L3 E - S5 2018-19 --- TD ordinateurs --- séance 3 ——solutions

1 Un peu d'analyse et de représentations graphiques

L'instruction *pylab inline* permettra l'insertion des graphiques dans la feuille notebook (au lieu de les avoir dans des fenêtres "surgissantes").

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

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

On fait appel à la bibliothèque *matplotlib* pour utiliser les fonctionnalités graphiques de python :

```
In [2]: from matplotlib.pylab import *
```

```
In [3]: import os
        os.chdir('.......................')
```

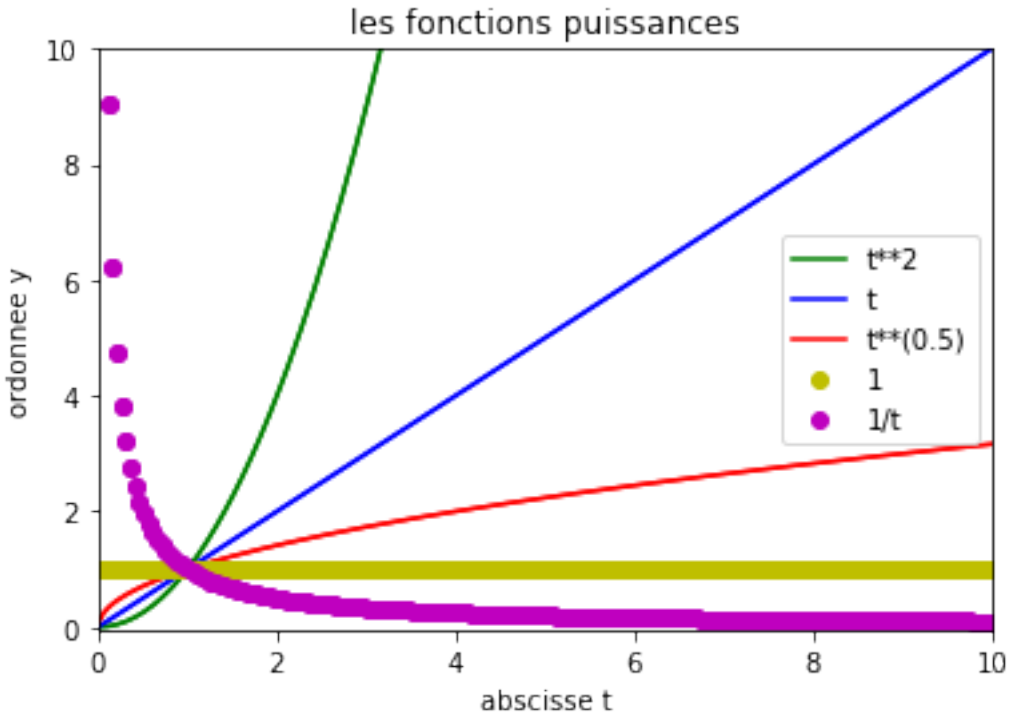
2 Exercice 1

Tous les graphiques réalisés sont à sauvegarder dans le répertoire personnel

Tracer sur un même dessin et pour $t > 0$ les graphes des fonctions $x \mapsto x^a$ pour $a \in \{-1, 0, 1/2, 1, 2\}$~: on distinguera les représentations graphiques en prenant des couleurs différentes et on fera un dessin avec une fenêtre avec $x \in]0, 3]$ et un autre dessin avec une fenêtre avec $x \in]0, 10]$.

```
In [4]: from matplotlib.pylab import *

        def g1(t):
            return t**2
        def g2(t):
            return t
        def g3(t):
            return t**(0.5)
        def g4(t):
            return t**0
        def g5(t):
            return t**(-1)

        t = linspace(0.01, 10, 200)
        y1 = g1(t)
        y2 = g2(t)
        y3 = g3(t)
        y4 = g4(t)
        y5 = g5(t)
```

```
plot(t, y1, "g-")
plot(t, y2, "b-")
plot(t, y3, "r-")
plot(t, y4, "yo")
plot(t, y5, "mo")
axis([0, 3, -0.05, 3])
xlabel("abscisse t")
ylabel("ordonnee y")
legend(["t**2", "t","t**(0.5)","1","1/t"])
title("les fonctions puissances")
savefig("fonctions-puissances-3.pdf")

plot(t, y1, "g-")
plot(t, y2, "b-")
plot(t, y3, "r-")
plot(t, y4, "yo")
plot(t, y5, "mo")
axis([0, 10, -0.05, 10])
xlabel("abscisse t")
ylabel("ordonnee y")
legend(["t**2", "t","t**(0.5)","1","1/t"])
title("les fonctions puissances")
savefig("fonctions-puissances-10.pdf")
```



3 Exercice 2

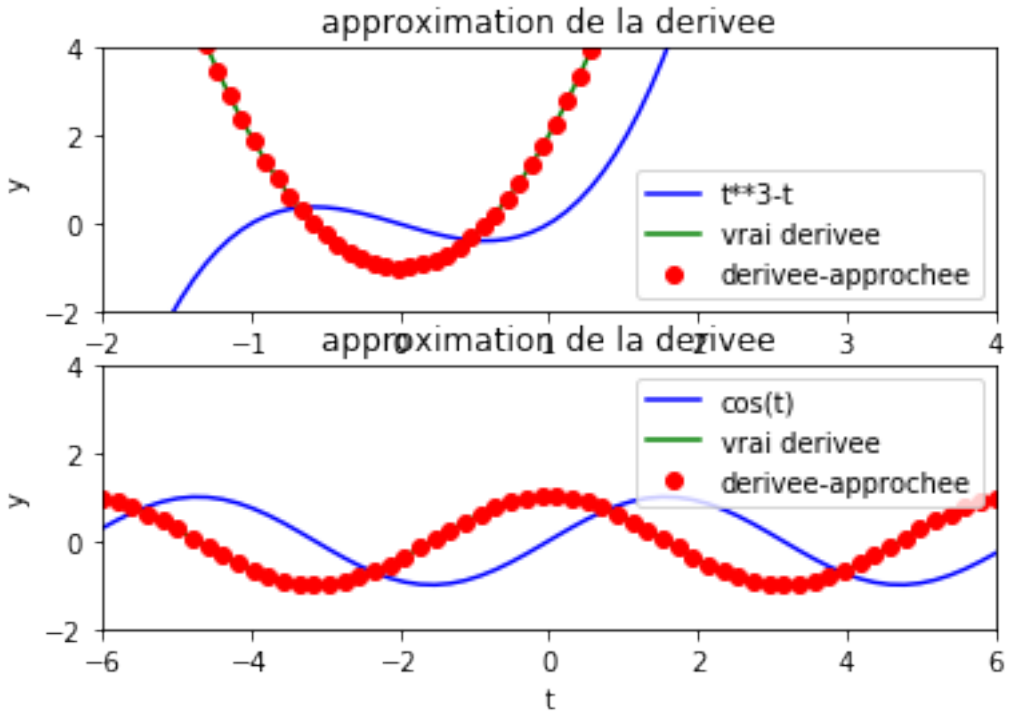
On se donne une application de classe C^1 sur un intervalle $[a,b]$. Supposant qu'on ne connaît pas sa dérivée, on va la calculer en considérant, pour h petit, la fonction suivante, définie pour $x \in [a+h, b-h]$~:

$$x \mapsto \frac{f(x+h)-f(x-h)}{2h}$$

1. Ecrire une fonction `ApproximationDerivee(f,h,t)` qui retourne $\frac{f(x+h)-f(x-h)}{2h}$.
2. Tester la qualité de cette approximation pour la fonctions $f(t) = t^3 - t$, puis pour $g(t) = \sin(t)$. Chaque fois, on tracera sur un même graphique les graphes de f, f' et de l'approximation de f'. De plus, on fera apparaître les 2 graphiques dans une même fenêtre.

```
In [5]: def approximation_derivee(f,h,t):
            return (f(t+h)-f(t-h))/(2*h)

        figure()  # pour faire des figures séparées
        subplot(2, 1, 1)  ### la 1ère figure


        def f(t):
            return t**3-t

        def f1(t):
            return 3*(t**2)-1

        t = linspace(-3, 5, 100)
        y=f(t)
        y1=f1(t)
        y2=approximation_derivee(f,0.1,t)

        plot(t,y,"b-")
        plot(t,y1,"g-")
        plot(t,y2,"ro")

        xlabel("t")
        ylabel("y")
        legend(["t**3-t","vrai derivee","derivee-approchee"])
        axis([-2, 4, -2, 4])   # [tmin, tmax, ymin, ymax]
        title("approximation de la derivee")
        savefig("approximation-derivee_poly.pdf")

        subplot(2, 1, 2)   ### la 2ème figure

        def g(t):
            return sin(t)
```

```
def g1(t):
    return cos(t)

t = linspace(-6, 6, 60)
z=g(t)
z1=g1(t)
z2=approximation_derivee(g,0.1,t)
plot(t,z,"b-")
plot(t,z1,"g-")
plot(t,z2,"ro")

xlabel("t")
ylabel("y")
legend(["cos(t)","vrai derivee","derivee-approchee"])
axis([-6, 6, -2, 4])    # [tmin, tmax, ymin, ymax]
title("approximation de la derivee")
savefig("approximation-derivee_cos.pdf")
```



4 Exercice 3

On se donne une fonction strictement convexe. On rappelle que cela signifie que sa courbe repr´esentative est toujours en-dessous de ses cordes :

1. Ecrire une fonction `Pente(f,a,b)` qui retourne le taux d'accroissement de la fonction f sur le segment $[a,b]$, c'est-à-dire $\frac{f(b)-f(a)}{b-a}$.

2. Si f est strictement convexe sur $[a,d]$, on sait qu'elle admet un unique minimum. Ecrire une fonction `ComparePente(f,a,b,c,d)` qui retourne, parmi les 4 intervalles $[a,b]$, $[a,c]$, $[b,d]$,$[c,d]$, l'intervalle dans lequel se trouve son minimum (voir dessins ci-dessous où on appelle x_0 le minimum).

3. Ecrire une fonction `ApproximationMinimum(f,a,b,`ϵ`)` qui, pour une fonction strictement convexe f sur un intervalle $[a,b]$, retourne une approximation de son minimum à ϵ près.

```
In [6]: def Pente(f,a,b):   ## calcul de la pente de f entre les points d'abscisses a et b
            return (f(b)-f(a))/(b-a)

        def ComparePente(f,a,b,c,d):  ## recherche de l'intervalle où s'annule la dérivée
            pab=Pente(f,a,b)
            pbc=Pente(f,b,c)
            pcd=Pente(f,c,d)
            if pab >=0:
                return a,b
            elif pbc >=0:
                return a,c
            elif pcd >=0:
                return b,d
            else:
                return c,d

        def Minimum(f,a,b,epsilon):    ## on cherche le minimum de f convexe sur [a,b]
                                       ##  on se contene d'une approximation à epsilon/2 près
            while b-a > epsilon:
                pas=(b-a)/3            ## on coupe [a,b] en 3 sous-intervalles
                a1,b1=a+pas,b-pas
                a,b=ComparePente(f,a,a1,b1,b)
            return (a+b)/2

        def f(t):
            return (t-2)**2

        def g(t):
            return t**2


        print(ComparePente(f,-1.0,2.0,3,6))

        print(Minimum(f,-4.0,6,0.1))
        print(Minimum(g,-4.0,6,0.1))
```

```
(-1.0, 3)
1.9922926533707415
0.0077073466292586455
```

In [ ]: