CSOPT : Calcul scientifique et optimisation
Travaux pratiques
Optimisation sans contraintes

Yassine Jamoud Samy Haffoudhi

29 octobre 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 :

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

   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 $f_0(x) = \frac{1}{2}r_1^2(x) + \frac{1}{2}r_2^2(x)$

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

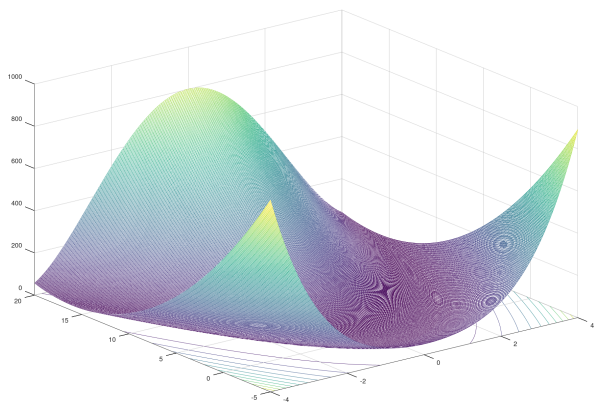
   Le Jacobien s'exprime comme $J = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} \\ \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} \end{bmatrix}$

   Après calcul on obtient alors :

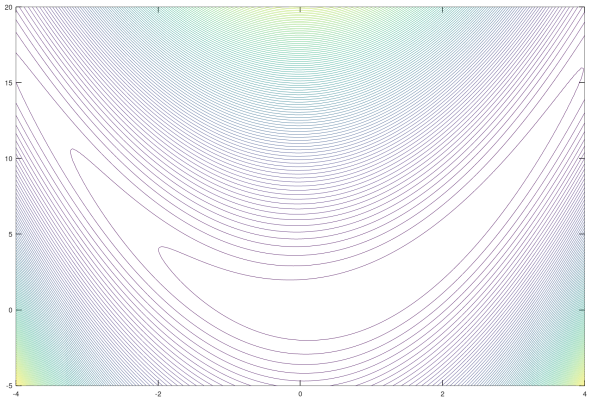
$$J = \begin{bmatrix} -2x_1\sqrt{2b} & \sqrt{2b} \\ \\ -\sqrt{2} & 0 \end{bmatrix}$$

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

1. 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 :

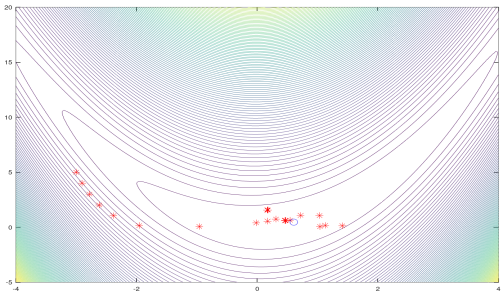


FIGURE 2 – Descente de pas 1

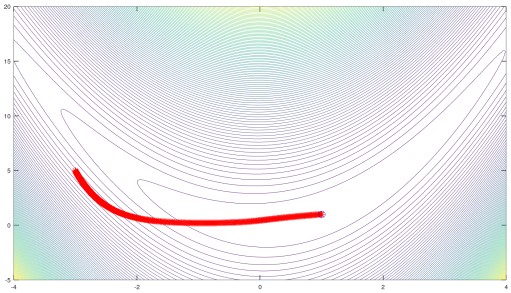


FIGURE 3 – Descente de pas 10^{-2}

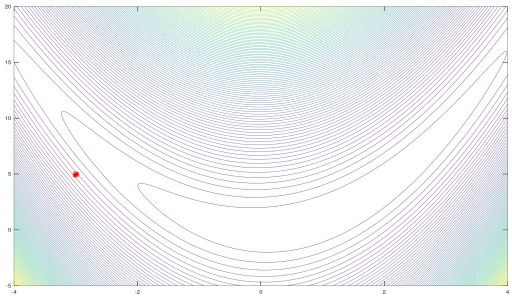


FIGURE 4 – Descente de pas 10^{-4}

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 $c = 10^{-4}$.

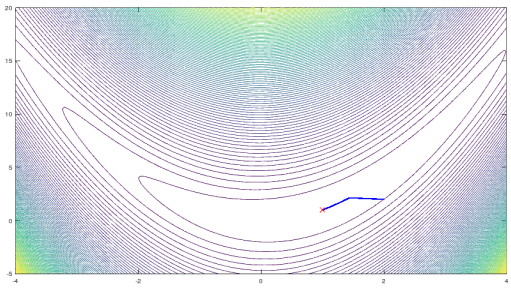


FIGURE 5 – technique de rebroussement

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 :

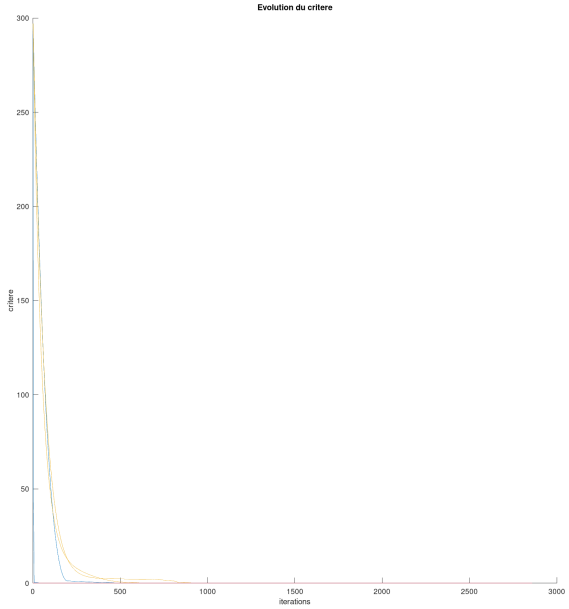


FIGURE 6 – Évolution du critère

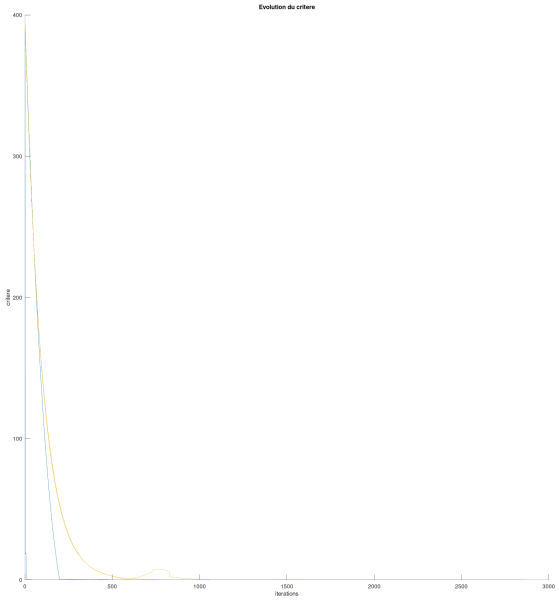


FIGURE 7 – Évolution de la norme du gradient

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

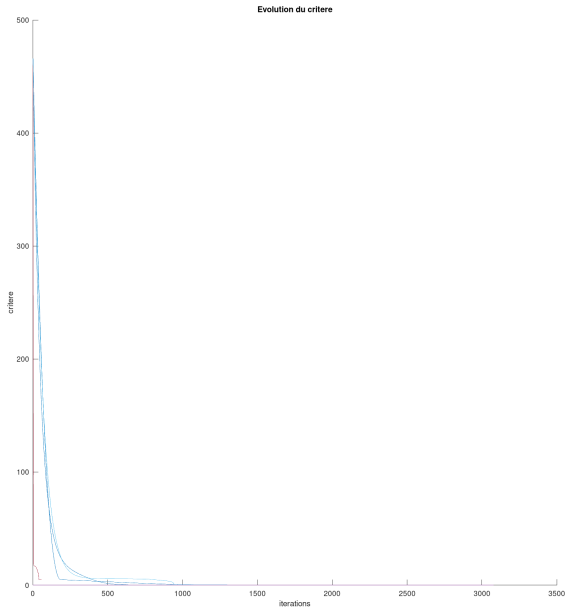


FIGURE 8 – Évolution du critère

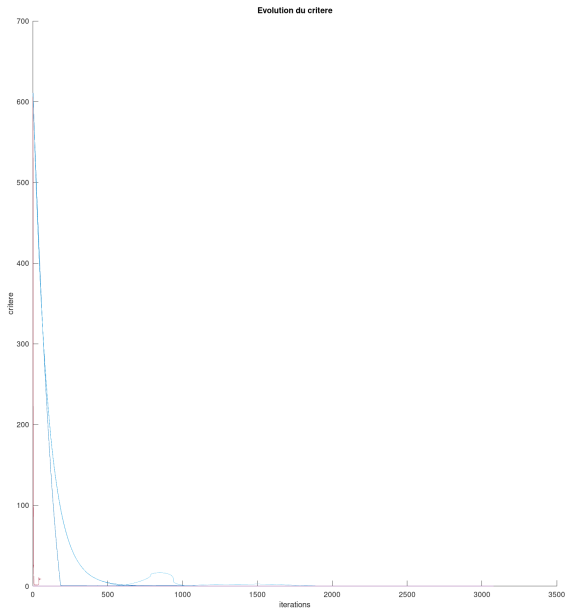


FIGURE 9 – Évolution de la norme du gradient

Nous avons cependant plusieurs problèmes d'affichage de ces graphiques sous Octave (nous n'arrivons par exemple pas à afficher la légende ou à interagir avec les tracés). On observe tout de même 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 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 λ, la méthode est alors quasiment équivalente à celle de Gauss-Newton.

2 Ajustement d'une courbe non-linéaire

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

4 Inversion numérique d'une transformée de Laplace par optimisation sous contraintes

A Minimisation de la fonction de Rosenbrock

A.1 rosenbrock.m

```
function [y, g, h, j] = rosenbrock(x, params)
    x1 = x(1);
    x2 = x(2);
    b = params.b;

    y = b * (x2 - x1 ^ 2) ^ 2 + (1 - x1) ^ 2;
    g = [-4 * x1 * b * (x2 - x1 ^ 2) - 2 * (1 - x1); 2 * b * (x2 - x1 ^ 2)];
    h = [-4 * b * (x2 - x1 ^ 2) + 8 * b * x1 ^ 2 + 2, -4 * b * x1; -4 * b * x1, 2 *
    b];
    j = [-2 * x1 * sqrt(2 * b), sqrt(2 * b); -sqrt(2), 0];
end
```

A.2 optimdescent.m

```
function [xh,result,xval] = optimdescent(critfun,params,options,x0)
    %
    % critfun : nom du fichier .m ?valuant la fonction objectif, son gradient et
    hessien
    % params : param?tres pour l'?valuation de la fonction objectif
    % options : options n?cessaires pour la mise en oeuvre des algorithmes
    %    options.method : 'gradient', 'gradient conjugue', 'newton', 'Quasi-Newton',
     ...
    %    options.pas : 'fixe', 'variable'
    %    options.const : constante d'armijo
    %    options.beta : taux de rebroussement
    %    options.tolX, options.tolF, options.tolG, options.maxiter
    %    options.pasInit, valeur du pas initial
    %    options.lambda, valeur de lambda pour la m thode de levenberg-marquardt
    % x0 : point initial
    % xh : point final
    % result : structure contenant les r?sultats
    %    result.iter : nombre d'iterations r?alis?es
    %    result.crit : valeur du crit?re par iteration
    %    result.grad : norm du gradient par iteration
    %    result.temp : temps de calcul
    %    result.stop : condition d'arret ('TolX', 'TolG', 'TolF', 'Maxiter')
    % xval : valeurs des it?r?es
    %
    tic

    k = 1;
    x = x0;
    stop = 0;
    fval = [];
    gval = [];
    xval = [];

    while not(stop)
        [y, g, h, j] = feval(critfun, x, params);
        if strcmp(options.method, 'gradient')
            d = -g / norm(g);
        elseif strcmp(options.method, 'gradient_conjugue')
            if k > 1
                x_old = xval(:,length(xval)-1);
                x_new = x;
                [y, g_old, h, j] = feval(critfun, x_old, params);
```

```
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='rosenbrock';
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='gradient';
12 options.pas='fixe';
13 options.pasInit = 1;
14 options.beta = 0.75;
15 options.const = 10 ^ (-4);
16 options.lambda = 0;
17
18 % OPTIMISATION
19 x0 = [2; 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

```
x1 = -4:0.05:4;
x2 = -5:0.05:20;
F = zeros(length(x2), length(x1));
params.b = 2;
for n1 = 1:length(x1)
    for n2 = 1:length(x2)
        x = [x1(n1); x2(n2)];
        y = rosenbrock(x, params);
        F(n2, n1) = y(1);
    end
end

subplot(2,1,1);
meshc(x1, x2, F);
L = 100;
subplot(2,1,2);
contour(x1, x2, F, L);
colorbar
```

A.5 comparaison.m

```
close all
clear all

params.fonction='rosenbrock';
params.b = 2;
options.maxiter = 10 ^ 4;
options.tolX = 10 ^ (-8);
options.tolG = 10 ^ (-8);
options.tolF = 10 ^ (-8);
options.pas='variable';
options.pasInit = 0.01;
options.beta = 0.75;
options.const = 10 ^ (-4);
options.lambda = 0.001;

x0 = [5; 10];
methodes = {'gradient'; 'gradient_conjugue'; 'newton'; 'BFGS'; 'gauss-newton'; '
    levenberg-marquardt'};

% figure(1)
% hold on
% title('Evolution du critere')
% xlabel('iterations')
% ylabel('critere')

for i = 1:6
    options.method = methodes(i);
    [xh,result,xval] = optimdescent(params.fonction,params,options,x0);

    % critere = zeros(result.iter-1);
    % grad = zeros(result.iter-1);
    % for i = 1:length(critere)
    %     [y, g, h, j] = rosenbrock(xval(:,i), params);
    %     critere(i) = y; grad(i) = norm(g);
    % end
    % plot(critere)

```

```
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
```