TP3 : Gradient descent

XU Kaiyuan

December 31st, 2022

Contents

0 Preliminary

The aim is to simulate several descent methods for the solution of a linear system

$$A\mathbf{x} = \mathbf{b} \tag{1}$$

where A is a $(n \times n)$ matrix assumed to be **symmetric** ($A^T = A$) and **positive definite** ($\forall\, \mathbf{x} \neq \mathbf{0}$, $\mathbf{x}^T A\mathbf{x} > 0$) ; $\mathbf{b}$ is a $(n \times 1)$ vector that is assumed to be known.

Proposition 1 After calculation, A is invertible. So we note

$$\overline{\mathbf{x}} = A^{-1}\mathbf{b} \tag{2}$$

the only solution for this linear system.

Proposition 2 If B is a invertible matrix, the matrix

$$A = BB^{-T} \tag{3}$$

is symmetric and positive definite.

1 Contouring

We define the cost function as

$$E(\mathbf{x}) = \|\mathbf{x} - \overline{\mathbf{x}}\|_A^2 \tag{4}$$

with

$$\|\mathbf{x}\|_A^2 = \mathbf{x}^T A \mathbf{x} \tag{5}$$

In the case n = 2, the following program represent in 3-d the cost function $E(x_1, X_2)$ in the case where $A = \begin{bmatrix} 5 & 11 \\ 11 & 25 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

The **MATLAB** code

```
clear all; close all;
A=[2 1;1 2]; b=[0;0];
xopt=inv(A)*b;
% generates A, b and xopt
xa=(-1 :0.01 :1)+xopt(1); ya=(-1 :0.01 :1)+xopt(2);
[XA,YA]=meshgrid(xa,ya);
% generates the 3D curve
z=A(1,1)*XA.*XA+A(2,2)*YA.*YA+A(1,2)*XA.*YA+A(2,1)*YA.*XA ...
    -2*XA*b(1)-2*YA*b(2)+xopt'*b;
% draws the 3D curve
subplot(1,2,1)
contour3(XA,YA,z,0:0.1:2,'k');
subplot(1,2,2)
contour(XA,YA,z,0:0.1:2,'k');
axis("equal")
hold on;
[u,v]=eig(A)
line([0; u(1,1)],[0; u(2,1)]);
line([0; u(1,2)],[0; u(2,2)]);
```

Output **MATLAB**

```
>> tp4

u =

   -0.7071    0.7071
    0.7071    0.7071


v =
```

$$\begin{matrix} 1 & 0 \\ 0 & 3 \end{matrix}$$

We have :

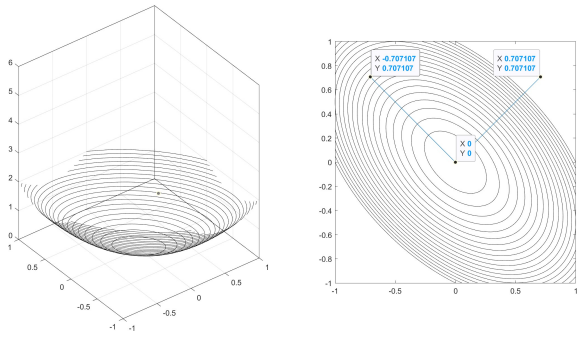


Figure 1: Contouring

Remarks 1 The eigenaxes of the ellipses coincide with the eigendirections of the matrix A.

Remarks 2 The eigenvectors of the matrix A are orthogonal and that its eigenvalues are stricktly positive.

2 Condition number

In the case of a positive defined symmetric matrix,

$$\text{cond}(A) = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{6}$$

Hint We will use the matrix $A = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}$ to generate the matrix A^k with $k = 2, 3, 4$

The **MATLAB** code

```
clear all; close all;
A=[1  0.4;0.4  1]; b=[0;0]; A0=[1  0.4;0.4  1];
for k=1:4
    subplot(1,4,k)
COND(k)=cond(A);
xopt=inv(A)*b;
% generates A, b and xopt
xa=(-1 :0.01 :1)+xopt(1); ya=(-1 :0.01 :1)+xopt(2);
[XA,YA]=meshgrid(xa,ya);
% generates the 3D curve
z=A(1,1)*XA.*XA+A(2,2)*YA.*YA+A(1,2)*XA.*YA+A(2,1)*YA.*XA ... -2*XA*b(1)-2*YA*b(
% draws the 3D curve
%subplot(1,2,1)
%contour3(XA,YA,z,0:0.1:2,'k');
%subplot(1,2,2)
contour(XA,YA,z,0:0.1:2,'k');
axis("equal")
hold on;
[u,v]=eig(A);
line([0; u(1,1)],[0; u(2,1)]);
line([0; u(1,2)],[0; u(2,2)]);
A=A*A0;
end
COND
```

Output **MATLAB**

```
>> conditionnement

COND =

    2.3333    5.4444   12.7037   29.6420
```

We have :

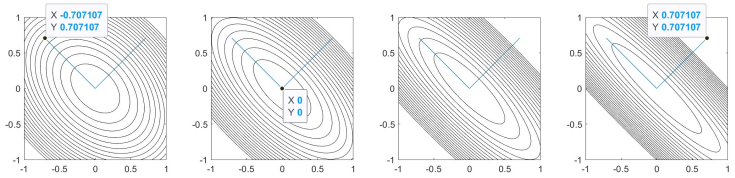


Figure 2: cond(A) | cond(A^2) | cond(A^3) | cond(A^4)

Note Personally, when we increase the cond(A), the rate of Gradient descent will be faster but the accuracy will decrease.

3 Algorithm

We will simulate and comment on the behaviour of the following algorithms. We choose a reasonable condition matrix A, for example

$$\text{cond}(A) \simeq 30 \tag{7}$$

If we choose the matrix $A = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}$ to generate, we have found that $\text{cond}(A^4) = 29.642 \simeq 30$

3.1 Local optimal step size gradient algorithm

Local optimal step size gradient algorithm calculate

$$\mathrm{x}_{k+1} = \mathrm{x}_k + \alpha_k \mathrm{r}_k \tag{8}$$

with a constant step size α_k chosen as

$$\alpha_k = \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{r}_k\|_A^2} \tag{9}$$

The steps of this algorithm are as follows:
1. initialize $\mathbf{x}_0$,
2. calculate, for $k \geq 0$,
2.1 the residual vector

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k \tag{10}$$

2.2 the optimal local step

$$\alpha_k = \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{r}_k\|_A^2} \tag{11}$$

with

$$\|\mathbf{x}\|_A^2 = \mathrm{x}^T A \mathrm{x} \tag{12}$$

2.3 the new candidate

$$\mathrm{x}_{k+1} = \mathrm{x}_k + \alpha_k \mathbf{r}_k \tag{13}$$

The **MATLAB** code

```
clear all; close all;
A=[1 0.4;0.4 1]; b=[0;0];
A=A*A*A*A; % pour un conditionement de cond(A*A)
xopt=inv(A)*b;
% generates A, b and xopt
xa=(-1 :0.01 :1)+xopt(1);
ya=(-1 :0.01 :1)+xopt(2);
[XA,YA]=meshgrid(xa,ya);
% generates the 3D curve
z=A(1,1)*XA.*XA+A(2,2)*YA.*YA+A(1,2)*XA.*YA+A(2,1)*YA.*XA ...
    -2*XA*b(1)-2*YA*b(2)+xopt'*b;
% draws the 3D curve
%contour3(XA,YA,z,0:0.1:2,'k');
contour(XA,YA,z,0:0.1:2,'k');
%si la numerotation des axes est trop petite
%set(gca, "fontsize", 100);
%contour(XA,YA,z,[0:0.1:2],'k');
axis("equal");
hold on;
[u,v]=eig(A)
line([0; u(1,1)],[0; u(2,1)]); % directions propres
line([0; u(1,2)],[0; u(2,2)]);

N=10;
x=ones(2,N);
x(1,1)=1; x(2,1)=-0.5;

for k=1:N
    r=b-A*x(:,k);
    alphak=(r'*r)/(r'*A*r+0.0000001);
    x(:,k+1)=x(:,k)+alphak*r;
    %plot(x(:,k),'o');
    % trajectoire de l'algorithme
    plot([x(1,k),x(1,k+1)],[x(2,k),x(2,k+1)],'or-');
    text(x(1,k),x(2,k),num2str(k)) ;
end
```

Output **MATLAB**

```
>> tp4_3

u =

   -0.7071    0.7071
    0.7071    0.7071


v =

    0.1296         0
         0    3.8416
```

With the initial condition: $\text{cond}(A^4)$, $\mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, $N = 10$, we have :

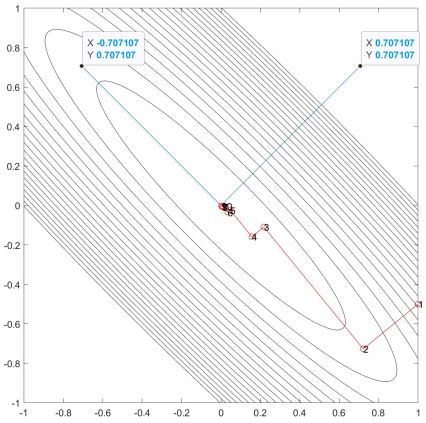


Figure 3: $\text{cond}(A^4)$ | $\mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ | $N = 10$

Test 1: change the vector b Just change the vector $\mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ by $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, and we just do this algorithm for 10 times so with $N = 10$ for both; of crouse cond(A^4) for both.

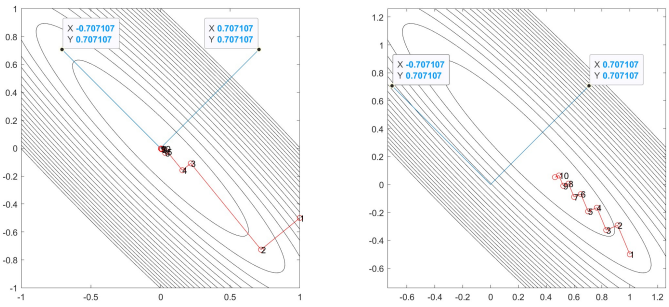


Figure 4: $\mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ | $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$

We find that the initial vector **b** influence the rate of the algorithm. Because with $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, after 10 times, the algorithm has not reached the local minimum.

Test 2: change the cond(A^4) After **Test1**, we keep $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, and $N = 10$. And now we want this algorithm reaching the local minimum within 10 times, we can just increase the cond(A^4) by cond(A^6), for example.

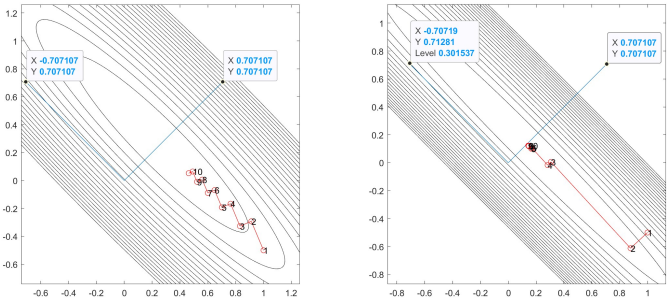


Figure 5: cond(A^4) | cond(A^6)

Test 3: too large $\text{cond}(A^6)$ After **Test2**, we keep $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, we replace $\text{cond}(A^6)$ by $\text{cond}(A^{12})$, we find that even if we do this algorithm for $N = 1000$ times, it can not reach the local minimum :

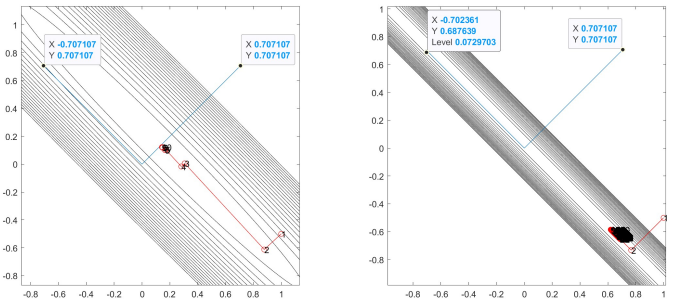


Figure 6: $\text{cond}(A^6)$ | $\text{cond}(A^{12})$

3.2 Constant step size gradient algorithm

Constant step size gradient algorithm calculate

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{r}_k \tag{14}$$

with a constant step size α chosen as follows:
Proposition 1 The constant step size should obey the condition

$$\alpha \in \left(0, \frac{2}{\lambda_{\max}}\right) \tag{15}$$

with $\lambda_{\max} = \max\{\lambda_i\}$, the largest of the eigenvalues of A.
Proposition 2 The constant step that ensures the fastest convergence is

$$\alpha_{opt} = \frac{2}{\lambda_{\min} + \lambda_{\max}} \tag{16}$$

The **MATLAB** code

```
clear all; close all;
A=[1  0.4;0.4  1]; b=[0;0];
A=A*A*A*A*A*A*A; % pour un conditionement de 161.38
xopt=inv(A)*b;
% generates A, b and xopt
xa=(-1:0.01:1)+xopt(1);
ya=(-1:0.01:1)+xopt(2);
[XA,YA]=meshgrid(xa,ya);
% generates the 3D curve
z=A(1,1)*XA.*XA+A(2,2)*YA.*YA+A(1,2)*XA.*YA+A(2,1)*YA.*XA ...
    -2*XA*b(1)-2*YA*b(2)+xopt'*b;
% draws the 3D curve
%contour3(XA,YA,z,0:0.1:2,'k');
contour(XA,YA,z,0:0.1:2,'k');
%si la numerotation des axes est trop petite
%set(gca, "fontsize", 100);
%contour(XA,YA,z,[0:0.1:2],'k');
axis("equal");
hold on;
[u,v]=eig(A)
line([0; u(1,1)],[0; u(2,1)]); %directions propres
line([0; u(1,2)],[0; u(2,2)]);

N=1000;
x=ones(2,N);
x(1,1)=0.3   ; x(2,1)=-0.9;
alpha=0.22; %pas constant
for k=1:N
  r=b-A*x(:,k);
  x(:,k+1)=x(:,k)+alpha*r;
  % trajectoire de l'algorithme
  plot([x(1,k),x(1,k+1)],[x(2,k),x(2,k+1)],'or-');
  text(x(1,k),x(2,k),num2str(k)) ;
end
```

Output **MATLAB**

```
>> pasconstant

u =

   -0.7071    0.7071
    0.7071    0.7071


v =

    0.0467         0
         0    7.5295
```

With the initial condition: $\text{cond}(A^6)$, $\mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, $N = 1000$, we have :

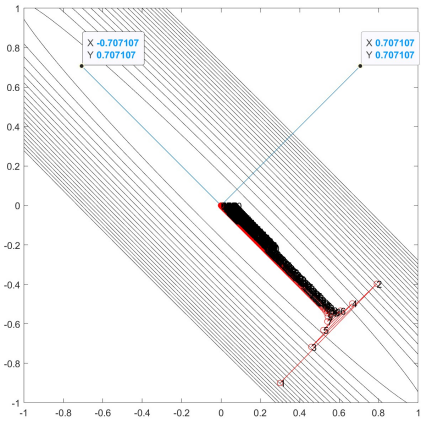


Figure 7: $\text{cond}(A^6)$ | $\mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ | $N = 1000$ | $\alpha = 0.22$

Remark Here the $\alpha \in \left(0, \frac{2}{\lambda_{\max}}\right) = \left(0, \frac{2}{7.5295}\right) = (0, 0.2656)$. For us, we choose $\alpha = 0.22 \in (0, 0.2656)$, so this algorithm works.

Test 1: case $\alpha = \alpha_{opt}$ We know that for us, $\alpha_{opt} = \frac{2}{\lambda_{\min}+\lambda_{\max}} = \frac{2}{0.0467+7.5295} = 0.2640$, we test it just replace $\alpha = 0.22$ by $\alpha = \alpha_{opt} = 0.2640$ without changing other conditions :

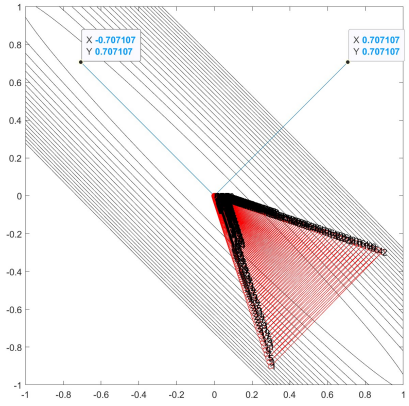


Figure 8: $\text{cond}(A^6) \mid \mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T \mid N = 1000 \mid \alpha = \alpha_{opt} = 0.2640$

Test 2: case $\alpha = 0.3 > \frac{2}{\lambda_{\max}} = 0.2656$ After **Test 1**, we replace $\alpha = \alpha_{opt} = 0.2640$ by $\alpha = 0.3$, we do this algorithm just for **5** times, we find that the vector $\mathbf{x}_k$ diverge very fast :

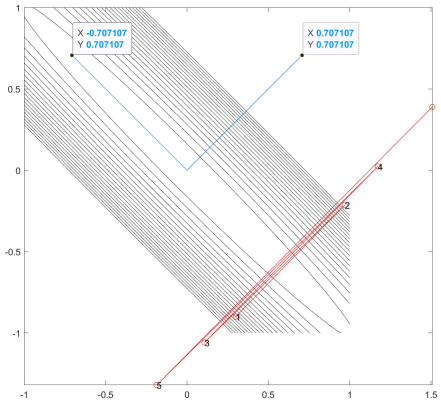


Figure 9: $\text{cond}(A^6) \mid \mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T \mid N = 5 \mid \alpha = 0.3 > \frac{2}{\lambda_{\max}}$

3.3 Conjugate gradient algorithm

Conjugate gradient algorithm calculate

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{17}$$

with a constant step α_k chosen as

$$\alpha_k = \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{p}_k\|_A^2} \tag{18}$$

the direction of descent equal to

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{p}_k \tag{19}$$

and the β_{k+1} chosen as

$$\beta_{k+1} = \frac{\|\mathbf{r}_{k+1}\|^2}{\|\mathbf{r}_k\|^2} \tag{20}$$

The steps of this algorithm are as follows:
1. Choose $\mathbf{x}_0$ and $\mathbf{p}_0 = \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
2. for $k \geq 0$, calculate
2.1 the optimal local descent step size

$$\alpha_k = \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{p}_k\|_A^2} \tag{21}$$

2.2 the new candidate

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{22}$$

2.3 the gradient

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k \tag{23}$$

2.4 the parameter

$$\beta_{k+1} = \frac{\|\mathbf{r}_{k+1}\|^2}{\|\mathbf{r}_k\|^2} \tag{24}$$

2.5 the direction of descent

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{p}_k \tag{25}$$

The **MATLAB** code

```
clear all; close all;
A=[1  0.4;0.4  1]; b=[0;0];
A=A*A*A*A; % pour un conditionement de cond(A*A)
xopt=inv(A)*b;
```

```
% generates A, b and xopt
xa=(-1 :0.01 :1)+xopt(1);
ya=(-1 :0.01 :1)+xopt(2);
[XA,YA]=meshgrid(xa,ya);
% generates the 3D curve
z=A(1,1)*XA.*XA+A(2,2)*YA.*YA+A(1,2)*XA.*YA+A(2,1)*YA.*XA ...
    -2*XA*b(1)-2*YA*b(2)+xopt'*b;
% draws the 3D curve
%contour3(XA,YA,z,0:0.1:2,'k');
contour(XA,YA,z,0:0.1:2,'k');
%si la numerotation des axes est trop petite
%set(gca, "fontsize", 100);
%contour(XA,YA,z,[0:0.1:2],'k');
axis("equal");
hold on;
[u,v]=eig(A)
line([0; u(1,1)],[0; u(2,1)]); % directions propres
line([0; u(1,2)],[0; u(2,2)]);

N=10;
x=ones(2,N);
x(1,1)=1; x(2,1)=-0.5;
r(:,1)=b-A*x(:,1);
p(:,1)=b-A*x(:,1);
for k=1:N
    %r(:,k)=b-A*x(:,k);
    alphak(k)=(r(:,k)'*r(:,k))/(p(:,k)'*A*p(:,k)+0.0000001);
    x(:,k+1)=x(:,k)+alphak(k)*p(:,k);
    %plot(x(:,k),'o');
    % trajectoire de l'algorithme
    plot([x(1,k),x(1,k+1)],[x(2,k),x(2,k+1)],'or-');
    text(x(1,k),x(2,k),num2str(k)) ;
    r(:,k+1)=r(:,k)-alphak(k)*A*p(:,k);
    beta(k+1)=(r(:,k+1)'*r(:,k+1))/(r(:,k)'*A*r(:,k)+0.0000001);
    p(:,k+1)=r(:,k+1)+beta(k+1)*p(:,k);
end
```

Output **MATLAB**

```
>> conju

u =

   -0.7071    0.7071
    0.7071    0.7071
```

v =

$$\begin{matrix} 0.1296 & 0 \\ 0 & 3.8416 \end{matrix}$$

With the initial condition: $\text{cond}(A^4)$, $\mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, $N = 10$, we have :

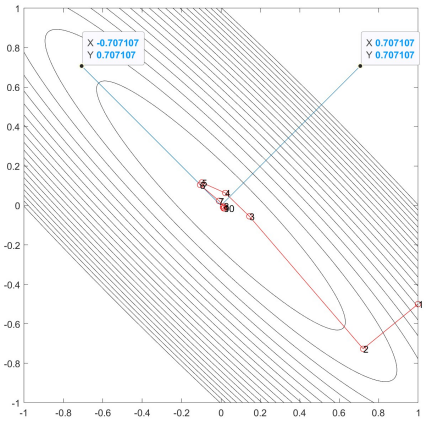
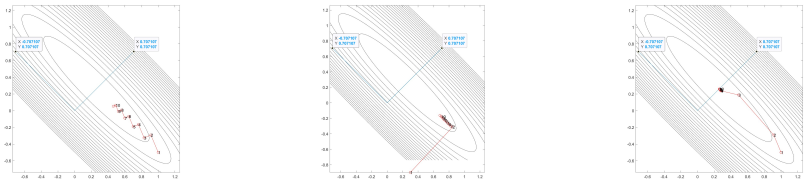


Figure 10: $\text{cond}(A^4)$ | $\mathbf{b} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ | $N = 10$

Comparison With the initial condition: $\text{cond}(A^4)$, $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, $N = 10$, we want to compare three algorithms above. By the way, we choose $\alpha = \alpha_{opt} = 0.2640$ in the constant step size gradient algorithm :



(a) Local optimal step (b) Constant step (c) Conjugate gradient

Figure 11: Comparison with three algorithms above

Conlusion The conjugate gradient algorithm is the best for me, I will choose it.

4 A non-linear example: the Rosenbrock function

The Rosenbrock function is defined on $\mathbb{R}^2$ by

$$f(x,y) = (x-1)^2 + 100\left(y - x^2\right)^2 \tag{26}$$

In the case of a non-quadratic function the constant step size gradient algorithm is written as:
1. initialize $\mathbf{x}_0$ and choose $\alpha = 0.001$ for example
2. calculate for $k \geq 0$,
2.1 the gradient vector

$$\mathbf{r}_k = \nabla f(x,y)_{|\mathbf{x}_k} \tag{27}$$

2.2 the new candidate

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{r}_k \tag{28}$$

We will test it with initial vector $\mathbf{x}_0 = [-1,1]^T$ as follows :

The **MATLAB** code

```
clear all; close all;
x=linspace(-2,2,201);
y=linspace(-1,3,201);
[X,Y] = meshgrid(x,y);
Z = (X-1).^2 + 100*(Y-X.^2).^2;
%ch=contour(X,Y,Z,20);
%clabel(ch);
[C,h]=contour(X,Y,Z,20);
clabel(C,h,'FontSize',8.5,'Color','red')
axis("equal");
hold on;

N=5000;
x=ones(2,N);
x(1,1)=-1   ; x(2,1)=1;
alpha=0.001; %pas constant
for k=1:N
   r(1,k)=2*x(1,k)-2+100*(4*x(1,k).^3-4*x(1,k)*x(2,k));
   r(2,k)=100*(2*x(2,k)-2*x(1,k).^2);
   x(:,k+1)=x(:,k)-alpha*r(:,k);
   % trajectoire de l'algorithme
   plot([x(1,k),x(1,k+1)],[x(2,k),x(2,k+1)],'or-');
   if ismember(k,[1:N/5:N]);
   text(x(1,k),x(2,k)+0.4,num2str(k))  ;
   end
end
```

Output **MATLAB**

>> Rosenbrock

We have :

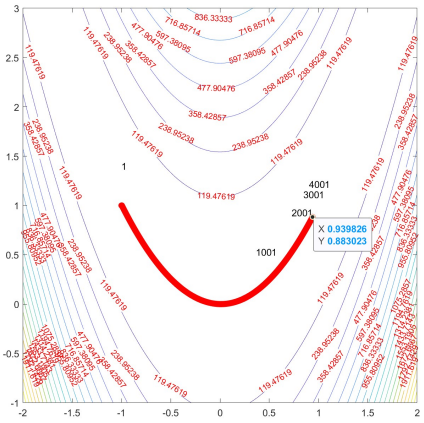


Figure 12: Gradient descent for the Rosenbrock function

Remark

$$\nabla f(x,y) = \begin{bmatrix} 2x - 2 + 100(4x^3 - 4xy) \\ 100(2y - 2x^2) \end{bmatrix} \tag{29}$$

We can find the global minimum in $(1,1)$, that's why we choose a constant step size $\alpha = -0.001 < 0$.

5 Conclusion

We reviewed the invertible matrix;
We learnt to draw the contour line;
Then we studied three algorithms : Local optimal step size gradient algorithm, Constant step size gradient algorithm and Conjugate gradient algorithm.
Finally, we take the Rosenbrock function as an example, we drew the contour using Constant step size gradient algorithm.

Happy New Year

Best wishes