M.P.S.I.2 2014 **Vendredi 6 mars** 2015 Charlemagne — 1-phô

MPSI 2/2014 **RACINES** 1-phô

L'objectif est la résolution d'une équation $f(x) = 0$ d'inconnue x, avec f continue sur un segment $[a,\ b]$, avec évidemment $f(a)$ et $f(b)$ de signes opposés. On va mettre en place divers algorithmes.
On suppose que f est donnée par une procédure telle que

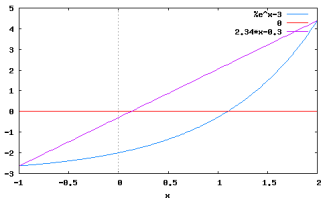
```
def f(x) :
....return(...)
```

pouvant faire appel à une boucle pour une somme, des tests...
Corrigez le premier algorithme que voici : •1 pt.•

```
precision = float(input(...))
aa, fa, bb, fb = float(a), f(aa), float(b), f(bb)
while abs(bb-aa) > precision :
....c = (a+b)/2
....fc = f(c)
....if fa*fc > 0 :
........bb, fb = c, fc
....else :
........aa, fa = c, fc
print(aa, '< racine <', bb)
```

Ajoutez un morceau de script qui demande la saisie de `a` et `b` et vérifie $f(a) * f(b) < 0$ *(en boucle jusqu'à avoir effectivement* $f(a).f(b) < 0$*)*. •1 pt.•
Expliquez pourquoi on a mis `aa, fa, bb, fb = float(a), f(aa), float(b), f(bb)`. •1 pt.•
Par quoi pourrait on remplacer `if fa*fc < 0 :` pour que le programme "tourne" plus vite ? •1 pt.•
Pourquoi a-t-on proposé `aa, fa = c, fc` plutôt que `aa, fa = c, f(c)`. •1 pt.•

On se propose d'utiliser la méthode dite "des sécantes". On pose $A(aa,\ f(aa))$ et $B(bb,\ f(bb))$ et on note s l'abscisse du point d'intersection de la droite $(A\ B)$ avec l'axe des abscisses.

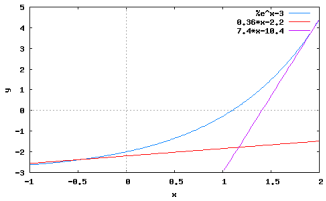


Mettez vous dans la peau de chacun des élèves suivants pour calculer s et détaillez leurs calculs : •3 pt.•

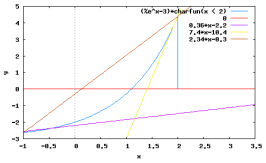
élève	méthode	réponse
Accinée	je calcule l'équation de la sécante, puis...	
Endual-Ennemi	j'écris une condition d'alignement par un déterminant	
Espa-Sasfabric-Encorh	j'utilise le théorème de Thalès	

Vérifiez que s est entre aa et bb. •1 pt.•
Réécrivez alors l'algorithme. •2 pt.•
Expliquez pourquoi cette méthode est plus judicieuse. •1 pt.•

On suppose qu'on a aussi la procédure qui calcule $f'(x)$ pour x donné. On utilise alors la méthode de Newton : on détermine `Ta` *(et `Tb`)*, intersection(s) de l'axe des abscisses avec la tangente en `A` *(et avec la tangente en `B`)*. Donnez la formule qui calcule `Ta` et `Tb`. •3 pt.•



On peut démontrer mathématiquement qu'avec `c` ou `s`, la convergence est linéaire *(la précision double quand on double le nombre d'étapes)*, alors qu'avec la méthode de Newton, une fois qu'on est "assez proche de la racine", la convergence est quadratique.
Adaptez le script pour qu'à chaque itération on retienne l'intervalle dont les extrémités sont à choisir parmi `a`, `b`, `c`, `s`, `Ta` et `Tb`, le plus court possible sur lequel il y a un changement de signe de f. •5 pt.•



MPSI 2/2014 — ANNIVERSAIRES — 1-phô

Objectif : on donne une liste `L` composée d'entiers *(des références produits ou clients dans une commande, la liste des 9 nombres d'une ligne ou colonne de Su-Do-Ku, des dates d'anniversaire...)*, il faut savoir si il y a un élément en double dans la liste.
On va donc écrire une procédure introduite sous la forme `def test(L) :` qui donnera par `return` la valeur `True` si tous les éléments de la liste sont différentes, et `False` si il y a un élément en double *(en option, le programme pourra indiquer la valeur trouvée en double)*.
On rappelle les fonctions et méthodes sur une liste `L` :

fonctions et affectations :

`len(L)`	longueur	entier
`L[k] = a`	remplace l'élément d'indice `k`	
`L[k]`	lit l'élément d'indice `k`	
`a in L`	test d'appartenance	`True` ou `False`

méthodes :

`L.append(x)`	ajoute `x` en fin de liste `L`	allonge la liste
`L.insert()`	insère x en position k dans L	allonge la liste
`L.pop(k)`	sort l'élément d'indice `k` [1] de `L`	raccourcit la liste et donne un élément
`L.sort()`	trie la liste	modifie la liste en un temps n.log(n)

Ecrivez un algorithme en Python. Expliquez en quelques mots l'algorithme que vous avez conçu. •3 pt.•
Indiquez le nombre maximum d'étapes en fonction des données *(ici la longueur de la liste)*. •1 pt.•
Attention, si vous risquez de modifier la liste par votre procédure, pensez à travailler sur une copie.

On va exploiter cette fonction pour un problème classique : quelle est la probabilité que deux élèves de la classe aient la même date d'anniversaire ?
On connaît la résolution rigoureuse de cette question *(dans un modèle simplifié où on ne tient pas compte des années bissextiles et où on estime que les dates de naissance sont équiprobables* [2]*)*. Mais on va ici utiliser l'ordinateur et son module `random`.
Je vous demande de tirer des listes de 48 nombres aléatoires compris entre 0 et 364 et de compter le pourcentage de listes dans lesquelles toutes les dates sont différentes. •4 pt.•
Variante : on suppose qu'il y a un mois de l'année où la probabilité de naissances double. Revoyez l'algorithme. •2 pt.•

[2] *dans les villages de mon enfance, on notait un pic de naissances trois mois avant la fête du village, ou plutôt neuf mois après*

2014-15

Algorithme par dichotomie. 1-phô

C'est l'algorithme classique qui cherche à chaque fois le milieu de l'intervalle et prend comme nouvel intervalle celui sur lequel il y a un changement de signe.

`precision = float(input(...))`

On saisit la précision ε avec laquelle on veut encadrer la racine. `input` ne suffirait pas, il faut s'assurer qu'on a bien un flottant.

`aa, fa, bb, fb = float(a), f(aa), float(b), f(bb)`

On initialise avec le premier segment $[a, b]$ et on regarde le signe à chaque extrémité *(en espérant que ce soit deux signes opposés).*

`while abs(bb-aa) > precision :`

On met en boucle tant que la longueur de l'intervalle $[aa, bb]$ est plus grande que ε *(il faut en effet s'assurer qu'on n'entre pas dans boucle folle).*

`....c = (aa+bb)/2`

et non pas `c = (a+b)/2`

`....fc = f(c)`

On cherche le milieu du segment et on calcule la valeur de f au milieu du segment.

```
....if fa*fc > 0 :
........bb, fb = c, fc
```

C'est là qu'est l'erreur. Si il n'y a pas eu un changement de signe entre `a` et `c`, alors il est entre `c` et `b`. Le nouvel intervalle doit être `[c, b]`. Comme `bb` ne change pas, c'est `aa` qu'on doit remplacer par `c`, et dans le même temps `fa` par `fc` *(déjà calculé).*

```
....else :
........aa, fa = c, fc
```

Dans l'autre cas, c'est l'autre moitié d'intervalle qu'on garde.

`print(aa, '< racine <', bb)`

Une fois qu'on a obtenu `abs(bb-aa)<precision`, on sort de la boucle et on affiche que la racine est entre `aa` et `bb`.

Ce qu'on aurait dû proposer :

```
....if fa*fc < 0 : #c'est le bon test
........bb, fb = c, fc
....else :
........aa, fa = c, fc
```

Ce qu'on pouvait mettre au début pour saisir `a` et `b` :

```
boucle = True
while boucle :
....a = float(input('Saisissez la premiere abscisse a :'))
....b = float(input('Saisissez la seconde abscisse b :'))
....fa, fb = f(a), f(b)
....if fa*fb > 0 :
........print('Mauvais intervalle, recommencez...')
....else :
........boucle = False
```

La boucle est effectuée tant que le booléen `boucle` est à la valeur `True` *(qu'on lui a donnée initialement).* Quand on a enfin $f(a).f(b) < 0$, on met `boucle` à `False` et on sort de la boucle.

`aa, fa, bb, fb = float(a), f(aa), float(b), f(bb)` c'est pour s'assurer que aa et bb sont bien des float et non pas des int (sinon, avec des int, la division par 2 risque de nous faire tomber brutalement sur 0, à tort).

D'autre part, on travaille donc sur une copie de a et b, et on garde les valeurs saisies de côté.
On stocke aussi f(a) et f(b) dans deux variables fa et fb pour ne pas les recalculer inutilement par la suite.

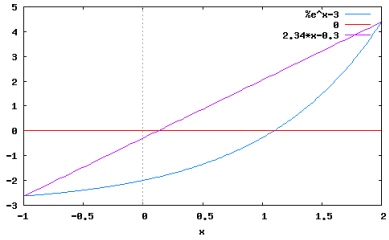
Le test `if fa*fc < 0 :` est inutilement lourd. Il calcule le produit de deux réels, alors que seul nous importe le signe. On perd donc un temps fou à calculer des décimales inutiles.
On va plutôt extraire le signe de chacune et calculer le produit des signes :
`if (fa<0 and fc>0) or (fa>0 and fc<0) :`

`aa, fa = c, fc` est préférable `aa, fa = c, f(c)` car dans ce dernier, on recalcule $f(c)$ qui a déjà été calculé auparavant. L'ordinateur est bête et obéissant. Il recalcule. Il fait donc deux fois un même calcul. Or, si le calcul de $f(x)$ est "long", c'est idiot. Or, si f n'est pas définie par une simple formule mais par une somme infinie que l'on tronque à un certain rang, ou comme une intégrale *(si si, ça existe)*, c'est un peu idiot.

2014-15

Méthode des sécantes. 1-phô

On dispose de deux points $A(aa,\ fa)$ et $B(bb,\ f(bb))$ ou même $A(aa,\ fa)$ et $B(bb,\ fb)$ avec nos notations.



Le premier élève a été pollué par le programme idiot du secondaire qui transforme tout en équations *(la dictature du signe "égale")*. Il cherche l'équation de la droite sous la forme la plus simple qui soit *(j'espère que c'est celle des programmes officiels)* :
$y = coeff.(x - a) + fa$ qui tient compte du coefficient directeur et se débrouille pour que l'on ait $y = fa$ pour $x = a$: $\boxed{y = \dfrac{fb - fa}{bb - aa}.(x - a) + fa}$

J'avoue que je renierai totalement l'élève (mais en est ce encore un ?) *qui passera par : l'équation est de la forme* $y = \alpha.X + \beta$ *et résoudra* $\alpha.aa + \beta = fa$ *et* $\alpha.bb + \beta = fb$. *Je crois que pour celui là, il n'y a plus rien à faire, à part une école de chimie* (avec le seul espoir qu'il n'ait ensuite aucun emploi, car je tiens à préserver les générations futures d'un tel danger).

Bref, ayant l'équation $y = \dfrac{fb - fa}{bb - aa}.(x - aa) + fa$, on résout $\dfrac{fb - fa}{bb - aa}.(\underline{x} - aa) + fa = 0$ et on trouve $s = aa - fa.\dfrac{bb - aa}{fb - fa}$ sachant que $fb - fa$ est non nul *(fa et fb sont de signes opposés).*
Un géomètre reconnaît qu'on part de l'abscisse aa et qu'on avance avec un certain coefficient directeur pour compenser l'ordonnée fa.

Le second élève a une approche d'algébriste. Il prend trois points $A(aa,\ fa)$, $B(bb,\ fb)$ et $S(s,\ 0)$. Il détermine deux vecteurs : $\overrightarrow{SA} = \begin{pmatrix} aa - s \\ fa \end{pmatrix}$ et $\overrightarrow{AB} = \begin{pmatrix} bb - aa \\ fb - fa \end{pmatrix}$ et il leur demande d'être colinéaires : $\begin{vmatrix} aa - s & bb - aa \\ fa & fb - fa \end{vmatrix} = 0$. On retrouve très vite le même calcul :

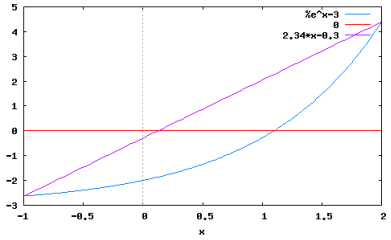
$$\boxed{s = aa - fa.\frac{bb - aa}{fb - fa} = \frac{fb.aa - fa.bb}{fb - fa} = \frac{f(b).a - f(a).b}{f(b) - f(a)}}$$

Sous cette forme plus compacte, on reconnaît d'ailleurs une moyenne de aa et b pondérée de coefficients fb et $-fa$ (tous deux positifs).

Le dernier élève a une approche de géomètre, avec des mesures d'angles ou ce qu'on appelle des

triangles semblables pour faire peur.

Il se place en S et mesure la tangente des deux côtes : $\frac{oppose}{adjacent} = \frac{fb}{bb-s} = \frac{fa}{aa-s}$. Il retrouve $aa.fb - bb.fa = s.(fb - fa)$.



Enfin, l'élève qui "voit les mathématiques" dit que s est bien une moyenne de aa et bb avec les coefficients correspondant eux ordonnées. Comme les plateaux d'une balance.

Disposant de cette formule, on modifie l'algorithme :

```
while abs(bb-aa) > precision :
....s = (aa*fb-bb*fa)/(fb-fa)
....fc = f(c)
....if fa*fc < 0 :
........bb, fb = c, fc
....else :
........aa, fa = c, fc
print(aa, '< racine <', bb)
```

On évitera bien sûr la formule `s = (aa*f(bb)-bb*f(aa))/(f(bb)-f(aa))` qui sollicite bien trop la fonction f pour des calculs répétitifs.

Cette méthode va nous rapporcher de la solution plus vite que la dichotomie. Elle va en effet couper l'intervalle en deux parts inégales et c'est dans le plus court des deux segments que se trouvera la racine *(sauf fonction particulière, changeant de concavité)*.

En effet, si $|f(aa)|$ est plus petit que $|f(bb)|$, l'abscisse s est plus proche de aa que de bb.

2014-15

Méthode de Newton 1-phô

Cette fois, on suppose f dérivable. On peut donc considérer la tangente en B et lui faire intersecter Ox (sauf si elle lui est parallèle).

Cette tangente a pour équation $y = f'(b).(x - b) + f(b)$. C'est direct : coefficient directeur, et exigence que y vaille $f(b)$ pour $x = b$.

On intersecte avec Ox : $f'(b).(x - b) + f(b) = 0$ et on trouve $Tb = b - \frac{f(b)}{f'(b)}$

La formule est homogène : $x - \frac{y}{dy/dx}$. Pour $f(b)$ et $f'(b)$ positifs, elle place Tb avant b.

On peut aussi effectivement raisonner sur les triangles et angles : $coefficient\ directeur = \frac{oppose}{adjacent} = \frac{fb}{b - Tb}$.

On a de même $Ta = a - \frac{f(a)}{f'(a)}$.

On isère de nouvelles lignes dans le programme :

```
fprimea, fprimeb = f'(a), f'(b)
if fprimea != 0 :
....Ta = a - fa/fprimea
else :
....Ta = a
if fprimeb != 0 :
....Tb = b - fb/fprimeb
```

```
else :
....Tb = b
```
On évite de devoir par un hasard énorme à diviser par 0.

L'optimisation pour chercher l'intervalle le plus court avec changement de signe sera plus lourde.
On met a, b, c, s, Ta et Tb dans une liste `Lx` (de longueur 6) et on met les images dans une autre.
```
Lx = [aa, bb, c, s, Ta, Tb]
Ly = [fa, fb, fc, fs, f(Ta), f(Tb)]
```
On parcourt cette liste sans prendre deux fois le même élément :
```
for i in range(1, 6) :
....for j in range(i) :
```
On regarde si il y a un changement de signe, et si oui, on mesure la longueur du segment :
```
........if Ly[i]*Ly[j] < 0
........long = abs(Lx[i]-Lx[j])
```
On regarde si on a trouvé une longueur optimale :
```
........if long < min :
```
Et on mémorise alors ce segment :
```
............bas, haut, min = min(Lx[i], Lx[j]), max(Lx[i], Lx[j]), long
```
A la fin, on a mis dans bas et haut les deux extrémités de l'intervalle (dans le bon sens) et on est sûr d'avoir pris le plus court.

```
Lx = [aa, bb, c, s, Ta, Tb]
Ly = [fa, fb, fc, fs, f(Ta), f(Tb)]
min = abs(bb-aa)
for i in range(1, 6) :
....for j in range(i) :
........if Ly[i]*Ly[j] < 0
........long = abs(Lx[i]-Lx[j])
........if long < min :
............bas, haut, min = min(Lx[i], Lx[j]), max(Lx[i], Lx[j]), long
```

2014-15

Savoir si il y a un élément en double dans une liste. 1-phô

Comme on va sûrement modifier la liste, on va travailler sur une copie appelée `double` :
```
def test(L) :
....double = L[ : ]
```
Une solution va constister à prendre un par un les éléments de la liste et à voir si on les retrouve dans le reste de la liste.
On notera qu'une fois qu'un élément a été testé, il n'est plus utile de le garder dans la liste, puisqu'on le sait unique.
Pour tester si le dernier élément de la liste `L` est en double :
```
dernier = L.pop()
if dernier in L : #puisque L ne contient plus dernier
....return(False) #et même éventuellement return(False, dernier)
```

Pour mettre ce processus en boucle :
```
while len(double) > 0 :
....dernier = double.pop()
....if dernier in L :
........return(False)
return(True) #si on est sorti de la boucle while c'est qu'aucun élément n'était en double
```
Attention, ne pas mettre un
```
....if dernier in L :
........return(False)
....else :
........return(True)
```

qui aurait pour effet de nous faire sortir de la procédure dès le premier test, même si il est favorable.

On pouvait aussi travailler en mode impératif :

```
for i in range(len(L)-1) : #on n'a pas de teste à faire sur le dernier élément de L
....for j in range(i+1, len(L)) : #on ne teste que les éléments qui suivent
........if L[i] == L[j] :
............return(False)
return(True) #si on a fait a double boucle sans retourner False, c'est que la liste est "injective"
```

On peut aussi être lourd :

```
L.sort() #on trie la liste
for i in range(len(L)-1) :
....if L[i] == L[i+1] :
........return(False)
```

Que ce soit par la boucle `for j in range()` que par le test `dernier in double`, on parcourt la liste dans son entier. Mais à chaque étape, la liste a diminué en taille.

L	dernier	test	nombre d'étapes	total
`[0, 2, 5, 7, 4, 8]`	8	`8 in [0, 2, 5, 7, 4]`	5	5
`[0, 2, 5, 7, 4]`	4	`4 in [0, 2, 5, 7]`	4	5+4
`[0, 2, 5, 7]`	7	`7 in [0, 2, 5]`	3	5+4+3
`[0, 2, 5]`	5	`5 in [0, 2]`	2	5+4+3+2
`[0, 2]`	2	`2 in [0]`	1	5+4+3+2+1
				n.(n+1)/2

En notant classiquement `N=len(L)`, le nombre maximum d'étapes est $\boxed{\frac{N.(N+1)}{2}}$ dans le cas de la liste sans doublon, ou dans la cas d'une liste comme `[5, 5, 2, 5, 7, 4, 8]`.

On passe au problème des dates de naissances. On importe la fonction `randrange` du module `random`.
On crée une liste de dates par `[randrange(365) for eleve in range(48)]`
On fait le test sur chacune de ces listes et quand il est favorable, on incrémente un compteur *(qu'on a initialisé à 0)*. On affiche au final le pourcentage :

```
from random import randrange
for k in range(1000) : #on va créer 1000 classes
....classe = [randrange(365) for eleve in range(48)]
....if test(classe) : #et pas if test(classe) == True qui est idiot
........compt += 1 #on augmente le compteur de 1
print(compt/1000.) #et pas compt/1000 qui donne une division entière, donc 0
```

L'application numérique donne **quatre vingt quinze pour cent** [3] des cas où deux élèves ont la même date d'anniversaire.
La résolution passant par "probabilité que les dates soient toutes distinctes" indique qu'à partir de vingt quatre élève, la probabilité dépasse 1/2. Ce que beaucoup de personnes interprètent en "à partir de vingt quatre élèves, il y en a deux qui ont le même annversaire".
La formule est $\boxed{1-\frac{365.364.363\ldots(365-N+1)}{365.365.365\ldots365}}$ à vous de la justifier.

Si on suppose maintenant que la répartition des dates n'est pas uniforme, il faut modifier le ranrange. On va suppose que le mois de décembre est renforcé *(par symétrie des rôles)*. Comme l'énoncé propose simplement d'en doubler la probabilité, on va travailler avec une année de 365+31 jours, et on va identifier les derniers, ceux de décembre bis avec ceux de décembre.

On remplace le tirage `ranrange(365)` par

```
jour = randrange(365+31)
if jour > 365 :
```

[3] allez voir la chanson de Georges Bassens intitulée "quatre quinze fois sur cent", même si elle n'a rien à voir

`....jour = jour-31`
Tous les jours au delà de 365 sont ramenés en jours du mois de décembre qui héritent donc d'une probabilité doublée.
On notera que cela ne change guère les valeurs mentionnées plus haut.