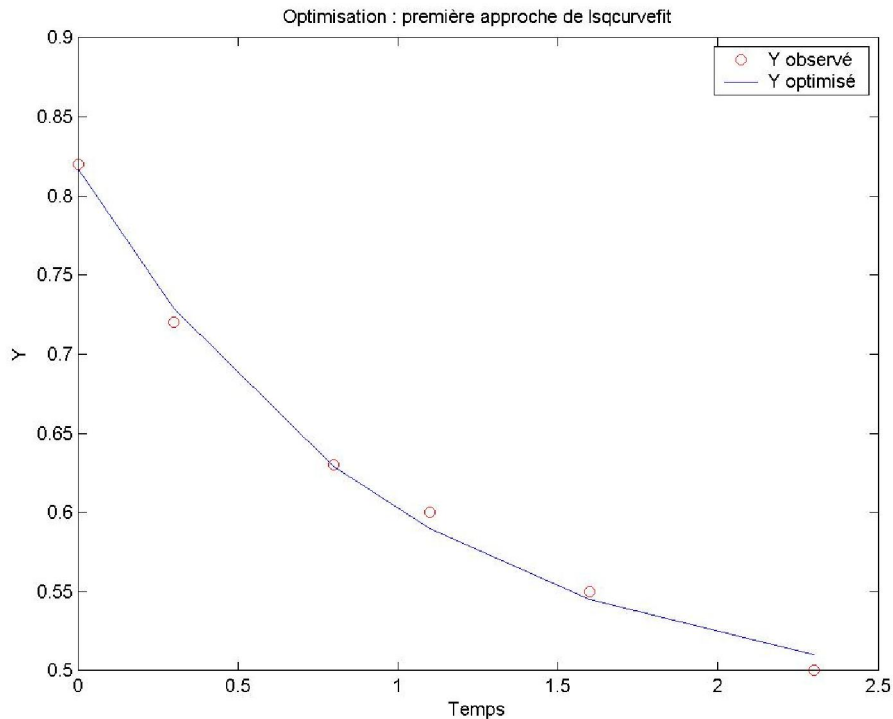


Figure 2 : Optimisation avec la fonction "lsqcurvefit"

La cohérence entre les deux figures est significative. Les deux opérations effectuées sur la série « Yobs » a permis de d'établir la caractérisation du modèle adopté et d'obtenir les coefficients optimisés qui correspondent le mieux avec les valeurs observées. Ainsi, en ce qui concerne le second programme nous obtenons les mêmes coefficients optimisés :

$$\begin{matrix} \text{xparam} = & 0.3413 \\ & 0.4760 \end{matrix}$$

2.1.3. Contraintes Upper Bound et Lower Bound

Après avoir effectué cette première optimisation nous pouvons nous poser la question de la performance de celle-ci et en particulier la possibilité de l'améliorer. La fonction « lsqcurvefit » permet de régler certains autres paramètres de l'optimisation. Parmi ceux-ci, cette fonction propose à l'utilisateur de définir les limites du domaine de définition des paramètres. En fonction des résultats obtenus au paragraphe précédent, nous avons choisi différentes valeurs pour les deux paramètres $\text{xparam}(1)$ et $\text{xparam}(2)$.

- Le premier test a consisté à limiter le domaine de définition des paramètres sans que, à priori, ceux-ci n'en soient affectés. C'est ainsi que nous avons choisi pour la limite minimale et la limite maximale :

$$0 \leq xparam(\lambda) \leq 1 \quad i=1,2.$$

Les résultats obtenus sont conformes à ceux supputés. Nous n'observons aucune différence entre avec l'optimisation sans limite.

- Nous avons ensuite testé l'optimisation avec une limite aval supérieure à un des deux termes retourné par l'optimisation sans limite. Le choix de la limite des paramètres a donc été le suivant :

$$0.4 \leq xparam(\lambda) \leq 1$$

Nous obtenons donc le résultat suivant pour les valeurs des paramètres optimisés :

xparam = 0.4000
0.4483

Ce qui conduit à la représentation graphique :

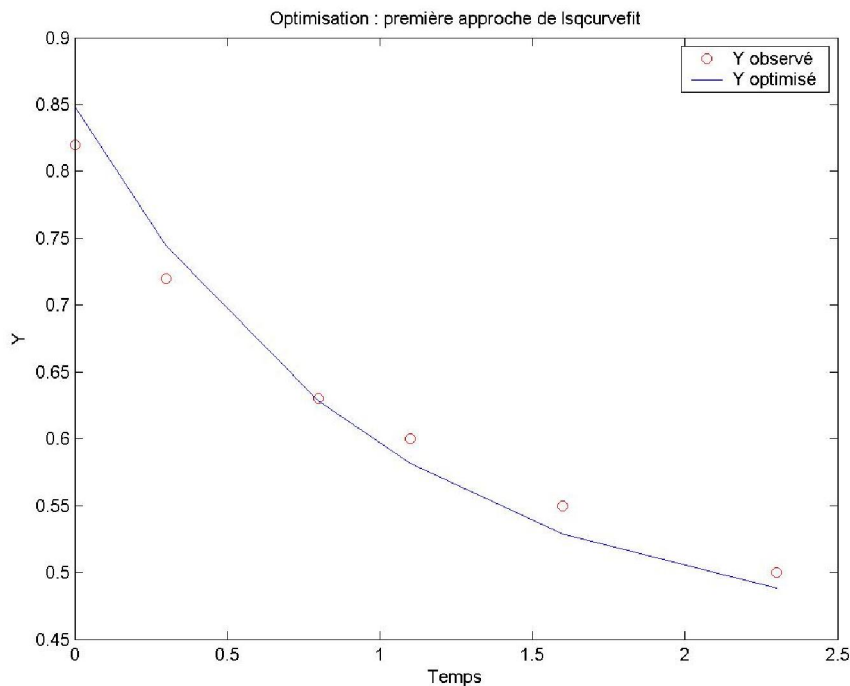


Figure 3 : Optimisation sous contrainte "Lower" et "Upper" bounds

La qualité du résultat est nettement moins bonne que sans cette contrainte supplémentaire. Nous pouvons appuyer cette affirmation par la valeur pour epsilon :

epsilon = 0.0483

Cette valeur signifie que le résultat est plus de deux fois et demie moins précis que celui sans contrainte de limite.

- Pour un choix des deux limites basse et haute fixée à 0 et à 0.4, nous n'observons aucun changement par rapport à l'optimisation sans contrainte de limite.

Nous avons donc vu que dans la plupart des cas, le fait d'ajouter des limites basses et hautes, contraignait le système un peu plus. Nous passons d'une optimisation libre à une optimisation liée. L'optimisation se fait donc en ajoutant comme condition une contrainte supplémentaire qui rend l'optimisation moins précise. Le coefficient epsilon prend alors des valeurs plus élevés puisque l'écart entre la valeur observée et la valeur optimisée est plus grande.

2.1.4. Contraintes indirectes

Nous pouvons contraindre le système à passer par les points $x = 0$ et $x = 1$ en modifiant la série de valeurs en entrée. En effet en ajoutant des points aux alentours de ces abscisses $x = 0$ et $x = 1$, nous forçons le système à passer par ces abscisses.

2.2. Analyse de la méthode

L'analyse de la méthode de programmation utilisée montre que ce sont les paramètres du modèle qui sont optimisés. Lors de l'insertion de la fonction « Isqcurvefit » dans le programme, nous précisons quelle est la matrice des vecteurs à optimiser. Les sorties de cette fonction sont donc les valeurs numériques de l'ensemble de la matrice des coefficients dont dépend le problème. C'est pour cette raison que pour visualiser l'effet de l'optimisation nous devons tracer de nouveau la fonction avec les paramètres que « Isqcurvefit » donne en sortie. Enfin, nous pouvons qualifier la qualité de l'optimisation grâce à la commande « resnorm » que nous demandons d'afficher à la fonction « Isqcurvefit ». Celle-ci retourne la norme du résidu dont nous pouvons extraire la valeur d'epsilon en effectuant la racine carrée.

2.3. Propagation du souffle de l'explosion AZF

Nous allons nous intéresser dans ce paragraphe à la propagation du souffle de l'explosion de l'usine AZF du 21 Septembre 2001 à Toulouse. L'objectif de cette

étude consiste en une représentation du champ de surpression dans la sphère de choc.

Pour ce faire nous allons utiliser le modèle de Sedov-Landau. Celui-ci permet de prédire le champ de pression à l'intérieur de la sphère de choc et à la surface de la sphère de choc, en supposant l'air peu perturbé à l'extérieur de la sphère.

Nous avons choisi d'optimiser l'explosion de l'usine AZF à l'aide de deux types de fonction : une optimisation avec une fonction sinusoïdale et une autre effectuée avec une fonction polynomiale.

2.3.1. Utilisation d'une fonction sinusoïdale

Nous allons nous intéresser à l'optimisation du souffle de l'explosion avec une fonction sinusoïdale. Nous choisissons d'intégrer quinze paramètres pour essayer de représenter au mieux la réalité.

Voici le programme qui nous a permis d'optimiser les valeurs du souffle de l'explosion :

```

clear all; close all;
xdata1 = [0;0.5;0.62;0.7;0.8;0.88;0.96;0.98;0.99;1];
ydata1 = [0.365;0.365;0.365;0.385;0.425;0.5;0.68;0.775;0.87;1];
xparam(1)=0.1;
xparam(2)=0.1;
xparam(3)=0.1;
xparam(4)=0.1;
xparam(5)=0.1;
xparam(6)=0.1;
xparam(7)=0.1;
xparam(8)=0.1;
xparam(9)=0.1;
xparam(10)=0.1;
xparam(11)=0.1;
xparam(12)=0.1;
xparam(13)=0.1;
[xparam,resnorm]=lsqcurvefit(@f_azf_sinus, [xparam(1) xparam(2)
xparam(3) xparam(4) xparam(5) xparam(6) xparam(7) xparam(8)
xparam(9) xparam(10) xparam(11) xparam(12) xparam(13) xparam(14)
xparam(15)], xdata1, ydata1);
epsilon1=(resnorm).^(1/2)
grid on
plot(xdata1,ydata1,'or ')
x=0:0.001:1;
y_opt=xparam(1)+xparam(2).*exp(x)+xparam(3).*exp(2.*x)+xparam(4).*exp(
3.*x)+xparam(5).*exp(4.*x)+xparam(6).*exp(5.*x)+xparam(7).*exp(6.*x)+x
param(8).*exp(7.*x)+xparam(9).*exp(8.*x)+xparam(10).*exp(9.*x)+xparam(
11).*exp(10.*x)+xparam(12).*exp(11.*x)+xparam(13).*exp(12.*x)+xparam(1
4).*exp(13.*x)+xparam(15).*exp(14.*x);
hold on
plot(x,y_opt)
TITLE('Optimisation du champ de pression de l`explosion AZF - Fonction
Sinus')
legend('Champ de pression intérieur observé','Champ de pression
intérieur optimisé',0)
XLABEL('Coordonnée spatio-temporelle réduite')
YLABEL('champ de pression intérieur P2(r,t)')
    
```

```
function yfun = f_azf_sinus(xparam,xdata1)
```

```

yfun=xparam(1)+xparam(2).*exp(xdata1)+xparam(3).*exp(2.*xdata1)+xparam(4).*exp(3.
*xdata1)+xparam(5).*exp(4.*xdata1)+xparam(6).*exp(5.*xdata1)+xparam(7).*exp(6.*xd
ata1)+xparam(8).*exp(7.*xdata1)+xparam(9).*exp(8.*xdata1)+xparam(10).*exp(9.*xdata
    
```

Nous obtenons la visualisation suivante :

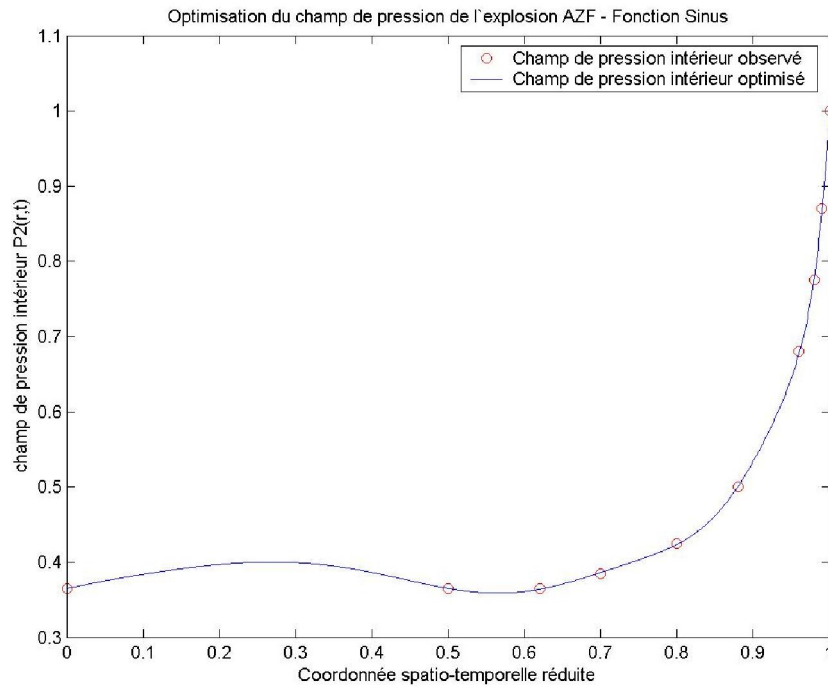


Figure 4 : Champ de pression AZF - fonction sinus

Nous remarquons que l'optimisation effectuée ne se comporte pas comme les valeurs observées entre les abscisses $x = 0$ et $x = 0.5$. Cependant l'écart entre les valeurs observées et les valeurs optimisées, epsilon, vaut :

$$\text{epsilon} = 0.0062$$

Nous décidons de contraindre le système à passer par un plus grand nombre de points entre $x = 0$ et $x = 0.5$.

```

clear all; close all;
xdata1 = [0;0.1;0.2;0.3;0.4;0.5;0.62;0.7;0.8;0.88;0.96;0.98;0.99;1];
ydata1 =
[0.365;0.365;0.365;0.365;0.365;0.365;0.365;0.385;0.425;0.5;0.68;0.775;
0.87;1];
xparam(1)=0.1;
xparam(2)=0.1;
xparam(3)=0.1;
xparam(4)=0.1;
xparam(5)=0.1;
xparam(6)=0.1;
xparam(7)=0.1;
xparam(8)=0.1;
xparam(9)=0.1;
xparam(10)=0.1;
xparam(11)=0.1;
xparam(12)=0.1;
xparam(13)=0.1;
[xparam,resnorm]=lsqcurvefit(@f_azf_sinus, [xparam(1) xparam(2)
xparam(3) xparam(4) xparam(5) xparam(6) xparam(7) xparam(8)
xparam(9) xparam(10) xparam(11) xparam(12) xparam(13) xparam(14)
xparam(15)], xdata1, ydata1);

epsilon1=(resnorm).^(1/2) % On affiche epsilon
grid on
plot(xdata1,ydata1,'or ')
x=0:0.001:1;
y_opt=xparam(1)+xparam(2).*exp(x)+xparam(3).*exp(2.*x)+xparam(4).*exp(
3.*x)+xparam(5).*exp(4.*x)+xparam(6).*exp(5.*x)+xparam(7).*exp(6.*x)+x
param(8).*exp(7.*x)+xparam(9).*exp(8.*x)+xparam(10).*exp(9.*x)+xparam(
11).*exp(10.*x)+xparam(12).*exp(11.*x)+xparam(13).*exp(12.*x)+xparam(1
4).*exp(13.*x)+xparam(15).*exp(14.*x);
hold on
plot(x,y_opt)

```

```
function yfun = f_azf_sinus(xparam,xdata1)
```

```

yfun=xparam(1)+xparam(2).*exp(xdata1)+xparam(3).*exp(2.*xdata1)+xparam(4).*exp(3.
*xdata1)+xparam(5).*exp(4.*xdata1)+xparam(6).*exp(5.*xdata1)+xparam(7).*exp(6.*xdata1)+xparam(8).*exp(7.*xdata1)+xparam(9).*exp(8.*xdata1)+xparam(10).*exp(9.*xdata1)

```

Nous obtenons alors la représentation graphique suivante :

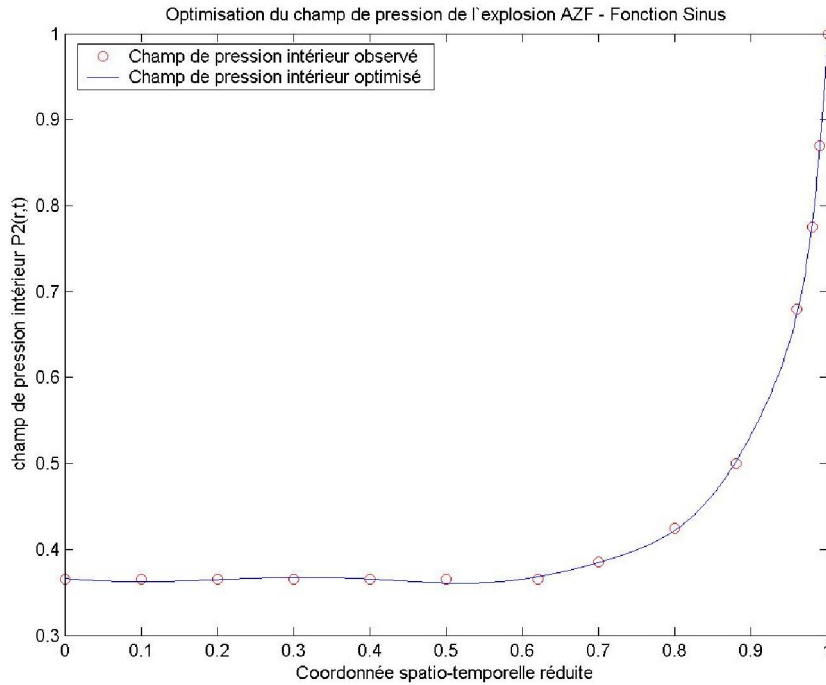


Figure 5 : Champ de pression AZF fonction sinus - données étendues

La valeur de epsilon pour cette optimisation correspond à :

$$\text{epsilon} = 0.0109$$

En augmentant les vecteurs initiaux, préalablement fixé à 0.1 dans l'expérience précédente, à une valeur dix fois plus élevée ($x_{\text{param}(i)}=1$), l'optimisation retourne une valeur pour epsilon plus faible. On obtient ainsi :

$$\text{epsilon} = 0.0103$$

De la même manière, nous arrivons à une valeur pour epsilon très faible pour des paramètres initiaux fixés à 10. Dans ce cas là, nous observons :

$$\text{epsilon} = 0.0087$$

Les différents cas testés ultérieurement semblent préciser cette valeur minimale pour epsilon. En effet, nous obtenons les résultats suivants :

- Pour $x_{\text{param}(i)} = 5$: $\text{epsilon} = 0.0090$
- Pour $x_{\text{param}(i)} = 20$: $\text{epsilon} = 0.0102$
- Pour $x_{\text{param}(i)} = 50$: $\text{epsilon} = 0.0094$

Ces tests effectués sur les vecteurs initiaux permettent de conclure sur l'existence d'une dépendance des résultats sur les valeurs initiales de l'optimisation en ce qui concerne le type de fonction utilisé.

Nous pouvons également tester le modèle aux conditions limites sur les paramètres. Nous choisissons de préciser une limite basse à 0 (contrainte de positivité) par exemple, et nous fixons la limite haute à 100. Sous ces hypothèses, nous n'observons aucune modification de la part de la réponse du système.

Nous avons alors fixés les limites avec une contrainte explicite de négativité de sorte que :

$$-100 \leq x_{param}(\lambda) \leq -10$$

Pour ce cas, nous obtenons une valeur pour epsilon plus faible que sans contrainte à savoir :

$\epsilon = 0.0083$

La représentation correspondant à cette simulation donne le graphique suivant :

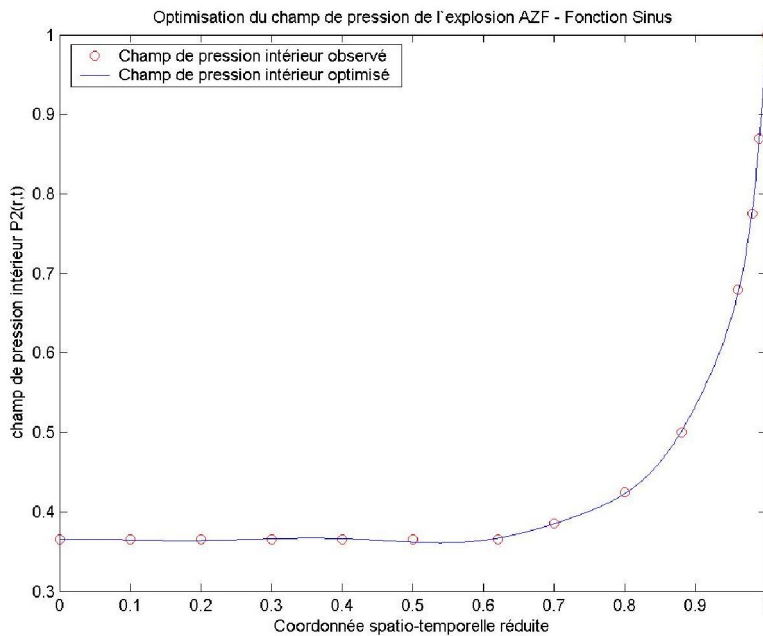


Figure 6 : Champ de pression AZF - fonction sinus - Contraintes sur les paramètres

Nous avons donc trouvé un cas où la contrainte de limite imposée au système permettait d'obtenir une optimisation meilleure.

2.3.2. Utilisation d'une fonction polynomiale

Nous avons ensuite choisi de modéliser le souffle de l'explosion de l'usine AZF en se basant sur une fonction de type polynôme dont l'écriture fait apparaître neuf coefficients d'optimisation. Le programme suivant développe l'algorithme :

```
clear all; close all;

xdata1 = [0;0.5;0.62;0.7;0.8;0.88;0.96;0.98;0.99;1];
ydata1 = [0.365;0.365;0.365;0.385;0.425;0.5;0.68;0.775;0.87;1];

xparam(1)=0.1;
xparam(2)=0.1;
xparam(3)=0.1;
xparam(4)=0.1;
xparam(5)=0.1;
xparam(6)=0.1;
xparam(7)=0.1;
xparam(8)=0.1;
xparam(9)=0.1;

[xparam,resnorm]=lsqcurvefit(@f_azf, [xparam(1) xparam(2) xparam(3)
xparam(4) xparam(5) xparam(6) xparam(7) xparam(8) xparam(9)], xdata1,
ydata1);
epsilon1=(resnorm).^(1/2)
grid on
plot(xdata1,ydata1,'or ')
x=0:0.001:1;
y_opt=(xparam(1)+xparam(2).*x+xparam(3).*x.^2+xparam(7).*x.^3)./(xparam(4)+xparam(5).*x+xparam(6).*x.^2+xparam(8).*x.^3);
hold on
plot(x,y_opt)
TITLE('Optimisation du champ de pression de l'explosion AZF - Fonction Polynomiale')
legend('Champ de pression intérieur observé','Champ de pression intérieur optimisé',0)
XLABEL('Coordonnée spatio-temporelle réduite')
YLABEL('champ de pression intérieur P2(r,t)')
```

```
function yfun = f_azf(xparam,xdata1)

yfun=(xparam(1)+xparam(2).*xdata1+xparam(3).*xdata1.^2 +
xparam(7).*xdata1.^3)./(xparam(4)+xparam(5).*xdata1+xparam(6).*xdata1.^2+xparam(8).
*xdata1.^3);
```

Pour des paramètres initiaux fixés à 0.1, nous obtenons l'optimisation suivante :

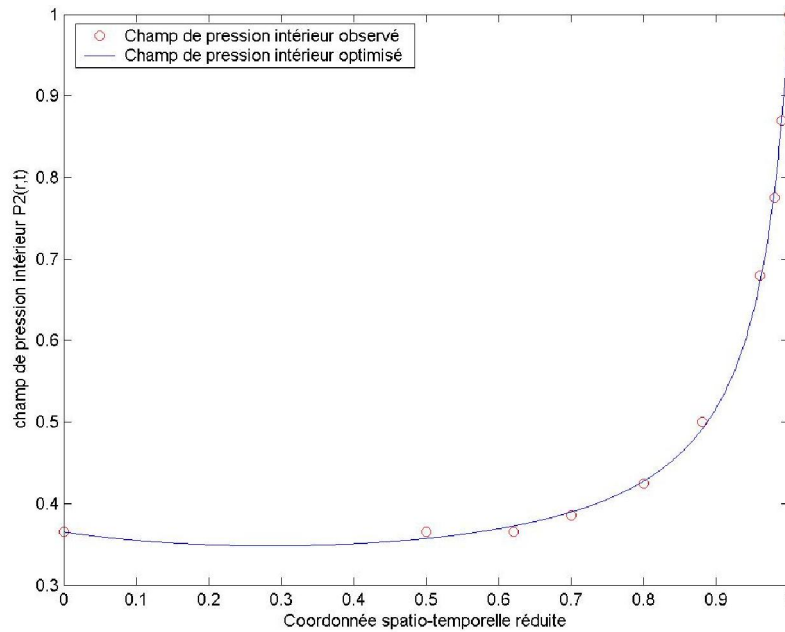


Figure 7 : Champ de pression - Fonction polynome

Les différents essais effectués sur les valeurs initiales des paramètres initiaux de l’optimisation ont mis en évidence une constante pour la valeur d’epsilon ; nous obtenons ainsi :

$$\text{epsilon} = 0.0202 \text{ pour } \text{xparam}(i) = 0.001, 0.1, 1 \text{ et } 10$$

Pour $\text{xparam}(i) = 100$, la valeur pour epsilon diffère de cette dernière et nous obtenons 0.0218. Nous pouvons donc en conclure que pour des faibles valeurs initiales des paramètres $\text{xparam}(i)$, la valeur pour epsilon ne dépend pas des valeurs initiales des paramètres contrairement à la fonction sinusoïde.

Nous pouvons réajuster cette optimisation en ajoutant des données en entrée. Pour des valeurs de la coordonnée spatio-temporelle réduite comprise entre 0 et 0.62, la valeur du champ de pression intérieure reste constante et égale 0.365. Nous ajoutons donc quatre points pour forcer la courbe à s’ajuster sur ces nouvelles valeurs.

On obtient un résultat plus régulier qui correspond mieux au phénomène physique, cependant cette réduction de liberté augmente le coefficient epsilon. La nouvelle valeur pour epsilon vaut donc pour une série plus longue que celle fournie :

$$\text{epsilon} = 0.0230$$

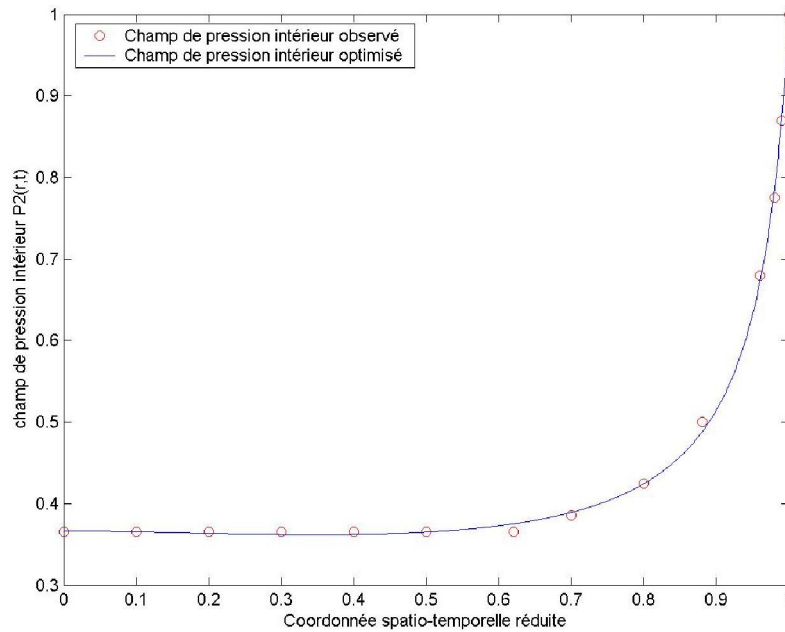


Figure 8 : Champ de pression - Fonction polynome - données étendues

Après avoir testé deux différents modèles d'optimisation du souffle de l'explosion de l'usine AZF, nous pouvons comparer ces deux modélisations. Le modèle non linéaire qui s'ajuste le mieux est donc le modèle polynomial en ayant ajouté quatre points supplémentaires. Le modèle avec la fonction sinusoïde n'est pas adapté pour décrire une telle variation, cependant le choix des multiples coefficients se révèle très efficace pour des coordonnées spatio-temporelles réduites supérieures à 0.6 ; le coefficient epsilon est alors divisé par un facteur 2.23 par rapport au modèle polynomial.

Après avoir effectué l'ensemble des différents tests ci-dessus, nous pouvons maintenant finaliser cette étude en traçant la modélisation quasi-analytique du souffle de l'explosion de l'usine AZF.

Le graphique suivant présente le résultat du champ de surpression à l'intérieur de la sphère de choc pour le temps $T_0 = 14$ ms.

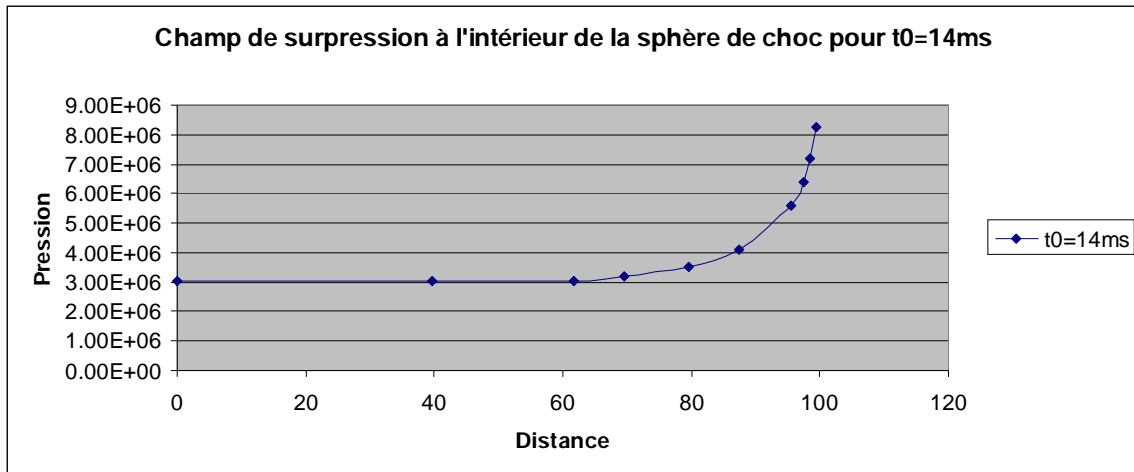


Figure 9 : Champ de surpression à l'intérieur de la sphère de choc

Nous pouvons maintenant tracer ce champ en trois dimensions à l'aide de Matlab, dont voici les lignes de commandes :

```

% Champ de surpression dans la sphère de choc
r0 = 1.04*((50.10^12*(14*10^(-3))^2)/1.225)^(1/5);
u0= 2*r0/(5*14*10^(-3));
gamma = 7/5;
v20=(2/(gamma+1))*u0;
rho1=1.225;
rho2=rho1*(gamma+1)/(gamma-1);
p20=(2/(gamma+1))*u0^2*rho1;
p2=y_opt*p20;

x=-0.5:0.001:0.5; % Expression du vecteur x
y=-0.5:0.001:0.5; % Expression du vecteur y

r=zeros(length(x),length(y)); % Coordonnee radiale
p_dessin=zeros(length(x),length(y)); % Initilisation

figure
for i=1:length(x)
    for j=1:length(y)
        r(i,j)=sqrt(y(j).^2+x(i).^2);
    end
    y_opt_2 = ( xparam(1) + xparam(2).* r(i,:) + xparam(3).*r(i,:).^2
    + xparam(7).*r(i,:).^3)./( xparam(4) + xparam(5).*r(i,:) +
    xparam(6).*r(i,:).^2 + xparam(8).*r(i,:).^3);
    p_dessin(i,:)=y_opt_2 *p20;
end

[XX,YY] = meshgrid(x,y);
colormap(jet);
meshc(XX,YY,p_dessin) % Tracer en 3D
    
```

Nous obtenons pour epsilon :

$$\text{epsilon} = 0.0230$$

Le script précédent à conduit à la représentation :

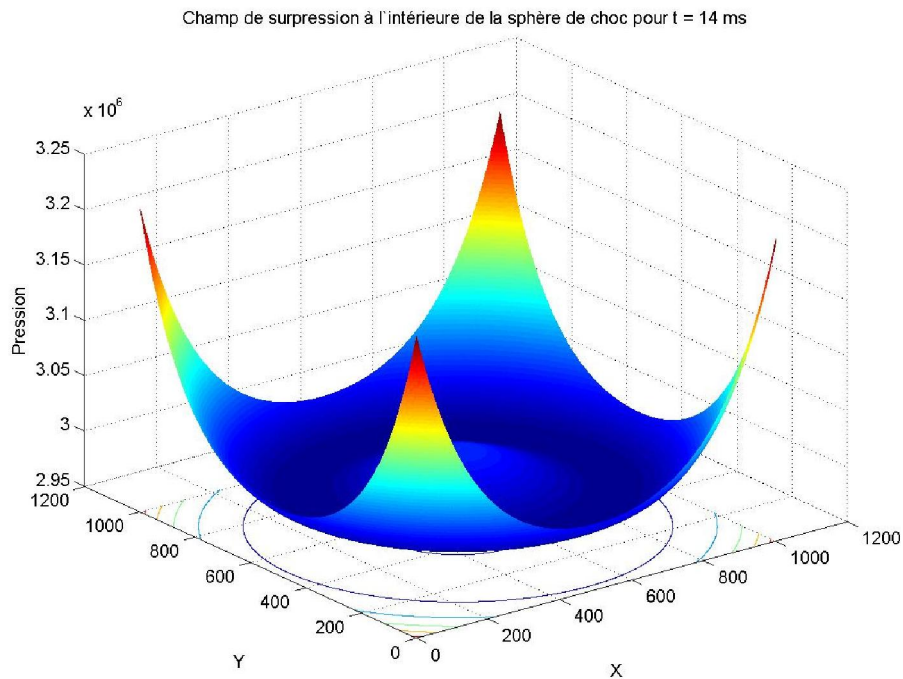


Figure 10: Champ de surpression à l'intérieur de la sphère de choc - 3D

Nous observons sur le graphique des pressions de l'ordre de $3 \cdot 10^6$ Pascal, ce qui correspond des pressions de l'ordre de 30 bars. Il semblerait que nous ayons commis une erreur d'un facteur 10. A ce programme, il faudrait ajouter une condition à l'extérieure de la sphère de choc qui stipule que la pression au delà de la sphère de choc est égale à la pression atmosphérique.

3. CALAGE DE MODELES RESERVOIRS PLUIES-DEBITS

3.1. Méthode de résolution

3.1.1. Formulation et intégration analytique du problème différentiel

Le système se met sous la forme :

$$\begin{cases} \frac{dh}{dt} = p(t) - q(t) \\ q(t) = k_0 * h(t) \end{cases}$$

Ce qui implique l'équation suivante :

$$\frac{dh}{dt} = p(t) - k_0 * h(t)$$

- Résolution de l'équation sans second membre ($p = 0$)

La résolution de l'équation sans second membre $\frac{dh}{dt} = -k_0 * h(t)$ conduit à la solution :

$$h(t) = A * e^{-k_0 * t}$$

- Méthode variation de la constante :

Soit $A = \alpha(t)$, nous obtenons : $h(t) = \alpha(t) * e^{-k_0 * t}$

D'où :

$$\begin{aligned} \frac{dh}{dt} &= \alpha'(t) * e^{-k_0 * t} - k_0 * \alpha(t) * e^{-k_0 * t} \\ \Leftrightarrow \frac{dh}{dt} &= p(t) - k_0 * \alpha(t) * h(t) \\ \Leftrightarrow \frac{dh}{dt} &= \alpha'(t) * e^{-k_0 * t} - k_0 * \alpha(t) * e^{-k_0 * t} \end{aligned}$$

On obtient donc : $\alpha'(t) = p(t) e^{k_0 * t}$

Et donc : $\alpha(t) = \int_0^t P_{obs}(s) e^{k_0 * s} ds$

- Solution particulière

Nous avons pour $h_{particulière}$:

$$h_{part} = \int_0^t P_{obs}(s) e^{k_0 * s} ds * e^{-k_0 * t}$$

Ce qui se met sous la forme : $h_{part} = \int_0^t P_{obs}(s) e^{-k_0(t-s)} ds$

$$h_{sol} = h_{ss \text{ second membre}} + h_{part}$$

$$h_{sol} = A * e^{-k_0 * t} + \int_0^t P_{obs}(s) e^{-k_0(t-s)} ds$$

Nous avons donc : $q_{sol} = k_0 * h_{sol} \Rightarrow q_{sol}(0) = k_0 * h_{sol} = A = q_0$

Et nous obtenons finalement :

$$q_{\text{mod}}(t) = q_0 * A * e^{-k_0 * t} + k_0 \int_0^t P_{\text{obs}}(s) e^{-k_0(t-s)} ds$$

$$S = \int_0^t P_{\text{obs}}(s) e^{-k_0(t-s)} ds$$

3.1.2. Discrétisation du modèle

Nous avons optimisé une fonction réservoir à l'aide de deux paramètres qui représentent q_0 et k_0 . La première boucle ouvre une itération sur le temps : nous effectuons alors 8157 itérations avec un intervalle de temps de 1800 secondes. Pour chaque pas de temps, nous procédons à l'évaluation de l'intégrale décrite par le modèle équationnel ci-dessus. Pour résoudre cette intégrale nous faisons appel à la méthode des rectangles qui pour chaque pas de temps exprime l'aire sous la courbe du rectangle formé par ce pas de temps. Cette somme est initialisée à la valeur nulle à chaque itération. Le calcul de l'intégrale, nous permet alors d'exprimer le second terme de l'équation, à savoir :

$$S = \int_0^t P_{\text{obs}}(s) e^{-k_0(t-s)} ds$$

Nous pouvons voir que la taille de la matrice S dépend principalement du nombre de pas de temps défini. Ainsi le grand nombre de pas de temps du phénomène modélisé a nécessité un temps de calcul conséquent. Il existe une fonction Matlab qui permet de calculer cette somme de Riemann (appelé par la syntaxe « rsums ») mais les tests effectués n'ont pas modifié le temps de calcul.

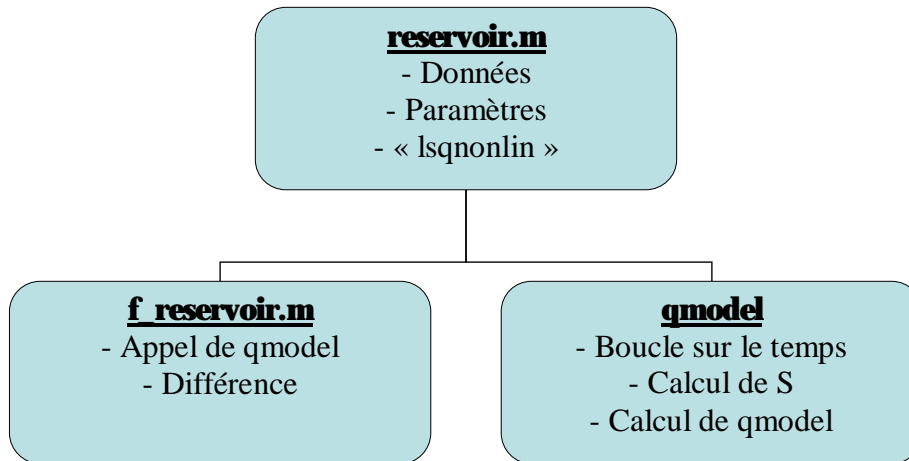
Voici les lignes du code qui ont permis cette discrétisation avec notamment le calcul de la somme de Riemann dont l'ensemble des valeurs sont incrémentés dans la variable « S ».

```
% boucle generale pour calculer qmodel
for j=0:nb_dt-1
    % Calcul de la somme avec la methode des rectangles
    S = 0;
    for i=0:j
        S = S + pobs(i+1)*exp(-coeff_reservoir(1)*(j-i)*dt)*dt;
    end

    % Calcul de Qmodel
    y_model(i+1) = coeff_reservoir(2)*exp(-
coeff_reservoir(1)*j*dt)+coeff_reservoir(1)*S; % calcul de qmodel
end
```

3.1.3. Méthode de programmation

Nous présentons dans cette partie l'organigramme qui nous a servi de base dans la confection du code de calcul.



Le programme débute par le fichier « reservoir.m », qui charge les données « pobs » et « qobs », qui correspondent respectivement à la pluie et au débit observé, en lisant les fichiers textes. Ces deux variables seront définies en « global » qui permet de partager la variable avec d'autre programme ou sous-programme. Il définit ensuite les paramètres qui seront nécessaires à l'exécution du programme. Ensuite la requête d'optimisation « lsqnonlin » appelle la fonction « f_reservoir.m ». Celle-ci s'adresse à la fonction « qmodel.m » qui effectue une boucle sur le temps pour le calcul, en premier lieu, de la somme de Riemann « S », puis de la fonction « qmodel ». C'est à l'intérieur de cette dernière que la discrétisation de l'équation différentielle établie au paragraphe précédent est développée. Une fois cette matrice obtenue, le programme retourne sur le fichier « f_reservoir.m » pour calculer la différence entre « qmodel.m » qui dépend des paramètres d'optimisation et « qobs ». Enfin le programme retourne vers « reservoir.m » où la valeur d'epsilon lui est demandée d'être affichée. Les dernières lignes du code permettent d'effectuer une représentation graphique de l'optimisation effectuée.

Voici les lignes de commandes qui nous permis d'optimiser le débit du réservoir considéré :

```

% =====
% Optimisation du debit d'un reservoir
% -----
% Quentin ARAUD - Renaud CHAMPREDONDE
% =====
clear all; close all;

global pobs qobs ;

pobs = load('PAL93_SEQ.txt');
qobs = load('QAL93_SEQ.txt');

% initialisation des parametres connus
dt = 1800; % Pas de temps en seconde
nb_dt = 8157; % Nombre de pas de temps
t=0:dt:8157*dt;

coeff_reservoir(1)=0.1;
coeff_reservoir(2)=0.1;

[coeff_corriges,resnorm]=lsqnonlin(@f_reservoir,[coeff_reservoir(1)
coeff_reservoir(2)]);

epsilon1=(resnorm).^(1/2)

q_optim=qmodel(coeff_corriges);

% Trace l'hydrogramme sur le premier graph
subplot(2,1,1) , plot(t,pobs)
subplot(2,1,2) , plot(t,qobs)
hold on
subplot(2,1,2), plot(t,q_optim,'-g')

% Trace le graph des deux debits :
subplot(2,1,1) , plot(t,pobs)
TITLE('Précipitations sur le réservoir')
legend('Pluie observée',0)
AXIS([0 14.99e+6 -2 80])
XLABEL('Année 1993 (intervalle 23 jours)')
YLABEL('Pluie (en m ???)')
set(gca,'XTickLabel',{'10 Juil.','2 Aout','25 Aout','17 Sept.','10
Oct.','2 Nov.','25 Nov.','18 Déc.'})
subplot(2,1,2) , plot(t,qobs)
hold on
subplot(2,1,2), plot(t,q_optim,'-r')
TITLE('Optimisation du débit du réservoir')
legend('Débit observé','Débit optimisé',0)
AXIS([0 14.99e+6 -2 35])
XLABEL('Année 1993 (intervalle 23 jours)')
YLABEL('Débit (en m^3/s)')
set(gca,'XTickLabel',{'10 Juil.','2 Aout','25 Aout','17 Sept.','10
Oct.','2 Nov.','25 Nov.','18 Déc.'})

% Trace le graph de la difference entre qobs et q optimise :
difference = qobs - q_optim;
plot(t,difference)
TITLE('Différence entre le débit observé et le débit modélisé')
legend('Différence Q_o_b_s - Q_m_o_d_e_l',0)
AXIS([0 14.99e+6 -8 25])
XLABEL('Année 1993 (intervalle 23 jours)')
YLABEL('Débit (en m^3/s)')
set(gca,'XTickLabel',{'10 Juil.','2 Aout','25 Aout','17 Sept.','10
Oct.','2 Nov.','25 Nov.','18 Déc.'})

```

3.1.4. Résultats de l'optimisation

Ces requêtes produisent donc le résultat de l'optimisation tracée sur le graphique ci-dessous :

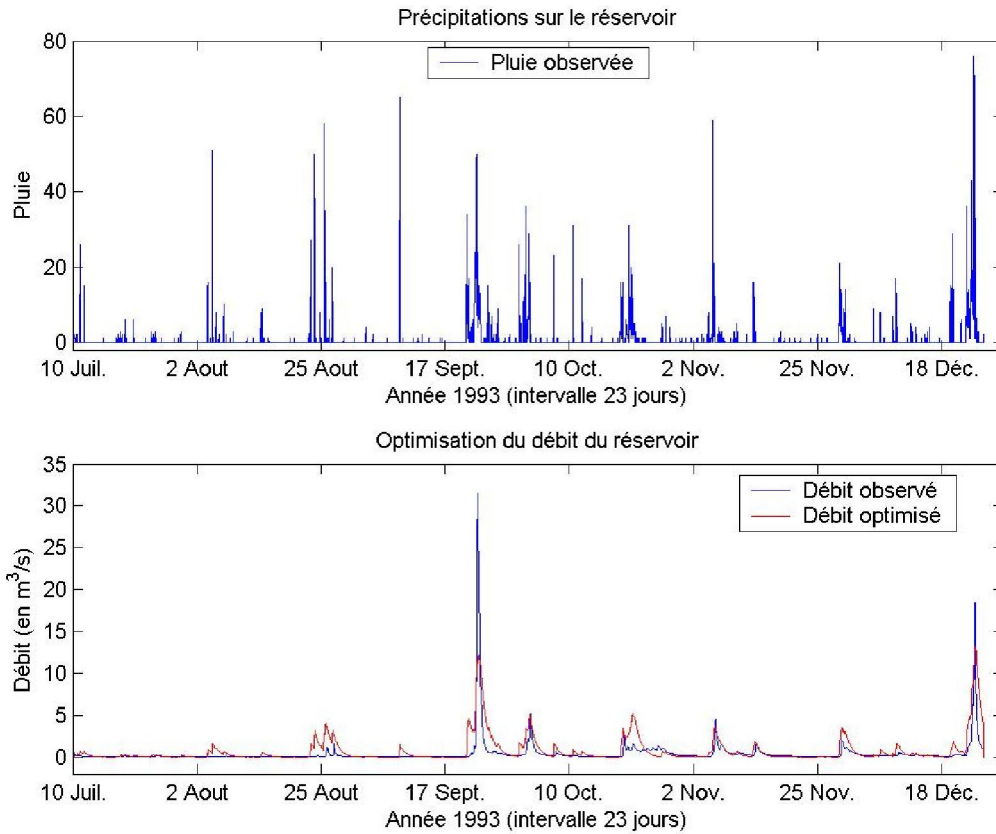


Figure 11 : Optimisation du débit du réservoir

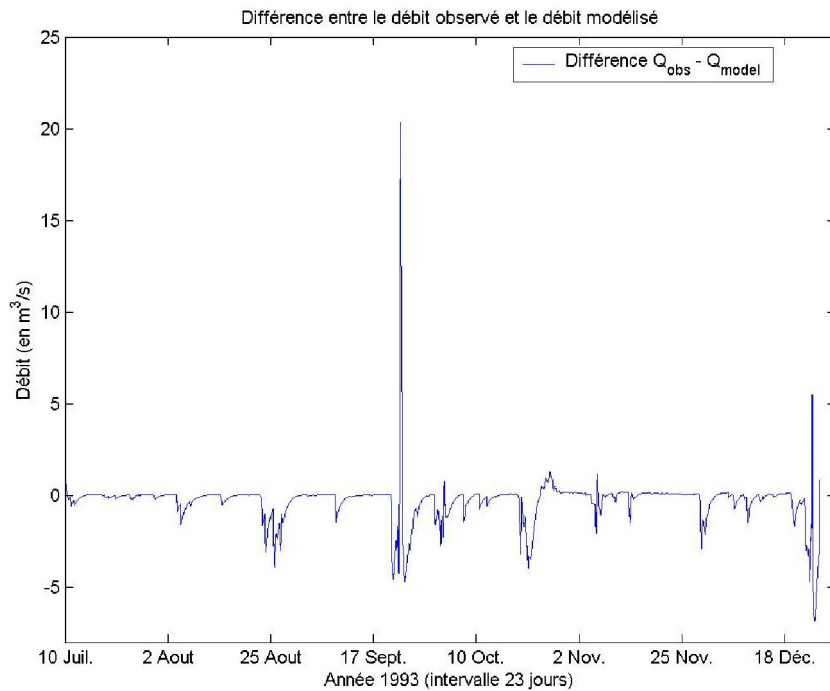


Figure 12 : Différence entre débit observé et débit modélisé

Pour cette modélisation, nous obtenons une valeur relativement élevée pour epsilon :

$$\text{epsilon} = 112.35$$

3.2. Commentaires et critiques des résultats

3.2.1. Différences entre Q_{model} et Q_{obs}

On observe que le débit modélisé représente relativement correctement le débit observé. Les différences qui apparaissent sur ce graphique correspondent le plus souvent à un débit modélisé supérieur à celui observé. Cette constatation peut s'expliquer par la très grande capacité de rétention de l'eau des systèmes karstiques. En effet, le système étudié ne réagit pas nécessairement à un évènement ponctuel comme par exemple ceux de début août et mi-août. La réponse du débit modélisé du réservoir est ressentie sur chaque pic de pluie, alors que le débit observé, lui, ne correspond pas tout fait à ces pics. Cette capacité à retenir l'eau dans ses nombreux vides, entraîne pour le système karstique, lors d'évènements très intenses, des débits observés nettement supérieurs à ceux modélisés : ce phénomène est représenté sur le graphique par les deux pics positifs de la différence entre le débit observé et le débit modélisé qui apparaissent autour du 24 Septembre et du 22 Décembre. Pour le premier pic (24 Septembre), cette différence de près de $20 \text{ m}^3/\text{s}$, s'explique par la rétention des pluies des trois évènements ponctuels de la saison sèche (2 Août, 25 Août, début Septembre) et par une forte accumulation de pluie dans le réservoir pendant l'évènement autour du 24 Septembre. Le débit qui sort alors du système karstique est alors trois fois plus important que celui modélisé.

Le second pic obtenu pour la fin décembre est généré par une pluie de forte intensité qui produit des débits moins importants en intensité que ceux obtenus pour l'évènement du 24 Septembre. Cette constatation peut s'expliquer par une plus forte saturation du système karstique en hiver qu'en été, et donc la capacité de rétention de l'eau dans ces systèmes est moins importante durant la période de fin Décembre. Le système réagit donc beaucoup plus linéairement par rapport à la pluie qui entre en son sein. Cette hypothèse est vérifiée par la faible différence entre le débit observé et le débit modélisé lors du pic du 24 Septembre ($5 \text{ m}^3/\text{s}$). Cette modélisation montre la complexité des systèmes karstiques.

3.2.2. Perspective du modèle

Le modèle utilisé ne permet pas une bonne représentation du modèle karstique étudié. Pour étudier les transferts dans les milieux discontinus comme les karsts, les modèles de type boîte-noire s'avèrent trop restrictifs. Il existe également d'autres concepts pour modéliser de tels milieux : les modèles à réseaux de conduits discrets. Cependant ils se révèlent inadaptés à l'échelle des réservoirs. Les modèles les plus adaptés sont des modèles pour milieux continus qui peuvent fournir des résultats satisfaisant à condition d'adapter le système aux spécificités de chaque karst.

Nous pourrions envisager d'ajouter un réservoir à la modélisation. Celui-ci aurait des capacités de stockage plus ou moins prononcé, ce qui permettrait de fournir des débits élevés à certaines périodes, par exemple pour le débit observé du 24 Septembre. On pourrait ensuite penser à une loi de vidange du réservoir qui ne s'articule pas nécessairement sur une loi linéaire. Cependant, ces hypothèses d'évolution de l'outil seraient à adapter à chaque modèle vu la complexité et l'hétérogénéité des systèmes karstiques. Toute fois ces améliorations auraient pour but de modifier la conception du karst assimilé à une boîte noire.

Nous avons fait l'hypothèse que le bassin suivait une loi de vidange linéaire du type $q(t)=k \cdot h(t)$. Cependant, il semble judicieux d'envisager que la loi de vidange puisse être non linéaire, du type $q(t)=k \cdot h(t)^a$. Un milieu karstique est un milieu complexe comportant un grand nombre de poches d'eau et d'écoulements souterrains. Il semble donc possible de considérer le bassin comme une somme de réservoirs. De plus, il semble logique que chaque réservoir suive une loi de vidange qui lui est propre avec des constante $k_1, k_2 \dots k_n$ différentes, ainsi que des valeurs de l'exposant « a » qui varient. En imaginant ces réservoirs en parallèle on aurait dès lors :

$$q_{\text{total}} = q_1 + q_2 + \dots + q_n$$

$$q_{\text{total}} = k_1 \cdot h_1(t)^{a_1} + k_2 \cdot h_2(t)^{a_2} + \dots + k_n \cdot h_n(t)^{a_n}$$

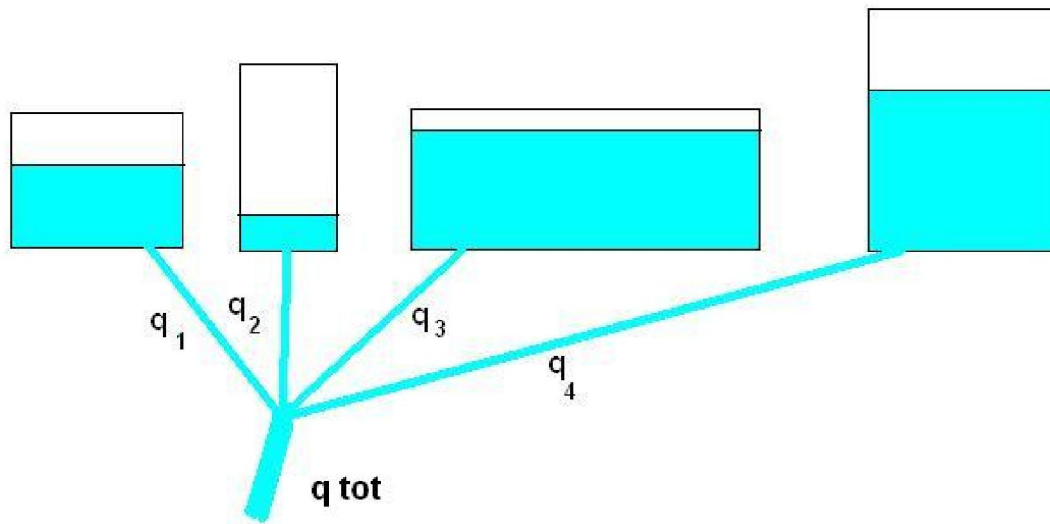


Figure 13 : Perspective du modèle - Réservoir en parallèle

On pourrait aussi concevoir un autre modèle dans lequel les n réservoirs seraient en série. Le flux sortant d'un réservoir, serait le flux entrant du suivant. On aurait alors une équation du type :

$$q_{\text{total}} = q_n(t) = k_n * h_n(t)^{2n}$$

En effet un tel modèle de h_n dépend directement de $q(n-1)$ et donc de $h(n-1)$.

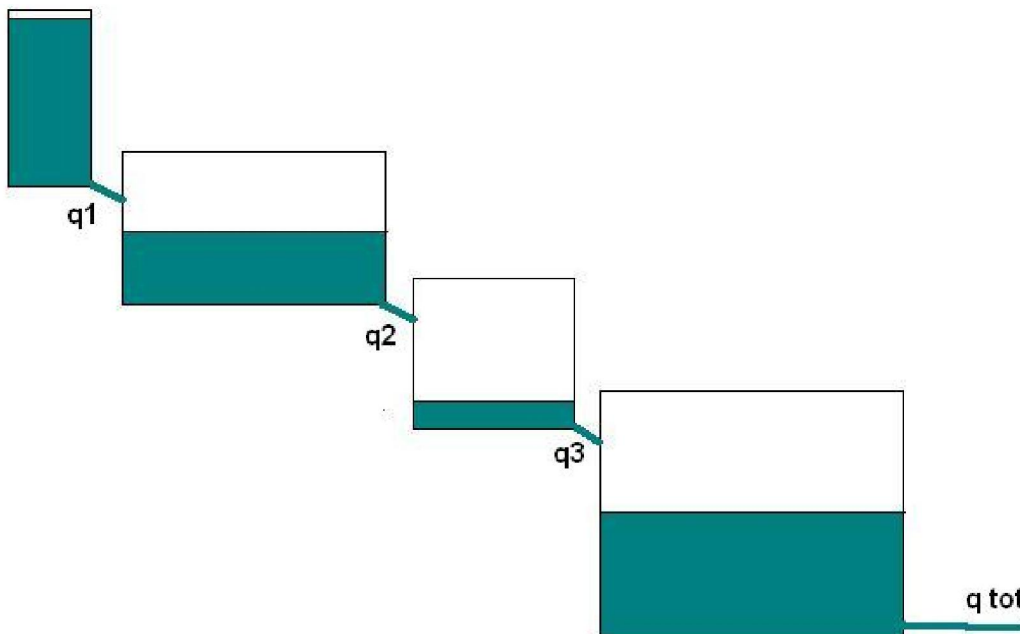


Figure 14 : Perspective du modèle - Réservoir en série

Afin de complexifier et de représenter au mieux la réalité, nous pouvons aussi imaginer un système de seuil dans la modélisation d'un réservoir. En effet, chaque réservoir pourrait comporter un seuil minimal (i.e. il n'y a aucun débit sortant tant que

$h < h_{\min}$) ainsi qu'un seuil maximal (dès que $h > h_{\max}$, le débit entrant n'est pas stocké, mais se rajoute directement au débit sortant). Ces deux types de modèle étant du type « if/else », ils sont très fortement non linéaires. Il apparaît également que le coefficient K_0 de conductivité hydraulique peut lui aussi être modélisé par une loi non linéaire dans l'hypothèse où l'on augmente le nombre d'équation à résoudre. Dans ce cas, il faudrait orienter la programmation vers l'utilisation des fonctions « ode23 » ou « ode45 » qui résolvent des équations du type :

$$M(t,y) \cdot y' = f(t,y)$$

Cette équation est alors à mettre en parallèle avec la loi de Darcy.

4. CONCLUSION

Au cours de ces séances, nous avons pu mettre en pratique les éléments du cours d'optimisation dans le but d'identifier les valeurs des paramètres qui gouvernent les systèmes à l'aide du logiciel Matlab.

La fonction « Isqcurvefit » a été utilisée notamment pour modéliser le souffle de l'explosion de l'usine AZF. Les conditions initiales ainsi que les conditions de limites hautes et limites basses qui incombent aux paramètres du système ont été testées.

Enfin le calage d'un modèle de réservoir pluies-débits a nécessité l'utilisation d'une autre fonction d'optimisation des problèmes fortement non linéaire qui est appelée par la syntaxe « Isqnonlin ». Après avoir exprimé la formulation du problème et son intégration sous forme équationnelle, ce modèle s'est vu discrétiser en une somme sur des valeurs indicielles. Celle-ci a calculé pour chaque pas de temps la résolution de l'intégrale de Riemann du système. Les résultats montrent une relativement bonne résolution du problème pour une première approche. Cependant, ils montrent la complexité des systèmes de types karstiques à fournir une réponse hydrologique réellement prévisible. La critique des résultats et les perspectives développées en dernière partie ouvrent des chemins pour une modélisation plus fidèle de la réalité en intégrant notamment la résolution d'équations différentielles non linéaires liées à la loi de Darcy.