Détermination du pH d'un acide faible et méthodes de continuation

Auteur: TSANA FOBING Sinclair (*Etudiant en 2ème année Bachelier Sciences Mathématiques Université de Mons*)
Enseignant: M. Christophe TROESTLER

Objectifs du projet:

Ce projet est réalisé dans le cadre du cours d'Introduction à l'Analyse numérique dispensé en BAB2 à l'université de Mons. Il est question d'aborder un problème concret grâce aux techniques numériques vues au cours que dont certaines serons adaptées de manière à répondre à la série des 11 questions proposées.

Résultats attendues:

> 1. Un bref rapport écrit qui résume les développements mathématiques et les solutions apportées aux problèmes posées
> 2. Les conclusions des expériences numériques.

Note: Ce notebook constitut une réponse à ces deux points. Nous y présenterons à la fois le `code python` ainsi que les détails de nos développements mathématiques.

Importation des librairies nécessaires

In [12]:
```
import matplotlib.pyplot as plt   # pour les représentations graphiques
import math                       # contient les fonctions mathématiques usuels tels que log, sin, et
c...
import random
import array
import numpy as np                # pour les calculs numériques
from sympy import *               # pour les calculs symboliques
from math import log10
from scipy.optimize import brentq
from scipy.integrate import solve_ivp
from numpy.linalg import inv
from PIL import Image
```

In [2]:
```
# Quelques constantes utilisées
k_a = 1.8*10**(-5)     # Constante d'acidité  propre à l'acide
k_e = 1.0116*10**(-14) # La constante ionique de l'eau
```

In [14]:
```
Image1 = Image.open("logo-umons.png");
Image2 = Image.open("tv1.png");
Image3 = Image.open("tv2.png");
Image.show()
```

On s'intéresse à la variation du pH d'une solution aqueuse d'un acide faible AH (formé d'un atome d'hydrogène H et d'une autre partie notée A) en fonction de $-log_{10}c$ où $c > 0$ représente la concentration de l'acide. L'analyse de ce problème conduit aux équations suivantes :

(1) $[H_3O^+][OH^-] = k_e$

(2) $[H_3O^+] = [OH^-] + [A^-]$

(3) $[AH]k_a = [A^-][H_3O^+]$

(4) $[AH] + [A^-] = c$

Question 1

1.a - Détermination du polynôme

On procède par élimination successive de $[OH^-]$, $[A^-]$ et $[AH]$ des équations ci-dessus.

(1) $\Rightarrow [OH^-] = \frac{k_e}{[H_3O^+]}$

Dans (2) on a:

$[H_3O^+] = \frac{k_e}{[H_3O^+]} + [A^-] \iff [A^-] = [H_3O^+] - \frac{k_e}{[H_3O^+]}$ $(\ast)$

Dans (3) on a:

$[AH] = \frac{1}{k_a}([H_3O^+]^2 - k_e$

Dans (4) on a:

$[A^-] = c - \frac{1}{k_a}([H_3O^+]^2 - k_e)$ $(\ast\ \ast)$

$(\ast)$ et $(\ast\ \ast)$ donnent:

$[H_3O^+] - \frac{k_e}{[H_3O^+]} = c - \frac{1}{k_a}([H_3O^+]^2 - k_e)$

En multipliant par $k_a[H_3O^+]$, on obtient:

$k_a[H_3O^+]^2 - k_ak_e = ck_a[H_3O^+] - [H_3O^+]^3 + k_e[H_3O^+]$

$\iff [H_3O^+]^3 + k_a[H_3O^+] - (ck_a[H_3O^+] + k_e[H_3O^+]) - k_ak_e = 0$

D'ou l'équation $X^3 + k_aX^2 - (k_e + ck_a)X - k_ak_e = 0$ où X représente la concentration en H_3O^+

1.b - Montrons que le polynôme P possède une unique racine positive.

Considérons la fonction polynôme $f_c(x) = x^3 + k_ax^2 - (k_e + ck_a)x - k_ak_e$ où c est un paramètre réel strictement positif.

- **Etudions les variations f_c**

La fonction f_c est dérivable sur $\mathbb{R}$ comme fonction polynôme.

On a: $f_c'(x) = 3x^2 + 2k_ax - ck_a - k_e$

$f_c'(x) = 0 \iff 3x^2 + 2k_ax - ck_a - k_e = 0$

On obtient un polynôme du secon degrée dont le discriminant est:

$\Delta = 4k_a^2 + 12(ck_a + k_e) > 0$

Les points critiques sont donc:

$x_1 = \frac{-2k_a+\sqrt{\Delta}}{6}$ et $x_2 = \frac{-2k_a-\sqrt{\Delta}}{6}$

On a donc le tableau de variations généralisé suivant:

`Image2.show()`

Pour tout x positif, x_1 est potentiellement l'unique point critique à condition que l'on est $x_1 \geq 0$.

$x_1 \geq 0 \iff -2k_a + \sqrt{\Delta} \geq 0$

$\iff \Delta \geq 4k_a^2$

$\iff 4k_a^2 + 12(ck_a + k_e) \geq 4k_a^2$

$\iff c \geq \frac{-k_e}{k_a}$

Le réel c étant positif, on aura toujours x_1 positif.

Ainsi, pour tout c positif, le tableau de variations de la fonction f_c sur [0,+∞[est le suivant:

`Image3.show()`

On voit que la fonction f_c est à valeurs négatives sur $[0, x_1]$ et f_c croit infiniment sur $]x_1, +\infty[$.

Donc la courbe de la fonction f_c rencontrera l'axe des abscisses une seule fois sur l'intervalle $]x_1, +\infty[$.

On conclut que pour tout réel c positif, le polynôme $x^3 + k_ax^2 - (k_e + ck_a)x - k_ak_e$ admet **une unique** racine positive sur l'intervalle]0,+∞[.

1.c - Algorithme de Horner pour évaluer P

On considère le polynôme $P(x) = a_nx^n + a_{n-1}x^{n-1} + \ldots + a_1x^1 + a_0$

La méthode de Horner consiste a écrire P(x) sous la forme $P_1(x).x + a_0$ et de repéter cette opération sur P_1 et ainsi de suite jusqu'a obtenir le polynôme constant $P_n(x) = a_n$. Son implémentation en python est la suivante:

In [3]:
```
def horner(A,x): # A est le vecteur contenant les coéfficients (a_i)_{i=0...n}
    n = len(A)-1
    p = A[n]
    for i in range(1,n+1):
        p = p*x + A[n-i]
    return p
```

1.d - Test de l'Algorithme de Horner et méthode des invariants

In [4]:
```
c = random.uniform(0,1) # On génère un nbre aléatoire entre 0 et 1 pour tester notre fonction
A = [1, k_a, -(k_e+c*k_a), -k_a*k_e]
```

In [5]:
```
horner(A,1)== sum(A)   # on teste l'image de 1
```

Out[5]:
```
True
```

Méthode des invariants

Un invariant de boucle est une propriété Q qui vérifie:

- **Est vraie avant l'entrée dans la boucle.**
- **Est toujours vraie après chaque itération de la boucle.**
- **Vaut la propriété P désirée a la fin de l'exécution de la boucle.**

Cas de l'Algorithme de Horner

Pour la preuve d'exactitude, nous utilisons les mêmes notations que celles du programme écrit précédement

Entrée: $A = [a_n, a_{n-1}, \ldots, a_0]$

Invariant de boucle:

Nous proposons comme invariant la propriété $Q \equiv p(A[i], x) = \sum_{j=0}^{i} A[n-j]x^{i-j}$

Preuve:

> **Avant la boucle**: $i = 0$
>
> Et on a $p(A[0], x) = \sum_{j=0}^{0} A[n-j]x^{i-j} = A[n]$
>
> **A l'entrée de la boucle:**
>
> Supposons que la proposition Q soit vraie une étape i. C'est à dire $p(A[i], x) = \sum_{j=0}^{i} A[n-j]x^{i-j}$.
>
> Montrons qu'elle est vraie à la prochaine itération $i + 1$. En effet à chaque itération, on a:
>
> $p(A[i], x).x + A[n-i] = (\sum_{j=0}^{i} A[n-j]x^{i-j}).x + A[n-i]$
>
> $= \sum_{j=0}^{i} A[n-j]x^{i-j+1} + A[n-i]$
>
> $= \sum_{j=0}^{i'} A[n-j]x^{i'-j}$ où $i' = i + 1$
>
> On a donc $p(A[i+1], x) = p(A[i], x).x + A[n-i]$
>
> **Après la boucle**: $i = n$
>
> et on a $p(A[n], x) = \sum_{j=0}^{n} A[n-j]x^{n-j} = A[n]x^n + A[n-1]x^{n-1} + \ldots + A[1]x + A[0]$ qui est la proriété désirée.

On conclut que la méthode de d'Horner est correcte.

1.e - Valeurs de $c > 0$ en fonction de k_e et k_a pour que $[H_3O^+] \leqslant 1$

D'après la question (1-a), la fonction f_c est décroissante sur $[0, x_1]$ pour tout c positif.

Ainsi sur cet intervalle, on a par décroissance de f_c:

$$[H_3O^+] \leqslant 1 \implies f_c([H_3O^+]) \geq f_c(1)$$

.

Mais on a $f_c([H_3O^+]) = 0$ et $f_c(1) = 1 + k_a - ck_a - k_e - k_ak_e$

Donc il suffit que $f_c(1) \leqslant 0$

C'est-à-dire $1 + k_a - ck_a - k_e - k_ak_e \leqslant 0 \iff ck_a \geq 1 + k_a - k_e - k_ak_e$

$\iff ck_a \geq (1 + k_a)(1 - k_e)$

$\iff c \geq \frac{(1+k_a)(1-k_e)}{k_a}$

Mais comme $c > 0$, on a finalement: $\boldsymbol{c \in]0, \frac{(1-k_e)(1+k_a)}{k_a}]}$

1.e - Représentation du polynôme p sur [0; 0.005] pour c $\in$ {1., 0.1, 0.01, 10^{-12}}

In [6]:
```
C = np.array([1.,0.1, 0.01, 10**(-12)])
coefs = [] # vecteur contenant les coefficients du polynôme pour les différentes valeurs de C
for i in C:
    res = np.array([-k_a*k_e, -(k_e+i*k_a), k_a, 1])
    coefs.append(res)
    coefs
```

In [7]:
```
coefs
```

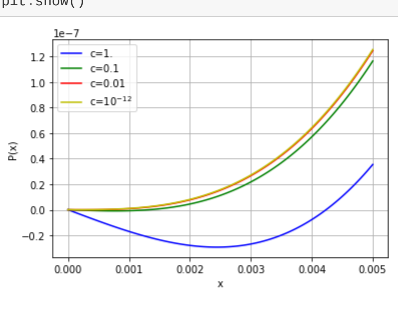
Out[7]:
```
[array([-1.82088e-19, -1.80000e-05,  1.80000e-05,  1.00000e+00]),
 array([-1.82088000e-19, -1.80000001e-06,  1.80000000e-05,  1.00000000e+00]),
 array([-1.8208800e-19, -1.8000001e-07,  1.8000000e-05,  1.0000000e+00]),
 array([-1.82088e-19, -1.01340e-14,  1.80000e-05,  1.00000e+00])]
```

In [8]:
```
x = np.linspace(0,0.005, 100)
px1 = [];px2 = [];px3 = [];px4 = []; # initialisation des vecteurs  qui recevront les valeurs des pol
ynômes
for i in range(len(x)):
    px1.append(horner(coefs[0],x[i]))
    px2.append(horner(coefs[1],x[i]))
    px3.append(horner(coefs[2],x[i]))
    px4.append(horner(coefs[3],x[i]))

px = [px1,px2,px3,px4]
colors = ['b','g','r','y']
labs = ["c=1.","c=0.1", "c=0.01", "c=$10^{-12}$"]

for i in range(0,4):
    plt.plot(x,px[i],color=colors[i],label=labs[i])

plt.xlabel('x')
plt.ylabel('P(x)')
plt.grid()
plt.legend()
plt.show()
```



1.f - Courbe du PH en fonction de $-\log_{10}(c)$

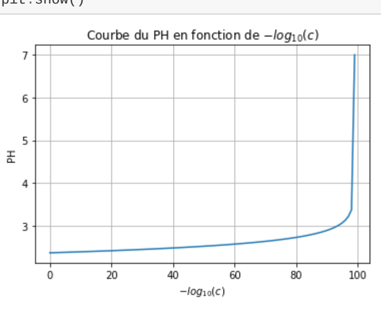
A partir de la suite $x_{n+1} = x_n + h\frac{f'(x_n)}{f(x_n)}$, on écris une routine `nextPoint()` qui, avec un pas constant h, détermine le point de la prochaine itération. Dans notre cas nous prenons un pas $h = 10^{-4}$

In [9]:
```
def nextPoint(x0,fxn,fprim,h=10**(-4)):
    xnext = x0+h*fprim/fxn
    return xnext
```

In [10]:
```
C = np.linspace(0,1, 100)
coef3 = [] # vecteur contenant le 3e coéfficient du polynôme pour les différentes valeurs de C
for i in C:
    coef3.append((k_e+i*k_a))
    coef3

racines = []
n=len(coef3)-1
xnext = 1+1.0e-15 # on initialise par une valeur superieur à 1
for i in range(n,-1,-1):
    f = lambda x: x**3+k_a*x**2-coef3[i]*x-k_a*k_e
    fp = lambda x: 3*(x**2)-2*k_a*x-coef3[i]
    xnext = nextPoint(xnext,f(xnext),fp(xnext))
    res = brentq(f, 0, xnext)
    racines.append(res)
```

In [11]:
```
x = [-log10(r) for r in racines]
plt.plot(x)
plt.title("Courbe du PH en fonction de $-log_{10}(c)$")
plt.xlabel("$-log_{10}(c)$")
plt.ylabel("PH")
plt.grid()
plt.show()
```



Explications:

> On remarque que pH est une fonction croissante de $-\log_{10}(c)$. Ce qui signifie que lorsque $-\log_{10}(c)$ augmente (i.e c diminue), le pH augmente aussi. Autrement dit plus la concentration c diminue, moins la solution est acide.

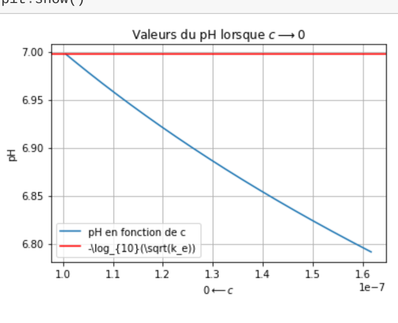
1.g) Preuve des propriétés constatées graphiquement

- **Continuité**:
- **Monotonie**:
- **Dérivabilité**:

Preuve que pH $\longrightarrow -log_{10}\sqrt{k_e} \approx 7$ lorsque $c \longrightarrow 0$

Commencons par une analyse gaphique. Le code suivant nous permet de représenter la courbe de la variation du pH losque c tend vers 0.

In [8]:
```
v = np.arange(10**(-7),0,-10**(-10))
res = []
for c in v:
    f = lambda x: x**3+k_a*x**2-(c*k_a+k_e)*x-k_a*k_e
    sol = brentq(f, 0, 1)
    res.append(sol)
x = res
y = [-log10(r) for r in res]
s = -log10(math.sqrt(k_e))
lab = '-\log_{10}(\sqrt(k_e))'
plt.plot(x,y,label = "pH en fonction de c")
plt.axhline(y = s, color = 'r',label = lab)
plt.title("Valeurs du pH lorsque $c\longrightarrow 0$")
plt.xlabel("$0 \longleftarrow c$")
plt.ylabel("pH")
plt.grid()
plt.legend()
plt.show()
```



Cette représentation nous permet de voir que lorsque c tend vers 0, l'écart entre le pH et droite $y = -\log_{10}\sqrt{k_e}$ se réduit.

Preuve de l'observation

Lorsque $c \longrightarrow 0$, notre équation de départ devient $x^3 + k_ax^2 - k_ex + k_ak_e = 0$.

Posons $Q(x) = x^3 + k_ax^2 - k_ex + k_ak_e$, on remarque que $Q(-k_a) = 0$

Grace à cette racine, on obtient la forme factorisée suivante : $Q(x) = (x + k_a)(x^2 - k_e)$

Donc $Q(x) = 0 \iff x + k_a = 0$ ou $x^2 - k_e = 0 \iff x = -k_a$ ou $x = \pm\sqrt{k_e}$

Mais $x > 0$, on a donc $x = \sqrt{(k_e)}$

Ainsi,lorsque $c \longrightarrow 0$, on a $[H_3O^+] \longrightarrow \sqrt{(k_e)}$.

Et comme $pH = -log_{10}[H_3O^+]$

On en déduit que $pH \longrightarrow -log_{10}\sqrt{k_e}$ lorsque $c \longrightarrow 0$

> Ce resultat **est logique d'un point de vue chimique** puisque nous avons à faire à un acide et donc son PH est donc inférieur ou égale à 7. mais cet acide est faible donc son pH se rapproche de celui de l'eau et l'eau est équilibrée(i.e de $pH \approx 7$)

Question 2: Adapdation de la méthode du point fixe

- **Montrons que la suite $(x_n)_{n\in\mathbb{N}}$ est de Cauchy:**

Soit $x_0 \in B$, comme B est stable par $\varphi(c, .)$ par défintiion de $\varphi(c, .)$, on peut poser $x_n = \varphi(c, x_{n-1})$ pour tout $n \in \mathbb{N}^*$

On a alors:

$\|x_{n+1} - x_n\| = \|\varphi(c, x_n) - \varphi(c, x_{n-1})\| \leqslant L\|x_n - x_{n-1}\|$

On obtient par récurrence sur n que

$\|\varphi(c, x_{n+1}) - \varphi(c, x_n)\| \leqslant L^n\|x_1 - x_0\|$

Ainsi, pour tout $n \in \mathbb{N}$ et $k \in \mathbb{N}^*$, on a :

$\|x_{n+k} - x_n)\| \leqslant \sum_{i=0}^{k-1}\|x_{n+i+1} - x_{n+i}\|$.

$\leqslant \sum_{i=0}^{k-1}L^{n+i}\|x_1 - x_0\|$

$\leqslant \|x_1 - x_0\|\sum_{i=0}^{k-1}L^{n+i}$

$\leqslant L^n\frac{1-L^k}{1-L}\|x_1 - x_0\|$

$\leqslant \frac{L^n}{1-L}\|x_1 - x_0\| \longrightarrow 0$ quand n $\longrightarrow +\infty$ car $0 < L < 1$

Ainsi la suite $(x_n)_{n\in\mathbb{N}}$ est de Cauchy.

- **Convergence:**

Comme $\mathbb{R}^n$ est complet, alors il existe $x^* \in \mathbb{R}$ tel que $x_n \longrightarrow x^*$.

De plus comme B est fermé, cette limite $x^* \in B$ et pour tout $n \in \mathbb{N}$, on a $x_{n+1} = \varphi(c, x_n)$

Par ailleurs, en utilisant la continuité de $\varphi(c, .)$ et en passant à la limite, on a $x^* = lim_{n\longrightarrow+\infty}x_{n+1} = lim_{n\longrightarrow+\infty}\varphi(c, x_n) = \varphi(c, lim_{n\longrightarrow+\infty}x_n) = \varphi(c, x^*)$

- **Unicité:**

Soit $L \in]0, 1[$ tel que:

$\forall c \in I, \forall x_1, x_2 \in B\|\varphi(c, x_1) - \varphi(c, x_2)\| \leqslant L\|x_1 - x_2\|$

Supposons que x et y sont deux points fixes de $\varphi(c, .)$, alors on a:

$\|x - y\| = \|\varphi(c, x) - \varphi(c, y)\| \leqslant L\|x - y\|$ ce qui implique $\|x - y\|(1 - L) \leqslant 0$

Mais on a $1 - L > 0$ car $L \in]0, 1[$

Donc $\|x - y\| = 0$ et donc x=y. D'ou l'unicité de x^*

On peut ainsi définir la fonction $\gamma : I \longrightarrow B$

$c \longmapsto$ l'unique x^* tel que $x^* = \varphi(c, x^*)$

Question 3: Montrons que la fonction φ vérifie les hypothèses du théorème du point fixe

On sait que pour tout $c \in I$, la fonction $\varphi(c, .)$ est **continue sur B**

De plus, l'hypothèse (a) nous dit que $\partial_x\boldsymbol{\varphi(c, x)}$ **existe** en tout point $c \in I$ et $x \in B$

Ces deux hypothèses montrent que pour tout $c \in I$ la fonction $\boldsymbol{\varphi(c, .)}$ **est de classe C^1 sur B**

Sous ces hypothèses, **l'Inégalité des Accroissement Finis(IAF)** nous assure que pour tout $x_1, x_2 \in B$:

$\|\varphi(c, x_1) - \varphi(c, x_2)\| \leqslant \|x_1 - x_2\| \sup_{x\in intB}\|\partial_x\varphi(c, x)\|$

$\leqslant \|x_1 - x_2\| \sup_{c\in I}(\sup_{x\in intB}\|\partial_x\varphi(c, x)\|)$ $(\ast)$

Comme B est fermé, alors pour tout compact $K \subseteq B$, il existe $L \in]0, 1]$ tel que $\sup_{x\in intB}\|\partial_x\varphi(c, x)\| \leqslant L \sup_{x\in K}\|\partial_x\varphi(c, x)\|)$

$(\ast)$ devient : $\|\varphi(c, x_1) - \varphi(c, x_2)\| \leqslant \|x_1 - x_2\| \sup_{c\in I}(L \sup_{x\in K}\|\partial_x\varphi(c, x)\|)$

$\leqslant L\|x_1 - x_2\| \sup_{c\in I}(\sup_{x\in K}\|\partial_x\varphi(c, x)\|$

$\leqslant L\|x_1 - x_2\|$ Car $\sup_{c\in I}(\sup_{x\in intB}\|\partial_x\varphi(c, x)\|) < 1$ d'après l'hypothèse (b)

Question 4: Preuve du Théorème des Fonctions Implicites

Etape 1: Montrons que la fonction φ est différentiable dans un voisinage approprié de (c_0, x_0).

- **Montrons que la fonction $\gamma = \varphi(c, .)$ est Lipschitzienne**

Comme $\frac{\partial f}{\partial x}(c, .)$ est continue en c_0, alors $\frac{\partial\varphi}{\partial x}(c, .)$ l'est aussi.

Soit $q \in]0, 1[$, on a ,par définition de la continuité de $\frac{\partial\varphi}{\partial x}$ au point c_0,qu'il existe $r > 0$ et $R > 0$ tel que:

$\forall x \in B, \|c - c_0\| \leqslant r, \|x - c_0\| \leqslant R \implies \|\frac{\partial\varphi}{\partial x}(c, x) - \frac{\partial\varphi}{\partial x}(c_0, c_0)\| \leqslant q$ (i)

Par définition de la continuité de γ au point c_0,on a que:

Pour tout $\epsilon > 0$, il existe $\delta > 0$ tel que: $\|c - c_0\| \leqslant \delta \implies \|\gamma(c) - \gamma(c_0)\| = \|\varphi(c, c_0) - \varphi(c_0, c_0)\| \leqslant \epsilon$

En particulier pour $\epsilon = R(1 - q)$, on a :$\|c - c_0\| \leqslant \delta \implies \|\gamma(c) - \gamma(c_0)\| = \|\varphi(c, c_0) - \varphi(c_0, c_0)\| \leqslant R(1 - q)$

Posons $r_0 = min(r, \delta)$,on a:

$\|c - c_0\| \leqslant r_0, \|x - c_0\| \leqslant R \implies \|\gamma(x) - \gamma(c_0)\| \leqslant \|\varphi(c, x) - \varphi(c, c_0)\| + \|\varphi(c, c_0) - \varphi(c_0, c_0)\|$

$\implies \|\gamma(c) - \gamma(c_0)\| \leqslant q\|x - c_0\| + R(1 - q)$ d'après (i) et l'IAF

$\implies \|\gamma(c) - \gamma(c_0)\| \leqslant qR + R(1 - q) = R$

Ainsi, $\|\gamma(c) - \gamma(c_0)\|_\infty = \sup\{\|\varphi(c, x) - \varphi(c_0, c_0)\|, \|x - c_0\| \leqslant r_0\}$

Et pour tout $x_1, x_2 \in B(c_0, R)$, on a:

$\|\gamma(x_1) - \gamma(x_2)\|_\infty = \sup\{\|\varphi(c, x_1) - \varphi(c, x_2)\|, \|x - c_0\| \leqslant r_0\}$

$\leqslant q \sup\{\|x_1 - x_2\|, \|x - c_0\| \leqslant r_0\}$ d'après (i) et l'IAF

$\leqslant q\|x_1 - x_2\|_\infty$

Donc la fonction γ est Lipschitzienne.

La fonction γ est continue et dérivable pour tout $c \in B(c_0, R)$.Ce qui nous permet de dire que la fonction φ est **différentiable** pour tout $(c, x) \in B(c_0, r) \times B(c_0, R)$

Etape 2: Montrons que φ est lipschitzienne.

On a $\varphi(c, x) = x - (\partial_xf(c_0, c_0))^{-1}(\frac{\partial f}{\partial x}(c, x))$

Ainsi, pour tout $(c, x) \in B(c_0, r) \times B(c_0, R)$, $\frac{\partial\varphi}{\partial x}(c, x)$ existe et on a:

$\frac{\partial\varphi}{\partial x}(c, x) = I_2 - (\partial_xf(c_0, c_0))^{-1}(\frac{\partial f}{\partial x}(c, x))$ où $I_2 : \mathbb{R}^2 \longrightarrow \mathbb{R}$ est l'application identité.

Au point c_0, on a: $\frac{\partial\varphi}{\partial x}(c_0, c_0) = I_2 - (\partial_xf(c_0, c_0))^{-1}(\frac{\partial f}{\partial x}(c_0, c_0)) = 0$

Donc $\frac{\partial\varphi}{\partial x}(c_0, c_0) = 0$ et (i) devient:

$\forall x \in B\|c - c_0\| \leqslant r, \|x - c_0\| \leqslant R \implies \|\frac{\partial\varphi}{\partial x}(c, x)\| \leqslant q$

Ainsi d'après l'Inégalité des Accroissements Finis, on a:

$\forall x_1, x_2 \in B, \|c - c_0\| \leqslant r, \|x - c_0\| \leqslant R \implies \|\varphi(c, x_1) - \varphi(c, x_2)\| \leqslant q\|x_1 - x_2\|$

Donc la fonction \varphi est Lipschitzienne.

On conclut que la fonction \varphi (c,x) vérifie (3)

Question 5:Preuve d'existence d'un prolongement maximal par le Lemme de Zorn

Soit E l'ensemble des fonctions qui prolongent \gamma_0.

On définit l'ordre **"prolonger"** que nous notons \preceq telle que:

Pour tout i\neq j, \hspace{10px}\gamma_i\preceq \gamma_j signifie \hspace{10px}"\gamma_j prolonge \gamma_i"

De toute suite d'éléments \gamma_i : I_i\longmapsto \mathbb{R}^N, i\in \{1,2,...,n\} de E, on peut extraire une chaîne telle que

\gamma_n\preceq \gamma_{n-1}...\gamma_1\preceq \gamma_0

Ainsi, (E,\preceq) est ensemble inductif.

On peut donc faire appel au lemme de Zorn qui garanti l'existence d'un prolongement maximal de \gamma _0.

Question 6:

Supposons par l'absurde qu'il existe une autre extension \gamma':I' \longmapsto \mathbb{R}^N de \gamma_0

Alors on distingue les cas suivants:

- Si \gamma' prolonge \gamma:

> Alors \gamma' est un prologement de \gamma_0. Ce qui contredirait la maximalité de \gamma_0

- Si \gamma prolonge \gamma'

> Alors on a d'une part I'\subseteq I et pour tout c\in I' \gamma(c)=\gamma'(c) </br> D'autre part pour tout c\in I\setminus I' \gamma(c)\neq\gamma'(c).</br>
>
> > Donc il existe x_1\neq x_2 tels que x_1=\gamma(c) et x_2=\gamma'(c). Ce qui contredit le Théorème des Fonctions Implicites et donc l'inversibilité de \partial _{c} f(c,\gamma (c)) pour tout c\in I.

On a donc nécessairement \gamma=\gamma'

On conclut que \gamma est l'unique extension de \gamma_0

Question 7: Dérivée de la fonction \gamma

On suppose que pour tout c\in I \partial _{c} f(c,\gamma (c)) est inversible.

La fonction c\in I\longmapsto f(c,\gamma (c)) étant différentiable comme composée de fonctions différentiables, la règle de dérivation par chaîne (chain rule) assure qu'en tout point de \Omega:

\frac{\partial f}{\partial c}(c,\gamma (c))+\frac{\partial f}{\partial x}(c,\gamma (c)).\partial _c \gamma (c)=0 \iff \partial _c \gamma (c)= (\frac{\partial f}{\partial x}(c,\gamma (c)))^{-1}.\frac{\partial f}{\partial c}(c,\gamma(c))

Question 8: Routine pour la Résolution de l'EDO

En considérant la relation précédente comme une Equation différentielle Ordinaire avec la condition initiale \gamma(0)=x_0, nous utilisons la fonction `solve_ivp` du module `scipy` pour implémenter une routine que nous appelons `solveEDO()`. Notre routine prends en entrée x_0 et détermine la courbe des solution \gamma.

Note: Pour traiter le cas où \partial _{c} f(c,\gamma (c)) n'est pas inversible, notre routine signale ce fait et demande une nouvelle valeur de x_0 à l'utilisateur.

In [9]:
```
def f(c,x):
    return x**3+k_a*x**2-(c*k_a+k_e)*x-k_a*k_e  # Polynome P(c,x)
def dfdc(c,x):
    return -k_a*x                               # dérivée partielle de P par rapport à c
def dfdx(c,x):
    return 3*x**2+2*k_a*x-(c*k_a+k_e)          # dérivée partielle de P par rapport à x
```

In [10]:
```
def solveEDO(x0):
    def gama(c,x):      #définition de la fonction
        res = -dfdc(c,x)/dfdx(c,x)
        if np.isnan(res):    # Traitement du cas ou dfdx n'est pas inversible
            x0 = input("Erreur! - dérivée nulle. Essayez avec une autre valeur de x0")
            try:
                solveEDO(x0)
            except ZeroDivisionError:
                print("Division par zero!")
        return res
    points = np.linspace(0,1,100) # points d'évaluation
    sol = solve_ivp(fun=gama,t_span=[0, 1], y0=[x0],t_eval=points)
    return sol.y
```

Testons notre routine pour x_0 = 10^{-6}

In [11]:
```
gamma = solveEDO(x0=10**(-6))
gamma[0][:20]    # On affiche les 20 premières valeurs calculées par notre routine
```

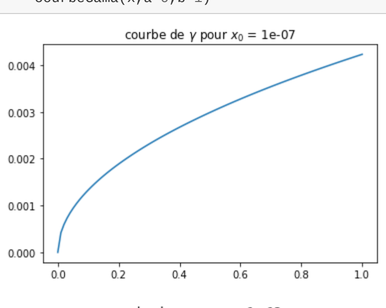
Out[11]:
```
array([1.00000000e-06, 4.17285412e-04, 5.93776083e-04, 7.29070228e-04,
       8.43381180e-04, 9.44228395e-04, 1.03495322e-03, 1.11835730e-03,
       1.19641120e-03, 1.26979829e-03, 1.33909706e-03, 1.40484692e-03,
       1.46753825e-03, 1.52761244e-03, 1.58546185e-03, 1.64142968e-03,
       1.69570352e-03, 1.74834972e-03, 1.79946966e-03, 1.84916060e-03])
```

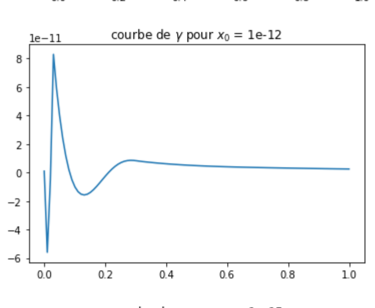
Le programme `courbeGama()` ci-dessous prend en entrée trois paramètres x_0, a et b et permet de représenter la courbe de \gamma sur l'intervalle [a,b] avec la condition initiale \gamma_0 = x_0

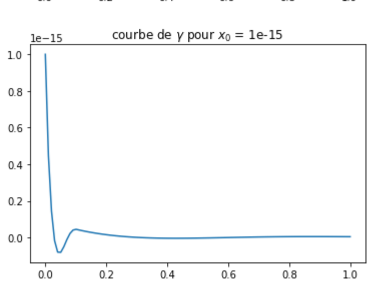
In [20]:
```
def courbeGama(x0,a,b):
    pts_test = np.linspace(a,b,100) # points de test
    gama = solveEDO(x0).T      # transposée
    plt.plot(pts_test, gama)
    plt.title("courbe de $\gamma$ pour $x_0$ = {}".format(x0))
    plt.show()
    return None
```

La portion de code suivant nous permet de visualiser la courbe de la fonction \gamma respectivement pour x_0 = 10^{-7},x_0 = 10^{-12},x_0 = 10^{-15}.

In [21]:
```
xinit=[10**(-7),10**(-12),10**(-15)]
for x in xinit:
    courbeGama(x,a=0,b=1)
```







Question 9: Influence des paramètres `rtol` et `atol`

Dans cette partie, nous voulons savoir l'éffet des paramètres `rtol` et `atol` qui désignent respectivement `l'erreur relative` et l' `erreur absolue`

On a pour f(0,x) = x^{3}+k_ax^2-k_ex-k_ak_e et f(1,x) = x^{3}+k_ax^2-(k_a+k_e)x-k_ak_e

et f(c,x) = (1-c)f(1,x)+cf(0,x)

Posons F(x) = f(1,x) et G(x)=f(0,x), comme F et G sont des fonctions continues, on peut définir l'Homotopie H telle que:

Pour tout (c,x)\in [0,1]\times\mathbb{R}, H(c,x)=(1-c)F(x)+cG(x)

Pour répondre à cette question, nous allons prendre une suite de valeurs de c allant de 0 à 1 avec un pas constant \delta (\delta=0.01 dans notre cas) et determiner la plus grande valeur de la norme \left\lVert f(c,\gamma (c) \right\lVert avec \gamma la fonction obtenu par la résolution de l'EDO de la question 8. La procédure `MaxNorm()` ci dessous nous permet d'obtenir cette valeur.

In [22]:
```
def F(x):
    return f(1,x)
def G(x):
    return f(0,x)
def H(c,x):
    return (1-c)*F(x)+c*G(x)
```

In [23]:
```python
def maxNorm(delta):
    c = np.arange(0,1,delta)
    a = []
    for s in gamma[0]:
        for i in c:
            N = np.linalg.norm(H(i,s))
            a.append(N)
    return max(a)
```

In [26]:
```python
maxNorm(delta=0.01)
```

Out[26]: 7.538140351016274e-08

__En particulier pour c = 1,

In [288]:
```python
b = []
for s in gamma[0]:
    b.append(H(1,s))

np.linalg.norm(b)
```

Out[288]: 3.82432220266069e-07

Interprétation des résultats: ces résultats nous montrent que dans le pire des cas on a $\left\lVert f(c,\gamma (c) \right\lVert\leqslant 3.824\times 10^{-7}$. Ce qui peut etre acceptable dans notre cas. Mais cette erreur peut justement etre améliorer en précisant la valeur de `rtol` qui contrôle le nombre de chiffres corrects tandis que `atol` qui mesure l'écart absolu entre la valeur exacte et la valeur approchée.

Question 10: Méthode de Newton pour une fonction $h:\mathbb{R}^{N}\longmapsto \mathbb{R}^{N}$

Considérons une fonction $h:(x_1,...,x_n)\in\mathbb{R}^{N}\longmapsto (f_1,...,f_n)$

Pour répondre à cette question, nous allons d'abord écrire une routine `MJacobienne()` qui prends deux paramètres:

- `vlist` de type `string` qui représente liste de variables $(x_1,...,x_n)$
- `flist` qui represente les $(f_1,...,f_n)$

Nous allons utiliser le module `sympy` de python qui permet de faire du calcul arithmétique formel basique, de l'algèbre et les mathématiques différentielles

In [27]:
```python
import sympy as sym
```

In [31]:
```python
def MJacobienne(vlist, flist):
    vars = sym.symbols(vlist)
    f = sym.sympify(flist)
    J = sym.zeros(len(f),len(vars))
    for i, fi in enumerate(f):
        for j, s in enumerate(vars):
            J[i,j] = sym.diff(fi, s)
    return J
```

Test de notre routine

In [54]:
```python
MJacobienne('c x',['x^3+k_a*x^2-(c*k_a+k_e)*x-k_a*k_e']).T
```

Out[54]: $\displaystyle \left[\begin{matrix}- k_{a} x\\- c k_{a} + 2 k_{a} x - k_{e} + 3 x^{2}\end{matrix}\right]$

In [55]:
```python
MJacobienne('u1 u2', ['2*u1 + 3*u2','2*u1 - 3*u2'])
```

Out[55]: $\displaystyle \left[\begin{matrix}2 & 3\\2 & -3\end{matrix}\right]$

On peut à présent utiliser notre routine pour implémenter la méthode de Newton

In [56]:
```python
def newton(h,x0, tol=10**(-8), max_iter=1000):
    for n in range(0,max_iter):
        hx0 = h(*x0)
        if abs(hx0) < tol:
            print("solution trouvée après {} itérations.".format(n))
            return x0
        Jhxn = MJacobienne(h,x0)
        if Jhxn == 0: # Traitement du cas ou dfdx n'est pas inversible
            x0 = input("Erreur! - dérivée nulle. Essayez avec une autre valeur de x0")
            try:
                newton(h,*x0)
            except ZeroDivisionError:
                print("Division par zero!")
            return None
        xn = xn - hxn/Jhxn
    print("Aucune solution trouvée après {} itérations".format(max_iter))
    return None
```

Question 11:

(a) Approximation à l'ordre 0:

On a $p(0,x)=G(x)= x^3+k_ax^2-k_ex-k_ak_e$

In [319]:
```python
def G(x):
    return x**3+k_a*x**2-k_e*x-k_a*k_e

x_tild = brentq(G, 0, 1)
x_tild
```

Out[319]: 1.0057832841862215e-07

(b) Approximation à l'ordre 1:

On a $c=1$ et $p(1,x)=F(x)= x^3+k_ax^2-(k_a+k_e)x-k_ak_e$

L'homotopie de Newton s'obtient en posant $G(x) = F(x)-F(\tilde{x})$

Ce qui donne $H(c,x) = (1-c)(F(x)-F(\tilde{x}))+cF(x)=F(x)-(1-c)F(\tilde{x})=0$

In [320]:
```python
def dHdx(x,x0=x_tild):
    return 3*x**2+2*k_a*x-k_a*k_e

def dHdc(x0=x_tild):
    return F(x0)
```

L'equation de la tangente à γ en un point c s'ecrit:

$y = \gamma(c)+(t-c)\gamma'(c)$ où t est la variable

Mais on a pour tout $c\in I$, $\gamma'(c) = \partial _c \gamma (c)=-(\frac{\partial f}{\partial x}(c,\gamma (c)))^{-1}.\frac{\partial f}{\partial c}(c,\gamma(c))$

Donc $y = \gamma(c)-(t-c)(\frac{\partial f}{\partial x}(c,\gamma (c)))^{-1}.\frac{\partial f}{\partial c}(c,\gamma(c))$

Pour $t = c+\delta$ ($\delta>0$) et $\gamma(c)=\tilde{x}$ on obtient:

$y = \tilde{x}-\delta.(\frac{\partial f}{\partial x}(c,\tilde{x}))^{-1}.\frac{\partial f}{\partial c}(c,\tilde{x})$

In [321]:
```python
def tangente(x=x_tild,delta=0.01):
    res = x - delta*(dHdc(x)/dHdx(x))
    return res
```

In [322]:
```python
tangente()
```

Out[322]: 0.00495854151923436

Conclusion

Ce travail est proposé en vue d'une appréciation par le corps enseignant. Il est donc possible qu'il y quelques erreurs qui seront bien évidemment rectifiées progressivement. Pour toute autre question contqctez-moi à l'adresse

Reférences:

1. Cours Introduction to Numerical Analysis and Statistical Modeling BAB2 Umons par Christophe Troestler
2. Python Programming and Numerical Methods - A Guide for Engineers and Scientists by Qingkai Kong, Timmy Siauw, Alexandre Bayen (Authors)
3. Earl A. Coddington, Norman Levinson - Theory of ordinary differential equations-R.E. Krieger (1984)
4. Wikipedia Numerical differentiation et Numerical methods for ordinary differential equations