Etudes numériques

Suites et sommations

Pour $k > 0$ donné, on considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par récurrence par : $u_{n+1} = \dfrac{u_n + k}{k{u_n}^2 + 1}$ avec $u_0 = 0$

Compléter le code pour qu'il permette de calculer les valeurs u_n lorsque n est donné en entrée :

```
def suitu(k):
        u=0
        for ..........................
        ...
        return(......)
```

Donner les valeurs de u_1 et u_2 :

```
u1 = ...
u2 = ...
```

Mais obtenir la liste des termes u_{10} à u_{15} est plus intéressant :

```
...
...
...
```

Et ces valeurs sont (fournir une liste de valeurs approchées à 10^{-3} près :

```
...
```

Et enfin, un traditionnel sigma défini comme $S_n = \sum_{i=0}^{n} u_i$ pour lequel on demande une programmation des valeurs :

```
def SumS(k,n):
        S=0
        ..............................
        ..............................
        ..............................
```

Puis vous remplirez ce tableau de valeurs de tests :

valeur $S_n(k)$:	k= 1	k=2.5	k=5	k = 10	k=0.1	k=0.25	k=0.4
n=10							
n = 25							
n=100							

Une fonction, plus en large qu'en travers

On considère les fonctions f et g définies par :

$$\forall x \in \mathbb{R} \quad f(x) = xe^{-\frac{x^2}{2}} \quad \text{et} \quad g(x) = x^3 + x^2 + x + 1$$

Sans oublier les packages :

```
....................................
....................................
import matplotlib.pyplot as plt
```

On peut maintenant définir la fonction f :

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

puis la fonction g

```
...
...
...
```

Et on commence le travail d'analyse par la création de tableaux de valeurs :

```
x=np.linspace(-5,5,101)
yf=.............................
yg=.............................
...
...
```

Au fait, nous venons d'échantillonner les fonctions f et g sur quel intervalle I ?

```
I = .............................
```

Et pour visualiser une courbe, vous feriez comment ?

```
Graphf=.........................
Graphg=.........................
plt.show()
```

Niveau équation, comme g est polynomiale ça a l'air plus facile. Et comme g est de degré 3, on peut assurer l'existence d'une solution α à $g(x) = 0$. Localiser une racine α de g sur un intervalle d'amplitude au plus 5 (comprendre : $b - a \leq 5$ et $\exists x \in [a;b] \quad g(x) = 0$:

```
ag=...................
bg=...................
```

Et au moyen de la méthode de dichotomie, produire une solution à 7 chiffres après la virgule près :

```
#méthode de dichotomie pour g(x)=0
while ......................
        c=(ag+bg)/2
        if g(ag)*g(c)<0 :
                bg=...........
        else
                ag=...........
print("solution approchée :",...............)
```

Et merci de produire votre proposition :

```
alpha = .............................
```

On passe à la résolution d'équation avec $f(x) = 0.1$. Attention ! Il y a deux solutions réelles : merci de rechercher la plus grande des deux et cette fois, on cherche des bornes avec une amplitude de 1 :

```
af=...................
bf=...................
```

Et toujours au moyen de la méthode de dichotomie, produire une solution à 7 chiffres après la virgule près :

```
#méthode de dichotomie pour f(x)=0.1
while ...................
        c=...................
        if ................<0 :
                bf=...........
        else
                af=............
print("solution approchée :",..............)
```

Et merci de produire votre proposition :

```
solution pour f = ........................
```

Remarque : La méthode de dichotomie est un attendu explicite du programme et a constitué, à plusieurs reprises, des questions dans les sujets de concours ECT anciens et nouveaux.

Exprimez-vous :

utilisez cet espace vierge pour noter ce qui pourrait vous être utile

Systèmes Sans Peine grâce aux matrices

On utiliser le module de calcul matriciel de Python pour résoudre des systèmes en un éclair. Mais les solutions seront approchées (étonnant !) Les packages nécessaires seront :

```
from math import*
import numpy as np
```

Voici un système à résoudre (ne pas le résoudre à la main !)

$$(S):\begin{cases} 4x & -5y & +6z & +8t & = & 0 \\ -x & +6y & +2z & -5t & = & 1 \\ 3x & -2y & -5z & +10t & = & 2 \\ 2x & +7y & +z & -2t & = & -3 \end{cases}$$

il ne fait pas envie du tout par contre écrire sa représentation matricielle $AX = B$ est accessible aisément (écrire les matrices) :

```
...
...
...
...
```

Et il vous faut déclarer en Python la matrice A et la colonne B :

```
A = np.array(.....................................................)
B = ....................................
```

On se souvient que la solution est $X = A^{-1}B$ et on va admettre que A est inversible. Un peu hors programme mais commode, testons la commande `np.linalg.inv(A)` qui inverse directement la matrice A :

```
Ainv=.......................
```

Et pour multiplier deux matrices compatibles M et N on utilise la commande au programme `np.dot(M,N)`. On espère que vous avez retenu quelle multiplication faire pour obtenir vos solutions :

```
X=.......................
```

et vous donnez les-dites solutions :

```
x=..................
y=..................
z=..................
t=..................
```

Et là vous vous dites : *Les matrices c'est magique !*
Alors une fois, peut-être, dans l'avenir, on parlera de ce qu'on appelle, pour de vrai, les matrices *magiques* mais là, on risque de dévoiler les trucs cachés derrière les tours de mentalistes.