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.

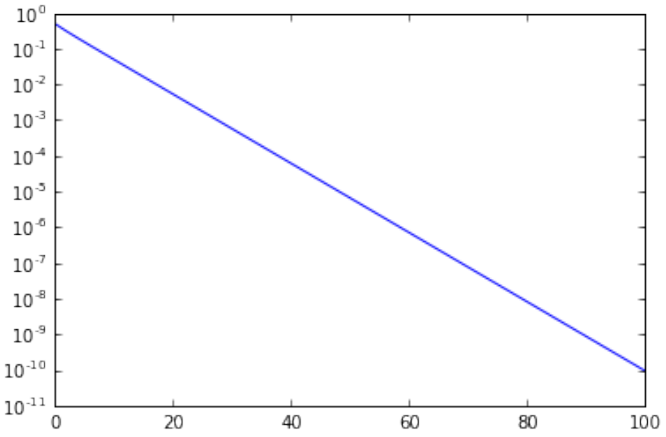
```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

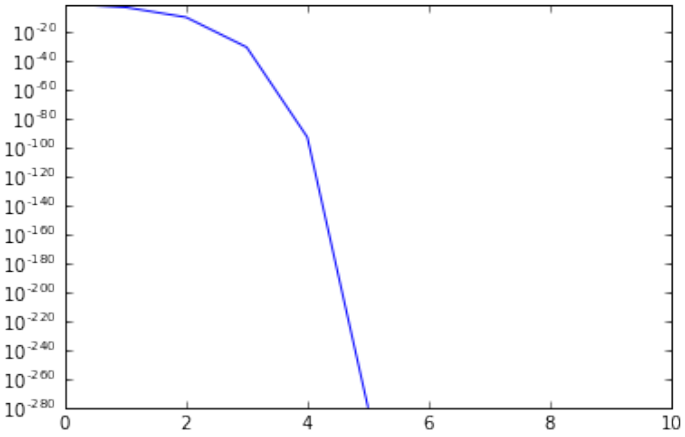
1. Retour en dimension un

(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.

```
In [2]: alpha=.1
        x=1/2
        l=[x]
        for i in range(100):
            x=x-alpha*(2*x+x**3)
            l.append(x)
        semilogy(l)
        show()
```

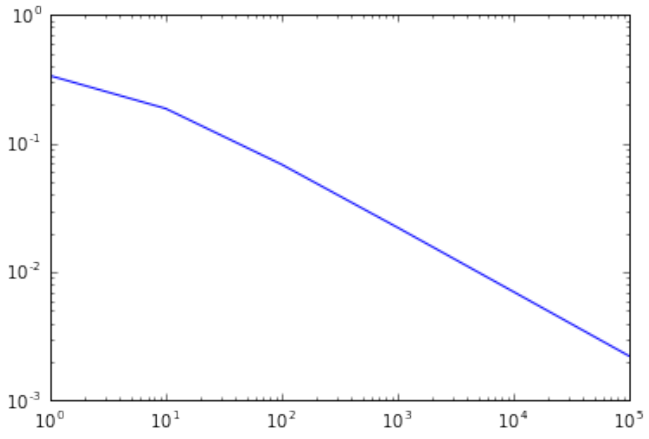



```
In [3]: alpha=1/2
        x=.1
        l=[x]
        for i in range(10):
            #x=x-alpha*(2*x+x**3)
            x=(1-2*alpha)*x-alpha*x**3 # On évite des erreurs d'arrondi
            l.append(abs(x))
        semilogy(l)
        show()
```



(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).

```
In [4]: alpha=1
        x=.4
        l=[]
        listei=[1,10,100,1000,10000,100000]
        for i in range(1,100001):
            x=x-alpha*(2*x-x**3)
            if i in listei:
                l.append(abs(x))
        loglog(listei,l)
        show()
        print(l)
        print([l[i]*sqrt(listei[i]) for i in range(5)])
```

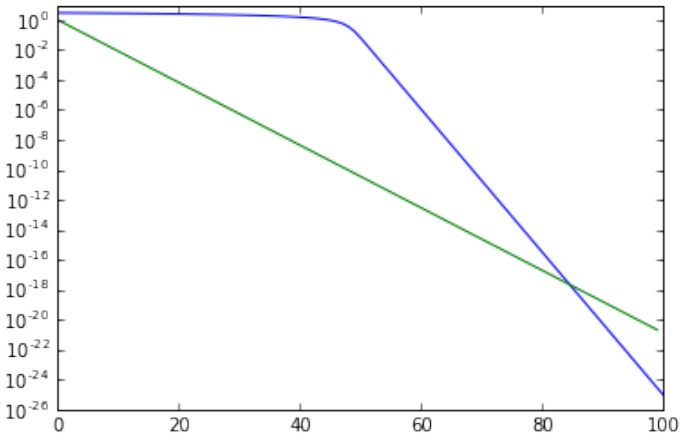



```
[0.33599999999999997, 0.18653372574181265, 0.06871675568927713, 0.022276413148933805, 0.007067
[0.33599999999999997, 0.58987143378130957, 0.68716755689277131, 0.70444203649554504, 0.7067781
```

On observe d'abord que ça ressemble asymptotiquement à une droite en loglog, donc $ln(|x_n|) \approx -C \ln n$. La pente C semble être $\frac{1}{2}$: on multiplie n d'un facteur 100 pour gagner un facteur 10 sur $|x_n|$. On conjecture donc que $|x_n| \approx \frac{a}{\sqrt{n}}$. On voit que $\sqrt{n}|x_n|$ a l'air de tendre vers 0.707. Ne serait-ce pas $\frac{1}{\sqrt{2}}$?

(c) Le taux de convergence est censé être $|1-2\alpha|$ (cf. question **(d)** du TD), donc ici $\frac{1}{3}$.

```
In [5]: x=3
        alpha=1/3
        l=[x]
        for i in range(100):
            x=x*(1-2*alpha/(1+x**2)**2)
            l.append(abs(x))

        semilogy(l)
        semilogy(((1+sqrt(5))/2)**(-arange(100)))
        show()
```



La pente asymptotique est certes meilleure que dans le cas de la section dorée (pour lequel on sait que le taux de convergence linéaire est $\frac{1}{\varphi} \approx 0,618 > \frac{1}{3}$), mais en pratique comme le démarrage est très lent, on arrive bien plus vite au minimum avec la méthode de la section dorée (on sait que cela ne sert à rien de chercher des précisions de 10^{-16}, surtout si la dérivée est approximée...).

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

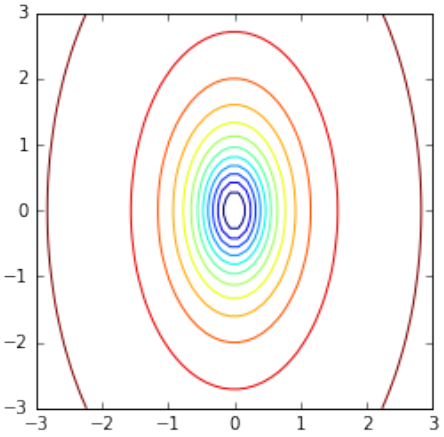
On va tester la méthode de descente de gradient sur la fonction $f_a : (x_0, x_1) \mapsto 1 - \frac{1}{1+ax_0^2+x_1^2}$, où $a > 0$ est un paramètre qu'on pourra changer pour voir comment se comporte la méthode en fonction de a.

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) Ici pas besoin du mot-clé `global`, puisqu'on ne modifiera pas a dans les fonctions.

```
In [6]: a=3
        def fa(X):
            return 1-1/(1+a*X[0]**2+X[1]**2)
```

```
In [7]: aX0=linspace(-3,3)
        aX1=linspace(-3,3)
        Z=array([[fa(array([x0,x1])) for x0 in aX0] for x1 in aX1])
        contour(aX0,aX1,Z,12)
        axis('scaled')
        show()
```



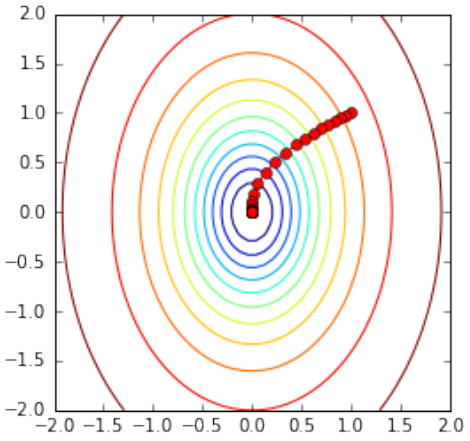
(b) On peut utiliser la fonction `norm` pour vérifier ce qu'on obtient. Le bon pas ε est de l'ordre de la racine de la précision machine, soit ici 10^{-8}.

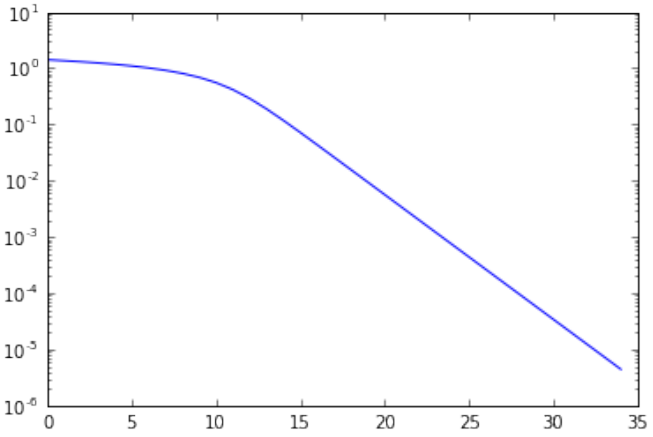
```
In [8]: epsilon=1e-8
        def gradientApprox(f,x):
            fx=f(x)
            gra=zeros(size(x))
            for i in range(size(x)):
                veps=zeros(size(x))
                veps[i]+=epsilon
                gra[i]=(f(x+veps)-fx)/epsilon
            return gra

        g=gradientApprox(fa,array([1,1]))
        print(g)
        norm(g-array([2*a/(2+a)**2,2/(2+a)**2]))

[ 0.24  0.08]


Out[8]: 5.0601018074952776e-09
```

(c) La seule chose à faire attention pour observer que le gradient est orthogonal aux lignes de niveaux, c'est que les axes aient la même échelle, sinon les angles sont déformés.

```
In [9]: def methodePasFixe(f,X0,alpha,tol,N=200):
            lX=[X0]
            X=X0
            g=gradientApprox(f,X0)
            n=0
            while norm(g)>tol and n<N:
                n+=1
                X=X-alpha*g
                g=gradientApprox(f,X)
                lX.append(X)
            if n==N:
                print("Nombre d'itérations maximal atteint : ",N)
            return lX
```

```
In [10]: a=2
         l=methodePasFixe(fa,[1,1],.2,1e-5)
         lx0=[X[0] for X in l]
         lx1=[X[1] for X in l]
         aX0=linspace(-2,2)
         aX1=linspace(-2,2)
         Z=array([[fa(array([x0,x1])) for x0 in aX0] for x1 in aX1])
         contour(aX0,aX1,Z,12)
         plot(lx0,lx1,"-ro")
         axis('scaled')
         show()
         semilogy([norm(X) for X in l])
         show()
```





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 $M_0, \ldots, M_{n-1}$ 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.

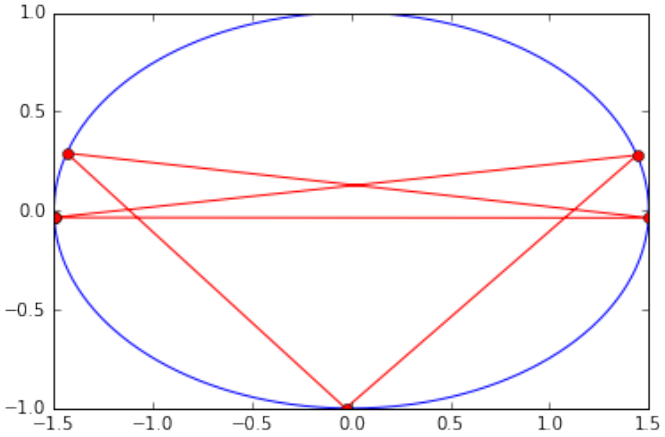
(a) On simplifie en prenant une version modifiée de la méthode qui ne renvoie que le point final (et la liste des normes des gradients).

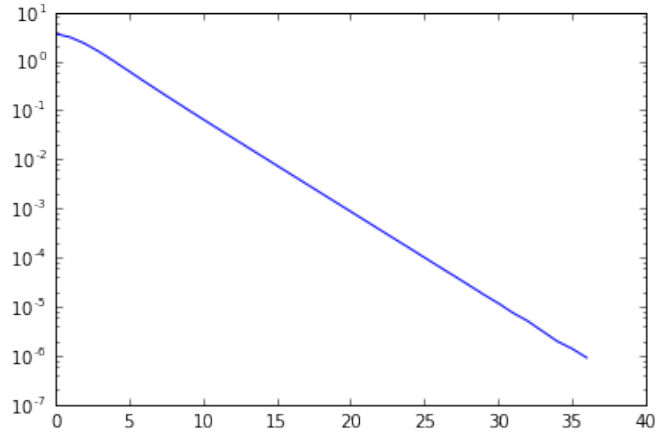
La méthodologie pour prendre le pas qui convienne est de le prendre suffisamment petit. Comme la fonction prend des valeurs de l'ordre de grandeur de 1, on peut espérer que ce soit la même chose pour la dérivée seconde, donc si on prend par exemple un pas de l'ordre de 10^{-1}, on peut espérer qu'il soit plus grand que $\frac{2}{L}$, où L est la plus grande valeur propre de la Hessienne.

```
In [11]: A=1.5;B=1
         def gamma(t):
             return array([A*cos(t),B*sin(t)])

         def moinsL(vt):
             n=size(vt)
             return -norm(gamma(vt[n-1])-gamma(vt[0]))-sum(
                 [norm(gamma(vt[i])-gamma(vt[i-1])) for i in range(1,n)])
```

```
In [12]: def methodePasFixeNormeGradient(f,X0,alpha,tol,N=200):
             X=X0
             g=gradientApprox(f,X0)
             normeg=norm(g)
             lnormeg=[normeg]
             n=0
             while normeg>tol and n<N:
                 n+=1
                 X=X-alpha*g
                 g=gradientApprox(f,X)
                 normeg=norm(g)
                 lnormeg.append(normeg)
             if n==N:
                 print("Nombre d'itérations maximal atteint : ",N)
             return X,lnormeg
```

```
In [13]: vt,listeNormes=methodePasFixeNormeGradient(moinsL,randn(5),.2,1e-6)

         xGamma=[gamma(t)[0] for t in linspace(0,2*pi,150)]
         yGamma=[gamma(t)[1] for t in linspace(0,2*pi,150)]
         plot(xGamma,yGamma)
         xTrajet=[gamma(t)[0] for t in vt];xTrajet.append(gamma(vt[0])[0])
         yTrajet=[gamma(t)[1] for t in vt];yTrajet.append(gamma(vt[0])[1])
         plot(xTrajet,yTrajet,"-ro")
         axis("scaled")
         show()
         semilogy(listeNormes)
         show()
```

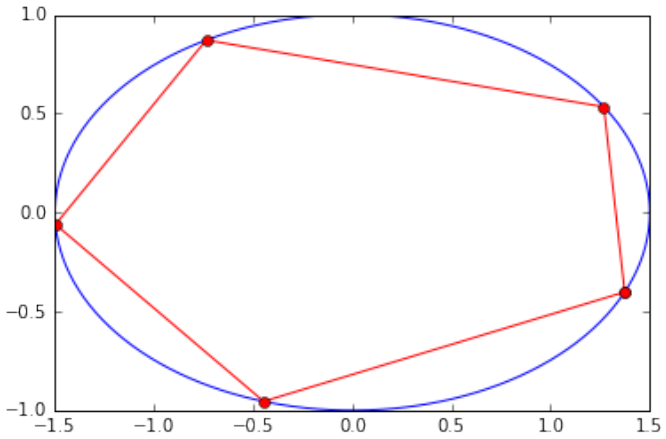




On voit bien que la norme des gradients converge linéairement vers 0. Le choix du pas était donc correct.

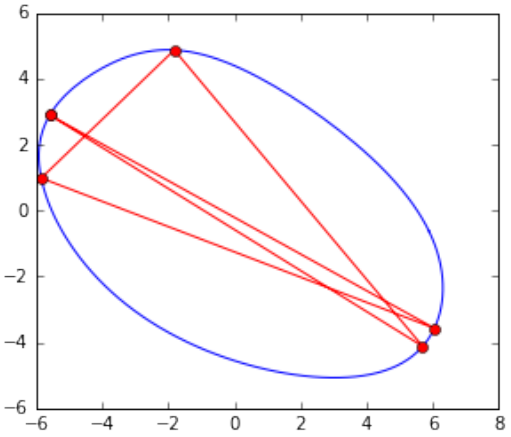
(b) Pour $n = 4$, on n'observe que des allers-retours entre les deux points opposés sur le plus grand axe de l'ellipse. Pour $n = 5$, on peut aussi obtenir une trajectoire décroisée en prenant les t_i initiaux dans l'ordre dans $[0, 2\pi]$.

```
In [14]: vt,listeNormes=methodePasFixeNormeGradient(moinsL,[0,1,2,3,4],.2,1e-6)
         plot(xGamma,yGamma)
         xTrajet=[gamma(t)[0] for t in vt];xTrajet.append(gamma(vt[0])[0])
         yTrajet=[gamma(t)[1] for t in vt];yTrajet.append(gamma(vt[0])[1])
         plot(xTrajet,yTrajet,"-ro")
         axis("scaled")
         show()
```



(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 [15]: 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

In [16]: gamma=genereGamma()
         vt,listeNormes=methodePasFixeNormeGradient(moinsL,randn(5),.1,1e-6)

         xGamma=[gamma(t)[0] for t in linspace(0,2*pi,150)]
         yGamma=[gamma(t)[1] for t in linspace(0,2*pi,150)]
         plot(xGamma,yGamma)
         xTrajet=[gamma(t)[0] for t in vt];xTrajet.append(gamma(vt[0])[0])
         yTrajet=[gamma(t)[1] for t in vt];yTrajet.append(gamma(vt[0])[1])
         plot(xTrajet,yTrajet,"-ro")
         axis("scaled")
         show()
```


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8