

CSOPT : Calcul scientifique et optimisation  
Travaux pratiques  
Optimisation sans contraintes

Yassine Jamoud                    Samy Haffoudhi

12 novembre 2021

# 1 Minimisation de la fonction de Rosenbrock

On s'intéresse à la fonction de Rosenbrock définie par :

$$f_0(x) = \sum_{i=1}^{n-1} b(x_{i+1} - x_i^2)^2 + (1 - x_i)^2, \quad \forall x \in \mathbb{R}^n \text{ et } b \in \mathbb{R}^+.$$

## 1.1 Travail préliminaire

1. On a  $\forall n \geq 2, \forall x \in \mathbb{R}^n$  et  $b \in \mathbb{R}^+$  :

$$\nabla f_0(x) = \begin{bmatrix} -4bx_1(x_2 - x_1^2) - 2(1 - x_1) \\ 2b(x_2 - x_1^2) - 4bx_2(x_3 - x_2^2) - 2(1 - x_2) \\ \vdots \\ 2b(x_{n-1} - x_{n-2}^2) - 4bx_{n-1}(x_n - x_{n-1}^2) - 2(1 - x_{n-1}) \\ 2b(x_n - x_{n-1}^2) \end{bmatrix}$$

Soit pour  $n = 2, f_0(x) = b(x_2 - x_1^2)^2 + (1 - x_1)^2, \forall x \in \mathbb{R}^2$  et  $b \in \mathbb{R}^+$ .

$$\nabla f(x) = \begin{bmatrix} -4bx_1(x_2 - x_1^2) - 2(1 - x_1) \\ 2b(x_2 - x_1^2) \end{bmatrix} \quad \nabla^2 f(x) = \begin{bmatrix} -4b(x_2 - x_1^2) + 8bx_1^2 + 2 & -4bx_1 \\ -4bx_1 & 2b \end{bmatrix}$$

2. Pour trouver le minimiseur, on cherche à annuler le gradient :

$$\nabla f_0(x_0) = 0 \iff \begin{cases} -4bx_1(x_2 - x_1^2) - 2(1 - x_1) = 0 \\ 2b(x_2 - x_1^2) = 0 \end{cases} \iff \begin{cases} x_2 = x_1^2 \\ 1 - x_1 = 0 \end{cases}$$

D'où  $x_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$  avec  $f_0(x_0) = 0$

Or,  $f_0$ , comme somme de carrés, est à valeurs positives donc,  $x_0$  est bien minimiser de cette fonction.

3. On pose  $f_0$  sous la forme d'un critère de moindres carrés non-linéaires tel que  $f_0(x) = \frac{1}{2}r_1^2(x) + \frac{1}{2}r_2^2(x)$

$$\text{On a alors } r(x) = \begin{bmatrix} \sqrt{2b}(x_2 - x_1^2) \\ \sqrt{2}(1 - x_1) \end{bmatrix}$$

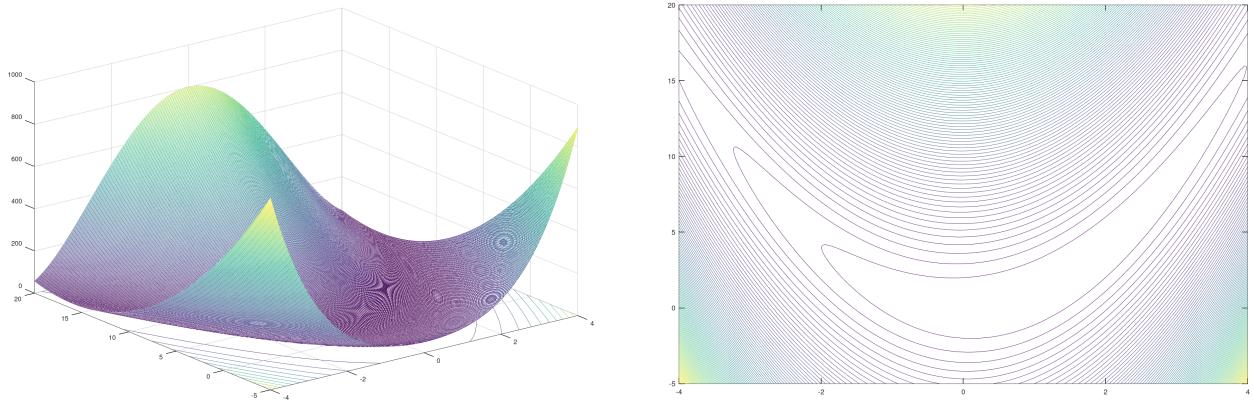
Le Jacobien s'exprime comme  $J = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} \end{bmatrix}$

Après calcul on obtient alors :

$$J = \begin{bmatrix} -2x_1\sqrt{2b} & \sqrt{2b} \\ -\sqrt{2} & 0 \end{bmatrix}$$

## 1.2 Visualisation de la fonction objectif

Après avoir complété les script `rosenbrock` et `visualisation` disponibles en Annexe, on trouve alors pour  $x_1 \in [-4, 4]$  et  $x_2 \in [-5, 20]$  :



(a) Visualisation de la fonction de Rosenbrock en 3D (b) Visualisation des lignes de niveaux de la fonction

### 1.3 Minimisation par descente itérative

- Pour commencer, on utilise la méthode de descente de plus forte pente avec un pas fixe. On obtient alors en 1000 itérations les différents figures :

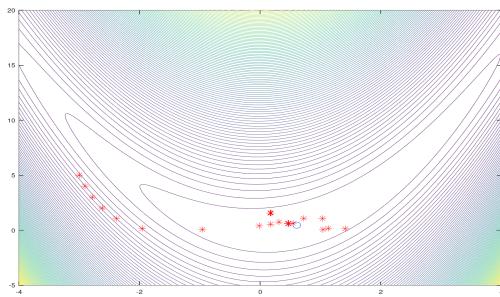


FIGURE 2 – Descente de pas 1

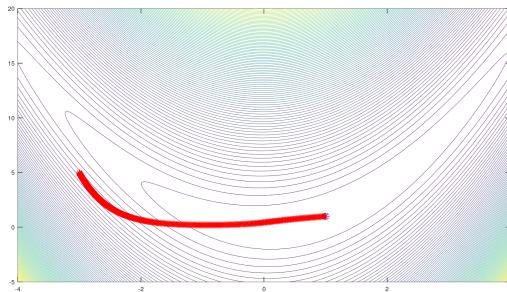


FIGURE 3 – Descente de pas  $10^{-2}$

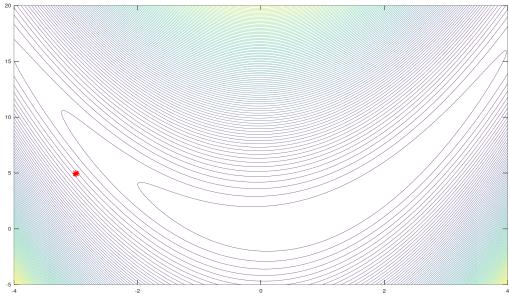


FIGURE 4 – Descente de pas  $10^{-4}$

On remarque que :

- Avec le pas de  $10^{-4}$ , les 1000 itérations ne sont pas suffisante pour tendre vers  $x_h$
- Pour les deux autres valeurs du pas, la méthode converge vers  $x_h$  mais avec le pas de 1, la méthode converge bien plus rapidement qu'avec le pas de  $10^{-2}$ .

2. On ajoute alors une recherche de pas par une technique de rebroussement avec un taux  $\beta = 0.75$  pour assurer la condition d'Armijo avec  $c = 10^{-4}$ .

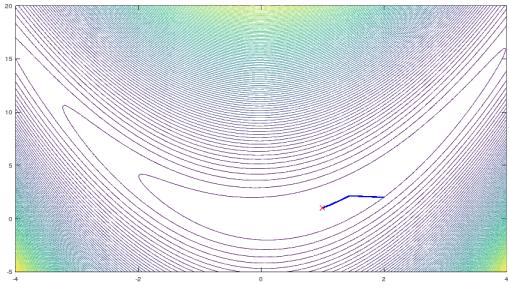


FIGURE 5 – technique de rebroussement

On observe alors que le pas varie effectivement au cours des itérations, de sorte à assurer la condition d'Armijo, ce qui garantit une décroissance suffisante et permet donc de ne pas avoir à se soucier des problèmes vus plus haut (pas trop faible ou trop élevé). Cependant, cette méthode est plus stable mais nécessite plus de calculs que pour la méthode du pas fixe, elle est par exemple bien plus lente que la méthode de pas fixe avec un pas de 1 pour cet exemple précis.

4. Pour la valeur :  $x_0 = (4, 4)$ , on obtient les tracés suivants :

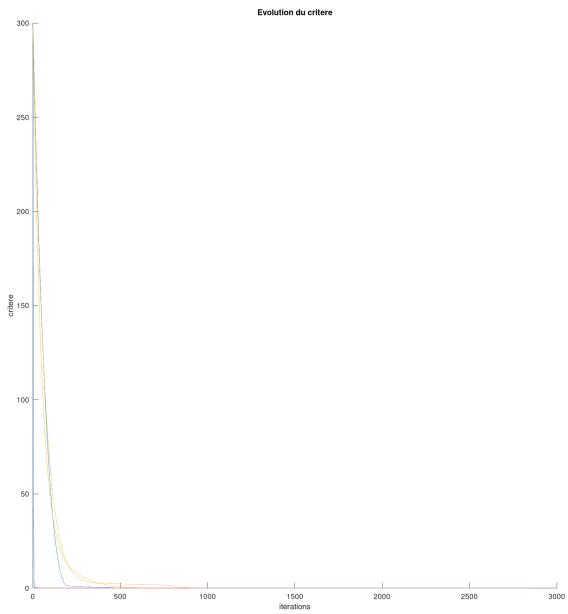


FIGURE 6 – Évolution du critère

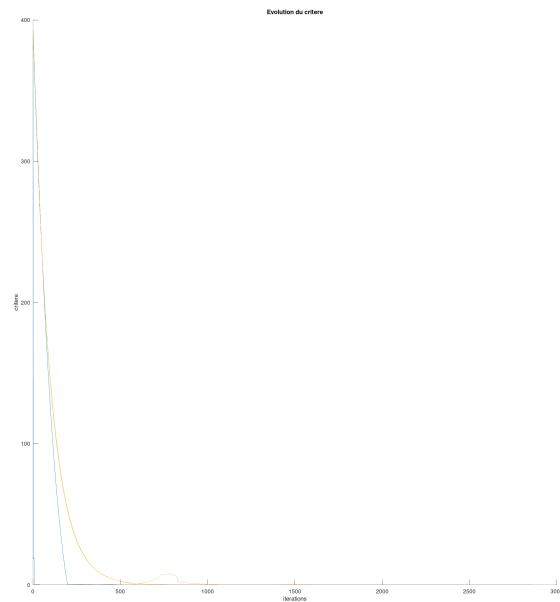


FIGURE 7 – Évolution de la norme du gradient

Pour la valeur :  $x_0 = (5, 10)$  et avec un pas variable, on obtient les tracés suivants :

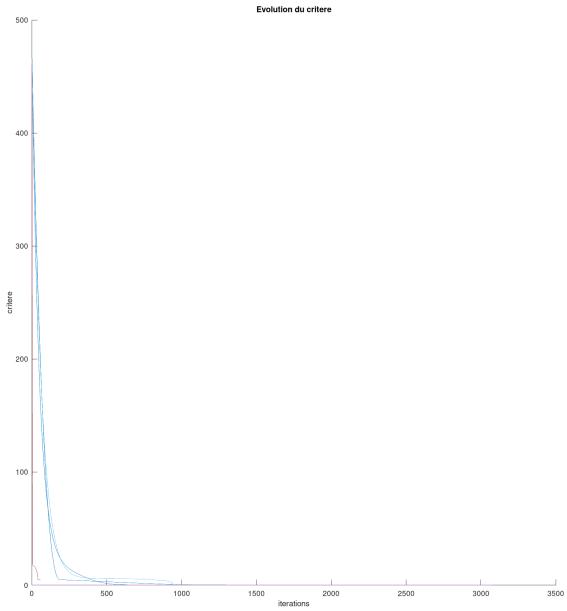


FIGURE 8 – Évolution du critère

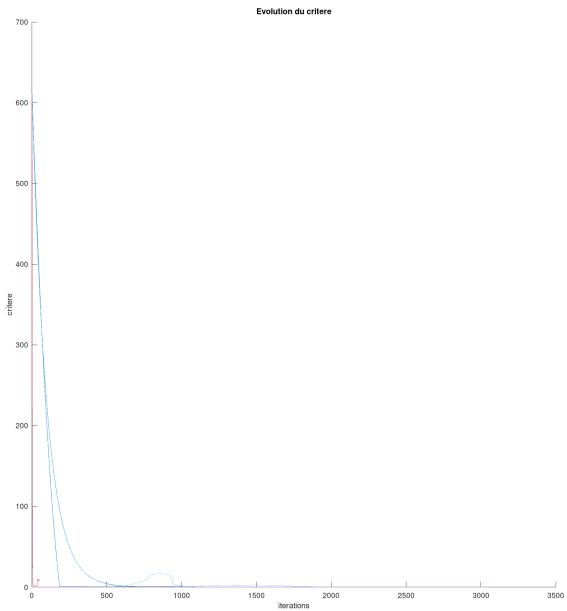


FIGURE 9 – Évolution de la norme du gradient

On observe que pour ces 4 tracés le critère et le gradient s’annulent rapidement pour chaque méthode, il faudrait choisir des valeurs de tolérance plus adaptées pour gagner en temps d’exécution, quitte à être légèrement moins précis. Pour comparer les performances de chaque méthode représentons le nombre d’itérations et le temps d’exécution dans un tableau :

Pour  $x_0 = (4, 4)$  :

Méthode	Nombre d'itérations	Temps (s)
Descente de plus forte pente	911	0.51
Gradient conjugué	451	0.18
Newton	1989	0.37
BFGS	2862	0.75

Pour  $x_0 = (5, 10)$  :

Méthode	Nombre d'itérations	Temps (s)
Descente de plus forte pente	1526	0.70
Gradient conjugué	x	x
Newton	2043	0.64
BFGS	3000	1.30

On remarque alors que :

- Pour la première valeur de  $x_0$ , la méthode du gradient conjuguée est la plus performante que ce soit en temps de calcul ou en nombre d'itérations
- Mais la méthode du gradient conjuguée ne converge pas vers  $x_h$  pour une position initiale plus éloignée.
- Les méthodes de Newton et BFGS sont celles nécessitant le plus d'itérations
- La méthode de Newton est cependant plus rapide que celle du gradient de plus forte pente

6. Pour  $x_0 = (4, 4)$  :

Méthode	Nombre d'itérations	Temps (s)
Gauss-Newton	1603	0.31
levenberg-marquardt	1594	0.33

Pour  $x_0 = (5, 10)$  :

Méthode	Nombre d'itérations	Temps (s)
Gauss-Newton	1622	0.32
levenberg-marquardt	1607	0.35

On remarque alors que :

- Les deux méthodes convergent bien vers  $x_h$  pour les deux valeurs de  $x_0$ .
- Les performances de ces deux méthodes sont très similaires pour les deux valeurs de  $x_0$ , ce qui n'est pas le cas des autres méthodes et qui représente un avantage important.
- Ce sont les méthodes les plus rapides pour la deuxième valeur de  $x_0$  et la deuxième plus rapide pour la première valeur de  $x_0$ .
- La méthode de levenberg-marquardt nécessite le choix d'une valeur pour un paramètre. Dans ce cas ci, nous obtenons des résultats satisfaisant uniquement pour des valeurs très faibles de  $\lambda$ , la méthode est alors quasiment équivalente à celle de Gauss-Newton.

## 2 Ajustement d'une courbe non-linéaire

On cherche à minimiser la fonction objectif :

$$f_0(\alpha, \beta) = \frac{1}{2} \sum_{i=1}^N (R_i - \beta E_i^{-\alpha})^2$$

1. Après calcul, on trouve :

$$\nabla f(x) = \begin{bmatrix} \sum_{i=1}^N \beta \ln(E_i) E_i^{-\alpha} (R_i - \beta E_i^{-\alpha}) \\ - \sum_{i=1}^N E_i^{-\alpha} (R_i - \beta E_i^{-\alpha}) \end{bmatrix}$$

$$\nabla^2 f(x) = \begin{bmatrix} \sum_{i=1}^N \beta E_i^{-\alpha} \ln(E_i)^2 (2\beta E_i^{-\alpha} - R_i) & \sum_{i=1}^N E_i^{-\alpha} \ln(E_i) (R_i - 2\beta E_i^{-\alpha}) \\ \sum_{i=1}^N E_i^{-\alpha} \ln(E_i) (R_i - 2\beta E_i^{-\alpha}) & \sum_{i=1}^N E_i^{-2\alpha} \end{bmatrix}$$

2. Pour les courbes de niveau, on la trace en échelle log pour  $0 < \alpha < 2$  et  $0 < \beta < 4$

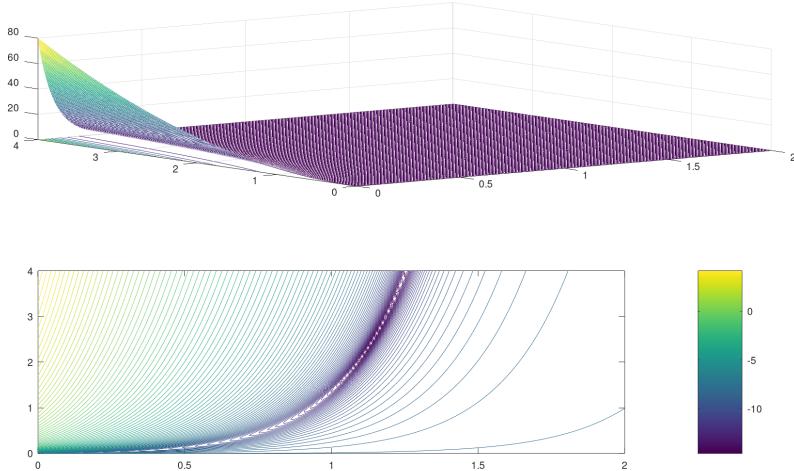


FIGURE 10 – Courbe de niveau du log-critère

3. Par la suite on applique les algorithmes vu à l'exercice 1 pour trouver les paramètres optimaux.  
On retrouve ainsi les résultats suivants :

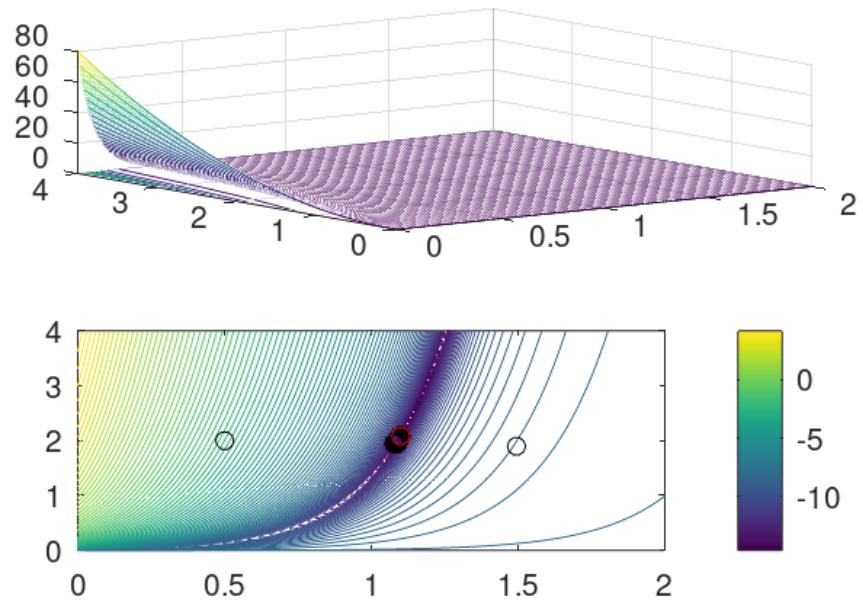


FIGURE 11 – Convergence pour une direction de plus forte pente avec pas variable

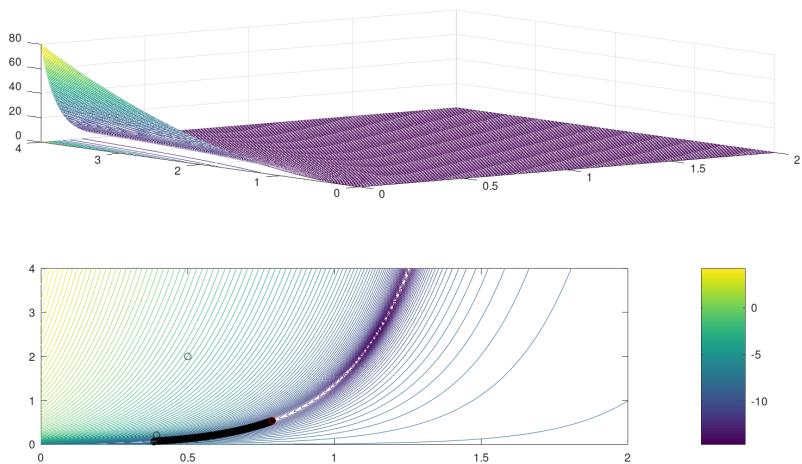


FIGURE 12 – Convergence pour une direction de Newton avec pas variable

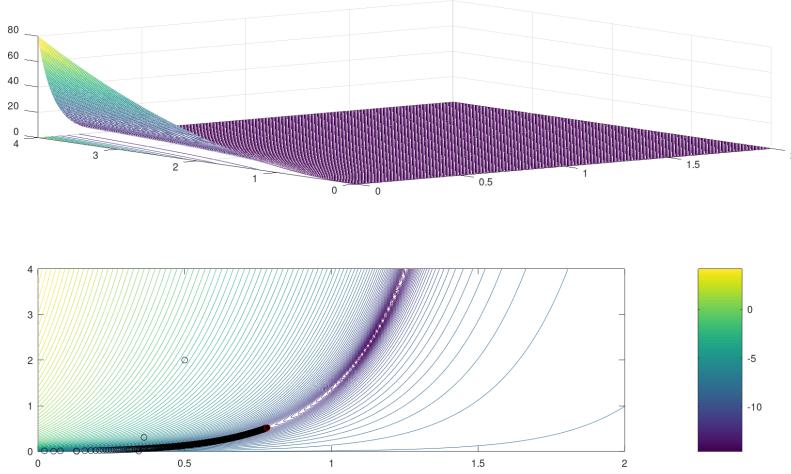


FIGURE 13 – Convergence pour une direction de Newton avec pas fixe

4. On peut reprendre les valeurs finales dans un tableau en prenant  $x_0 = [0.5; 2]$  :

Méthode	Nombre d'itérations	Temps (s)	Valeur finale
Descente de plus forte pente	1002	2	(1.10, 2.08)
Gradient conjugué	1002	10.7	(1.11, 2.15)
Newton	1002	5.9	(0.79, 0.54)
BFGS	39	0.38	(1.15, 2.55)

5. En prenant les différents  $x_h$ , on obtient la caractéristique théorique suivante en rouge par rapport à celle de mesure en bleue :

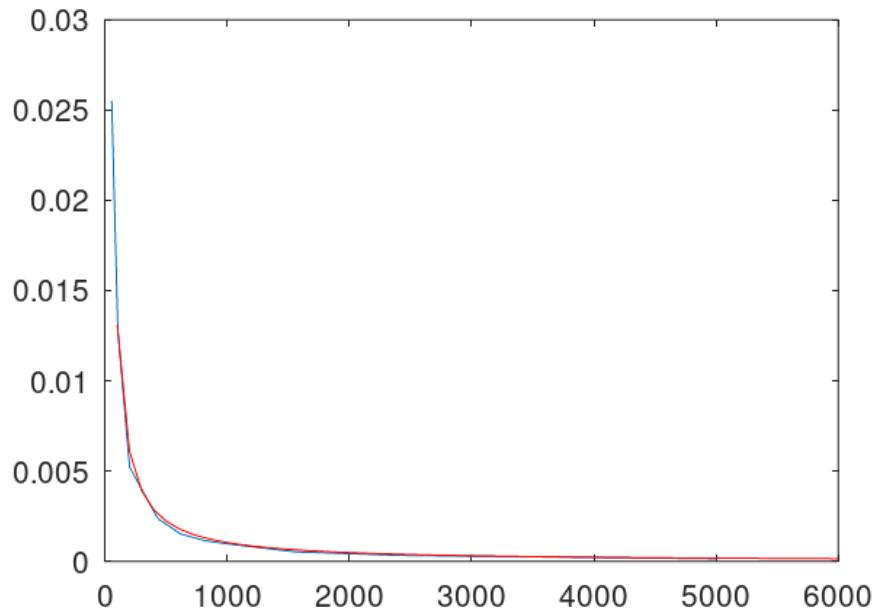


FIGURE 14 – Caractéristiques théorique en et expérimentale pour  $xh = [1.10, 2.08]$

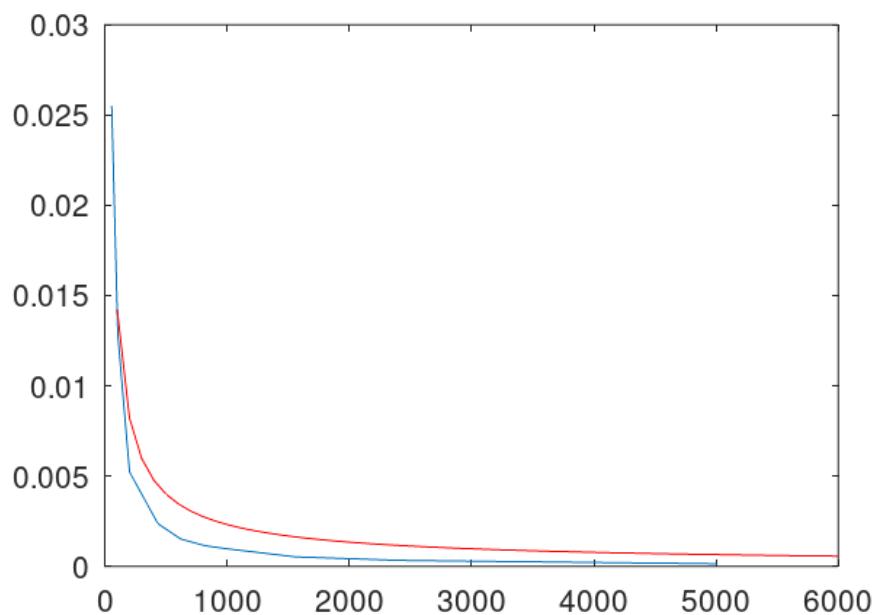


FIGURE 15 – Caractéristiques théorique en et expérimentale pour  $xh = [0.79, 0.54]$

### 3 Débruitage d'un signal par minimisation d'un critère composite

#### 3.1 Simulation des signaux

1. Simulons une réalisation du vecteur  $x$  similaire au spectre vibrationnel d'une molécule chimique.

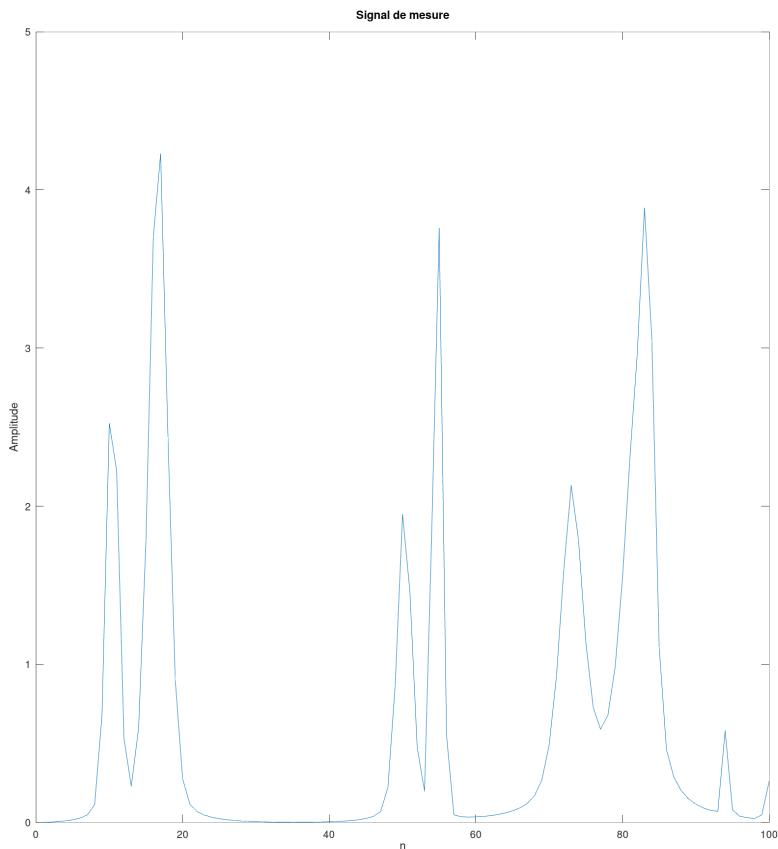


FIGURE 16 – Signal  $x$

2. Après calcul on obtient une variance égale à  $\frac{E_x}{100}$ .
3. Le signal en présence de bruit à l'allure suivante :

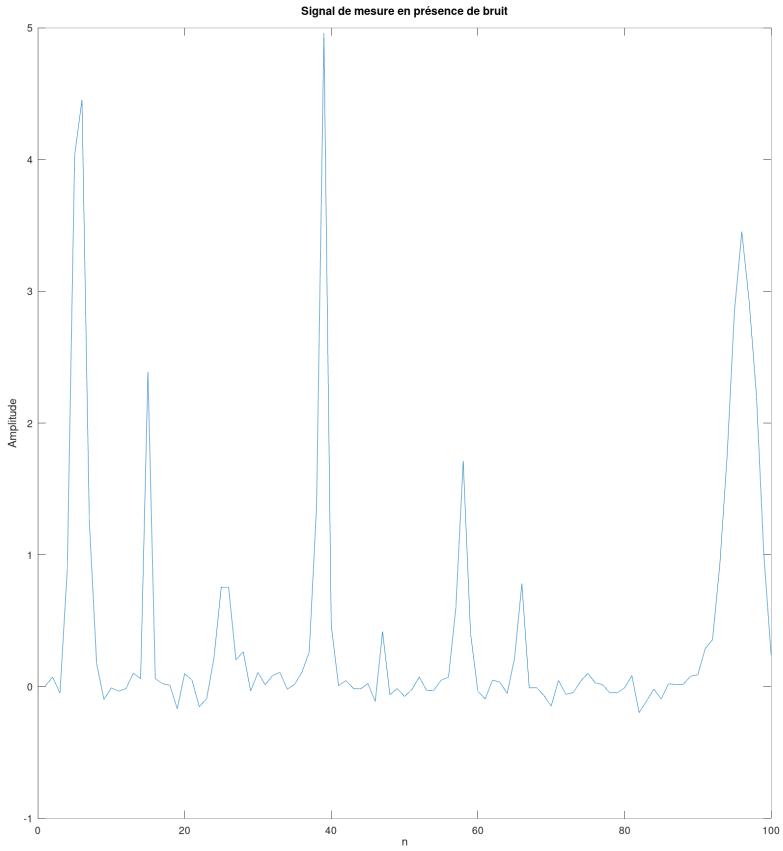


FIGURE 17 – Signal bruité

4. Le modèle de mesure peut s'écrire sous la forme  $y = Hx + e$  avec  $H = I_n$ .

### 3.2 Débruitage du signal

1. Par minimisation d'un critère de moindre carré on obtiendrait une solution égale au signal bruité lui même ce qui n'est pas voulu, il faut introduire un autre critère.
2.  $D$  est une matrice bidiagonale composée de 1 sur la diagonale principale et de -1 sur la diagonale inférieure.

On a  $\nabla f_1(x) = (D + D^T)x$  Or on montre que  $D + D^T$  est inversible donc la solution est obtenue pour  $x^* = (0, \dots, 0)$  ( $f_1$  est à valeurs positives c'est donc bien un minimum).

3. La solution du problème d'optimisation du critère  $f_\lambda$  est :

$$x^* = (I + \lambda D^T D)^{-1}y$$

En utilisant une méthode de plus forte pente à pas variable et pour  $lambda = 0.5$ , on obtient les résultats suivants :

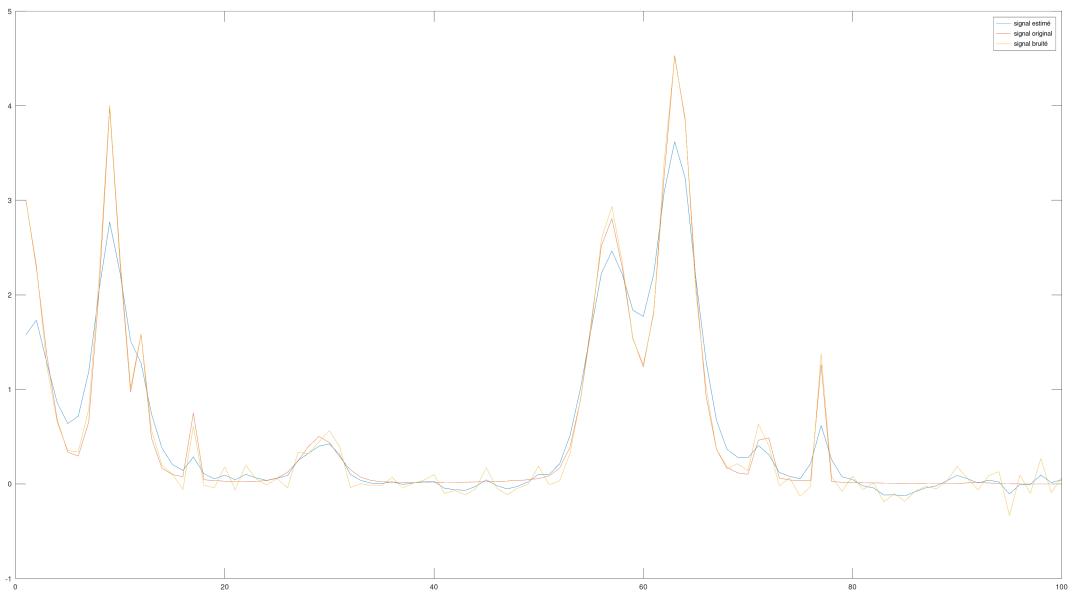


FIGURE 18 – Résultats pour  $\lambda = 0.5$

4. Représentons l'évolution de l'erreur pour différentes valeurs de  $\lambda$  :

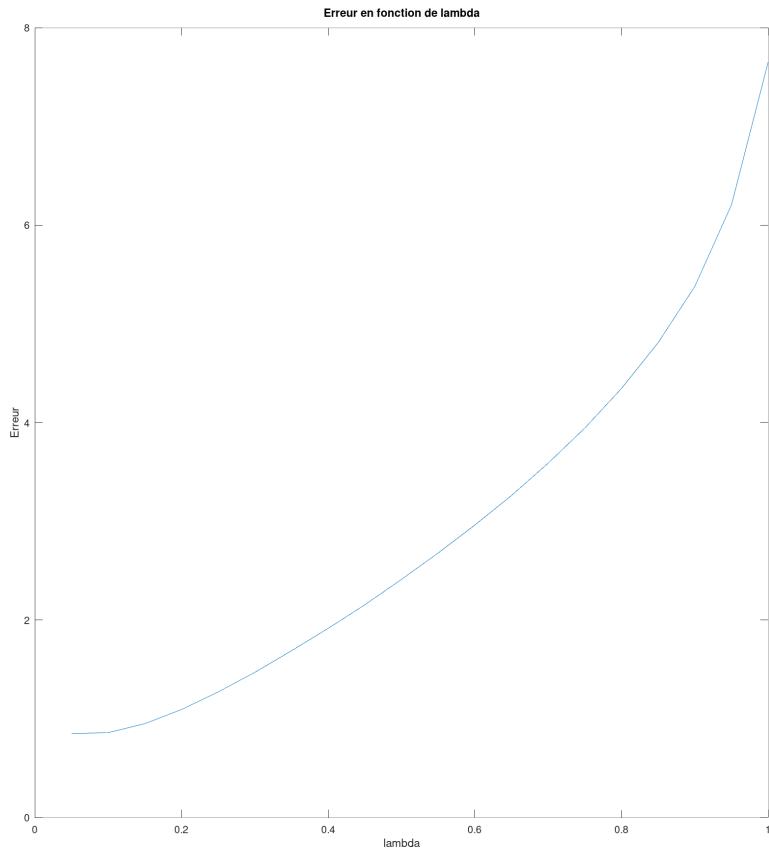


FIGURE 19 – Évolution de l'erreur en fonction de  $\lambda$

En terme d'erreur d'estimation, les meilleurs résultats sont obtenus pour de faibles valeurs de lambda, c'est-à-dire pour un problème de minimisation de  $f_0$  plus important que celui de la minimisation de  $f_1$ , cependant comme précisé en 2. nous n'obtiendrons pas un signal debruité pour ces valeurs de  $\lambda$ . Il faut prendre en compte les deux contraintes.

5. Traçons les valeurs de  $f_0$  en fonction de celles de  $f_1$  :

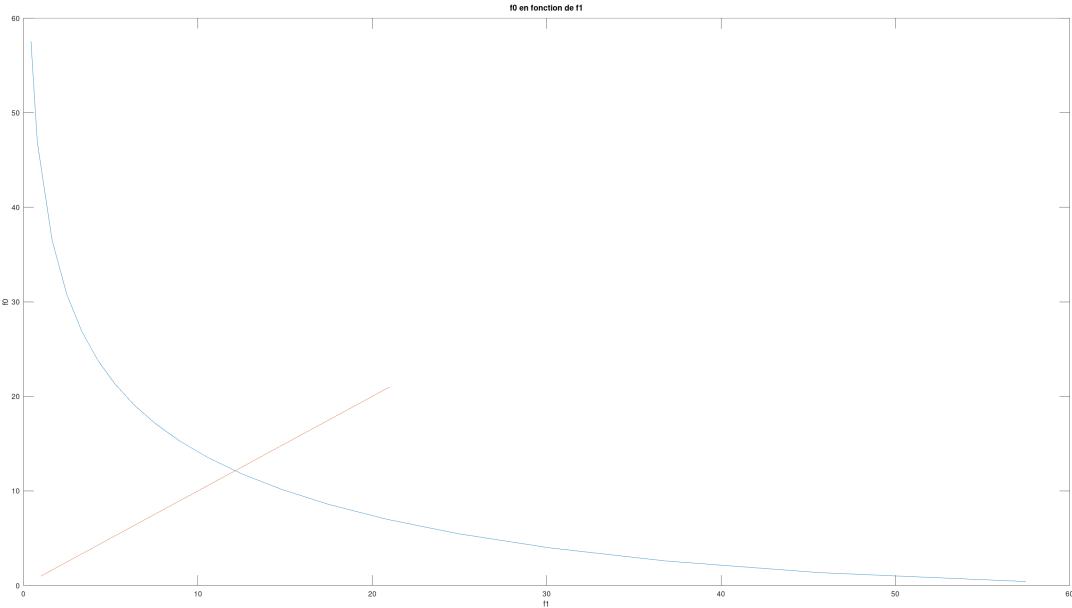


FIGURE 20 – Valeurs de  $f_0$  en fonction de celles de  $f_1$

On peut considérer que la meilleure valeur de  $\lambda$  correspond au point d'intersection de la courbe avec la première bissectrice. En effet, on accorde dans ce cas la même importance aux deux critères.

On obtient alors pour valeur optimale :  $\lambda = 0.45$ .

# A Minimisation de la fonction de Rosenbrock

## A.1 rosenbrock.m

```
1 function [y, g, h, j] = rosenbrock(x, params)
2     x1 = x(1);
3     x2 = x(2);
4     b = params.b;
5
6     y = b * (x2 - x1 ^ 2) ^ 2 + (1 - x1) ^ 2;
7     g = [-4 * x1 * b * (x2 - x1 ^ 2) - 2 * (1 - x1); 2 * b * (x2 - x1 ^ 2)];
8     h = [-4 * b * (x2 - x1 ^ 2) + 8 * b * x1 ^ 2 + 2, -4 * b * x1; -4 * b * x1, 2 *
b];
9     j = [-2 * x1 * sqrt(2 * b), sqrt(2 * b); -sqrt(2), 0];
10 end
```

## A.2 optimdescent.m

```
1 function [xh, result, xval] = optimdescent(critfun, params, options, x0)
2 %
3 % critfun : nom du fichier .m ?valuant la fonction objectif, son gradient et
4 % hessien
5 % params : param?tres pour l?'valuation de la fonction objectif
6 % options : options n?cessaires pour la mise en oeuvre des algorithmes
7 %     options.method : 'gradient', 'gradient conjugue', 'newton', 'Quasi-Newton',
8 % ...
9 %     options.pas : 'fixe', 'variable'
10 %    options.const : constante d'armijo
11 %    options.beta : taux de rebroussement
12 %    options.tolX, options.tolF, options.tolG, options.maxiter
13 %    options.pasInit, valeur du pas initial
14 %    options.lambda, valeur de lambda pour la methode de levenberg-marquardt
15 % x0 : point initial
16 % xh : point final
17 % result : structure contenant les r?sultats
18 %     result.iter : nombre d'iterations r?alis?es
19 %     result.crit : valeur du crit?re par iteration
20 %     result.grad : norm du gradient par iteration
21 %     result.temp : temps de calcul
22 %     result.stop : condition d'arret ('TolX', 'TolG', 'TolF', 'Maxiter')
23 % xval : valeurs des it?r?es
24 %
25 tic
26
27 k = 1;
28 x = x0;
29 stop = 0;
30 fval = [];
31 gval = [];
32 xval = [];
33
34 while not(stop)
35     [y, g, h, j] = feval(critfun, x, params);
36     if strcmp(options.method, 'gradient')
37         d = -g / norm(g);
38     elseif strcmp(options.method, 'gradient_conjugue')
39         if k > 1
40             x_old = xval(:, length(xval)-1);
41             x_new = x;
42             [y, g_old, h, j] = feval(critfun, x_old, params);
43             g = g - g_old;
44             h = h - h_old;
45             j = j - j_old;
46         end
47         % Calcul de la direction de descente
48         % ...
49         % Calcul de la norme du gradient
50         % ...
51         % Calcul de la condition d'arret
52         % ...
53     end
54     x = x + d;
55     fval = [fval; y];
56     gval = [gval; g];
57     xval = [xval; x];
58     k = k + 1;
59 end
```

```

41 [y, g_new, h, j] = feval(critfun, x_new, params);
42 beta = (norm(g_new)/norm(g_old))^2;
43 d = -g_new+beta*d;
44 else
45     d = -g;
46 end
47 elseif strcmp(options.method, 'newton')
48     d=inv(h)*g;
49 elseif strcmp(options.method, 'BFGS')
50     if k>1
51         x_old=xval(:,length(xval)-1);
52         x_new=x;
53         [y1, g_old, h, j] = feval(critfun, x_old, params);
54         [y2, g_new, h, j] = feval(critfun, x_new, params);
55         s=x_new-x_old;
56         y3=g_new-g_old;
57         B=B-(B*s*s'*B)/(s'*B*s)+(y3*y3')/(y3'*s);
58         d=inv(B)*g_new;
59     else
60         B=norm(g)*eye(length(x));
61         d=inv(B)*g;
62     end
63 elseif strcmp(options.method, 'gauss-newton')
64     d = -(j'*j)^-1*g;
65 elseif strcmp(options.method, 'levenberg-marquardt')
66     A = j'*j;
67     d = -(A+options.lambda*[A(1,1) 0; 0 A(2,2)])^-1*g;
68 else
69     error("Methode non reconnue")
70 end
71 if d'*g>0
72     d=d;
73 end
74 if strcmp(options.pas, "fixe")
75     alpha = options.pasInit;
76 elseif strcmp(options.pas, "variable")
77     alpha = pas(critfun, options.beta, d, options.const, x, options, params);
78 else
79     error("Methode de choix du pas non reconnue")
80 end
81 xval = [xval, x];
82 fval = [fval, y];
83 gval = [gval, norm(g)];
84 x = x + alpha * d;
85 nb_col = size(xval);
86 nb_col = nb_col(2);
87 if k > options.maxiter
88     stop = 1;
89     result.stop = "Maxiter";
90 elseif norm(g) < options.tolG
91     stop = 1;
92     result.stop = "TolG";
93 elseif length(fval) > 1 && abs(y - fval(k-1)) / abs(fval(k-1)) < options.tolF
94     stop = 1;
95     result.stop = 'TolF';
96 elseif nb_col > 1 && norm(x - xval(:,k-1)) / norm(xval(:,k-1)) < options.tolX
97     stop = 1;

```

```

98         result.stop = 'TolX';
99     end
100    k = k + 1;
101 end
102
103 result.iter = k;
104 result.crit = fval;
105 result.grad = gval;
106 xh = x;
107 result.temps = toc;
108 end
109
110 function alpha = pas(critfun, beta, d, c, x, options, params)
111     alpha=options.pasInit;
112     [y, g, h, j] = feval(critfun, x, params);
113     [y1, g, h, j] = feval(critfun, x+alpha*d, params);
114     while y1>y+c*alpha*g'*d
115         alpha=beta*alpha;
116         [y1, g, h, j] = feval(critfun, x+alpha*d, params);
117     end
118 end

```

### A.3 main\_tpcsopt\_optim1.m

```

1 close all
2 clear all
3
4 % DEFINITION DES PARAMETRES DU PROBLEME
5 params.fonction='capteur';
6 params.b = 2;
7 options.maxiter = 10 ^ 3;
8 options.tolX = 10 ^ (-8);
9 options.tolG = 10 ^ (-8);
10 options.tolF = 10 ^ (-8);
11 options.method='newton';
12 options.pas='variable';
13 options.pasInit = 1;
14 options.beta = 0.75;
15 options.const = 10 ^ (-4);
16 options.lambda = 0;
17
18 % OPTIMISATION
19 x0 = [0.5; 2];
20 [xh, result, xval] = optimdescent(params.fonction, params, options, x0);
21
22 % AFFICHAGE DES RESULTATS
23 fprintf("xh = (%0.2f, %0.2f)\n", xh(1), xh(2))
24 if isequal(params.fonction, 'rosenbrock')
25     visualisation
26     hold on;
27     plot(xval(1,:),xval(2,:), 'b.');
28     plot(xh(1),xh(2), 'rx');
29 elseif isequal(params.fonction, 'capteur')
30     visualisation_2
31     hold on;
32     plot(xval(1,:),xval(2,:), 'ko');
33     plot(xh(1),xh(2), 'ro');
34     figure;
35     visualisation_3(xh(1),xh(2));
36 end

```

## A.4 visualisation.m

```
1 x1 = -4:0.05:4;
2 x2 = -5:0.05:20;
3 F = zeros( length(x2) , length(x1));
4 params.b = 2;
5 for n1 = 1:length(x1)
6     for n2 = 1:length(x2)
7         x = [x1(n1); x2(n2)];
8         y = rosenbrock(x, params);
9         F(n2, n1) = y(1);
10    end
11 end
12
13 subplot(2,1,1);
14 meshc(x1, x2, F);
15 L = 100;
16 subplot(2,1,2);
17 contour(x1, x2, F, L);
18 colorbar
```

## A.5 comparaison.m

```
1 close all
2 clear all
3
4 params.fonction='capteur';
5 params.b = 2;
6 options.maxiter = 10 ^ 4;
7 options.tolX = 10 ^ (-8);
8 options.tolG = 10 ^ (-8);
9 options.tolF = 10 ^ (-8);
10 options.pas='variable';
11 options.pasInit = 0.01;
12 options.beta = 0.75;
13 options.const = 10 ^ (-4);
14 options.lambda = 0.001;
15
16 x0 = [-0.5; 2];
17 methodes = {'gradient'; 'gradient_conjugue'; 'newton'; 'BFGS'; 'gauss-newton'; 'levenberg-marquardt'};;
18
19 % figure(1)
20 % hold on
21 % title('Evolution du critere')
22 % xlabel('iterations')
23 % ylabel('critere')
24
25 for i = 1:6
26     options.method = methodes(i);
27     [xh, result, xval] = optimdescent(params.fonction, params, options, x0);
28
29     % critere = zeros(result.iter-1);
30     % grad = zeros(result.iter-1);
31     % for i = 1:length(critere)
32     %     [y, g, h, j] = rosenbrock(xval(:, i), params);
33     %     critere(i) = y; grad(i) = norm(g);
34     % end
35     % plot(critere)
36
```

```

37   fprintf('M thode : %s\n', char(options.method))
38   fprintf("Cause de fin : %s\n", result.stop)
39   fprintf("xh = (%0.2f, %0.2f)\n", xh(1), xh(2))
40   fprintf("Nb iterations : %i\n", result.iter)
41   fprintf("Temps : %f\n\n", result.temps)
42 end

```

## B Ajustement d'une courbe non-linéaire

### B.1 capteur.m

```

1 function [f,g,h,J] = capteur(x,params)
2 %
3 % x : valeur de la variable d'optimisation
4 % params : structure contenant les param?tres de la fonction objectif
5 % f,g,H,J : fonction objectif , gradient , hessian et Jacobien
6 %
7 %
8 E = [5013 2415 1558 1000 820 621 433 201 105 55];
9 R = [0.141 0.329 0.525 0.970 1.140 1.511 2.362 5.224 12.826 25.512];
10 R=R/10^3;
11 f = 0;
12 g = [0;0];
13 h = zeros(2,2);
14 J = zeros(2,2);
15 for i=1:length(E)
16     f = f + 0.5*(R(i)-x(2)*E(i)^(-x(1)))^2;
17     g(1) = g(1) + (R(i)-x(2)*E(i)^(-x(1)))*x(2)*log(E(i))*E(i)^(-x(1));
18     g(2) = g(2) - (R(i)-x(2)*E(i)^(-x(1)))*E(i)^(-x(1));
19     h(1,1) = h(1,1)+ x(2)*log(E(i)) * (-log(E(i))*E(i)^(-x(1))*(R(i)-x(2)*E(i)^(-x(1)))) + x(2)*log(E(i))*E(i)^(-x(1));
20     h(1,2) = h(1,2) + E(i)^(-x(1))*log(E(i))*(R(i)-2*x(2)*E(i)^(-x(1)));
21     h(2,1) = h(1,2);
22     h(2,2) = h(2,2)+E(i)^(-x(1)*2);
23 end
24 end

```

### B.2 visualisation\_2.m

```

1 pas = 0.01;
2 alpha = 0:pas:2;
3 beta = 0:pas:4;
4 params = 0;
5
6 F = zeros(length(beta), length(alpha));
7 for n1 = 1:length(alpha)
8     for n2 = 1:length(beta)
9         x = [alpha(n1); beta(n2)];
10        y = capteur(x, params);
11        F(n2, n1) = y(1);
12    end
13 end
14
15 subplot(2,1,1);
16 meshc(alpha, beta, F)
17 L = 100;
18 subplot(2,1,2);
19 contour(alpha, beta, log(F), L)
20 colorbar

```

### B.3 visualisation\_3.m

```
1 function visualisation_3(alpha , beta)
2     E = [5013 2415 1558 1000 820 621 433 201 105 55];
3     R = [0.141 0.329 0.525 0.970 1.140 1.511 2.362 5.224 12.826 25.512];
4     R=R/10^3;
5     plot(E,R,'-');
6     E_theo=0:100:6000;
7     y_theo = beta*E_theo.^-alpha ;
8     hold on;
9     plot(E_theo,y_theo,'-r');
10 end
```

## C Débruitage d'un signal par minimisation d'un critère composite

### C.1 ex3.m

```
1 close all
2 clear all
3
4 n = 100;
5
6 x = simsignal(n, 'spectre');
7
8 figure(1)
9 plot(x)
10 title('Signal de mesure')
11 xlabel('n')
12 ylabel('Amplitude')
13
14 E = std(x)^2;
15 variance = E / 100;
16 b = sqrt(variance)*randn([1, n]);
17 y = x + b;
18
19 figure(2)
20 plot(y);
21 title('Signal de mesure en présence de bruit')
22 xlabel('n')
23 ylabel('Amplitude')
24
25 options.maxiter = 10 ^ 3;
26 options.tolX = 10 ^ (-4);
27 options.tolG = 10 ^ (-4);
28 options.tolF = 10 ^ (-4);
29 options.method='gradient';
30 options.pas='variable';
31 options.pasInit = 0.01;
32 options.beta = 0.75;
33 options.const = 10 ^ (-4);
34
35 critfun = 'critere_debruitage';
36 params.y = y';
37 params.lambda = 0.5;
38 x0 = y';
39
40 [xh, result, xval] = optimdescent(critfun, params, options, x0);
```

```

41
42 figure(3)
43 plot(xh)
44 hold on
45 plot(x)
46 plot(y)
47 legend('signal estim ','signal original','signal bruit ')
48
49 lambdas = 0:0.05:1;
50 err = zeros(1,length(lambdas));
51 xmin = zeros(n,length(lambdas));
52 for i = 1:length(lambdas)
53     params.lambda = lambdas(i);
54     [xh,result,xval] = optimdescent(critfun,params,options,x0);
55     err(i) = sqrt(sum((x'-xh).^2));
56     xmin(:,i) = xh;
57 end
58
59 figure(4)
60 plot(lambdas, err)
61 title('Erreur en fonction de lambda')
62 xlabel('lambda')
63 ylabel('Erreur')
64
65 f0 = zeros(1,length(lambdas));
66 f1 = zeros(1,length(lambdas));
67 D = diag(ones(n, 1)) - diag(ones(n-1,1),-1);
68 for i = 1:length(lambdas)
69     f0(i) = norm(params.y - xmin(:,i)) ^ 2;
70     f1(i) = norm(D * xmin(:,i))^2;
71 end
72
73 figure(5)
74 plot(f1, f0)
75 title('f0 en fonction de f1')
76 xlabel('f1')
77 ylabel('f0')
78 hold on
79 plot(1:length(lambdas),1:length(lambdas))

```

## C.2 critere\_debruitage.m

```

1 function [f,g,h,J] = critere_debruitage(x, params)
2     f0 = norm(params.y - x) ^ 2;
3     n = length(x);
4     D = diag(ones(n, 1)) - diag(ones(n-1,1),-1);
5     f1 = norm(D * x)^2;
6     f = (1 - params.lambda) * f0 + params.lambda * f1;
7     g = -2 * (1 - params.lambda) * (params.y - x) + 2 * params.lambda * D' * D * x;
8     h = 2 * (1 - params.lambda) + 2 * params.lambda * D' * D;
9     J = [];
10 end

```