Les valeurs extrêmes: Utilisation du gradient

(Partie 2 de 3)

© Pierre Lantagne
Enseignant retraité du Collège de Maisonneuve

La première version de ce document est parue en février 2006. Ce deuxième document Maple a pour but de présenter une approche numérique dans la recherche d'un extremum relatif d'une fonction de deux variables. Nous allons utiliser le gradient d'une fonction en (a, b) afin de bâtir, point par point, un chemin dans le domaine de la fonction dont les point (x_k, y_k) permettront de se rapprocher aussi près que l'on voudra de celui donnant la valeur optimum de cette fonction.

Bonne lecture à tous !

* Ce document Maple est exécutable avec la version 2020.2

Initialisation

```
> restart;
> with(LinearAlgebra,Norm):
> with(plots,contourplot,display,gradplot,pointplot,pointplot3d,setoptions,
  setoptions3d):
  setoptions(size=[300,300],labels=[x,y],axesfont=[TIMES,ROMAN,8],labelfont=
  [TIMES,ROMAN,8]);
  setoptions3d(size=[400,400],labels=[x,y,z],axes=frame,axesfont=[TIMES,ROMAN,
  8],labelfont=[TIMES,ROMAN,8]):
```

L'initialisation suivante permettra d'avoir plus de lisibilité des nombres décimaux en supprimamt les zéros non significatifs à la fin d'un nombre.

```
> interface(typesetting=extended); # Pour s'assurer le niveau de composition
  étendue
  Typesetting:-Settings(striptrailing=true);
```

$$\mathit{extended}$$

$$\mathit{false} \tag{1.1}$$

```
> with(VectorCalculus,Gradient):
> Digits:=60;
  interface( displayprecision=20 );
```

$$\mathit{Digits} := 60$$

$$20 \tag{1.2}$$

Gradient

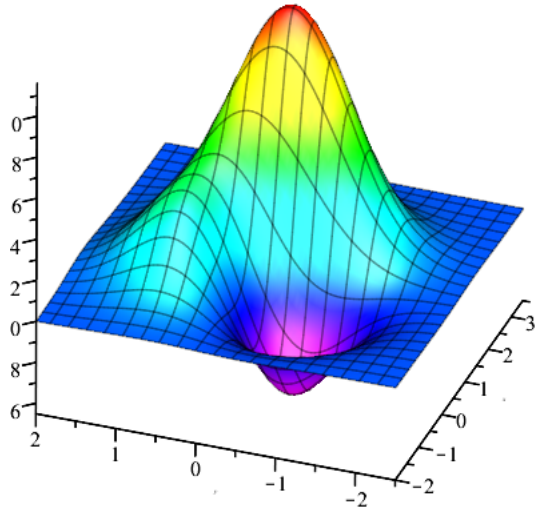
Considérons la fonction g définie par $g(x,y) = x\,e^{-(x-1)^2-\left(y+\frac{1}{4}\right)^2} + y\,e^{-x^2-y^2} + 3.$

```
> g:=(x,y)->x*exp(-(x-1)^2-(y+1/4)^2)+(y)*exp(-x^2-y^2)+3;
```

$$g := (x,y) \mapsto x\cdot e^{-(x-1)^2-\left(y+\frac{1}{4}\right)^2} + y\cdot e^{-x^2-y^2} + 3 \tag{2.1}$$

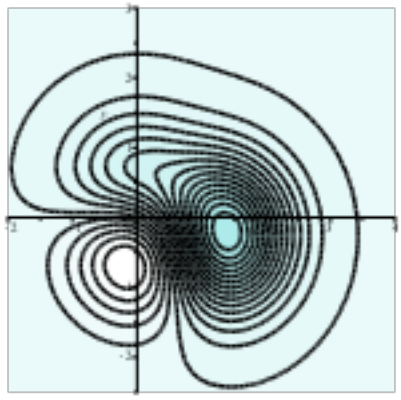
Traçons la surface d'équation $z = g(x, y)$ sur le pavé $[-2, 3.5] \times [-2.5, 2]$.

```
> Surface:=plot3d([x,y,g(x,y)],x=-2..3.5,y=-2.5..2,grid=[80,80]):
  display(Surface,labels=[x,y,z],
          lightmodel=light1,shading=zhue,color=navy,
          axes=framed,font=[TIMES,ROMAN,8],
          orientation=[-160,60]);
```



Cette surface possède clairement des extremums relatifs. Traçons alors des courbes de niveau pour mieux estimer les valeurs x et y qui permettent d'atteindre ces extremums.

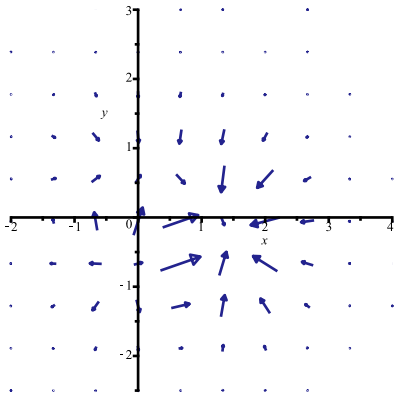
```
> Courbes_niveau:=contourplot(g(x,y),x=-2..4,y=-2.5..3,contours=20,grid=[60,
  60],coloring=[white,turquoise],filled=true):
  Courbes_niveau;
```



Les courbes de niveau nous montrent que le maximum est atteint avec (x, y) au voisinage du point $(1.4, -0.3)$ et que le minimum est atteint avec (x, y) dans le voisinage du point $(-0.2, -0.7)$.

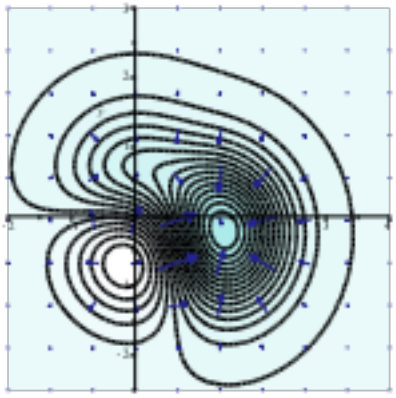
Visualisons le fait que le gradient en un point est toujours perpendiculaire à la courbe de niveau passant par ce point. Obtenons d'abord un champ de vecteurs gradient. La macro-commande `gradplot` de de la bliothèque `plots` sera utilisée.

```
> gp:=gradplot(g(x,y),x=-2..4,y=-2.5..3,grid=[10,10],arrows=slim,color=navy):
  gp;
```



Superposons ensuite ce champ de vecteurs au tracé des courbes de niveau.

```
> display([Courbes_niveau,gp]);
```



Nous observons que ces vecteurs sont toujours perpendiculaires aux courbes de niveau. Sélectionnez le graphique précédent et agrandissez le graphique avec les carrés de redimentionnement. Cette illustration nous indique la direction avec laquelle la fonction possède le plus grand taux de variation. Le sens de ces vecteurs confirme qu'il se présente effectivement un maximum relatif (flèches rentrantes) dans le voisinage du point $(1.4,-0.3)$ et un minimum relatif (flèches sortantes) dans le voisinage du point $(-0.2,-0.7)$.

Pour rechercher les couples (a,b) avec lesquels la fonction g présentera un extremum relatif, appliquons d'abord le test des dérivées secondes. En fait, nous nous contenterons de trouver les valeurs critiques et sur la base du tracé de la surface, nous identifierons la nature de chacun de ces points.

```
> gx:=D[1](g):
  gy:=D[2](g):
```

```
> Sol_symboliques:=solve({gx(x,y)=0,gy(x,y)=0,x<infinity},{x,y});
```

Warning, solutions may have been lost

$$Sol\_symboliques := ( ) \quad \textbf{(2.2)}$$

Il fallait s'y attendre, l'évaluateur n'a trouvé aucune solution à ce système. Plus souvent qu'autrement, il faut plutôt résoudre numériquement de tels systèmes d'équations. Résolvons d'abord numériquement ce système au voisinage de $(1.4,-0.3)$ avec la macro-commande `fsolve`.

```
> Digits:=15:
```

```
> xy_max:=fsolve({gx(x,y)=0,gy(x,y)=0},{x,y},{x=0..1.4,y=-0.3..0});
```

$$xy\_max := \{x = 1.39123548497918, y = -0.196144422243028\} \quad \textbf{(2.3)}$$

Résolvons ensuite numériquement le système au voisinage de $(-0.2,-0.7)$.

```
> xy_min:=fsolve({gx(x,y)=0,gy(x,y)=0},{x,y},{x=-0.2..0,y=-0.7..-0.5});
```

$$xy\_min := \{x = -0.160787316824713, y = -0.689220759504229\} \quad \textbf{(2.4)}$$

Contrôlons si chaque résultat précédent annule respectivement gx et gy.

```
> assign(xy_max);
  gx(x,y);
  unassign('x,y'):
```

$$1.02\,10^{-14} \quad \textbf{(2.5)}$$

```
> assign(xy_min);
  gx(x,y);
  unassign('x,y'):
```

$$1.\,10^{-15} \quad \textbf{(2.6)}$$

Ces valeurs sont satisfaisantes: elles sont presque nulles. Obtenons alors une approximation de ces extremums relatifs.

```
> z_max=g(op([1,2],xy_max),op([2,2],xy_max));
  z_min=g(op([1,2],xy_min),op([2,2],xy_min));
```

$$z\_max = 4.16308097005083$$

$$z\_min = 2.54787530079019 \quad \textbf{(2.7)}$$

Gradient et extremum relatif

Rappelons que le gradient en (a, b) indique la direction du plus grand taux et du plus petit taux de variation de la fonction. Nous allons donc utiliser le vecteur gradient pour rechercher un maximum relatif d'une fonction (en supposant qu'il existe, bien sûr). Soit le raisonnement proposé suivant: avec un point intérieur du domaine $P_k = (x_k, y_k)$, nous allons nous déplacer légèrement dans le domaine de la fonction g, **dans l'orientation donnée par le gradient en** (x_k, y_k), vers un deuxième point du domaine $P_{k+1} = (x_{k+1}, y_{k+1})$. Si, avec le point P_k on n'atteint pas encore le maximum relatif de la fonction, le point P_{k+1} nous permettra par contre de s'en rapprocher puisque nous nous déplacerons dans le sens du vecteur gradient au point P_k pour déterminer le point suivant P_{k+1}.

Transposons ce raisonnement en calcul. Soit donc l'équation vectorielle $OP = OP_k + \lambda\, grad(g)(x_k, y_k)$ de la droite passant par le point P_k et ayant $grad(g)(x_k, y_k)$ pour vecteur directeur. La valeur du pas $\lambda > 0$ déterminera l'importance du déplacement dans le direction de $grad(g)(x_k, y_k)$. Avec une valeur $\lambda > 0$ donnée, on déduira un point P_{k+1} de cette droite. Cet autre point intérieur du domaine de la fonction nous permettra de se rapprocher du point permettant d'atteindre la valeur maximum de la fonction.

Comme les calculs seront numériques, nous considèrerons que le point $P_k = (x_k, y_k)$ permet d'avoir un maximum relatif de la fonction si le gradient en ce point est égal au vecteur $(0, 0)$. Un façon élégante de tester ce fait, c'est de vérifier si la norme de ce vecteur gradient est presque nulle.

Pour les besoins de la cause, le point $P_0 = P(-2, 2)$ sera désigné point de départ. Nous allons traiter ce point et les suivants en tant que vecteurs positions afin de facilter les calculs.

```
> P[0]:=<-2.0|2.0>;
```

$$P_0 := \begin{bmatrix} -2. & 2. \end{bmatrix} \quad \textbf{(3.1)}$$

Contrôlons (ici, par principe) si ce point permet d'atteindre le maximum de la fonction. Utilisons une tolérance de 10^{-8}.

```
> tolérance:=Float(1,-8);
```

$$tolérance := 1.\,10^{-8} \quad \textbf{(3.2)}$$

Calculons maintenant la norme du vecteur gradient en $(-2, 2)$ pour déterminer si nous somme assez près du point (a, b) du domaine donnant la valeur maximum $f(a, b)$.

```
> Grad_longueur:=eval(Norm(Gradient(g(x,y),[x,y]),2),[x=P[0][1],y=P[0][2]]);
```

$$Grad\_longueur := 0.00355492053735411 \quad \textbf{(3.3)}$$

```
> evalb(Grad_longueur<tolérance);
```

$$false \quad \textbf{(3.4)}$$

On a utilisé la macro-commande Norm de la bibliothèque LinearAlgebra pour calculer la norme du gradient au point P(-2,2) du domaine. De plus, rappelons que `evalb` commande une évaluation booléenne. Le résultat vrai pour l'inégalité Grad_longueur < tolerance signifiera que le point $P(x, y)$ est, au niveau de tolérance admis, suffisamment près du point permettant d'avoir le maximum cherché.

Obtenons maintenant un deuxième point avec un pas $\lambda = 0,1$.

```
> lambda:=0.1;
```

$$\lambda := 0.1 \quad \textbf{(3.5)}$$

Obtenons le point P_1.

```
> Grad_P:=convert(eval(Gradient(g(x,y),[x,y]),[x=-2.0,y=2]),Vector[row]):
  P[1]:=P[0]+lambda*Grad_P;
```

$$P_1 := \begin{bmatrix} -1.99973248916151 & 1.99976587919451 \end{bmatrix} \quad \textbf{(3.6)}$$

Contrôlons si le point P_1 permet d'atteindre le maximum recherché (toujours avec la tolérance de 10^{-8}).

```
> Grad_longueur:=eval(Norm(Gradient(g(x,y),[x,y]),2),[x=P[1][1],y=P[1][2]]);
  evalb(Grad_longueur<tolérance);
```

$$Grad\_longueur := 0.00356113360793278$$

$$false \quad \textbf{(3.7)}$$

Il sera beaucoup plus commode d'utiliser une boucle de calcul pour continuer de se déplacer de point en point dans le domaine de la fonction. Prenons soins de limiter le nombre d'itérations à 1000 et rappelons les paramètres suivants qui seront utilisés dans cette boucle.

```
> n:=1000;
  lambda:=0.1;
  tolérance:=0.00000001;
```

$$n := 1000$$

$$\lambda := 0.1$$

$$tolérance := 1.\,10^{-8} \quad \textbf{(3.8)}$$

L'algorithme de cette boucle est le suivant.

1) Évaluer le gradient au point P_k.

2) Calculer la longueur (la norme) de ce vecteur gradient.

3) Contrôler si cette longueur est inférieur à la longueur tolérée.

4) Si tel est le cas, terminer les itérations

5) Sinon, calculer un prochain point P_{k+1}.

```
> P[0]:=<-2.0|2.0>;  # Point d'amorce
```

(3.9)

$$P_0 := \begin{bmatrix} -2. & 2. \end{bmatrix} \quad \textbf{(3.9)}$$

```
> for k from 0 to n-1 do
    Grad_longueur:=eval(Norm(Gradient(g(x,y),[x,y]),2),[x=P[k][1],y=P[k][2]]);

    if Grad_longueur<tolérance then break
     else Grad_P:=convert(eval(Gradient(g(x,y),[x,y]),[x=P[k][1],y=P[k][2]]),
  Vector[row]);
     P[k+1]:=P[k]+lambda*Grad_P;
    fi;
  od:
  gradient=convert(Grad_P,Vector[row]);
  eval(Norm(Gradient(g(x,y),[x,y]),2),[x=P[k][1],y=P[k][2]]);
  Nombre_itérations:=k;
  'P[k]'=P[k];
```

$$gradient = \begin{bmatrix} 3.45856429606606\,10^{-9} & -1.046615595926\,10^{-8} \end{bmatrix}$$

$$8.71640967415443\,10^{-9}$$

$$Nombre\_itérations := 812$$

$$P_k = \begin{bmatrix} 1.39123548367211 & -0.196144418287609 \end{bmatrix} \quad \textbf{(3.10)}$$

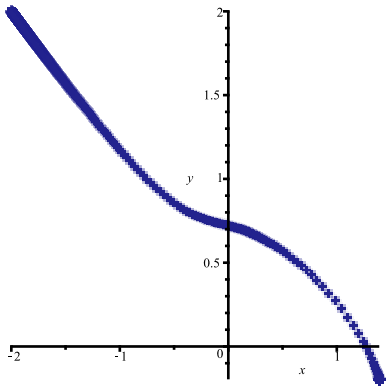
Comparons ce point avec celui xy_max obtenu à la section précédente.

```
> xy_max;
```

$$\{x = 1.39123548497918, y = -0.196144422243028\} \quad \textbf{(3.11)}$$

Traçons, dans le plan, sans les relier, le chemin de points (le point de départ plus les 812 points obtenus par les itérations précédentes).

```
> Courbe:=pointplot([seq(convert(P[k],list),k=0..Nombre_itérations)],color=
  navy):
  Courbe;
```

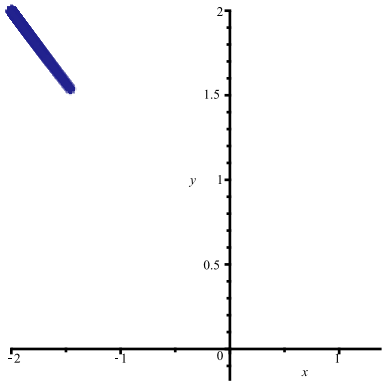


On observe très bien sur le tracé précédent, que plus le taux de variation de la fonction g est important, plus la distance séparant les points augmente. Cela découle du fait que le pas de déplacement λ est constant et que le taux de variation de la fonction g ne l'est pas.

Illustrons ce fait en réalisant une animation du tracé de ce chemin de points. Pour animer la partie significative du chemin de points, l'animation sera construite à partir du 600 ième point jusqu'au 750 ième. Nous

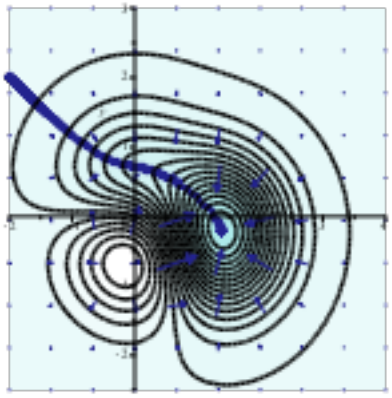
constaterons qu'avec un pas λ constant, plus la distance entre les points augmente, plus l'animation s'accélère.

```
> Animation:=seq(pointplot([seq((convert(P[n],list)),n=0..k)],color=navy),k=
  600..750):
> display(Animation,insequence=true);
```



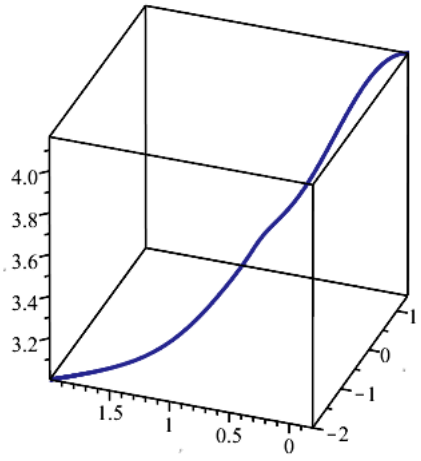
Superposons maintenant le tracé de ces points à ceux du champ de vecteurs gradients et des courbes de niveau. On aura alors une idée claire du chemin à parcourir dans le domaine, à partir du point $(-2, 2)$, pour se rapprocher du point (x, y) qui permet d'atteindre le maximum relatif de la fonction. Notons que le chemin obtenu est toujours, évidemment, perpendiculaire à toutes les courbes de niveau passant par ces points.

```
> display([Courbe,Courbes_niveau,gp]);
```



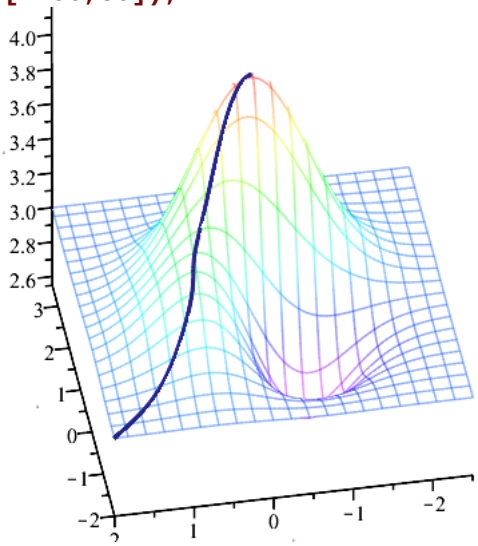
Obtenons l'image de ce tracé en ajoutant, à chacun des points calculés du domaine, une troisième composante (cote) dont la valeur sera celle de l'image par la fonction g. Relions cette fois-ci les points en précisant l'option `connect=true`.

```
> Courbe3d:=pointplot3d([seq([P[n][1],P[n][2],evalf(g(P[n][1],P[n][2]))],n=1..
  Nombre_itérations)],connect=true,thickness=3,axes=boxed,color=navy,
  orientation=[-160,60]):
  Courbe3d;
```



Superposons finalement le tracé de cette courbe à celui de la surface. L'effet est spectaculaire.

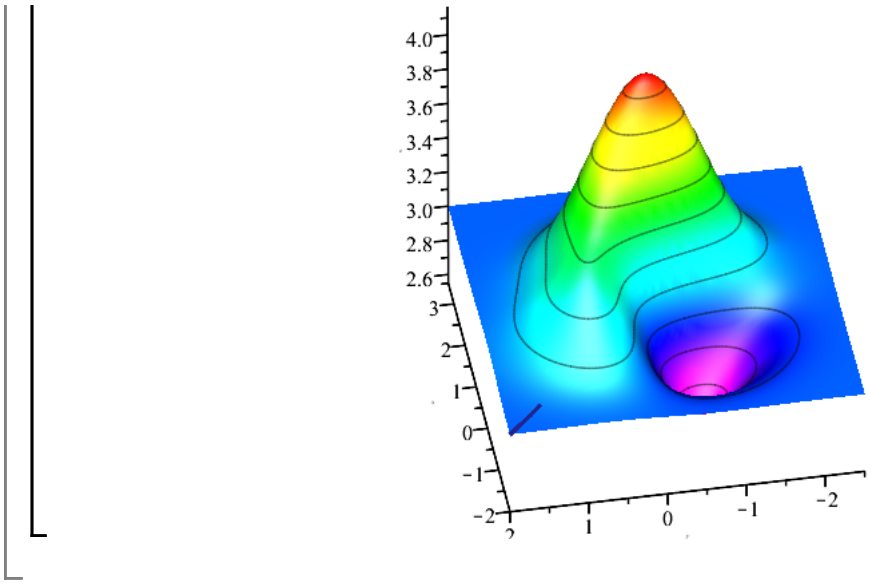
```
> display([Surface,Courbe3d], lightmodel=light2,style=hidden,
          shading=zhue,color=navy,
          orientation=[-190,50]);
```



Pour un effet davantage spectaculaire, animons cette superposition.

```
> Animation:=seq(display([Surface,pointplot3d([seq([P[n][1],P[n][2],evalf(g(P
  [n][1],P[n][2]))],n=0..k)],color=navy,thickness=3,connect=true)]),k=600.
  .750):
```

```
> display(Animation,insequence=true,
          lightmodel=light1,style=patchcontour,shading=zhue,color=navy,
          orientation=[-190,50]);
```


4.0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
3
2
1
0
-1
-2
2
1
0
-1
-2