TP 2 : Méthodes de descente de gradient

On commence toujours par charger les librairies. Les questions avec une étoile sont facultatives (et difficiles). Les parties 1 et 2 sont indépendantes.

```
In [ ]: %pylab inline
```

1. Retour en dimension un

(a) On considère la fonction $f : x \mapsto x^2 - \frac{1}{4}x^4$. On considère les points x_k correspondant à une descente de gradient à pas fixe α pour la fonction f. Programmer numériquement la suite x_k pour des valeurs de α et de x_0 où la convergence est linéaire ou superlinéaire. Illustrer les différents types de convergence à l'aide de graphiques.

(b*) Programmer la suite des x_k dans le cas où elle est convergente mais ne converge pas linéairement. Conjecturez un équivalent de $|x_n|$ lorsque $n \to \infty$. Indication : on pourra tracer $|x_n|$ en fonction de n avec des échelles logarithmiques en abscisses et en ordonnées (pour des valeurs de n assez grandes).

(c) On considère la fonction $f : x \mapsto \frac{x^2}{1+x^2}$. On considère les points x_k correspondant à la méthode de descente de gradient à pas fixe α. Programmer la suite x_k lorsque $x_0 = 3$ et $\alpha = \frac{1}{3}$. La convergence est-elle linéaire ? Si oui, quel est le taux de convergence ?

Discuter de l'intérêt ou non de ce genre de méthode par rapport par exemple à la méthode de la section dorée.

Indication : programmer au moins une centaine de points, et tracer également par exemple $\varphi^{-k}|x_0|$, où $\varphi = \frac{1}{2}(1+\sqrt{5})$ est le nombre d'or...

2. Méthode de gradient à pas fixe, cas test en dimension 2.

On va tester la méthode de descente de gradient sur la fonction $f_a : (x_0, x_1) \mapsto 1 - \frac{1}{1+ax_0^2+x_1^2}$, où $a > 0$ est un paramètre qu'on pourra changer pour voir comment se comporte la méthode en fonction de a.

Attention au changement de notation : ici x_i est la $i$ème coordonnée d'un vecteur $X \in \mathbb{R}^n$. On commence par $i = 0$ pour être cohérent avec la numérotation en python. On notera les itérées X_k. On utilisera deux indices si on veut préciser les coordonnées, par exemple en dimension 2 on écrit $X_k = (x_{0,k}, x_{1,k})$.

(a) Définir f_a en supposant que a est une variable extérieure à f_a. Elle ne doit prendre qu'un argument, la variable correspondant au vecteur X. La valeur de a sera définie en dehors de la fonction.

Tracer les lignes de niveau de la fonction f_a sur $[-5, 5] \times [-5, 5]$ pour différentes valeurs de a (faire plusieurs graphiques). On utilisera la fonction `contour`, qui peut s'appeler par exemple comme ceci : `contour(X0,X1,Z,12)`, où `X0` est de type `array` à une dimension (qui liste les abscisses x_0 des points du maillage), de même pour `X1` (pour les ordonnées), et où `Z` est un tableau de type `array` à deux dimensions qui contient les valeurs des $f_a(X)$, avec $X = (x_0, x_1)$ où x_0 parcourt la liste des abscisses et x_1 parcourt celle des ordonnées. Cela trace 12 lignes de niveau.

On pourra utiliser la commande `axis("scaled")` pour que l'échelle des ordonnées et des abscisses soit la même. Vérifier qu'on n'a pas interverti les abscisses et les ordonnées (on pourra regarder la direction du grand axe des ellipses représentant les courbes de niveau, en fonction de si $a > 1$ ou $a < 1$).

(b) Définir une variable ε qui correspondra à un paramètre d'approximation pour la formule des différences finies à droite (on n'aura pas besoin de le changer souvent, autant ne pas le mettre comme argument dans les méthodes). Quelle valeur prendre pour ε ? Définir une fonction qui prend en argument une fonction f dont on veut approximer le gradient, un point X (sous la forme d'un vecteur) et qui renvoie l'approximation du gradient par différences finies à droite (elle ne doit pas prendre en argument la variable ε, qui est définie en dehors de la fonction).

Vérifier que cela fonctionne quand on l'applique à f_a en un point donné, que cela renvoie bien un vecteur de type `array` qui est proche du gradient au point X.

(c) Définir une fonction qui prend en argument une fonction f à minimiser, un point initial X_0, un pas α, et une tolérance, et qui calcule la suite X_k correspondant à la méthode de descente de gradient à pas fixe pour f. On renverra la liste des X_k. On prendra comme critère d'arrêt le fait que la norme du gradient

est plus petite que la tolérance (utiliser la fonction `norm` pour calculer la norme). Et on se mettra une limite au nombre de passages dans la boucle, au cas où on ne convergerait pas.

La tester sur la fonction f_a. On pourra tracer la suite des points sur le graphique de f_a à l'aide de `plot` et de marqueurs. Vérifier graphiquement que le gradient est bien orthogonal aux lignes de niveau.

Illustrer le taux de convergence de X_k vers 0 par un graphique.

3. Application à la recherche de trajectoires fermées sur un billard.

On se donne un convexe de $\mathbb{R}^2$, dont le bord est noté Γ. On cherche à placer n points $M_0, \dots, M_{n-1}$ sur Γ qui correspondent à une trajectoire de billard parfaite : l'angle entre la normale au bord et la trajectoire avant rebond doit être le même que celui entre la normale et la trajectoire après rebond.

On modélise d'abord notre problème. On se donne $t \mapsto \gamma(t) \in \mathbb{R}^2$ une paramétrisation 2π-périodique du bord Γ. Par exemple si le convexe est une ellipse, on prendrait $\gamma(t) = (a\cos t, b\sin t)$. On suppose que γ est de classe C^1 et que $\gamma'(t) \neq 0$ pour tout $t \in \mathbb{R}$.

On pose L la fonction de $\mathbb{R}^n$ dans $\mathbb{R}$ qui donne la longueur totale de la trajectoire passant successivement pas les points (on ne se préoccupe pas de savoir si la trajectoire est une trajectoire de billard).

$$L(t_0, \dots, t_{n-1}) = \|\gamma(t_0) - \gamma(t_{n-1})\| + \sum_{i=1}^{n-1} \|\gamma(t_i) - \gamma(t_{i-1})\|.$$

On a vu en TD que la fonction L admet un maximum global sur $\mathbb{R}^n$ et que tout point de maximum local correspond à une trajectoire de billard parfaite. Il existe donc au moins une trajectoire parfaite, et on va essayer d'en approximer numériquement.

(a) Définir L comme une fonction prenant comme argument un seul vecteur, le vecteur des t_i. On définira γ en dehors de cette fonction (on peut par exemple commencer par une ellipse non circulaire).

Appliquer la méthode de gradient à pas fixe pour obtenir des trajectoires de billards parfaites, en 3, 4, 5 bandes ou plus... On pourra éventuellement pour simplifier prendre une version modifiée de la méthode qui ne renvoie que le point final.

Quelle méthodologie appliquer pour choisir un pas fixe α qui convienne ?

Afficher le billard et les trajectoires obtenues.

(b) Observer ce qui se passe lorsque l'on prend des points initiaux différents. Par exemple pour $n = 4$ ou $n = 5$, peut-on obtenir des trajectoires décroisées, des trajectoires croisées ?

(c*) Lire et comprendre la fonction de génération de courbes convexes ci-dessous. L'utiliser pour obtenir des trajectoires de billard parfaites sur un convexe généré de la sorte (on obtient des convexes moins symétriques).

```
In [ ]: def genereGamma(checkConvexe=True,Npoints=100):
            a=randn(5);b=randn(5);c=randn(5);d=randn(5)
            a[1]+=5;d[1]+=5

            def courbe(t):
                x=sum((a[i]*cos(i*t) + b[i]*sin(i*t))/i**2 for i in range(1,5))
                y=sum((c[i]*cos(i*t) + d[i]*sin(i*t))/i**2 for i in range(1,5))
                return array([x,y])

            if checkConvexe:
                t=linspace(0,2*pi,Npoints)
                xp=sum((-a[i]*sin(i*t) + b[i]*cos(i*t))/i for i in range(1,5))
                xs=sum((-a[i]*cos(i*t) - b[i]*sin(i*t)) for i in range(1,5))
                yp=sum((-c[i]*sin(i*t) + d[i]*cos(i*t))/i for i in range(1,5))
                ys=sum((-c[i]*cos(i*t) - d[i]*sin(i*t)) for i in range(1,5))
                if any(xp*ys-yp*xs<0):
                    return genereGamma(True,Npoints)
            return courbe
```

4. Descente de gradient à pas optimal

(a) Voici une fonction de recherche de pas optimal à l'aide de la méthode de la section dorée. Pour éviter d'évaluer les fonctions là où elles sont déjà calculées, on donne comme arguments la fonction f, le point x_k, la valeur de $f(x_k)$, la direction de descente d_k et un pas par défaut α ainsi qu'une tolérance.

On renvoie le point x_{k+1}, le nouveau pas α_k calculé, ainsi que la valeur de $f(x_{k+1})$ vu qu'elle a déjà été calculée.

```
In [ ]: def recherchePasOptimal(f,xk,fxk,dk,alpha,tol=1e-5):
            invphi=(sqrt(5)-1)/2
            a=0
            ha=fxk
            b=alpha
            hb=f(xk+b*dk)
            c=a+(1-invphi)*(b-a)
            d=a+taux*(b-a)
            hc=f(xk+c*dk)
            hd=f(xk+d*dk)
            while hc>ha:
                d,b=c,d
                hd,hb=hc,hd
                c=a+(1-invphi)*(b-a)
                hc=f(xk+c*dk)
            while hd>hb:
                c,d=d,b
                hc,hd=hd,hb
                b=b+invphi*(b-a)
                hb=f(xk+b*dk)

            while b-a>tol:
                if hc<hd:
                    d,b=c,d
                    hd=hc
                    c=a+(1-invphi)*(b-a)
                    hc=f(xk+c*dk)
                else:
                    a,c=c,d
                    hc=hd
                    d=a+unsurphi*(b-a)
                    hd=f(xk+d*dk)

            if hc<hd:
                return xk+c*dk,c,hc
            else:
                return xk+d*dk,d,hd
```

Questions :

— À quoi servent les deux premières boucles `while` ?

— Expliquer, si la fonction $t \mapsto h(t) = f(x_k + td_k)$ est unimodale sur $[0, +\infty[$ et que d_k est une direction de descente pour f, pourquoi ces deux boucles terminent.

— Montrer que dans ce cas on ne passe que dans l'une des deux boucles (si on passe au moins une fois dans la première, la condition de la deuxième ne sera pas satisfaite dès qu'on sort de la première).

(b) Voici maintenant une fonction qui effectue la méthode de descente de gradient à pas optimal à l'aide de la fonction de recherche précédente.

```
In [ ]: epsilon=1e-8
        def gradientApproxModif(f,x,fx):
            gra=zeros(size(x))
            for i in range(size(x)):
                veps=zeros(size(x))
                veps[i]+=epsilon
                gra[i]=(f(x+veps)-fx)/epsilon
            return gra
```

```
In [ ]: compteur=0
        a=3
        def facompteur(X):
            global compteur
            compteur+=1
            return 1-1/(1+a*X[0]**2+X[1]**2)
```

```
In [ ]: def descenteGradientOptimal(f,X0,alpha0,tol,tolrecherche=1e-5,N=200):
            global compteur
            compteur=0
            X=X0
            fX=f(X0)
            g=gradientApproxModif(f,X0,fX)
            lX=[X]
            lcompteur=[compteur]
            n=0
            alpha=alpha0
            while norm(g)>tol and n<N:
                n+=1
                X,alpha,fX=recherchePasOptimal(f,X,fX,-g,alpha,tolrecherche)
                g=gradientApproxModif(f,X,fX)
                lX.append(X)
                lcompteur.append(compteur)
            if n==N:
                print("Nombre d'itérations maximal atteint : ",N)
            return lX,lcompteur
```

Questions :

— Pourquoi a-t-on redéfini la fonction d'approximation du gradient ?

— Appliquer la méthode sur la fonction f_a et afficher les points obtenus sur un graphique (on pourra reprendre le code de la partie 2 ci-dessus). Observer que les directions de descentes d_k sont tangentes aux lignes de niveaux en x_{k+1} et orthogonales aux lignes de niveau en x_k.

— Observer le taux de convergence linéaire en traçant la norme de x_k en fonction de k (en échelle semi-logarithmique). Le taux de convergence linéaire paraît-il meilleur que celui obtenu à la partie 2 avec la méthode de gradient à pas fixe ?

(c) On s'intéresse maintenant au taux de convergence linéaire effectif.

— Tracer le graphique de la norme des itérées en fonction du nombre d'évaluations de fonctions (qui est renvoyé par la variable `lcompteur`) dans le code donné précédemment).

— Quel est le nombre d'évaluations de fonction par itération dans le cas de la méthode de descente de gradient à pas fixe ? Le taux effectif est-il meilleur dans le cas du gradient à pas optimal ?

— Observer comment se comporte le taux de convergence effectif si on change la tolérance dans la fonction de recherche de pas optimal (par exemple, si on augmente la tolérance, la recherche du pas optimal sera moins bonne, mais on fera moins d'évaluations de fonctions).

(d) Appliquer la méthode de descente de gradient à pas optimal pour obtenir des trajectoires de billards parfaites comme dans la partie 3. Observer également si cette méthode est efficace.