Correction du concours blanc

Informatique pour tous, première année

Julien Reichert

Exercice 1

```
def evalue_poly(l,x):
    xpuissancen = 1
    somme = 0
    for i in range(len(l)):
        somme += l[i] * xpuissancen # plus efficace que l[i] * x ** i
        xpuissancen *= x
    return somme

def convertit_fonction(l):
    def fonction(x):
        return evalue_poly(l,x) # ou coller le code de la fonction
    return fonction

def derive_poly(l):
    if len(l) == 1:
        return [0] # Au pire, si on assimilait [] au polynôme 0, il n'y aurait pas de contradiction
    ll = []
    for i in range(1,len(l)):
        ll.append(l[i]*i)
    return ll

def newton_poly(l,dep,eps):
    ll = derive_poly(l)
    xn = depart
    xnplusun = xn - evalue_poly(l,xn) / evalue_poly(ll,xn)
    while abs(xnplusun - xn) > epsilon:
        xn = xnplusun
        xnplusun = xnplusun - evalue_poly(l,xnplusun) / evalue_poly(ll,xnplusun)
    return xnplusun

def newton_poly_bis(l,dep,eps):
    fonction = convertit_fonction(l)
    derivee = convertit_fonction(derive_poly(l))
    xn = depart
    xnplusun = xn - fonction(xn) / derivee(xn)
    while abs(xnplusun - xn) > epsilon:
        xn = xnplusun
        xnplusun = xnplusun - fonction(xnplusun) / derivee(xnplusun)
    return xnplusun
```

Exercice 2

La ligne manquante est `y = y + (x_plus_h-x)*G(y,x)`.

Concernant la méthode d'Euler implicite, puisqu'on crée une équation dont la seule inconnue est la valeur qu'on est en train de déterminer, il suffit de résoudre cette équation à l'aide d'une méthode à la Newton (sans utiliser la dérivée de l'expression qu'on cherche à annuler, car on n'en dispose pas).

On peut alors écrire la méthode de la sécante, ou appeler directement `scipy.optimize.newton`.

Avec une fonction anonyme, le code est plus court :

```
from scipy.optimize import newton

def euler_implicite(G,y0,les_x):
    les_y = [y0]
    x_plus_h,y = les_x[0], y0
    for i in range(1,len(les_x)):
        x,x_plus_h = x_plus_h,les_x[i]
        y = newton(lambda yy : (x_plus_h-x)*G(yy,x_plus_h)-yy+y,y,10**-9) # !
        les_y.append(y)
    return les_y
```

Sinon, une fonction locale marche bien, la ligne marquée d'un point d'exclamation est à remplacer par les lignes suivantes :

```
        def fonction_a_newtoniser(yy):
            return (x_plus_h-x)*G(yy,x_plus_h)-yy+y
        y = newton(fonction_a_newtoniser,y,10**-9)
```

Deux réécritures possibles (avec renommage) de la fonction `newton` sans fournir la dérivée :

```
def secante(f,xn,e):
    xnp1 = xn+e
    while abs(xnp1 - xn) > e:
        xn,xnp1 = xnp1,xnp1-f(xnp1)/((f(xnp1)-f(xn))/(xnp1-xn))
    return xnp1

def newton_avec_accroissements(fonction,depart,epsilon):
    def derivee(xn):
        return (fonction(xn+epsilon)-fonction(xn-epsilon))/(2*epsilon)
    xn = depart
    xnplusun = xn - fonction(xn) / derivee(xn) # les divisions par zéro ne sont pas rattrapées
    while abs(xnplusun - xn) > epsilon: # autre version : abs(f(xn)) > epsilon
        xn = xnplusun
        xnplusun = xnplusun - fonction(xnplusun) / derivee(xnplusun)
    return xnplusun
```