TP 1 : Optimisation en dimension un.

On commence toujours par charger les librairies. Les questions avec une étoile sont facultatives (et difficiles). Les parties **1**, **2**, et **3** sont essentiellement indépendantes. La partie **4** nécessite d'avoir avancé sur la partie **2** ou la partie **3**.

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

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

1. Ordre de convergence des suites

Calculer numériquement les N premiers termes des trois suites suivantes (on pourra commencer avec $N = 10$). Produire un graphique qui permette d'illustrer leur ordre de convergence (on pourra utiliser une échelle semi-logarithmique avec la fonction `semilogy`). On pourra utiliser directement la limite si on la connaît, ou visualiser les écarts entre deux termes successifs. Observer si un changement de la condition initiale x_0 influence le comportement de la suite.

(a) $x_0 = 1$ et pour tout $n \in \mathbb{N}$,

$$x_{n+1} = x_n\left(1 - \frac{x_n}{2}\right) + 1.$$

(b) $x_0 = 1$ et pour tout $n \in \mathbb{N}$,

$$x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n}.$$

(c) $x_0 = 2$, $x_1 = 1$ et pour tout $n \in \mathbb{N}^*$,

$$x_{n+1} = \frac{x_n x_{n-1} + 2}{x_n + x_{n-1}}.$$

```
In [ ]:
```

2. Dichotomie et dérivée approchée.

(a) Écrire une fonction prenant en argument une fonction f et un point x et qui renvoie une approximation de la dérivée de cette fonction au point x par la formule des différences finies avec un paramètre ε. L'appliquer à une fonction dont vous connaissez la dérivée (non nulle), et afficher le comportement de l'erreur entre la valeur approchée et la valeur exacte en fonction de ε (pour une large plage d'ordre de grandeur de ε, par exemple au moins entre 10^{-1} et 10^{-15}). On pourra cette fois-ci prendre aussi une échelle logarithmique pour ε (en utilisant les fonctions `semilogx` ou `loglog`). Comment expliquer ce comportement ? Quel paraît être un bon choix de ε ? Quel est alors l'ordre de grandeur de l'erreur numérique ? Comparer avec la théorie du TD.

```
In [ ]:
```

(b) Même question avec la formule des différences finies centrées.

(c) Appliquer les fonctions codées précédemment à la fonction `sinapprox` suivante, qui calcule le sinus en simulant des erreurs d'arrondis plus grosses que celles de la machine. Que devient le bon choix de ε cette fois-ci ? Comment semble se comporter ce bon choix en fonction de la taille des erreurs d'arrondi ?

```
In [ ]: tailleErreur=1e-6
        def sinapprox(x):
            return (2*random_sample()-1)*tailleErreur + sin(x)
```

(d) Programmer une fonction qui prend pour argument une fonction f, deux réels correspondant aux extrémités d'un segment sur lequel f est strictement décroissante puis croissante (on l'appliquera à des fonctions C^1 sur ce segment), un pas ε et une tolérance, et qui calcule par dichotomie une approximation du minimum (en calculant des approximations de $f'(x)$ par différence finie et en résolvant $f'(x) = 0$).

(e) Définir une fonction f à optimiser (par exemple $f : x \mapsto \frac{x}{2} + \frac{1}{x}$ sur $[1, 2]$) en y incorporant une variable globale jouant le rôle de compteur du nombre de fois où cette fonction est évaluée. Tester la fonction programmée à la question **(d)** sur la fonction f. Illustrer par un graphique le taux de convergence effectif (qui illustre la décroissance de l'erreur en fonction du nombre d'évaluations de la fonction).

In [ ]:

3. Méthodes de réduction de triplets

(a) Programmer une fonction qui prend pour argument une fonction f, deux points initiaux correspondant aux extrémités d'un segment sur lequel f est unimodale, et une tolérance ε. La fonction doit renvoyer une approximation du minimum par la méthode de la section dorée.

(b) Tester la fonction programmée au **(a)** en l'appliquant à la fonction $f : x \mapsto \frac{x}{2} + \frac{1}{x}$ sur $[1, 2]$ (ou à une autre fonction de votre choix). Incorporer une variable globale jouant le rôle d'un compteur dans la définition de f pour pouvoir illustrer par un graphique le taux de convergence effectif (et vérifier qu'il est égal à $\alpha = \frac{1}{2}(\sqrt{5} - 1)$, et meilleur que le taux obtenu à la question **(e)** de la partie **2**.

(c*) Programmer de même une fonction qui prend les mêmes arguments que dans la question **(a)**, et qui renvoie cette fois-ci une approximation par la méthode de réduction du triplet par interpolation quadratique. Tester cette fonction et illustrer par un graphique l'ordre de convergence effectif obtenu.

In [ ]:

4. Application : résolution d'un problème de plus court chemin entre deux zones parcourues à deux vitesses différentes.

On cherche à modéliser un problème de plus court chemin entre deux zones de $\mathbb{R}^2$ parcourues à deux vitesses v_1 et v_2. On se donne par exemple une fonction f convexe de $\mathbb{R}$ dans $\mathbb{R}$ et on définit $Z_1 = \{(x, y), y \geqslant f(x)\}$ et $Z_2 = \{(x, y), y \leqslant f(x)\}$ les deux zones : au-dessus et en dessous de la courbe $\mathcal{C}$ d'équation $y = f(x)$. On se donne un point A dans Z_1, un point B dans Z_2, et un point M sur la courbe $\mathcal{C}$. On cherche à minimiser le temps de parcours de A à B, sachant qu'on se déplace en ligne droite sur chacune des zones aux vitesses v_1 et v_2 avec $v_1 > v_2$ (cf. le TD pour la partie théorique) :

$$\inf_{M \in \mathcal{C}} \frac{\|AM\|}{v_1} + \frac{\|BM\|}{v_2}.$$

(a) Utiliser les méthodes programmées dans les parties précédentes pour résoudre le problème, pour une fonction f de votre choix, et des points A et B (on pourra les prendre au hasard). Afficher un joli dessin représentant les zones et le chemin optimal.

(b*) On s'intéresse au problème de minimiser le chemin entre A et un autre point A' de Z_1. Faire la même chose qu'à la question **(a)** pour ce problème.

(c*) Illustrer le fait qu'il peut y avoir plusieurs points de minimum local dans certains cas, en trouvant des conditions initiales différentes pour lesquelles les méthodes ne donnent pas les mêmes points, dans le cadre des questions **(a)** et **(b)**.

In [ ]: