TD_sup

August 26, 2026

1 TD1_sup - code capytal : 02d1-7280221

1.1 Exercice 1 : Exponentiation naïve

Calculer x^n au moyen d'une fonction itérative (boucle for ou while) prenant pour argument x et n

[12]:
```
def expo(x,n):
    resultat=1
    for i in range(1,n+1):
        resultat=resultat*x
    return resultat
expo(10,3)
```

[12]:
```
1000
```

1.2 Exercice 2 : analyse vectorielle

On va travailler en base cartésienne, ainsi deux points A et B de coordonnées respectives $\binom{x_A}{y_A}$ et $\binom{x_B}{y_B}$ seront représentés informatiquement par deux listes $[x_A, y_A]$ et $[x_B, y_B]$.

On va traiter les vecteurs à l'aide de leurs coordonnées en base cartésienne. Par exemple, le vecteur $\vec{AB}$ à deux dimensions est défini par $\binom{x_B - x_A}{y_B - y_A}$ et sera traité informatiquement sous la forme d'une liste : $[x_B - x_A, y_B - y_A]$

1) Ecrire une fonction produit, qui prend en argument un vecteur $\vec{AB}$ et un coefficient k, et qui renvoie le vecteur $k\vec{AB} = \binom{k(x_B - x_A)}{k(y_B - y_A)}$.

[24]:
```
def produit(vec,k):
    return [k*vec[i] for i in range(len(vec))]
produit([2,3],4)
```

[24]:
```
[8, 12]
```

2) Ecrire une fonction somme, prenant pour argument deux vecteurs vec1 et vec2, et qui renvoie la somme de ces deux vecteurs.

[25]:
```
def somme(vec1,vec2):
    new_vec=[0,0]
    for i in range(len(vec1)):
        new_vec[i]=vec1[i]+vec2[i]
```

```
    return new_vec
somme([3,3],[4,5])
```

[25]: [7, 8]

[26]:
```
def somme2(vec1,vec2):
    return [vec1[i]+vec2[i] for i in range(len(vec1))]
somme2([3,3],[4,5])
```

[26]: [7, 8]

3) Ecrire une fonction scalaire, prenant pour argument deux vecteurs vec1 et vec2, et qui renvoie le produit scalaire entre ces deux vecteurs :

$$\vec{AB}.\vec{A'B'} = AB_x * A'B'_x + AB_y * A'B'_y$$

[27]:
```
def scalaire(vec1,vec2):
    s=0
    for i in range(len(vec1)):
        s=s+vec1[i]*vec2[i]
    return s
scalaire([1,2],[2,4])
```

[27]: 10

5) Ecrire une fonction opposé qui prend en argument un vecteur et qui calcule son opposé.

[28]:
```
def oppose(vec):
    return [-vec[i] for i in range(len(vec))]
oppose([2,3])
```

[28]: [-2, -3]

6) On considère des points A, B, C associés à des masses m_A, m_B, m_C. On définit le barycentre G de ces points par

$$\vec{OG} = \frac{m_A\vec{OA} + m_B\vec{OB} + m_C\vec{OC}}{m_A + m_B + m_C}$$

En utilisant une partie des fonctions précédentes, écrire une fonction qui prend en arguments les points A, B, C et les masses m_A, m_B, m_C associée et qui calcule le vecteur $\vec{OG}$.

[29]:
```
def barycentre(A,B,C,mA,mB,mC):
    m_tot=mA+mB+mC
    vec1= produit(A,mA/m_tot)
    vec2= produit(B,mB/m_tot)
    vec3= produit(C,mC/m_tot)
    vec4=somme(vec1,vec2)
    return somme(vec4,vec3)
```

7) Quel est la barycentre des trois points A, B et C de coordonnées respectives $\binom{x_A=0}{y_A=0}, \binom{x_B=2}{y_B=1}$ et $\binom{x_C=2}{y_C=2}$ de masse respective $m_A = 1kg, m_B = 2kg, m_C = 3kg$

[30]:
```
barycentre([0,0],[2,1],[2,2],1,2,3)
```

[30]:
```
[1.6666666666666665, 1.3333333333333333]
```

1.3 Exercice 3 : Traitement de données numériques

On dispose de notes sous forme d'une liste. Par exemple [10,5,14,5,16,20,18,13].

1) Ecrire une fonction moyenne, qui prend en argument une liste, et qui renvoie la moyenne L_{moy}. Tester votre fonction avec la liste ci-dessus.

[31]:
```
def moyenne(L):
    somme=0
    for i in L:
        somme=somme+i
    return somme/len(L)

moyenne([10,5,14,5,16,20,18,13])
```

[31]:
```
12.625
```

2) Ecrire une fonction variance, qui prend en argument une liste (d'au moins 2 termes), et qui renvoie la variance de cette liste :

$$V = \frac{\sum_1^n (L_i - L_{moy})^2}{n-1}$$

[32]:
```
def variance(L):
    moy=moyenne(L)
    somme=0
    for i in L:
        somme=somme+(i-moy)**2
    return somme/(len(L)-1)
```

[33]:
```
variance([0,1,2,3])
```

[33]:
```
1.6666666666666667
```

3) Ecrire une fonction écart-type qui calcule l'écart-type $\sigma = \sqrt{V}$ en prenant une liste L en argument

[34]:
```
def ecart_type(L) :
    return variance(L)**0.5
ecart_type([0,1,2,3,4])
```

[34]:
```
1.5811388300841898
```

4) Pour apprécier la variabilité de la moyenne, on calcule alors l'écart type par :

$$\sigma(L_{moy}) = \frac{1}{\sqrt{n}} \sqrt{\frac{\sum_1^n (L_i - L_{moy})^2}{n-1}}$$

Ecrire un programme qui permet de calculer cette quantité.

[35]:
```
def ecart_type2(L) :
    return (variance(L)**0.5)/(len(L)**0.5)
ecart_type2([0,1,2,3,4])
```

[35]:
```
0.7071067811865476
```

1.4 Exercice 4: Estimation de π

Leibnitz a démontré la relation suivante :

$$\frac{\pi}{4} = \sum_{i=0}^{\infty} \frac{(-1)^n}{2n+1}$$

Ecrire une fonction qui renvoie une liste L dont chaque élément d'indice i donne une estimation de π pour $n = i$

[36]:
```
def pi(n):
    L=[]
    val=0
    for i in range(0,n):
        val=val+(-1)**i/(2*i+1)*4
        L.append(val)
    return L
pi(100)[-1]
```

[36]:
```
3.1315929035585537
```

1.5 Exercice 5 : Transposée d'une matrice

La transposée d'une matrice carrée d'ordre 3 s'obtient en échangeant ses lignes et ses colonnes :

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \quad \Longrightarrow \quad A^{\mathsf{T}} = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}.$$

Dans cet exercice une matrice sera représentée par la liste de listes suivante :

[37]:
```
M1=[[1,2,3],
    [4,5,6],
    [7,8,9]]
```

1) Ecrire une fonction prenant pour argument une matrice carré $n*n$ et qui renvoie sa transposée

[38]:
```
def transposee(M):
    n=len(M)
    for i in range(n-1):
        for j in range(i+1,n):
            M[i][j],M[j][i]=M[j][i],M[i][j]
```

```
    return M
M1=[[1,2,3],
    [4,5,6],
    [7,8,9]]
transposee(M1)
```

[38]:
```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

[39]:
```
def transpose2(M):
    l,c=len(M),len(M[0])
    M_new=[[0 for i in range(l)] for j in range(c)]
    for i in range(l):
        for j in range(c):
            M_new[j][i]=M[i][j]
    return M_new
M1=[[1,2,3],
    [4,5,6],
    [7,8,9]]
transpose2(M1)
```

[39]:
```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

Considérons une matrice A de dimension 2×3 :

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}.$$

Sa transposée, notée A^{T}, est une matrice de dimension 3×2 :

$$A^{\mathsf{T}} = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{pmatrix}.$$

2) Ecrire une fonction réalisant la transposée d'une matrice $l * c$.

[40]:
```
def transpose3(M):
    l,c=len(M),len(M[0])
    M_new=[[0 for j in range(l)] for i in range(c)]
    for i in range(c):
        for j in range(l):
            M_new[i][j]=M[j][i]
    return M_new
M1=[[1,2],
    [4,5],
    [7,8]]
print(transpose3(M1))
M2=[[1, 4, 7], [2, 5, 8]]
print(transpose3(M2))
```

```
[[1, 4, 7], [2, 5, 8]]
[[1, 2], [4, 5], [7, 8]]
```

1.6 Exercice 6 : Produit matriciel

On donne le principe du produit matriciel:

Le produit de deux matrices carrées d'ordre 3 s'écrit :

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}.$$

Chaque coefficient c_{ij} est obtenu en multipliant les coefficients de la ligne i de la première matrice par ceux de la colonne j de la seconde matrice, puis en additionnant les produits obtenus :

$$c_{ij} = \sum_{k=1}^{3} a_{ik} b_{kj}.$$

Par exemple, le coefficient c_{11} est obtenu à partir de la première ligne de la première matrice et de la première colonne de la seconde :

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}.$$

Chaque élément c_{ij} du produit matriciel vérifie :

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Dans cet exercice, matrice est alors représentée par une liste de liste. Par exemple :

[41]:
```
M1=[[1,2,3],
    [4,5,6],
    [7,8,9]]
```

Ecrire un porgramme permettant d'effectuer un produit matriciel tel qu'il est présenté ci-dessus :

[42]:
```
def produit(M1,M2):
    l1,c1=len(M1),len(M1[0])
    l2,c2=len(M2),len(M2[0])
    if c1!=l2:
        raise ValueError("mauvaise dimension")
    M_new=[[0 for i in range(c2)] for j in range(l1)]
    for i in range(l1) :
        for j in range(c2):
            val=0
            for k in range(l2):
                val=val+M1[i][k]*M2[k][j]
```

```
            M_new[i][j]=val
    return M_new
M1=[[1,2,3],
    [4,5,6],
    [7,8,9]]
M2=[[1],
    [2],
    [3]]
print(produit(M1,M1))
print(produit(M1,M2))
```

```
[[30, 36, 42], [66, 81, 96], [102, 126, 150]]
[[14], [32], [50]]
```

[43]:
```
def produit3(M1,M2):
    return [[sum(a*b for a,b in zip(ligne,colonne)) for colonne in zip(*M2)] for ligne in M1]  #zip(*M2) permet d'accéder aux colonnes
A = [[1, 2],
     [3, 4]]
I = [[1, 0],
     [0, 1]]

print(produit3(I, A))
print(produit3(A, I))
```

```
[[1, 2], [3, 4]]
[[1, 2], [3, 4]]
```

1.7 Exercice 7: DL de exp(x)

On donne le développement limité de exp(x) :

$$exp(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!}$$

1) Ecrire une fonction factorielle de n qui prend en argument n et qui calcule la factorielle de n.
2) Ecrire une fonction qui prend en argument x et n et qui renvoie une liste dont chaque valeur d'indice i correspond à l'estimation de exp(x) à l'ordre i.

[44]:
```
def factorielle(n):
    if n==0 or n==1:
        return 1
    else :
        val=1
        for i in range(2,n+1):
            val=val*i
    return val
factorielle(5)
```

[44]: 120

[45]:
```
def expo(x,n):
    if n==0:
        return [1]
    else:
        val=1
        L=[val]
        for i in range(1,n+1):
            val=val+x**i/(factorielle(i))
            L.append(val)
    return L
expo(2,10)
```

[45]:
```
[1,
 3.0,
 5.0,
 6.333333333333333,
 7.0,
 7.266666666666667,
 7.355555555555555,
 7.3809523809523805,
 7.387301587301587,
 7.3887125220458545,
 7.388994708994708]
```

[46]:
```
def factorielle(n):
    if n==0 or n==1:
        return 1
    else:
        r=1
        while n>1:
            r=r*n
            n=n-1
    return r
def f(x,n):
    L=[1]
    for i in range(1,n+1):
        L.append(L[-1]+x**i/factorielle(i))
    return L
print(f(2,5))
import numpy as np
print(np.exp(2))
```

```
[1, 3.0, 5.0, 6.333333333333333, 7.0, 7.266666666666667]
7.38905609893065
```

[ ]: