MP - MP* / 2015 - 2016

Mis à jour le 16 mars 2016 à 13:40

Informatique: Niveau II

Guillaume CONNAN - Lycée Clemenceau

UNIVERSITÉ DE NANTES

IUT NANTES Université de Nantes

TABLE DES MATIÈRES

THÈME

1

Structures de données linéaires

En 1920, le mathématicien-logicien-philosophe polonais Jan Łukasiewicz (1878-1956) invente la notation préfixée alors qu'il est ministre de l'éducation (on a le droit de rêver...).
35 ans plus tard, le philosophe et informaticien (!) australien (!!!) Charles Hamblin (1922 - 1985) s'en inspire.
Il a en effet en sa possession l'un des deux ordinateurs présents à l'époque en Australie. Il se rend compte que la saisie de calculs avec les opérateurs en notation infixée induit de nombreuses erreurs de saisie par oubli de parenthèses. Il pense aussi à la (très petite) mémoire des ordinateurs de l'époque.
Il trouve alors son inspiration dans le travail de Łukasiewicz pour éviter les parenthèses et il pense à mettre les opérateurs en position préfixée : ainsi il introduit la notion de *pile* (ou *stack* ou Lifo) qui économise le nombre d'adresses mémoire nécessaires.
C'est la naissance de la Notation Polonaise Inversée (NPI ou RPN). Elle économise également la taille des composants électroniques des portes logiques.
Son seul inconvénient : les mauvaises habitudes prises d'utiliser des notations infixées...
Un autre grand avantage : il faut comprendre le calcul avant de l'exécuter :-)

Recherche 1 - 1 Centrale 2015

1. Donnez la valeur des expressions suivantes :
   i. `[1, 2, 3] + [4, 5, 6]`
   ii. `2 * [1, 2, 3]`
2. Écrivez une fonction Python `smul` à deux paramètres, un nombre et une liste de nombres, qui multiplie chaque élément de la liste par le nombre et renvoie une nouvelle liste : **`smul(2, [1, 2, 3])`** $\longrightarrow$ `[2, 4, 6]`.
3. Arithmétique des listes
   i. Écrivez une fonction Python `vsom` qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la somme terme à terme de ces deux listes :
   `vsom([1, 2, 3], [4, 5, 6])` $\longrightarrow$ `[5, 7, 9]`.
   ii. Écrivez une fonction Python `vdif` qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la différence terme à terme de ces deux listes :
   `vsom([1, 2, 3], [4, 5, 6])` $\longrightarrow$ `[-3, -3, -3]`.
4. Question subsidiaire
   Créez une fonction d'ordre supérieure qui prendra l'operation comme paramètre.
   Ainsi, `vsom` sera **`vop(addition)`**.

Recherche 1 - 2 Adresse

Commentez ! Quelles précautions prendre ?

```
In [22]: t = [0,1,2,3]

In [23]: id(t)
Out[23]: 140162835709256

In [24]: tt = t

In [25]: id(tt)
Out[25]: 140162835709256

In [26]: t += [4]

In [27]: t
Out[27]: [0, 1, 2, 3, 4]

In [28]: id(t)
Out[28]: 140162835709256

In [29]: tt
Out[29]: [0, 1, 2, 3, 4]

In [30]: t = t + [5]

In [31]: t
Out[31]: [0, 1, 2, 3, 4, 5]

In [32]: tt
Out[32]: [0, 1, 2, 3, 4]

In [33]: id(t)
Out[33]: 140162734366856

In [34]: id(tt)
Out[34]: 140162835709256

In [35]: import copy as cp

In [39]: ttt = cp.copy(t)

In [40]: id (ttt)
Out[40]: 140162835291592

In [41]: t += [6]

In [42]: t
Out[42]: [0, 1, 2, 3, 4, 5, 6]

In [43]: tt
Out[43]: [0, 1, 2, 3, 4]

In [44]: ttt
Out[44]: [0, 1, 2, 3, 4, 5]
```

Recherche 1 - 3 Performances

Commentez :

```
import numpy as np

def v1():
    l = [0,1,2]
    l += [3]
def v2():
    l = [0,1,2,None]
    l[3] = 3
def v3():
    l = [0,1,2]
    l.append(3)
def v4():
    l1 = np.array([0,1,2])
    l2 = np.array([3])
    np.concatenate((l1,l2))
def v5():
    l = np.array([0,1,2,None])
    l[3] = 2
def v6():
    l = np.arange(4)
    l[3] = 12
def v7():
    l = it.chain([0,1,2],[3])
```

```
In [45]: %timeit v1()
1000000 loops, best of 3: 320 ns per loop

In [46]: %timeit v2()
10000000 loops, best of 3: 179 ns per loop

In [47]: %timeit v3()
1000000 loops, best of 3: 246 ns per loop

In [48]: %timeit v4()
100000 loops, best of 3: 5.19 µs per loop

In [49]: %timeit v5()
1000000 loops, best of 3: 1.91 µs per loop

In [50]: %timeit v6()
1000000 loops, best of 3: 788 ns per loop

In [68]: %timeit v7()
1000000 loops, best of 3: 404 ns per loop
```

Recherche 1 - 4 Flavius Josèphe le fourbe

Flavius Josèphe (37 - 100) ou plutôt Yossef ben Matityahou HaCohen est un historien romain d'origine juive et de langue grecque et qui ne devait pas être trop mauvais en mathématiques si on en croit la sinistre anecdote suivante. Lors de la première guerre judéo-romaine, Yossef fut piégé dans une grotte avec 39 autres de ses compagnons en juillet 67. Ne voulant pas devenir esclaves, ils mirent au point un algorithme d'auto-destruction : il s'agissait de se mettre en cercle et de se numéroter de 1 à 40.

Chaque septième devait être tué jusqu'à ce qu'il n'en reste plus qu'un qui devait alors se suicider. Ce dernier fut Yossef lui-même...et il ne se suicida pas ! Après deux ans de prison, il fut libéré, il entra au service des romains comme interprète et acquit la citoyenneté romaine deux ans plus tard.
On voudrait savoir quel numéro portait Yossef. Vous allez créer un programme Python qui donne l'ordre des exécutions quelque soit le nombre de prisonniers et le nombre fatal
Vous pourrez utiliser dans un premier temps les listes de Python et la fonction **pop**.
Ensuite, vous pourrez utiliser la structure générale de liste chaînée vue en cours.
Dans une autre situation (pour ne pas vous donner la réponse quand même...), on doit obtenir avec 20 compagnons et 6 comme nombre sinistre suite :

```
La 1e victime est le no 6
La 2e victime est le no 12
La 3e victime est le no 18
La 4e victime est le no 4
La 5e victime est le no 11
La 6e victime est le no 19
La 7e victime est le no 7
La 8e victime est le no 15
La 9e victime est le no 3
La 10e victime est le no 14
La 11e victime est le no 5
La 12e victime est le no 17
La 13e victime est le no 10
La 14e victime est le no 8
La 15e victime est le no 2
La 16e victime est le no 9
La 17e victime est le no 16
La 18e victime est le no 13
La 19e victime est le no 1
Out[62]: 'Et le survivant est le no 20'
```

Peut-on prévoir qui sera le survivant sans dresser de liste avec une belle fonction récursive ?...

Recherche 1 - 5 Piles

Dans ce TD nous considèrerons disposer d'un type pile et des opérations suivantes :

Opération	Effet
`pile()`	Création d'une nouvelle pile (vide)
`p.pop()`	Retire l'élément au sommet de la pile `p` et renvoie sa valeur. Une erreur survient si la pile est vide.
`p.push(x)`	Ajoute l'élément `x` au sommet de la pile `p`.
`p.is_empty()`	Renvoie `True` si la pile est vide et `False` sinon.

Nous ne disposons d'aucune autre opération sur les piles.
Remarques liminaires.

— Vous indiquerez pour chaque fonction la complexité en temps et en espace.

— Certaines fonctions supposent que la pile est non-vide. On ne vérifiera pas que la pile est bien non-vide et on laissera l'erreur survenir si la pile est en fait vide.

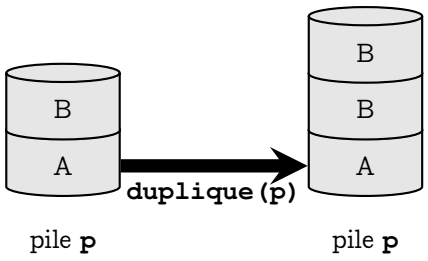
— Vous veillerez à bien faire la différence entre modifier une pile existante et créer une nouvelle pile.

— Vous n'utiliserez aucune autre structure de données que les piles. En particulier, vous n'utiliserez pas de listes.

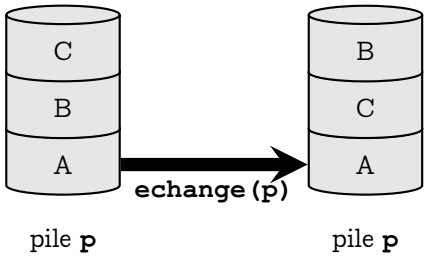
Implémenter les fonctions suivantes sur les piles :

1. `nonepop(p)` qui a le même effet que `p.pop()`, sauf si la pile est vide : dans ce cas, la pile reste inchangée et `nonepop(p)` renvoie `None`.

2. `duplique(p)` qui duplique l'élément au sommet de la pile.

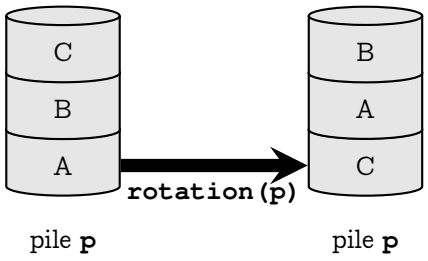


3. `echange(p)` qui échange les deux premiers éléments de la pile `p`.

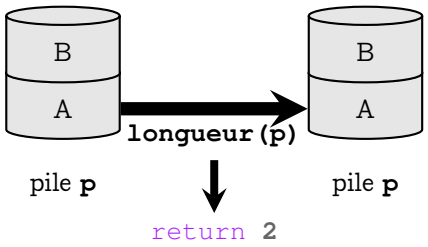


4. `ecrase(p)` qui efface tous les éléments de `p`.

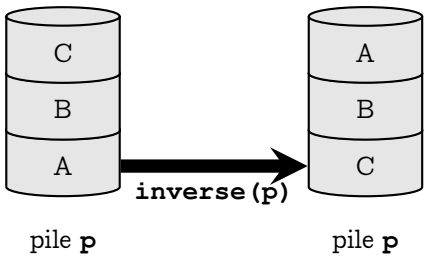
5. `rotation(p)` qui met au fond de la pile `p` l'élément qui est actuellement au dessus de la pile.



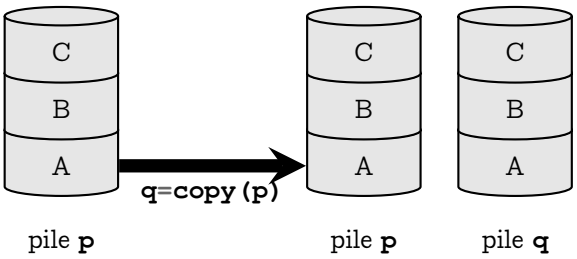
6. `longueur (p)` qui calcule la taille de la pile p.



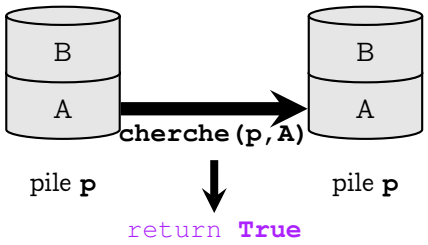
7. `inverse (p)` qui inverse l'ordre des éléments de p.



8. `copy (p)` qui renvoie une nouvelle pile, qui est une copie de p.



9. `cherche(p, x)` qui renvoie `True` si x est dans la pile p et `False` sinon.



Recherche 1 - 6 D'après une épreuve posée lors d'un entretien Google

Dans cet exercice on considère des chaînes de caractères avec 3 types de parenthèses () [] {} (par exemple s="{([[]])}").

1. Parmi les chaînes de caractères suivantes, lesquelles sont bien parenthésées ?

i. `()[]{()}`

ii. `([()])`

iii. `([][]{()}`

iv. `([][]{()})`

2. Écrire une fonction qui prend en argument une telle chaîne de caractères et vérifie qu'elle est bien parenthésée, c'est-à-dire qui renvoie `True` si elle est bien parenthésée et `False` sinon.

3. Indiquer votre complexité en temps et en espace.

4. Comment simplifier l'algorithme dans le cas où il n'y aurait qu'un seul type de parenthèses ? Quelle est alors la complexité en temps et en espace ?

5. Dans l'épreuve de Google, on suppose disposer de N types de parenthèses différentes. Comment traiter ce cas ?

Recherche 1 - 7 Notation polonaise inverse

La notation polonaise inverse ou notation postfixée consiste à placer les opérateurs après les opérandes, en cela il diffère de la notion infixe.

Exemples.

Notation infixe	Notation postfixe
$1+2$	$1\ 2\ +$
$10 \div 0.5$	$10\ 0.5\ \div$
2×3	$2\ 3\ \times$
$(1+2) \times 3$	$1\ 2\ +\ 3\ \times$
$1+(2 \times 3)$	$1\ 2\ 3\ \times\ +$
$(-2)-3$	$2\ -_1\ 3\ -_2$
$-(2-3)$	$2\ 3\ -_2\ -_1$

Remarques sur la notation postfixe.

— Pas besoin de parenthèses

— Il faut distinguer le $-$ unaire (noté ici $-_1$) du moins binaire (noté ici $-_2$). Le premier transforme un nombre en son opposé (exemple : -2) le second calcule la différence entre deux termes (exemple : $1-3$).

1. Convertir les expressions suivantes en notation polonaise inverse.

i. $1+2+3+4+5+6$

ii. $7+(1+3) \times (2+3)$

iii. $2 \times 3 \times 5 \div 7$

iv. $3+9 \times (5 \times (7+2)+2)$

2. Convertir en notation usuelle les expressions postfixes suivantes.

i. $2\ 3\ -_1\ -_2$

ii. $1\ 2\ 3\ +\ \times$

iii. $1\ 2\ 3\ +\ -_1\ \times$

iv. $7\ 5\ 3\ 2\ 1\ +\ \times\ \div\ +$

Recherche 1 - 8 HP15C

Créez une calculatrice qui effectue les calculs arithmétiques à la manière du *calculator* d'Emacs.
On pourra se contenter d'une version simplifiée : on donne comme argument la chaîne correspondant à l'opération à effectuer sous forme postfixée. La fonction renvoie le résultat de l'opération. Par exemple :

```
In [1]: seq = "2 3 4 + *"

In [2]: npi(seq)
Out[2]: 14
```

On pourra utiliser les fonctions du module `operator`.
Par exemple, `operator.add(2,3)` renvoie 5.
On peut alors créer un dictionnaire des opérations :

```
operations = {
    '+':  operator.add, '-':  operator.sub,
    '*':  operator.mul, '/':  operator.truediv,
    '%':  operator.mod, '**': operator.pow,
    '//': operator.floordiv
}
```

et on obtient l'opération d'addition à partir de la chaîne `'+'` :

```
In [3]: operations['+']
Out[3]: <function _operator.add>
```

Par exemple :

```
In [1]: npi("3 2 7 9 - * 3 + 2 / +")
Out[1]: 2.5

In [2]: npi("3 2 7 9 - * 3 + 2 /")
  File "<string>", line unknown
SyntaxError: Expression non valide
```

Vous pouvez ensuite proposer une calculatrice plus « interactive » :

```
In [1]: hp_inter()
->  |
rentrer un nombre ou un opérateur: 1
-> 1 |
rentrer un nombre ou un opérateur: 2
-> 2, 1 |
rentrer un nombre ou un opérateur: 3
-> 3, 2, 1 |
rentrer un nombre ou un opérateur: 4
-> 4, 3, 2, 1 |
rentrer un nombre ou un opérateur: '+'
-> 7, 2, 1 |
rentrer un nombre ou un opérateur: '-'
-> -5, 1 |
rentrer un nombre ou un opérateur: '+'
-> -4 |
rentrer un nombre ou un opérateur: 3
-> 3, -4 |
rentrer un nombre ou un opérateur: '*'
-> -12 |
rentrer un nombre ou un opérateur: 'fin'
```

Recherche 1 - 9 Casio/TI

Créez une calculatrice qui lit une expression arithmétique en notation infixe et effectue le calcul en utilisant une (ou des) pile(s). On utilisera les mêmes opérateurs **binaires** (i.e. d'arité 2) que précédemment (attention à séparer les opérandes, parenthèses et opératuers par des espaces). Par exemple :

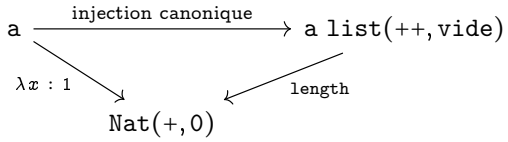
```
In [1]: casio("3 + ( ( ( 2 * ( 7 - 9 ) ) + 3 ) / 2 )")
Out[1]: 2.5

In [2]: casio("3 + ( (  2 * ( 7 - 9 )  + 3 ) / 2 )")
  File "<string>", line unknown
SyntaxError: expression non valide
```

Recherche 1 - 10 Morphisme de monoïde (ENS)

On a remarqué dans l'exercice précédent que elem, sum, length, map, foldl, filter, etc. fonctionnaient toutes selon le même principe et que l'on avait tendance à se répéter un peu.
La lumière va venir de l'algèbre et plus particulièrement des homomorphismes de monoïdes (derrière se cache en fait la notion de *catégorie*...).
Nous remarquons en effet que toutes ces fonctions sont des morphismes de monoïde.
Soit a notre type de base. Notons a list le type des listes d'éléments de type a. Si on munit a list de la concaténation des chaînes (que nous noterons ++), on obtient un monoïde dont l'élément neutre est la liste vide.
Considérons par exemple la fonction length. Elle est à valeurs dans Nat, le type des entiers non signés, qui, muni de l'addition, est aussi un monoïde.

$$\texttt{length(lst1 ++ lst2)} = \texttt{length(lst1)} + \texttt{length(lst2)} \qquad \text{et} \qquad \texttt{length(Vide)} = 0$$

Pour chaque élément, on peut considérer la fonction $\lambda x\ :\ 1$ dont length est le « gonflement » (une liste ne contenant qu'un élément est de longueur 1).
On a donc le diagramme suivant :



L'injection canonique transforme un élément quelconque en la liste constituée de cet unique élément.
Cela nous donne alors un moyen d'imaginer une construction d'une fonction générale qui construira le morphisme en fonction de la fonction de gauche et du morphisme du bas.

```
def gonfle(f,loiMonoide,neutreMonoide):
    """
    crée le morphisme de monoïde entre [a] et b si b est un monoïde
    i.e.  gonfle la fonction f : a -> b en le morphisme f* : [a] -> b
    loiMonoide : (b*b) -> b
    neutreMonoide : le neutre de b
    f traite les éléments, f* traite les listes de ces éléments
    f*([a])          = f(a)
    f*([as] ++ [as]) = f*([as]) loiMonoide f*([as])
    f*([])           = neutreMonoide
    ex:
    longueur =  gonfle(lambda x: 1, operator.add, 0)
    """
    ?????
```

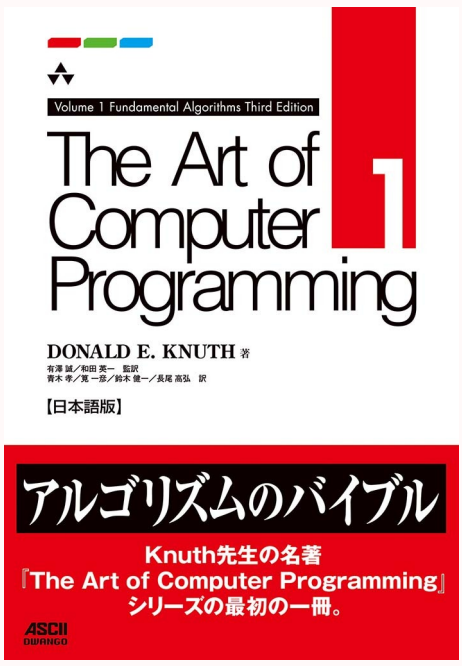
Construisez cette fonction.
Tracez les diagrammes de elem, sum, map, foldl, filter puis définissez-les sur Python à l'aide de la fonction gonfle.
Démontrez par récurrence que toute fonction créée avec gonfle renvoie le morphisme de monoïde défini par le diagramme général : *une preuve pour les démontrer toutes...*

Merci à Jean-Loup Carré *pour les exercices 5 et 6.*

THÈME

2

Tris - Complexité



Dans toute la suite, on se ramènera au cas du tri d'une permutation quelconque de $\mathfrak{S}_n$. On étudiera les danses hongroises disponibles en ligne, par exemple : https://www.youtube.com/watch?v=kDgvnbUIqT4

Tri sélectif...

On parcourt la liste, on cherche le plus grand et on l'échange avec l'élément le plus à droite et on recommence avec la liste privée du plus grand élément.

Version impérative

On commence par chercher l'indice du maximum d'une liste. On part de 0 et on compare à chaque élément de la liste en faisant évoluer l'indice du maximum si nécessaire.

```
def ind_maxi(xs):
    ind_tmp = 0
    n       = len(xs)
    for i in range(n):
        if xs[i] > xs[ind_tmp]:
            ind_tmp = i
    return ind_tmp
```

Ensuite, on copie la liste donnée en argument pour ne pas l'écraser. On parcourt la liste et on procède aux échanges éventuels entre le maximum et l'élément de droite.

```
def tri_select(xs):
    cs = xs.copy()
    n  = len(cs)
    for i in range(n - 1,0,-1):
        i_m = ind_maxi(cs[:i + 1])
        if i_m != i:
            cs[i_m],cs[i] = cs[i],cs[i_m]
        # print(cs) : pour suivre l'évolution
    return cs
```

On va utiliser `permutation` de la bibliothèque `numpy.random` qui renvoie une permutation uniformément choisie par les permutations de $\mathfrak{S}_n$.

```
In [5]: ls = list(permutation(range(10)))

In [6]: ls
Out[6]: [8, 0, 4, 3, 5, 2, 7, 1, 9, 6]

In [7]: tri_select(ls)
[8, 0, 4, 3, 5, 2, 7, 1, 6, 9]
[6, 0, 4, 3, 5, 2, 7, 1, 8, 9]
[6, 0, 4, 3, 5, 2, 1, 7, 8, 9]
[1, 0, 4, 3, 5, 2, 6, 7, 8, 9]
[1, 0, 4, 3, 2, 5, 6, 7, 8, 9]
[1, 0, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 0, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 0, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Out[7]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Correction

Il s'agit de démontrer la correction des deux fonctions.
Pour la première, l'invariant de boucle est : « ind_temp est l'indice du maximum des i + 1 premiers termes de la liste en argument » et se démontre par récurrence.
Pour la deuxième fonction, l'invariant de boucle est « la liste des élements d'indices entre i et n − 1 est triée ».

Complexité

Nous mesurerons la complexité en nombre de comparaisons.
La complexité de `ind_maxi(xs)` est en $\Theta(n)$ avec n la longueur de `xs`.
En effet, il y a une comparaison par itération et n itérations.
Pour `tri_select`, on effectue $n-1$ itérations, et chacune lance `ind_maxi` sur une liste de longueur $i+1$.
Or $\sum_{i=1}^{n-1} i+1 = \sum_{i=2}^{n} i = (n-2)\frac{2+n}{2} = \frac{n^2-4}{2}$
donc la complexité est en $\Theta(n^2)$.

```
In [10]: ls = list(permutation(100))

In [11]: %timeit tri_select(ls)
1000 loops, best of 3: 561 µs per loop

In [12]: ls = list(permutation(200))

In [13]: %timeit tri_select(ls)
100 loops, best of 3: 2.03 ms per loop

In [14]: ls = list(permutation(400))

In [15]: %timeit tri_select(ls)
100 loops, best of 3: 7.89 ms per loop
```

On observe effectivement que le temps est environ multiplié par 4 quand la taille de la liste double.

Tri par insertion

C'est le tri d'un jeu de carte : on insère un élément dans un tableau trié petit à petit en comparant le nouvel élément à insérer aux éléments déjà triés.
La version récursive est assez naturelle. On commence par créer une fonction qui insère un nouvel élément dans la liste :

```
def ins_rec(carte,Main):
    if Main == Vide or carte <= tete(Main):
        return [carte] + Main
    return [tete(Main)] + ins_rec(carte,queue(Main))
```

Ensuite, pour trier, on insère la tête dans la queue triée...

```
def tri_ins_rec(Pioche):
    if Pioche == Vide:
        return Vide
    return ins_rec(tete(Pioche),tri_ins_rec(queue(Pioche)))
```

Pour la version itérative, on module aussi :

```
def insere(y,xs):
    cs = (xs.copy()) + [y] # xs est triée et on insère y par la doite
    n = len(xs)
    i = n
    while cs[i] < cs[i - 1] and i > 0:
        cs[i - 1], cs[i] = cs[i], cs[i - 1]
        i -= 1
        # print(cs)
    return cs

def tri_insere(Pioche):
    Main = Vide
    for carte in Pioche:
        Main = insere(carte, Main)
        #print(Main)
    return Main
```

La justification et la complexité de ces fonctions seront à travailler en TD, mais la complexité semble encore quadratique :

```
In [29]: ls = list(permutation(100))

In [30]: %timeit tri_insere(ls)
1000 loops, best of 3: 941 µs per loop

In [31]: ls = list(permutation(200))

In [32]: %timeit tri_insere(ls)
100 loops, best of 3: 3.67 ms per loop

In [33]: ls = list(permutation(400))

In [34]: %timeit tri_insere(ls)
100 loops, best of 3: 13.7 ms per loop
```

Tri fusion

Cette fois, si le tableau a au plus une valeur, il est trié, sinon on coupe le tableau en deux, on trie ces deux moitiés et on fusionne.

```
def tri_fusion(xs):
    t = len(xs)
    if t < 2:
        return xs
    return fusion(tri_fusion(xs[:t//2]), tri_fusion(xs[t//2:]))
```

Il reste à définir la fusion :

```
def fusion(xs,ys):
    if xs == Vide or ys == Vide:
        return xs + ys
    if tete(xs) < tete(ys):
        return [tete(xs)] + fusion(queue(xs),ys)
    return [tete(ys)] + fusion(xs,queue(ys))
```

La version récursive est naturelle mais pose toujours des problèmes en Python : vous chercherez donc une version impérative à titre d'exercice...
L'étude de la terminaison, de la correction et de la complexité devra également être conduite par vos soins.
Au fait :

```
In [46]: ls = list(permutation(100))

In [47]: %timeit tri_fusion(ls)
1000 loops, best of 3: 296 µs per loop

In [48]: ls = list(permutation(200))

In [49]: %timeit tri_fusion(ls)
1000 loops, best of 3: 642 µs per loop

In [50]: ls = list(permutation(1000))

In [51]: %timeit tri_fusion(ls)
100 loops, best of 3: 3.88 ms per loop
```

Tri rapide

Le tri et ses comlexités

Observez et commentez :

```
def partition(pivot,seq):
    p,m,g = [],[],[]
    for item in seq:
        (p if item < pivot else (g if item > pivot  else m)).append(item)
    return p,m,g

def tri_rapide(xs):
    if estVide(xs):
        return Vide
    else:
        pivot = tete(xs)
        p,m,g = partition(pivot, xs)
        return (tri_rapide(p)) + m + (tri_rapide(g))
```

Pour la complexité en moyenne, démontrez que

$$K(n) = n - 1 + \frac{1}{n}\sum_{i=0}^{n-1}(K(i) + K(n-1-i))$$

et en déduire que

$$\frac{K(n)}{n+1} = 2\sum_{i=2}^{n}\frac{i-1}{i(i+1)}$$

Que se passe-t-il dans le pire des cas ?
Dans quel cas est-on sûr que le tri en $n \log n$?

La médiane en temps linéaire

Il serait très pratique d'avoir une médiane en temps linéaire : nous aurions alors un tri rapide en $n \log n$ dans le pire des cas.
La médiane d'un tableau de n éléments est l'élément de rang $\lfloor \frac{n+1}{2} \rfloor$ comme vous l'avez appris au collège.
L'idée est bien sûr d'éviter de trier le tableau pour trouver la médiane :-)
On cherche donc le tableau de rang $r_m = \lfloor \frac{n+1}{2} \rfloor$: on va utiliser la fonction partition en la modifiant légèrement pour qu'elle renvoie l'indice du pivot après partition. Si cet indice est égal à r_m on a gagné, s'il est plus petit on cherche la médiane sur la partie droite, sinon sur la partie gauche.
Y a plus qu'à...

RECHERCHES

ÉCOLE SUPÉRIEURE DE PHYSIQUE ET DE CHIMIE INDUSTRIELLES

=210mm =297mm

CONCOURS D'ADMISSION 2012

FILIÈRE MP HORS SPÉCIALITÉ INFO
FILIÈRE PC

COMPOSITION D'INFORMATIQUE – B – (XEC)

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

⋆ ⋆ ⋆

Sandwich au jambon

Le problème dit du *sandwich au jambon* ou bien encore appelé théorème de *Stone-Tukey* s'énonce de la manière suivante : un ensemble de n points en dimension d peut toujours être séparé en deux parties de cardinal au plus $\lfloor n/2 \rfloor$ par un hyperplan de dimension $d-1$ (certains points peuvent être dans l'hyperplan), où $\lfloor n/2 \rfloor$ désigne la partie entière de $n/2$. De manière concrète, un ensemble de points dans l'espace peut être séparé en deux parties quasi-égales par un plan. De même un ensemble de points dans le plan peut être séparé en deux par une droite et même en 4 à l'aide de deux droites. Ce sujet porte sur la résolution algorithmique de ce problème et de problèmes connexes selon différentes méthodes.

Dans tout le problème, les tableaux sont indicés à partir de 1. L'accès à la i-ème case d'un tableau *tab* est noté $tab[i]$. Quel que soit le langage utilisé, on suppose que les tableaux peuvent être passés comme arguments des fonctions. En outre, il existe une primitive $\mathsf{allouer}(m, c)$ pour créer un tableau de taille m dont chaque case contient c à l'origine, ainsi qu'une primitive $\mathsf{taille}(t)$ qui renvoie la taille d'un tableau t. Enfin, on disposera d'une fonction `floor`(x) qui renvoie la partie entière $\lfloor x \rfloor$ pour tout réel $x \geq 0$.

La complexité, ou le temps d'exécution, d'un programme P (fonction ou procédure) est le nombre d'opérations élémentaires (addition, soustraction, multiplication, division, affectation, etc...) nécessaires à l'exécution de P. Lorsque cette complexité dépend d'un paramètre n, on dira que P a une complexité en $\mathcal{O}(f(n))$, s'il existe $K > 0$ tel que la complexité de P est au plus $Kf(n)$, pour tout n. Lorsqu'il est demandé de garantir une certaine complexité, le candidat devra justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

Partie I. Grand, petit et médian

Dans cette partie, nous supposerons donné un tableau `tab` contenant n nombres réels. Les indices du tableau vont de 1 à n. Nous dénoterons par `tab[a..b]` le tableau pris entre les indices a et b c'est-à-dire les cellules $tab[a], tab[a+1], \ldots, tab[b-1], tab[b]$. Nous supposerons dans l'énoncé que $a \leq b$.

Nous utiliserons le tableau de taille 11 suivant pour nos exemples :

3	2	5	8	1	34	21	6	9	14	8

Question 1 Écrire une fonction `calculeIndiceMaximum`(tab, a, b) qui renvoie l'indice d'une case où se trouve le plus grand réel de $tab[a..b]$. Sur le tableau précédent avec $a = 1$ et $b = 11$, la fonction renverra 6 car la case 6 contient la valeur 34.

Question 2 Écrire une fonction `nombrePlusPetit`(tab, a, b, val) qui renvoie le nombre d'éléments dans le tableau `tab[a..b]` dont la valeur est plus petite ou égale à val. Sur le tableau exemple, pour une valeur de val égale à 5, et $a = 1, b = 11$, la fonction devra renvoyer la valeur 4 car seuls les nombres $3, 2, 5, 1$ sont inférieurs ou égaux à 5.

Nous allons maintenant calculer un médian d'un tableau. Rappelons qu'une valeur médiane m d'un ensemble E de nombres est un élément de E tel que les deux ensembles $E_{<m}$ (les nombres de E strictement plus petits que m) et $E_{>m}$ (les nombres de E strictement plus grands que m) vérifient $|E_{<m}| \leq \lfloor n/2 \rfloor$ et $|E_{>m}| \leq \lfloor n/2 \rfloor$. Notez que le problème du médian est une reformulation de problème dit du *sandwich au jambon* pris en dimension 1. Une méthode naïve consiste donc à parcourir les éléments de l'ensemble et à calculer pour chacun d'eux les valeurs de $|E_{<m}|$ et $|E_{>m}|$ mais il est possible de faire mieux comme nous allons le voir dans la partie suivante.

Partie II. Un tri pour accélérer

Une méthode plus efficace serait de trier le tableau par ordre croissant tout en prenant la cellule du milieu dans le tableau trié. Cette méthode certes rapide requiert $\mathcal{O}(n \ln n)$ opérations. Il existe une méthode optimale en temps linéaire $\mathcal{O}(n)$ pour trouver le médian d'un ensemble de n éléments. Cette partie a pour but d'en proposer une implémentation.

Une fonction annexe nécessaire pour cet algorithme consiste à savoir séparer en deux un ensemble de valeurs. Soit un tableau tab et un réel appelé pivot $p = tab[i]$, il s'agit de réordonner les éléments du tableau en mettant en premier les éléments strictement plus petits que le pivot $tab_{<p}$, puis les éléments égaux au pivot p, et en dernier les éléments strictement plus grands $tab_{>p}$. Sur le tableau exemple, en prenant comme valeur de pivot 8 on obtiendra le tableau résultat suivant :

3, 2, 5, 1, 6	**8, 8**	21, 34, 9, 14

Notez que dans le résultat les nombres plus petits que le pivot $3, 2, 5, 1, 6$ peuvent être dans n'importe quel ordre les uns par rapport aux autres.

Question 3 Écrire une fonction `partition`$(tab, a, b, indicePivot)$ qui prend en paramètre un tableau d'entiers $tab[a..b]$ ainsi qu'un entier $a \leq indicePivot \leq b$. Soit $p = tab[indicePivot]$. La fonction devra réordonner les éléments de $tab[a..b]$ comme expliqué précédemment en prenant comme pivot le nombre p. La fonction retournera le nouvel indice de la case ou se trouve la valeur p.
Dans cette question, on suppose que les modifications effectuées par la fonction sur le tableau tab sont persistantes, même après l'appel de la fonction.

Remarquons que le $\lfloor n/2 \rfloor$-ème élément dans l'ordre croissant d'un tableau de taille n est un élément médian du tableau considéré. Nous allons donc non pas programmer une méthode pour trouver le médian mais plus généralement pour trouver le k-ème élément d'un ensemble. Nous allons utiliser l'algorithme suivant :

On cherche le k-ème élément du tableau $tab[a..b]$.
- Si $k = 1$ et $a = b$ alors renvoyer $tab[a]$
- Sinon, soit $p = tab[a]$. Partitionner le tableau $tab[a..b]$ en utilisant le pivot p en mettant en premier les éléments plus petits que p. Soit i l'indice de p dans le tableau résultant.
  - Si $i - a + 1 > k$ chercher le k-ème élément dans $tab[a..i-1]$ et renvoyer cet élément.
  - Si $i - a + 1 = k$ renvoyer le pivot.
  - Si $i - a + 1 < k$ chercher le $(k - i + a - 1)$-ème élément dans $tab[i+1..b]$ et renvoyer cet élément.

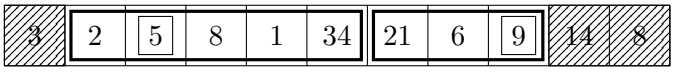
Question 4 Écrire une fonction `elementK`(tab, a, b, k) qui réalise l'algorithme de sélection du k-ème élément dans le tableau $tab[a..b]$ décrit précédemment et renvoie cet élément.

Question 5 Supposons que dans l'algorithme précédent nous voulions rechercher le premier élément mais qu'à chaque étape le pivot choisi est le plus grand élément, quel est un ordre de grandeur du nombre d'opérations réalisées par votre fonction ?

L'algorithme précédent ne semble donc pas améliorer le calcul du médian. Le problème vient du fait que le pivot choisi peut être mauvais c'est-à-dire qu'à chaque étape un seul élément du tableau a été éliminé. En fait, si l'on peut choisir un pivot p dans $tab[a..b]$ tel qu'il y ait au moins $(b-a)/5$ éléments plus petits et $(b-a)/5$ plus grands alors on peut montrer que l'algorithme précédent fonctionne optimalement en temps $\mathcal{O}(n)$.

Pour choisir un tel élément dans $tab[a..b]$, on réalise l'algorithme `choixPivot` suivant où chaque étape sera illustrée en utilisant le tableau donné en introduction en prenant $a = 2$ et $b = 9$.

- On découpe le tableau en paquets de 5 éléments plus éventuellement un paquet plus petit. On calcule l'élément médian de chaque paquet.



S'il n'y a qu'un paquet on renvoie son médian. Sinon on place ces éléments médians au début du tableau.

– On réalise `choixPivot` sur les médians précédents. Dans notre exemple on recommence donc les étapes précédentes en prenant $a = 2$ et $b = 3$.

3	5	9	2	8	1	34	21	6	14	8

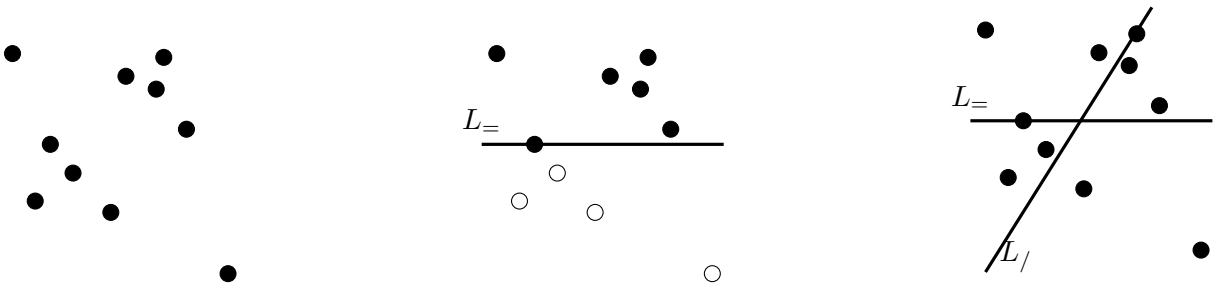
Question 6 Écrire la fonction `choixPivot`(tab, a, b) qui réalise l'algorithme précédent et renvoie la valeur du pivot.

Partie III. De la 1D vers la 2D, des nombres aux points.

Pour un réel $x \geq 0$, on note dans cette partie $\lceil x \rceil$ la partie entière supérieure de x, c'est-à-dire, le plus petit entier qui est plus grand ou égal à x : $\lceil x \rceil - 1 < x \leq \lceil x \rceil$. On supposera disposer d'une fonction `ceil`(x) qui renvoie la partie entière supérieure $\lceil x \rceil$ pour tout réel $x \geq 0$.

Dans la partie précédente, nous avons étudié le problème du médian en dimension 1. On supposera donc que vous disposez maintenant d'une fonction `indiceMedian`(tab, a, b) qui calcule un élément médian du tableau $tab[a..b]$ et renvoie l'indice de cet élément. Vous supposerez de plus que cette fonction ne modifie pas l'ordre des éléments du tableau tab.

Dans cette partie, nous généralisons l'algorithme de manière à trouver deux droites dans le plan séparant un ensemble de n points en 4 parties de cardinal plus petit que $\lceil n/4 \rceil$, Soit E un ensemble de n points tel qu'il n'existe pas trois points alignés. Nous allons chercher deux lignes $L_=$ et $L_/$ séparant cet ensemble de points en 4 parties comme le montre la troisième figure ci-dessous. En effet, dans cette figure chaque partie est composée d'exactement 2 points, les points situés sur les lignes n'étant pas pris en compte. Nous supposerons donnés dans cette partie deux tableaux $tabX$ et $tabY$ de taille n contenant les coordonnées des n points. Ainsi le point i a comme abscisse $tabX[i]$ et comme ordonnée $tabY[i]$.

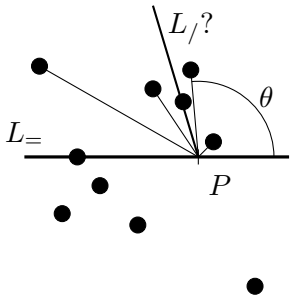


La première étape est de séparer les points en deux ensembles de même cardinal. Il suffit de remarquer que l'on peut toujours effectuer cette séparation par une ligne horizontale notée $L_=$ passant par un point d'ordonnée médiane comme le montre le second schéma ci-dessus.

Question 7 Écrire une fonction `coupeY`$(tabX, tabY)$ qui renvoie l'ordonnée d'une ligne horizontale séparant les points en deux parties de cardinal au plus $\lfloor n/2 \rfloor$.

La seconde ligne $L_/$ est plus difficile à trouver. Nous allons en réalité trouver le point d'intersection des deux lignes $L_=$ et $L_/$.

Soit P un point sur la droite horizontale $L_=$ précédente de coordonnées (x, y). On veut vérifier si ce point peut être le point d'intersection des deux lignes $L_=$ et $L_/$. Nous allons trouver dans un premier temps l'angle entre $L_=$ et $L_/$ en utilisant le fait que $L_/$ doit séparer en deux parties de cardinal proche les points au dessus de $L_=$. Ensuite nous allons vérifier si la droite $L_/$ ainsi devinée sépare les points en dessous de $L_=$ en deux parties presque égales.



Concrètement on considère les demi-droites partant de $P = (x, y)$ et joignant les k points de E au dessus strictement de $L_=$ comme indiqué sur le schéma ci-dessus. On calcule ensuite les angles θ entre $L_=$ et ces demi-droites. Remarquez alors que toute demi-droite d'angle médian partage en deux parties de cardinal $\leq \lfloor k/2 \rfloor$ les points au dessus de $L_=$.

Nous supposerons donnée une fonction `angle`$(x, y, x2, y2)$ qui calcule et renvoie l'angle formé par une demi-droite horizontale partant de (x, y) et allant vers la droite et le segment $(x, y) - (x2, y2)$. La valeur retournée sera comprise dans l'intervalle $[0, 2\pi[$.

Question 8 Écrire une fonction `demiDroiteMedianeSup`$(tabX, tabY, x, y)$ qui calcule et renvoie un angle médian entre $L_=$ et les segments reliant $P = (x, y)$ avec les points de E dont l'ordonnée est supérieure à y.

Pour un point P donné, nous avons déterminé l'angle que doit prendre $L_/$ pour couper les points au dessus de $L_=$ en 2 parties de taille au plus moitié. Il faut maintenant vérifier que cette droite $L_/$ coupe aussi les points en dessous de $L_=$ en 2 parties quasi-égales. Il suffit de vérifier que l'angle de $L_/$ partitionne les angles formés entre P et les ℓ points en dessous de $L_=$ en deux parties de cardinal $\leq \lceil \ell/2 \rceil$.

Question 9 Écrire une fonction `verifieAngleSecondeDroite`$(tabX, tabY, x, y, theta)$ qui calcule les ℓ angles formés entre $L_=$ et les points strictement au dessous de $L_=$. Votre fonction devra renvoyer :

- $\rightarrow$ 0 si theta est une médiane des ℓ angles.
- $\rightarrow$ -1 si le nombre d'angles strictement plus petits que theta est $> \lceil \ell/2 \rceil$
- $\rightarrow$ 1 si le nombre d'angles strictement plus grands que theta est $> \lceil \ell/2 \rceil$

Si $xmin$ est l'abscisse minimale des points de E et $xmax$ l'abscisse maximale, alors il est évident que l'intersection entre $L_=$ et $L_/$ ait une abscisse entre $xmin$ et $xmax$. Nous allons donc rechercher cette abscisse en utilisant l'algorithme suivant :

1. Trouver $L_=$ d'ordonnée y.
2. Soit $\alpha = xmin$ et $\beta = xmax$.
3. On calcule P le point au milieu de (α, y) et (β, y).
4. On calcule l'angle possible de la droite $L_/$ par la fonction `demiDroiteMedianeSup`.
5. Soit d la valeur donnée par la fonction $verifieAngleSecondeDroite$ avec l'angle trouvé précédemment. Si $d = 0$ alors on a trouvé $L_/$. Si $d = -1$ on recommence à partir du point 3 en prenant l'abscisse de P à la place de β. Si $d = 1$ on recommence mais en prenant l'abscisse de P à la place de α.

Question 10 Écrire la fonction `secondeMediane`$(tabX, tabY, y)$ qui à partir d'un ensemble E de points donnés par leurs coordonnées et de l'ordonnée de la droite $L_=$ calcule et renvoie un tableau contenant dans la première case l'abscisse x du point d'intersection de $L_=$ et de $L_/$ ainsi que l'angle de $L_/$ dans la seconde case.

* *
*

Complexité

Recherche 2 - 1 Maths II CCP MP 2015 : sujet 0

Non, non, ce n'est pas un extrait d'un livre de seconde, c'est un sujet 0 proposé pour l'épreuve de Math du concours CCP...

1. Écrire une fonction factorielle qui prend en argument un entier naturel n et renvoie $n!$ (on n'accepetra pas bien sûr de réponse utilisant la propre fonction factorielle du module math de Python ou Scilab).
2. Écrire une fonction seuil qui prend en argument un entier M et renvoie le plus petit entier naturel n tel que $n! > M$.
3. Écrire une fonction booléenne nommée est_divisible, qui pend en argument un entier naturel n et renvoie True si $n!$ est divisible par $n+1$ et False sinon.
4. On considère la fonction suivante nommée mystere :

```
def mystere(n):
    s = 0
    for k in range(1,n+1):
        s = s + factorielle(k)
    return s
```

   i. Quelle valeur renvoie mystere(4) ?

   ii. Déterminer le nombre de multiplications qu'effectue mystere(n).

   iii. Proposer une amélioration du script de la fonction mystere afin d'obtenir une complexité linéaire.

Recherche 2 - 2 Maths II CCP MP 2015

1. Donnez la décomposition en binaire (base 2) de l'entier 21.
   On considère la fonction `mystere` suivante :

```
def mystere(n, b) :
    """Données : n > 0 un entier et b > 0 un entier
       Résultat : ......."""
    t = [] # tableau vide
    while n > 0 :
        c = n % b
        t.append(c)
        n = n // b
    return t
```

   On rappelle que la méthode append rajoute un élément en fin de liste. Si l'on choisit par exemple `t = [4,5,6]`, alors, après avoir exécuté `t.append(12)`, la liste a pour valeur `[4,5,6,12]`.
   Pour $k \in \mathbb{N}^*$, on note c_k, t_k et n_k les valeurs prises par les variables c, t et n à la sortie de la k-ème itération de la boucle « while ».
2. Quelle est la valeur renvoyée lorsqu'on exécute `mystere(256,10)` ?
   On recopiera et complètera le tableau suivant en rajoutant les colonnes nécessaires pour tracer entièrement l'exécution :

k	1	1	...
c_k			...
t_k			...
n_k			...

3. Soit $n > 0$ un entier. On exécute `mystere(n,10)`. On pose $n_0 = n$.

   i. Justifier la terminaison de la boucle `while`.

   ii. On ote p le nombre d'itérations lors de l'exécution de `mystere(n, 10)`. Justifiez que pour tout $k \in [\![0, p]\!]$, on a $n_k \leqslant \frac{n}{10^k}$. Déduisez-en une majoration de p en fonction de n.
4. En vous aidant du script de la fonction `mystere`, écrivez une fonction `mystere` qui prend en argument un entier naturel et renvoie la somme de ses chiffres. Par exemple, `somme_chiffres(256)` devra renvoyer 13.

5. Écrivez une version récursive de la fonction **`somme_chiffres`** que vous nommerez **`somme_rec`**.

Recherche 2 - 3 Point fixe

On dispose d'un tableau de N entiers (relatifs) distincts rangés dans l'ordre croissant.
Déterminez un algorithme qui renvoie l'éventuel indice `i` tel que `a[i] = i`

Recherche 2 - 4 Recherche en dent de scie

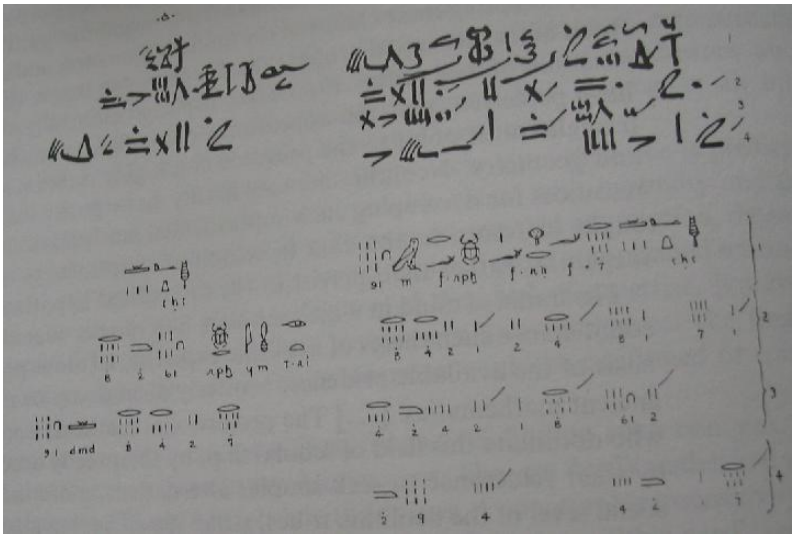
Un tableau est constitué d'une suite croissante d'entiers directement suivie d'une suite décroissante d'entiers. Tous les entiers sont supposés distincts. Déterminez une fonction qui détermine si un entier appartient ou non à un tableau en dent de scie. Votre programme doit effectuer $\sim 3\log N$ comparaisons dans le pire des cas.

Recherche 2 - 5 Écart mumixam

Étant donné un tableau de N réels, déterminez un algo en temps linéaire qui détermine la valeur maximum de `a[j] - a[i]` pour $j \geqslant i$.

Recherche 2 - 6 Multiplication du paysan russe

Voici ce qu'apprenaient les petits soviétiques pour multiplier deux entiers. Une variante de cet algorithme a été retrouvée sur le papyrus de Rhind datant de 1650 avant JC, le scribe Ahmes affirmant que cet algorithme était à l'époque vieux de 350 ans. Il a survécu en Europe occidentale jusqu'aux travaux de Fibonacci.



```
1  Fonction MULRUSSE(x: entier ,y: entier, acc:entier): entier
2  Si x==0 Alors
3  |  Retourner acc
4  Sinon
5  |  Si x est pair Alors
6  |  |  Retourner MULRUSSE(x/2,y*2,acc)
7  |  Sinon
8  |  |  Retourner MULRUSSE((x-1)/2,y*2,acc+y)
9  |  FinSi
10 FinSi
```

Que vaut acc au départ ?
Écrivez une version récursive de cet algorithme en évitant l'alternative des lignes 5 à 9.
Écrivez une version impérative de cet algorithme.
Prouvez la correction de cet algorithme.
Étudiez sa complexité.

En python, `x >> 1` décale l'entier x d'un bit vers la doite et `x << 1` décale x d'un bit vers la gauche en complétant par un zéro à droite, `x& y` renvoie l'entier obtenu en faisant la conjonction logique bit à bit des représentations binaires de x et y et `~x` renvoie le complément à 1 de x.
Ré-écrivez la multiplication russe en utilisant que ces opérations bit à bit (pas de division ni de multiplication).

Recherche 2 - 7 Produit de matrices correct

On a deux matrices A et B dont on a calculé le produit. On a trouvé C. On voudrait vérifier que le calcul a été bien mené.
Quelle est la complexité habituelle du calcul d'un produit de matrcies ?
On préfère choisir un vecteur $\boldsymbol{v}$ dans $\{0,1\}^n$ et effecteur le produit $D \times \boldsymbol{v}$ en notant $D = A \times B - C$.
D étant non nul, il a au moins un coefficient non nul. On peut se ramener au cas $d_{11} \neq 0$.
Exprimer alors v_1 en fonction des autres v_k et des d_{1k}.
En déduire que si D est non nulle, la probabilité que $D \times \boldsymbol{v} = 0_n$ est inférieure à $1/2$.
Conclure en terme de vérification et de complexité.

Recherche 2 - 8 Algorithme de Strassen

En 1969 (*« Gaussian elimination is not optimal » in Nümerische mathematik*), Volker Strassen propose un algorithme pour calculer le produit de deux matrices carrées de taille n.
Estimez tout d'abord la complexité en nombre d'additions et en nombre de multiplications de la multiplication usuelle que vous avez vue en Sup.
Considérons à présent le produit de deux matrices de taille 2×2 :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} w & x \\ y & z \end{pmatrix}$$

On note :

$$p_1 = a(f-h),\ p_2 = (a+b)h,\ p_3 = (c+d)e,\ p_4 = d(g-e),\ p_5 = (a+d)(e+h),\ p_6 = (b-d)(g+h),\ p_7 = (c-a)(e+f)$$

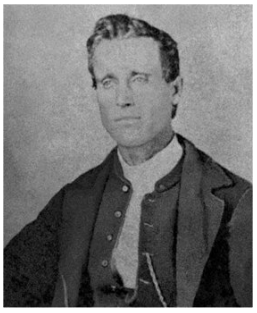
alors :

$$\begin{cases} w = p_4 + p_5 + p_6 - p_2 \\ x = p_1 + p_2 \\ y = p_3 + p_4 \\ z = p_1 + p_5 - p_3 + p_7 \end{cases}$$

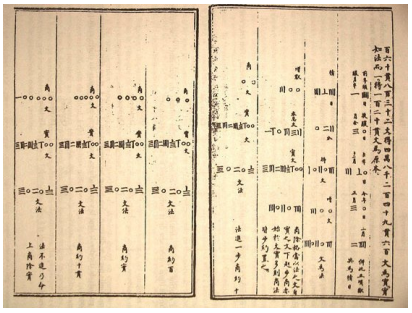
Comparez le nombre d'opérations effectuées.
Considérez maintenant des matrices carrées de taille 2^t (si ce n'est pas le cas, que peut-on faire ?).
On peut découper nos matrices en quatre blocs de taille 2^{t-1} et appliquer récursivement notre algorithme : il faut quand même vérifier quelque chose pour passer des formules sur les coefficients aux formules sur les matrices : quoi donc ?
Comptez le nombre de multiplications et d'additions nécessaires.
C'est une application de *divide et impera* : pourquoi alors ne pas diviser nos matrices en 9 blocs trois fois plus petits ?

Recherche 2 - 9 Schéma de Horner-Ruffini-Holdred-Newton-Al-Tusi-Liu-Hui...

Paolo Ruffini
(1765 - 1822)



William Horner
(1765 - 1822)



Neuf chapitres sur l'art mathématique

La méthode que nous allons voir porte le nom du britannique William George Horner (1786 - 1837) mais en fait elle fut publiée presque 10 ans auparavant par un horloger londonien, Theophilius Holdred et simultanément par l'italien Paolo Ruffini (1765 - 1822) mais fut déjà utilisée par Newton 150 ans auparavant et par le chinois Zhu Shije cinq siècles plus tôt (vers 1300) et avant lui par le Persan Sharaf al-Din al-Muzaffar ibn Muhammad ibn al-Muzaffar al-Tusi vers (1100) et avant lui par le Chinois Liu Hui (vers 200) révisant un des résultats présent dans *Les Neuf Chapitres sur l'art mathématique* publié avant la naissance de JC...
Il faut cependant noter que Ruffini l'avait employée en fait comme un moyen de calculer rapidement le quotient et le reste d'un polynôme par $(X-\alpha)$. C'est ce que nous allons (re)découvrir aujourd'hui...

Complexité

Dans toute la suite, un polynôme de dégré n sera représenté par le vecteur de ses coefficients. Par exemple, `[1 2 3]` correspond au polynôme $1+2x+3x^2$.
Prenons l'exemple de $P(x)=3x^5-2x^4+7x^3+2x^2+5x-3$. Le calcul classique nécessite 5 additions et 15 multiplications. On peut faire pas mal d'économies de calcul en suivant le schéma suivant :

$$\begin{aligned} P(x) &= \underbrace{a_n x^n + \cdots + a_2 x^2 + a_1 x}_{\text{on met } x \text{ en facteur}} + a_0 \\ &= \left(\underbrace{a_n x^{n-1} + \cdots + a_2 x}_{\text{on met } x \text{ en facteur}} + a_1 \right) x + a_0 \\ &= \cdots \\ &= (\cdots(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-3})x + \cdots)\, x + a_0 \end{aligned}$$

Ici cela donne $P(x)=(((((3x)-2)x+7)x+2)x+5)x-3$ c'est-à-dire 5 multiplications et 5 additions.
Comparez les complexités au pire du calcul de $P(t)$ pour $t\in\mathbb{K}$. Vous prendrez comme « unité de complexité » les opérations arithmétiques de base : `+` `-` `*`.
Déterminez une fonction `horner(P,t)` qui évalue le polynôme `P` en `t` selon le schéma de Horner.

Dérivées successives

On note $P_n=a_0+a_1X+\cdots+a_nX^n$ un polynôme de degré n.
Déterminez le lien entre l'algorithme de Horner et la division euclidienne de P_n par $(X-t)$. On utilisera par la suite la notation suivante :

$$P_n=(X-t)(b_nX^{n-1}+b_{n-1}X^{n-2}+\cdots+b_1)+b_0$$

Comment calculer les dérivées successives $\widetilde{P_n^{(j)}}(t)$ en utilisant uniquement des schémas de Horner ? Quel est l'intérêt informatique ? Voici un peu d'aide...
Considérons le développement de P_n selon les puissances croissantes de $(X-t)$:

$$P_n=t_n(X-t)^n+t_{n-1}(X-t)^{n-1}+\cdots+t_0$$

Montrez que $\widetilde{P_n^{(j)}}(t)=(j!)\times t_j$. Quel est le lien entre les t_j et nos schémas de Horner ?

Version récursive

Déterminez une fonction récursive `der_j(poly,j,t)` qui calcule $\widetilde{P_n^{(j)}}(t)$ en utilisant le schéma de HORNER.
Pour cela, on pourra commencer par modifier légèrement la fonction `horner` pour récupérer le vecteur $[b_0 b_1 \dots b_n]$.
Créez ensuite `quotient_horner(poly,t)` qui renvoie le quotient de `poly` par `(X-t)` puis `reste_horner(poly,t)`.
Déduisez-en une fonction récursive `jeme_poly(poly,j,t)` qui renvoie $\widetilde{P_n^{(j)}}(t)$.

Version itérative

Prenons par exemple le polynôme $P_4 = [6, -7, 1, -5, 2]$. Essayez de trouver un algorithme simple pour remplir le tableau suivant :

Restes	0	1	2	3	4	j	
/	6	-7	1	-5	2	a_j	P_4
-12	-9	-1	-1	2	/	b_j	Q_3
						c_j	Q_2
						d_j	Q_1
						e_j	Q_0

En déduire l'écriture de P_4 sous la forme $R_4 + R_3(X-2) + R_2(X-2)^2 + R_1(X-2)^3 + R_0(X-2)^4$ et faire le lien avec les dérivées successives.
Écrivez ensuite une fonction qui renvoie un tableau similaire puis transformez-la un peu pour obtenir les divisions successives en t.

```
jeme_poly_mat([6 -7 1 -5 2],5,2)
```

```
   0.     6.   - 7.     1.   - 5.     2.
 - 12.  - 9.   - 1.   - 1.     2.     0.
   1.     5.     3.     2.     0.     0.
   19.    7.     2.     0.     0.     0.
   11.    2.     0.     0.     0.     0.
   2.     0.     0.     0.     0.     0.
```

Recherche 2 - 10 Complexité de l'algorithme d'Euclide et nombre d'or...

On suppose dans toute la suite que a et b sont deux entiers naturels tels que $a \geqslant b$.

1. Rappeler le principe de l'algorithme et de sa preuve.
2. Soit $r_n = a \wedge b$ avec les notations habituelles : $r_0 = b$ puis $r_1 = a \bmod b$, $r_2 = r_1 \bmod r_0$ et plus généralement :

$$r_{k+1} = r_k q_k + r_{k-1}, \quad 0 \leqslant r_{k-1} < r_k$$

   Montrez que pour tout entier $k \in \{1, 2, \dots, n\}$, on a $q_k \geqslant 1$ puis que $q_n \geqslant 2$.
3. Notons $\Phi = \frac{1+\sqrt{5}}{2}$. Comparez Φ et $1 + \frac{1}{\Phi}$.
4. Montrez par récurrence que $r_k \geqslant \Phi^{n-k}$ pour tout entier $k \in \{0, 1, \dots, n\}$.
5. Déduisez-en que $b \geqslant \Phi^n$ puis que le nombre d'appels récursifs de l'algorithme d'Euclide est au maximum de $\frac{\ln b}{\ln \Phi}$.

Tris

Recherche 2 - 11 Tri sportif

Que pensez-vous de la complexité de cet algorithme de tri ?
(source : http://unclecode.blogspot.tw/2012/02/stupid-sort_2390.html) :

```
void StupidSort()
{
    int i = 0;
    while(i < (size - 1))
    {
           if(data[i] > data[i+1])
          {
                int tmp   = data[i];
                data[i]   = data[i+1];
                data[i+1] = tmp;
                i = 0;
          }
          else
         {
               i++;
         }
     }
}
```

Recherche 2 - 12 Mélange de cartes : le mélange de ASF Software

Savoir bien trier c'est aussi savoir bien mélanger.
Afin de convaincre les utilsateurs que leur algorithme de mélange était juste, la société ASF Software, qui produit les logiciels utilisés par de nombreux sites de jeu, avait publié cet algorithme :

```
procedure TDeck.Shuffle;
var
    ctr: Byte;
    tmp: Byte;

    random_number: Byte;
begin
    { Fill the deck with unique cards }
    for ctr := 1 to 52 do
        Card[ctr] := ctr;

    { Generate a new seed based on the system clock }
    randomize;

    { Randomly rearrange each card }
    for ctr := 1 to 52 do begin
        random_number := random(51)+1;
        tmp := card[random_number];
        card[random_number] := card[ctr];
        card[ctr] := tmp;
    end;

    CurrentCard := 1;
    JustShuffled := True;
end;
```

1. Considérez un jeu de 3 cartes. Dressez l'arbre de tous les mélanges possibles en suivant cet algorithme. Que remarquez-vous ?
2. Proposez un algorithme qui corrige ce problème. Dressez l'arbre correspondant pour un jeu de trois cartes.

3. Traduisez la fonction proposée en Python puis votre version corrigée en Python.

Recherche 2 - 13 Yet another sort of sort

```
Pour i de 1 à n Faire
    Pour j de n à i+1 parPasDe -1 Faire
        Si t[j]<t[j-1] Alors
            Échange t[j] et t[j-1]
        FinSi
    FinPour
FinPour
```

Qu'est-ce que c'est ? Qu'est-ce que ça fait ? Comment ça marche ? Complexité ? En python ? Donnez un nom à ce tri.

Recherche 2 - 14 Comparaisons de tris et révisions sur le tracé de graphiques

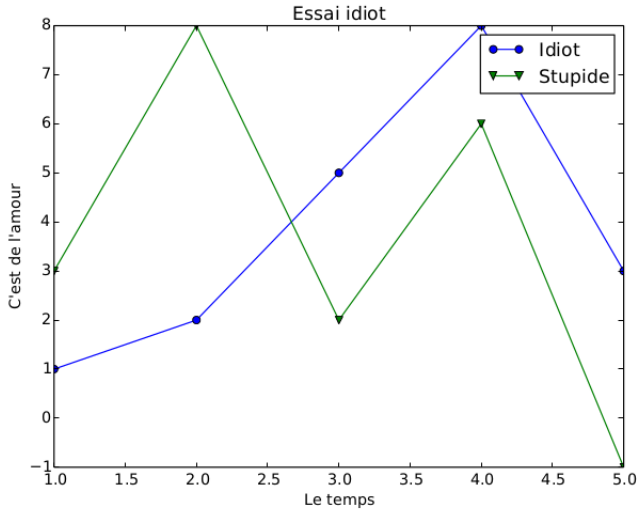
Tracez sur un même graphique les temps d'exécution des tris étudiés en fonction de la longueur d'une liste « mélangée ».

Sur un autre graphique, comparez ces tris avec des listes ordonnées dans le sens croissant, puis avec des listes ordonnées dans le sens décroissant.

Pour cela, voici quelques rappels sur Matplotlib.

On lance ipython avec l'option `-pylab` ou bien, si vous ne travaillez pas dans un confortable terminal Unix, vous pouvez toujours importer la chose :

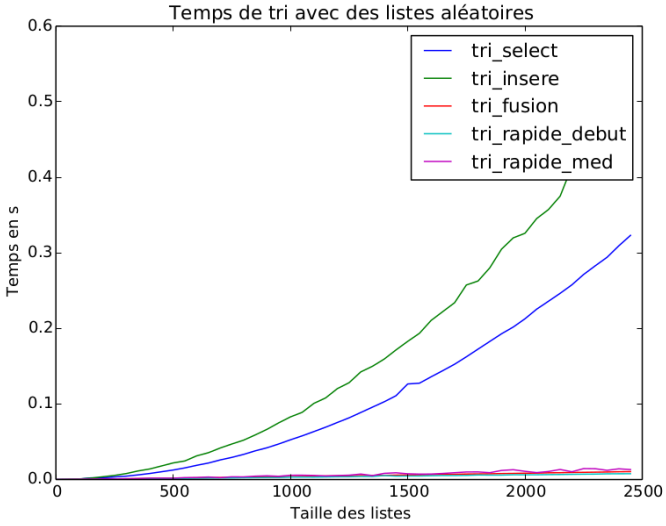
```
In [29]: import matplotlib.pyplot as plt

In [30]: p1 = plt.plot([1,2,3,4,5],[1,2,5,8,3], marker = 'o',label = "Idiot")

In [31]: p2 = plt.plot([1,2,3,4,5],[3,8,2,6,-1], marker = 'v',label = "Stupide")

In [32]: plt.legend()
Out[32]: <matplotlib.legend.Legend at 0x7fbc2c6b76d8>

In [33]: plt.title("Essai idiot")
Out[33]: <matplotlib.text.Text at 0x7fbc2babc668>

In [34]: plt.xlabel("Le temps")
Out[34]: <matplotlib.text.Text at 0x7fbc2c620898>

In [35]: plt.ylabel("C'est de l'amour")
Out[35]: <matplotlib.text.Text at 0x7fbc2baaa2b0>

In [36]: plt.show()

In [37]: plt.savefig("zig.pdf")

In [38]: plt.clf()
```



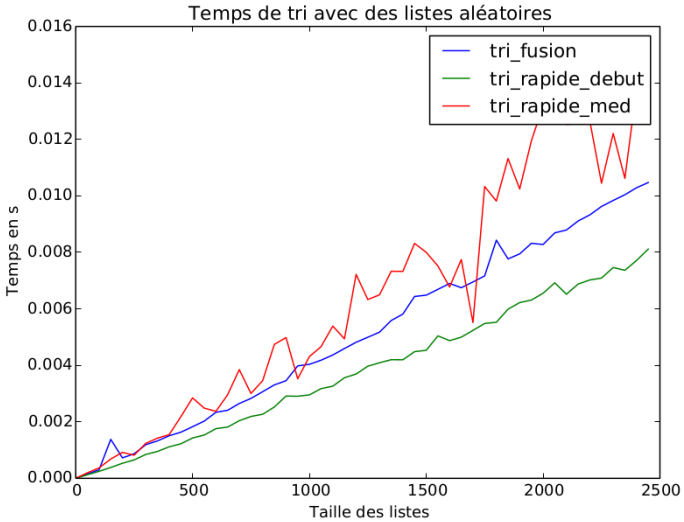
Comment mesurer le temps ? On utilise la fonction `perf_counter` de la bibliothèque `time` :

```
from time import perf_counter

def temps(tri,p):
    debut = perf_counter()
    tri(p)
    return perf_counter() - debut
```

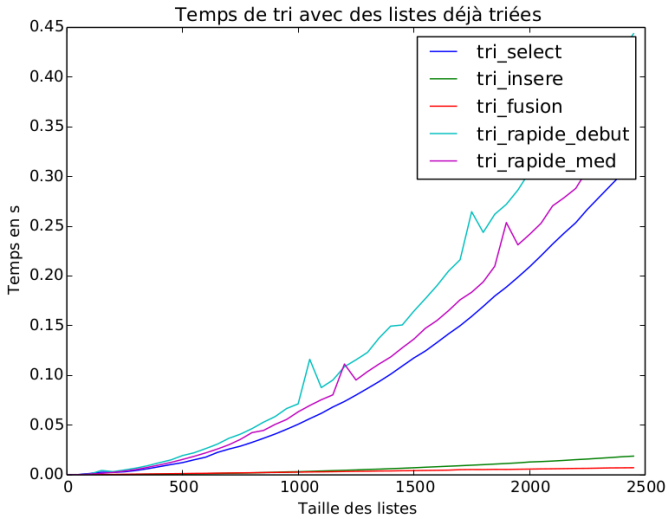
On devrait obtenir :



et en regardant les trois plus rapides :



mais si les listes sont déjà rangées :



Comme le nombre d'appels récursifs va augmenter, on va augmenter aussi le nombre d'appels récursifs autorisé :

```
from sys import setrecursionlimit
setrecursionlimit(100000)
```

Recherche 2 - 15 Politique américaine

Le Sénat américain est composé de deux sénateurs par état (il y a 50 états...). Vous allez récupérer les statistiques sur 46 votes du 109e congrès (2006). Un 1 signifie un « Yea », un −1 un « Nay » et un 0 une abstention. Il manque les votes de Jon Corzine, ssénateur du New-Jersey qui a été victime d'un accident de voiture cette année-là (cela fait une exception à gérer :-)).
On utilisera les fonctions **open, read, split** pour récupérer les données.

1. Créez un dictionnaire vs de type **{Nom : liste des votes}** et un autre, os, de type **Nom : [Parti, État]**. Par exemple :

```
In [18]: os['Obama']
Out[18]: ['D', 'IL']

In [20]: vs['Obama']
Out[20]: [1, -1, 1, 1, 1, -1, -1,...]
```

On pourra utiliser la fonction **int** et des définitions par compréhension.

2. Comment utiliser le produit scalaire pour mesurer l'accord ou le désaccord entre deux sénateurs ?

3. Écrivez une fonction **compare(sen1, sen2)** qui mesure le degré d'accord de deux sénateurs. Par exemple :

```
In [21]: compare('Obama', 'McCain')
Out[21]: 16
```

4. Écrivez une fonction **senat_parti(parti)** qui renvoie l'ensemble des noms des sénateurs d'un parti donné. Créez aussi une fonction qui renvoie le nom des sénateurs par état, le nom des états par parti.

5. Écrivez une fonction **compare_sen_etat(etat)** qui mesure la cohérence des votes des deux sénateurs d'un état donné.

```
In [23]: compare_etat('WA')
Out[23]: (39, 'WA', ('Cantwell', 'Murray'))
```

Classez les états dans l'ordre croissant des cohérences.

6. Créez une fonction **plus_eloigne(sen, ens)** qui renvoie le nom et la mesure de l'écart du sénateur d'un ensemble donné le plus en désaccord avec un sénateur donné.

```
In [33]: plus_eloigne('Obama',senat_parti('D'))
Out[33]: (22, 'Nelson2')

In [34]: plus_eloigne('Obama',senat_parti('R'))
Out[34]: (7, 'Sununu')
```

Faites de même avec le plus en accord.

7. Quel est le parti le plus « cohérent » ? Comment le déterminer à l'aide d'une fonction calculant une moyenne de cohérence.

Par souci d'efficacité, on pourra se souvenir que le produit scalaire est distributif sur l'addition des vecteurs.

8. Écrivez une fonction **moins_d_accord(ens)** qui renvoie le couple de sénateurs les moins d'accord d'un ensemble donné :

```
In [50]: moins_d_accord(senateurs)
Out[50]: (('Feingold', 'Inhofe'), -3)
```

Combien de calculs et de comparaisons sont effectués ? Pensez-vous pouvoir être plus efficace ?

9. Qui est le sénateur le plus Républicain ? L'État le plus Démocrate ? Classez les sénateurs/États selon ces critères.

10. Ouvrez le fichier **ONU_vote.txt** et étudiez-le d'une manière similaire à ce qui vient d'être fait.

Recherche 2 - 16 Algorithme probabiliste : amélioration du tri rapide

On peut introduire une dose de hasard dans un algorithme de deux façons : on choisit le pivot au hasard ou on prend toujours le premier élément de la liste mais on permute aléatoirement les autres éléments de la liste.
On distingue deux classes d'algorithmes probabilistes (nous ne rentrerons pas dans les détails des classes de complexité, des machines de Turing probabilistes,etc.) :

les algorithmes de Monte-Carlo : l'algorithme peut se tromper mais on connait la probabilité qu'il donne la bonne réponse : on peut donc diminuer cette probabilité en augmentant le nombre d'exécution.

les algorithmes de Las Vegas : ils donnent toujours la bonne réponse mais le temps d'exécution est une variable aléatoire.

On peut passer de Monte-Carlo à Las Vegas à la condition de pouvoir déterminer si l'issue de Monte-Carlo est correcte : comment ?
Revenons au cas du tri rapide.
Notons $\{x_1, x_2, ..., x_n\}$ la suite à trier et $\{_1, y_2, ..., y_n\}$ la suite triée. Soit $i < j$. On considère la var X_{ij} qui vaut 1 si y_i et y_j ont été comparés et 0 sinon.
Soit X le nombre total de comparaisons, alors :

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

Que vaut $\mathbb{E}(X)$? Que vaut $\mathbb{E}(X_{ij})$ compte-tenu de la nature de la var X_{ij} ? Maintenant, y_i et y_j sont comparés si, et seulement si, l'un des deux est choisi comme pivot de la sous-suite $y_i \dots y_j$.
En effet, si aucun des deux n'est pivot de cette suite, ils seront dans des sous-listes séparées et ne seront jamais comparés. Si, l'un des deux est choisi comme pivot de la sous-suite $y_i, \dots, y_j$, ils seront forcément comparés. Écrivez plus formellement cette preuve à l'aide de la logique des propositions. Quelle est donc la probabilité que $X_{ij} = 1$? En déduire $\mathbb{E}(X)$ en fonction de $H_n = \sum_{k=1}^{n} 1/k$. Concluez.
Observez et testez en vous inspirant de ce qui a été fait à l'exercice précédent.

Recherche 2 - 17 Pi au casino

Imaginez un moyen de calculer une valeur approchée de π avec un algorithme probabiliste.

Recherche 2 - 18 MURD3RS

Les américains ont un goût prononcé pour les tueurs en série. Le profil psychologique de certains de ces héros maléfiques correspond à une personne logique, froide, méthodique, dangereuse, qui connait bien son métier, qui travaille suffisamment lentement pour bien faire son boulot et suffisamment rapidement pour être efficace. Certains sont des sociopathes, d'autres sont très à l'aise en société : on retrouve exactement les qualités recherchées chez un programmeur...Bref, si on apprenait plus de Caml à l'école, les tueurs en série disparaîtraient. Un policier canadien, Kim Rossmo, s'est lui aussi passionné pour les mathématiques et la pro- grammation et devint en 1995 le premier policier canadien à obtenir un doctorat en crimonologie (https://en.wikipedia.org/wiki/Kim_Rossmo). Il a publié un article (http://www.popcenter.org/library/crimeprevention/volume_04/10-Rossmo.pdf) sur l'analyse géographique des crimes et proposa une formule pour localiser le lieu d'habitation d'un tueur en série, ou plutôt pour calculer le lieu le plus probable :

$$p_{i,j} = k \sum_{n=1}^{(\text{nb total de crimes})} \left[\underbrace{\frac{\phi_{ij}}{(|X_i - x_n| + |Y_j - y_n|)^f}}_{1^{\text{er}}\text{ terme}} + \underbrace{\frac{(1-\phi_{ij})(B^{g-f})}{(2B - |X_i - x_n| - |Y_j - y_n|)^g}}_{2^{\text{nd}}\text{ terme}} \right],$$

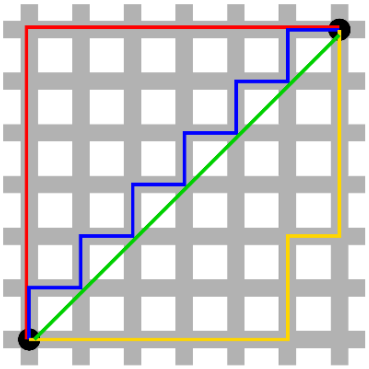
avec

$$\phi_{ij} = \begin{cases} 1, & \text{si } (|X_i - x_n| + |Y_j - y_n|) > B \\ 0, & \text{sinon} \end{cases}$$

Bon, plus synthétiquement, ça peut s'écrire :

$$P(x) = \sum_{\text{lieuducrimec}} \frac{\varphi}{d(x,c)^f} + \frac{(1-\varphi)B^{g-f}}{(2B - d(x,c))^g}$$

....euh...qui est qui là-dedans ? On discrétise un plan, de ville de préférence, à l'aide d'une grille. On calcule alors la probabilité que le tueur habite dans la case x.
Comme on travaille dans une ville, la distance d sera la distance du taxi de Manhattan :



ou si vous préférez :

B est le rayon de la « Buffer zone » ou zone tampon : c'est un « cercle » (au sens des taxis) autour du lieu du crime. La fonction φ est la fonction caractéristique de cette zone tampon : elle vaut 1 si la cellule est dans la zone et 0 sinon. f et g sont des paramètres arbitraires qui permettent de s'adapter aux données des crimes précédents. On cherche à présent à dresser une carte des probabilités. On va utiliser la fonction **pcolormesh** de **matplotlib.pyplot** . Vous pouvez regarder l'aide. Ce qui va motiver notre construction, c'est la nature des arguments de cette fonction :

```
def dessine_rossmo( lieux, vrai, B, f, g, taille_x, taille_y) :
    t_x = np.arange(taille_x)
    t_y = np.arange(taille_y)
    X, Y = np.meshgrid(t_x, t_y)
    Z = np.vectorize( proba_rossmo(lieux, B, f, g) )(X, Y)
    g = cg(lieux)
    plt.pcolormesh( X, Y, Z, cmap = cm.jet)# , shading='gouraud')
    plt.plot(vrai[0], vrai[1], 'ks', mew = 7)
    plt.plot(g[0],g[1], 'ko', mew = 7)
    plt.colorbar()
    plt.show()
```

Ici, `lieux` est l'ensemble des lieux des crimes, `vrai` est le véritable domicile du tueur, `taille_x` et `taille_y` sont les dimensions de la grille.

La fonction `cg` calcule les coordonnées du centre de gravité des lieux des crimes pour comparer avec le candidat de la méthode Rossmo.

`colorbar` permet d'avoir une échelle de couleur pour lire la carte. Il va falloir bien comprendre la construction de Z...À vous de menez l'enquête informatique.

Rossmo fournit des données pour deux tueurs en série célèbres :

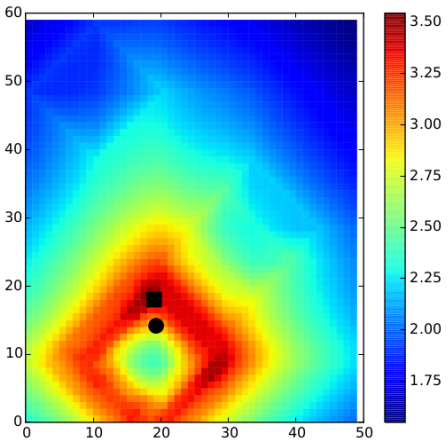
— le vampire de Sacramento (https://en.wikipedia.org/wiki/Richard_Chase)

```
vampire = {(3, 17), (15, 3), (19, 27), (21, 22), (25, 18)}
vrai_vampire = ([19],[17])
```

— L'étrangleur de Boston (https://en.wikipedia.org/wiki/Albert_DeSalvo)

```
boston = [[10, 48], [13, 8], [15, 11], [17, 8], [18, 7],
          [18, 9], [19, 4], [19, 8], [20, 9], [20, 10], [20, 11],
          [29, 23], [33, 28]]
vrai_boston = ([19],[18])
```

Vous obtiendrez des cartes qui ressemblent à ça :



THÈME

A

POO for dummies

Syntaxe ?

Vous vous êtes sûrement demandé pourquoi les appels de « fonctions » n'ont pas toujours la même syntaxe :

```
In [67]: e = [1,2]

In [68]: len(e)
Out[68]: 2

In [69]: e.len
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-69-8a0fbf71e19e> in <module>()
----> 1 e.len

AttributeError: 'list' object has no attribute 'len'

In [70]: append(e,3)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-70-18248a9bc3f7> in <module>()
----> 1 append(e,3)

NameError: name 'append' is not defined

In [71]: e.append(3)

In [72]: e
Out[72]: [1, 2, 3]
```

Même les nombres ont ce genre de « fonctions » bizarres qui suivent un point :

```
In [73]: 1.2.is_integer()
Out[73]: False

In [74]: 1.0.is_integer()
Out[74]: True
```

En fait, cette syntaxe mystérieuse provient du fait que Python est un langage de programmation orientée objet. Mais nous n'avons que quelques minutes pour en parler...

Vocabulaire

Une *classe* correspond à un « moule » permettant de créer un type d'objet.
Il ne faut pas confondre la classe avec l'objet (il vaut mieux manger le gâteau que le moule).
Prenons un exemple : vous êtes Saroumane et vous avez un moule à orques. Vous pouvez créer ainsi une infinité d'orques. Informatiquement la classe `Orque` permet d'*instancier* un objet de type orque.
Un orque a un nom et un poids : ce sont des *attributs* (ses caractéristiques). Il peut effectuer un salut : c'est une *méthode* (ce que l'objet peut faire).
On résume ceci dans un petit tableau :

Orque
nom poids maitre
salut()

En python, je pourrais faire ça :

```
class Orque:

    nom    = ""
    poids  = 0
    maitre = ""

    def salut(self):
        print("Ash nazg durbatulûk! Mon nom est %s" % self.nom)
```

et créer des orques :

```
In [76]: a = Orque()

In [77]: a.nom = "Grishnákh"

In [78]: a.salut()
Ash nazg durbatulûk! Mon nom est Grishnákh

In [79]: a.age = 2

In [80]: b = Orque()

In [81]: b.nom = "Uglúk"

In [82]: b.salut()
Ash nazg durbatulûk! Mon nom est Uglúk

In [83]: a.maitre = "Saroumane"

In [84]: b.maitre
Out[84]: 'Sauron'
```

Les attributs introduits dans la classe ont des valeurs arbitraires et on aurait pu s'en passer comme le montre la création ad hoc de l'attribut age.

Pour clarifier ceci, Python fournit la fonction `__init__()` (notez bien les deux horribles tirets avant et après `init`) qui est appelée à chaque création d'objet et donne une valeur initiale.
On aurait alors pu créer la classe comme ceci :

```
class Orque:

    def __init__(self):
        self.nom    = ""
        self.poids  = 0
        self.maitre = "Sauron"

    def salut(self):
        print("Ash nazg durbatulûk! Mon nom est %s" % self.nom)
```

On remarque aussi qu'une méthode voulant utiliser un attribut d'une classe doit faire précéder cet attribut du nom de l'instance qui la possède. En python, on utilise le nom standard `self`.

On peut aussi permettre à l'utilisateur(rice) de choisir lui(elle)-même les valeurs par défaut. On ajoute alors des arguments à `__init__` :

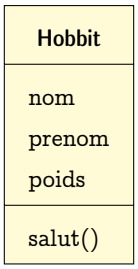
```
class Orque:

    def __init__(self, nom = "", poids = 0, maitre = "Sauron"):
        self.nom    = nom
        self.poids  = poids
        self.maitre = maitre

    def salut(self):
        print("Ash nazg durbatulûk! Mon nom est %s" % self.nom)
```

On peut l'utiliser ainsi :

```
In [106]: a = Orque("Truc",3)

In [107]: a.maitre
Out[107]: 'Sauron'

In [108]: a = Orque("Truc","Sauron")

In [109]: a.maitre
Out[109]: 'Sauron'

In [110]: a.poids
Out[110]: 'Sauron'

In [111]: a = Orque("Truc",100,"Sauron")

In [112]: a.poids
Out[112]: 100

In [113]: a.maitre
Out[113]: 'Sauron'

In [114]: a.nom
Out[114]: 'Truc'
```

Héritage

Imaginez maintenant que vous êtes JRR Tolkien et que vous voulez fabriquer des Hobbits. Vous vous dites qu'il suffit de créer une classe `Hobbit` :




```
class Hobbit:

    def __init__(self, nom = "", prenom = "", poids = 0):
        self.nom    = nom
        self.prenom = prenom
        self.poids  = poids

    def salut(self):
        print("Bonjour! Mon nom est %s %s" % (self.prenom, self.nom))
```

qui s'utilise comme d'habitude :

```
In [119]: a = Hobbit("Sacquet", "Bilbo", 50)

In [120]: a.salut()
Bonjour! Mon nom est Bilbo Sacquet
```

Et ensuite il y a les Elfes, les Nains, etc.
On remarque cependant que les Orques comme les Hobbits ont un nom et un poids.
On peut alors créer une « super-classe » `Etre` dont `Hobbit` et `Orque` serait des sous-classes qui *héritent* de ses attributs :

```
class Etre:

    def __init__(self, appellation = "", poids = 0, bonjour = ""):
        self.appellation = appellation
        self.poids       = poids
        self.bonjour     = bonjour

    def salut(self):
        print("%s ! Mon nom est %s" % (self.bonjour, self.appellation))

    def masse(self):
        print("Je pèse %f kg" % self.poids)


class Orque(Etre):

    def __init__(self,nom = "", poids = 0, maitre = "Sauron"):
        Etre.__init__(self,nom,poids,"Ash nazg durbatulûk")
        self.maitre = maitre

class Hobbit(Etre):

    def __init__(self, nom = "", prenom = "", poids = 0):
        Etre.__init__(self,prenom + ' ' + nom, poids, "Bonjour")
        self.prenom  = prenom
```

Alors `Hobbit` et `Orque` héritent des attributs et méthodes de `Etre` alors qu'ils ne sont pas définis dans les sous-classes :

```
In [137]: a = Hobbit("Sacquet", "Bilbo", 50)

In [138]: a.masse()
Je pèse 50.000000 kg

In [139]: a.salut()
Bonjour ! Mon nom est Bilbo Sacquet

In [140]: b = Orque("Uglúk",100)

In [141]: b.salut()
Ash nazg durbatulûk ! Mon nom est Uglúk
```

RECHERCHES

Manipulation des entiers comme chaînes de bits

Nous allons essayer de nous rapprocher du cœur de la machine (autant que faire ce peut avec Python...) en définissant les opérations arithmétiques à partir de listes de bits.

Classe de chaînes de bits

Nous allons manipuler les entiers signés comme des listes de bits de manière simple.
Par exemple, 13 sera représenté par Bits ([0,1,1,0,1]) , le premier bit représentant le signe : 0 pour + et 1 pour −.

```python
class Bits(list) :
    """
    Crée des objets pour représenter des chaînes de bits correspondant à des entiers signés à partir de
        ↪ listes de 0 et 1.
    Par exemple 13 sera Bits([0,1,1,0,1])
    """

    def __init__(self,bits = []) :
        """ Une chaine de bits est construite à partir de la liste de ses bits"""
        self.bits = bits

    def __iter__(self) :
        """ On itère sur la liste des bits"""
        return iter(self.bits)

    def __len__(self) :
        """ Permet de donner le nombre de bits avec la méthode len"""
        return len(self.bits)

    def __getitem__(self,cle) :
        """ Pour obtenir le bit en position cle avec b[cle] ou une
            séquence de bits avec b[deb..fin]"""
        return self.bits[cle]

    def __reversed__(self) :
        """Pour renvoyer la chaîne de bits inversée"""
        return Bits(self.bits[::-1])

    def norm(self) :
        """Normalise l'écriture d'une chaîne de bits en supprimant les
        bits non significatifs à gauche"""
        if len(self) == 0:
            return self
        if len(self) == 1:
            return Bits(self.bits + self.bits)
        tete = self[0]
        qs   = self[1:]
        if qs[0] == tete:
            return Bits(qs).norm()
        return self

    def __str__(self) :
        """On utilise la fonction str pour afficher la liste des bits"""
        return str(self.bits)

    def __repr__(self) :
        """Une belle représentation dans l'interpréteur
           Bits([0,1,1,0,1]) est affiché 0:1101"""
        n = self.norm()
        return str(n[0]) + ':' + ''.join(str(b) for b in n.bits[1:])
```

Somme de deux chaînes de bits

On voudrait additionner facilement ces nombres. Une idée serait d'utiliser l'addition posée comme à l'école primaire. Par exemple pour calculer $5+9$

```
    00101
+   01001
------------
    01110
```

On trouve bien 14.
Il faut faire attention à reporter la retenue.
Que se passe-t-il quand on additionne 3 bits ?

b_1	b_2	b_3	retenue	unité
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

On remarque que u = b1 ^ (b2 ^ b3) avec ^ l'opérateur du OU EXCLUSIF.
Quant à la retenue, elle vaut 1 si au moins deux des bits sont à 1. On va donc utiliser les opérateurs logiques ET et OU qui s'écrivent & et | en Python.

```python
def __add__(self, other) :
        """
        Additionne deux entiers en base 2 sous forme de la liste de leurs bits
        """
        # Formate les listes pour qu'elles soient de même longueur
        ????
        op1 = Bits([???)
        op2 = Bits(???)
        # stocke la retenue de la précédente somme de 2 bits
        retenue = 0
        # On stocke la somme intermédiaire
        res = [];
        # on itère dans l'ordre inverse de la liste des bits des opérandes
        ????
            # unité comme xor des deux bits et de la retenue
            res = ???
            # retenue : vaut 1 si au moins 2 bits sont à 1
            retenue = ???
        # on ajoute l'éventuelle dernière retenue à droite
        ???
        # on remet la liste dans le bon sens
        return Bits(res).norm()
```

Bon, essayons avec des nombres négatifs. Comment les représenter ? L'idée naïve est de mettre un 1 en premier puis de garder la même « valeur absolue ». Par exemple pour $5+(-9)$:

```
    00101
+   11001
------------
    11110
```

Mais on trouve $5+(-9)=-14$! Il faut trouver une autre idée sinon il faudrait créer une addition pour les positifs et une autre pour les nombres de signes différents ce qui ralentirait énormément le traitement des nombres sur machine.
Considérons n=11110 et calculons $n+2$.

```
    11110
+   00010
------------
 (1)00000
```

Si l'on ne garde que le nombre de bits correspondant au plus long nombre, on obtient 0 (sur machine, on travaille sur 64 bits par exemple et tout ce qui dépasse cette taille n'est pas pris en compte).
On peut donc considérer que 11110 est l'opposé de 00010 en conservant le même algorithme d'addition.
En fait, ajouter 1 à 1111111 ou n'importe quelle autre chaîne uniquement composée de 1 donnera zéro.
Pour obtenir cette chaîne uniquement composée de 1, on ajoute au nombre initial son *complément à 1*, c'est-à-dire qu'on remplace chaque bit par son complémentaire.
Par exemple, le complément à 1 de 01001 est 10110. En rajoutant 1 à ce nombre, on obtient donc l'opposé du nombre initial.

$$01001 \xrightarrow{\text{complément à 1}} 10110 \xrightarrow{+1} 10111$$

Ainsi -9 s'écrit 10111 et cette fois :

```
    00101
+   10111
------------
    11100
```

Pour savoir le nombre que cette chaîne représente, on effectue les transformations réciproques dans l'ordre inverse :

$$11100 \xrightarrow{-1} 11011 \xrightarrow{\text{complément à 1}} 00100$$

On trouve 4 : en effet, $5 + (-9) = -4$!
Créons les méthodes associées :

```
1   def map(self,fonc) :
2       """Applique une fonction aux bits d'une chaîne"""
3       return Bits(????)
4
5   def comp1(self) :
6       """Renvoie le complément à 1 d'une chaîne"""
7       return ????
8
9   def __neg__(self) :
10      """Opposé d'une chaîne en utilisant le symbole -"""
11      ????
12
13  def __sub__(self,other) :
14      """Différence de deux chaînes avec -"""
15      return self + (-other)
16
17  def __abs__(self) :
18      """Valeur absolue d'une chaîne"""
19      ????
```

Conversion en base 10

On rajoute quelques méthodes utiles :

```
1   def mantisse(self) :
2       """La liste des bits sans le bit de signe"""
3       ???
4
5   def __eq__(self,other) :
6       """Permet de tester l'égalité de deux chaînes à partir de leur forme normalisée"""
7       return self.norm().bits == other.norm().bits
8
9   def signe(self) :
10      """Renvoie le bit de signe"""
11      ?????
12
13  def pair(self) :
14      """Teste la parité d'une chaîne"""
15      ?????????
```

On va maintenant passer de cette représentation en base 2 à la représentation en base 10 mais sans utiliser d'opérations arithmétiques.
Il existe sur tout langage des *opérations bits à bits* qui travaillent directement sur la représentation en base 2 des nombres. Nous avons déjà vu les opérateurs OU EXCLUSIF, ET, OU. Il existe également les opérateurs de *décalage* vers la gauche << ou la droite >> qui décale la chaîne de bits d'un certain nombres de bits d'un côté ou de l'autre. Ainsi on peut manipuler des entiers Python comme nos objets Bits.
Un décalage << n correspondra donc à une multiplication par 2^n et un décalage >> n à une division par 2^n.

```
1       def to_dec(self) :
2         """Renvoie l'écriture décimale de la chaîne de bits"""
3         ?????????
```

Par exemple :

```
1  In [29]: treize = Bits([0,1,1,0,1])
2  In [30]: dix = Bits([0,1,0,1,0])
3
4  In [31]: treize
5  Out[31]: 0:1101
6
7  In [32]: treize.to_dec()
8  Out[32]: 13
9  In [33]: dix - treize
10 Out[33]: 1:01
11 In [34]: (dix - treize).to_dec()
12 Out[34]: -3
```

Multiplication de l'école primaire

Nous allons commencer par la multiplication telle qu'elle est en général enseignée dans nos écoles primaires.
On prend les chiffres un par un d'un nombre, on les multiplie par l'autre nombre et on ajoute les résultats obtenus en « décalant ».

```
1    def mul_primaire(self,other) :
2         """Multiplication avec l'algo appris à l'école primaire"""
3         ???????
```

On peut alors définir cette multiplication comme produit canonique :

```
1  def __mul__(self,other) :
2         """Définit la multplication de deux chaîne avec l'algo de l'école primaire"""
3         return self.mul_primaire(other)
```

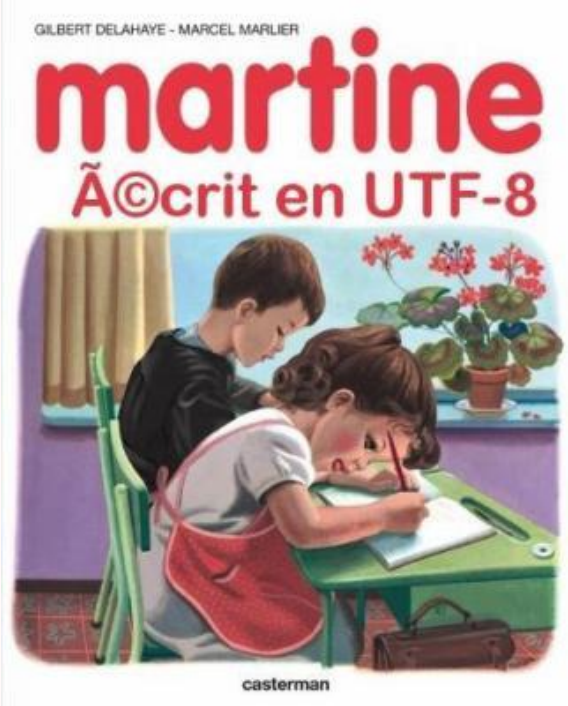
Par exemple :

```
1  In [38]: dix * -treize
2  Out[38]: 1:01111110
3  In [39]: (dix * -treize).to_dec()
4  Out[39]: -130
```

THÈME

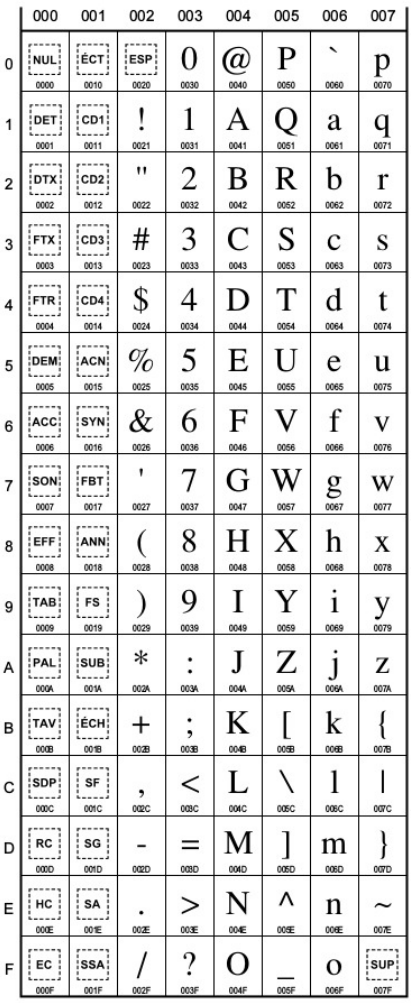
B

Théorie de l'information

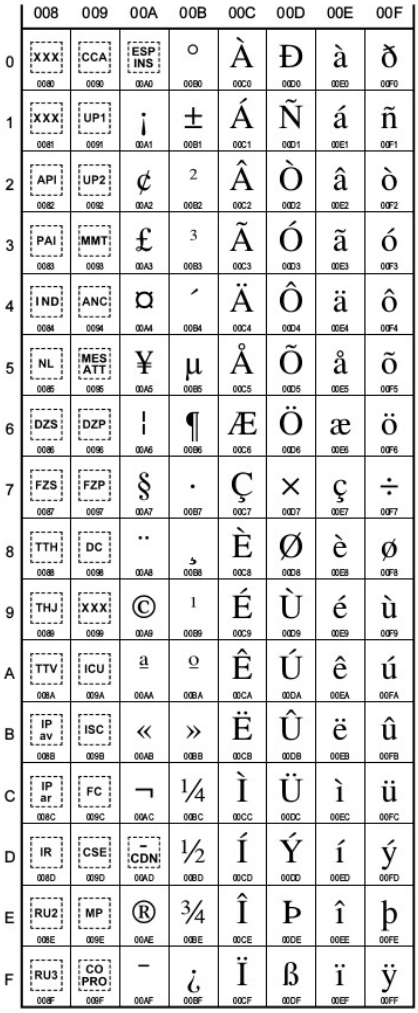


UTF-8

Début des tables de l'Unicode :



	000	001	002	003	004	005	006	007
0	NUL 0000	ÉCT 0010	ESP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	DET 0001	CD1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	DTX 0002	CD2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	FTX 0003	CD3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	FTR 0004	CD4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	DEM 0005	ACN 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACC 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	SON 0007	FBT 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	EFF 0008	ANN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	TAB 0009	FS 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	PAL 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	TAV 000B	ÉCH 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	SDP 000C	SF 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	RC 000D	SG 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	HC 000E	SA 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	EC 000F	SSA 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	SUP 007F



	008	009	00A	00B	00C	00D	00E	00F
0	XXX 0080	CCA 0090	ESP INS 00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
1	XXX 0081	UP1 0091	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	API 0082	UP2 0092	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	PAI 0083	MMT 0093	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	IND 0084	ANC 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	NL 0085	MES ATT 0095	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	DZS 0086	DZP 0096	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	FZS 0087	FZP 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	TTH 0088	DC 0098	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	THJ 0089	XXX 0099	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	TTV 008A	ICU 009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	IP av 008B	ISC 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	IP ar 008C	FC 009C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	IR 008D	CSE 009D	- CDN 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	RU2 008E	MP 009E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
F	RU3 008F	CO PRO 009F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

Fonctionnement de l'UTF-8 :

De	à	1er octet	2ème octet	3ème octet	4ème octet
$0x0$	$0x7F$	$0b_7b_6b_5b_4b_3b_2b_1$	/	/	/
$0x80$	$0x7FF$	$110b_{11}b_{10}b_9b_8b_7$	$10b_6b_5b_4b_3b_2b_1$	/	/
$0x800$	$0xFFFF$	$1110b_{16}b_{15}b_{14}b_{13}$	$10b_{12}b_{11}b_{10}b_9b_8b_7$	$10b_6b_5b_4b_3b_2b_1$	/
$0x10000$	$0x10FFFF$	$11110b_{21}b_{20}b_{19}$	$10b_{18}b_{17}b_{16}b_{15}b_{14}b_{13}$	$10b_{12}b_{11}b_{10}b_9b_8b_7$	$10b_6b_5b_4b_3b_2b_1$

Recherche B - 1 Décodage de texte brut

Vous allez travailler sur le fichier **`texte.txt`** à placer dans votre répertoire courant. Il s'agit d'un fichier rempli de 0 et de 1 encodé en ISO-8859-1. Commencez par décoder à la main le début du texte :

0100000101111010001110100011001010110111001110100011010010110111101101110000100000

0010000100100000010001010110111000100000011001100110000101101001011101000010000

0110001101100101001000000110110101100101011100110111001101100001011001110110010 1

Il va s'agit ensuite de créer un programme Python qui va déchiffrer ce texte automatiquement.
Vous importerez le message brut puis le tranformerez en une longue chaîne de caractères avec :

```
brut = open('texte.txt')
zero_un = brut.read()
```

Il s'agira ensuite de lire cette chaîne par paquets de 8 et de transformer chaque paquet de 8 en caractère ISO-8859-1 correspondant grâce à la fonction **chr**.

```
In [29]: chr(97)
Out[29]: 'a'
```

Vous pourrez construire une fonction **octet2int** :

```
In [27]: octet2int('10001100')
Out[27]: 140
```

Décodez alors **zero_un** et ajustez éventuellement votre programme ;-) Créez ensuite des fonctions qui permettront d'effectuer la démarche inverse.

Littérature et chaînes de Markov

Recherche B - 1

Андрей Андреевич МАРКОВ (1856 - 1922)

Grand mathématicien russe, initiateur des probabilités modernes, grand amateur de poésie, MARKOV n'hésita pourtant pas à s'impliquer dans la tumultueuse histoire de son pays. Il n'eut pas peur d'affirmer publiquement son antitsarisme et fut renvoyé de l'Université. Il fut réhabilité après la Révolution.
Nous allons illustrer de manière élémentaire quelques résultats de ses travaux sur les processus aléatoires (ou stochastiques), c'est-à-dire l'évolution dans le temps d'une variable aléatoire appelée *chaînes de* MARKOV dont les applications sont très nombreuses, notamment en informatique (reconnaissance de la langue d'un texte, techniques de compression, l'algorithme PageRank de Google, files d'attente, bioinformatique,...)
D'un point de vue plus probabiliste, les chaînes de MARKOV permettent d'étudier des variables aléatoires non indépendantes ce qui est très délicat à faire habituellement.
La particularité des chaînes de MARKOV est de représenter des processus où la probabilité permettant de passer à un instant donné d'un état présent à un état futur ne dépend du passé qu'à travers le présent soit : « la probabilité du futur sachant présent et passé est égale à la probabilité du futur sachant le présent ».
On peut produire des textes (ou de la musique...) aléatoirement à l'aide de chaînes de Markov. Par exemple, connaissant un couple de lettres consécutives, on peut calculer la probabilité qu'il soit suivi par telle lettre en faisant un calcul de fréquence sur un très grand texte.
Nous allon scréer un *dictionnaire* dont les clés seront des n-uplets de longueur arbitraire et les valeurs la liste des lettres (ou des mots) qui suivent ce n-uplet dans le texte.
Soit par exemple le texte :

Maman les petits bâteaux qui vont sur l'eau ont-ils des jambes ?

Nous allons créer un dictionnaire :

```
{
('Maman', 'les') : ['petits'],
('les', 'petts') : ['bâteaux'],
('petits', 'bâteaux') : ['qui'],
...
}
```

ou bien :

```
{
('M', 'a') : ['m'],
('a', 'm') : ['a', 'b'],
...
}
```

Ensuite, nous allons partir d'un n-uplet choisi aléatoirement puis construire un texte en prenant un nouveau caractère (mot) dans la liste des successeurs de la clé puis prendre comme nouvelle clé la clé précédente sans son premier terme et avec comme nouveau dernier terme le caractère (mot) choisi aléatoirement.
Par exemple : je tire c_1c_2. Dans les successeurs de la paire c_1c_2 je choisis aléatoirement c_3. Je mets c_1 dans ma châine et je prends comme nouvelle clé c_2c_3, etc.
Nous auront besoin de méthodes sur les chaînes de caractères :

```
In [1]: ??str.join
Docstring:
S.join(iterable) -> str

Return a string which is the concatenation of the strings in the
iterable.  The separator between elements is S.

In [2]: ??str.split
Docstring:
S.split(sep=None, maxsplit=-1) -> list of strings

Return a list of the words in S, using sep as the
delimiter string.  If maxsplit is given, at most maxsplit
splits are done. If sep is not specified or is None, any
whitespace string is a separator and empty strings are
removed from the result.
```

Par exemple :

```
In [9]: ''.join(['B','o','n','j','o','u','r'])
Out[9]: 'Bonjour'

In [10]: '/'.join(['B','o','n','j','o','u','r'])
Out[10]: 'B/o/n/j/o/u/r'

In [11]: "Bonjour le monde".split(' ')
Out[11]: ['Bonjour', 'le', 'monde']

In [12]: "Bonjour le monde".split('o')
Out[12]: ['B', 'nj', 'ur le m', 'nde']
```

On utilisera aussi la fonction **open** qui permet d'ouvrir un fichier sous forme d'une liste de caractères.
On utilisera enfin la fonction **numpy.randint(a)** qui renvoie un nombre entier naturel strcitement inférieur à a.
Vous pourrez créer :

— une fonction **n_uplets(liste, long_t)** qui revoie la liste de tous les n-uplets extraits de la liste.

— une fonction **repertoire(liste, long_t)** qui renvoie le dictionnaire **n-uplet : liste des suivants**.

— une fonction **`n-uplet : liste des suivants`** qui choisit aléatoirement selon une loi uniforme un élément d'une liste.

— une fonction **`clés(dico)`** qui renvoie la liste des clés d'un dictionnaire.

— enfin la fonction principale **`genere_Markov(fichier, atomes, long_t, nb_mots = 100)`** qui renvoie le texte généré, sachant que **`atomes`** permet de choisir les lettres ou les mots comme unité de génération.

Vous récupèrerez les fichiers **`txt`** pour travailler en vraie grandeur.

```
In [27]: genere_Markov('verne.txt', mots, 2)
Londres dans
plusieurs constructions du soir, le témoignaient de la dent du Jardin des ovules,
des volutes, des rhomboldes, des fourmis qui peuplent notre
planète, ou sept
mille cinq cent quarante brasses. Sous cette portion de Conseil.
 -- Remonter à demi pourris, des eaux ; elles battent ?
 -- Et le même pas été éteint, et avec lui :
 Â Ami Ned, il marchait à
une chimère, mais la venaison, dont il
ne restait de neuf cents mètres.
 Â Et c'est un trente-millième pour un mouvement à voix ne nous
écoute pas cela seul bruit des légumineuses et d'une ouverture qui s'est produit. J'ai beaucoup
hésité.

In [29]: genere_Markov('verne.txt', lettres, 4, 300)
Polypiers les battirière les jourserver, qu'il vivant libergé d'un sur déplat du
capitait le sait sa côtel Grahambé, formaladessaillo, frois, qui peut-être deux. Je n'en qu'il de capide
réservi.
 Â Calés de coché sont ce divisir
soures sent. Malaité.
 -- Aloré, foi, retent, je vites noire cachera
```

Cryptographie : chiffrement par blocs

Recherche B - 1

Cryptosystème

Commençons par une définition :

Définition B - 1

Cryptosystème (ou système de chiffrement ou chiffre)
Un cryptosystème est un quintuplet $\langle P, C, K, E, D\rangle$ tel que :

— L'ensemble P est l'espace des messages en clair (*plaintext* en anglais). En général, c'est l'ensemble des chaînes de bits.

— L'ensemble C est l'espace des messages chiffrés (*cyphertext* en anglais). En général, c'est aussi l'ensemble des chaînes de bits.

— L'ensemble K est l'espace des clés (*key*) ;

— E est la famille des fonctions de chiffrement, qui vont de P dans C ;

— D est la famille des fonctions de déchiffrement, qui vont de C dans P ;

Pour chaque élément $e \in K$, il existe un élément $d \in K$ tel que, pour tout message clair m de P, il existe $D_d \in D$ et $E_e \in E$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

1. On travaille sur des mots formés des 26 minuscules non accentuées de notre alphabet latin et d'une espace qui sont modélisés par les entiers de $E = \{0, 1, 2, \ldots, 26\}$.
Dans ce contexte, on remplace une « lettre » x par $kx[27]$ avec $k \in E$. Définit-on ainsi un cryptosystème ?

2. Un cryptosystème est un *chiffrement par blocs* si $P = C = \{0,1\}^n$. Est-ce que le chiffrement ROT-13 est un chiffrement par blocs ?

3. Pourquoi les fonctions de chiffrement d'un chiffrement par blocs sont des permutations ?

Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits. Une présentation de l'algorithme de chiffrement associé, RIJNDAEL, et de sa résistance aux attaques est présentée ici : http://www.cryptis.fr/assets/files/Canteaut-25ans-Cryptis.pdf.

Classe de permutations

Nous allons avoir besoin de travailler avec des permutations et plus précisément dans le groupe $(\mathfrak{S}_n, \circ)$.
Une permutation sera définie par la liste de ses images. Par exemple, on veut définir

$$p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 2 & 0 & 1 & 4 & 3 \end{pmatrix}$$

par

```
p = Perm([2, 0, 1, 4, 3])
```

Pour cela nous utiliserons la classe suivante dont on donne un squelette qu'il va falloir compléter :

```
class Perm :
    """ Groupe des permutations, la loi étant la composition """

    def __init__(self, images = None) :
        """ Permutation définie par son ensemble image """
        self.images = images

    def __repr__(self) :
        return (' ').join( [str(x) for x in self.images] )

    def __add__(self, other) :
        """ fabrique la composée self o other """
        return Perm( ??????? )

    def __neg__(self) :
        """ réciproque de self """
        im = self.images
        n = len(im)
        return Perm( ????? )

    def circulaire(self, rangs) :
        """ permutation circulaire de rangs rangs """
        im = self.images
        n = len(im)
        return Perm( ?????? )

    def permute(self,xs) :
        """ si xs = [xs[0], xs[1], xs[2],...], p.permute(xs) = [xs[p(0)], xs[p(1)], xs[p(2)],...] """
        return ??????
```

On aura peut-être besoin de créer une fonction qui trie les couples de nombres, une fonction qui crée la liste de couples $[(x_0, y_0), (x_1, y_1), \dots]$ à partir de $[x_0, x_1, \dots]$ et de $[y_0, y_1, \dots]$.

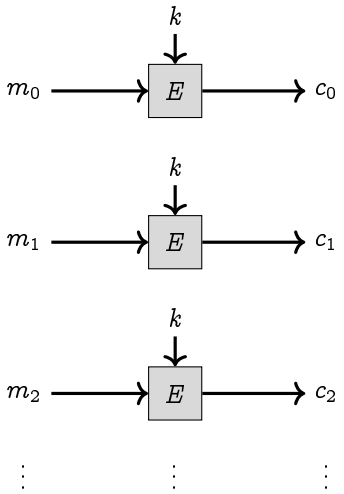
Mode ECB

ECB : *Electronic CodeBook*. C'est le plus simple...et le plus vulnérable des modes de chiffrement par blocs.
Voyons sur un exemple. On considère une chaîne de bits quelconque que l'on découpe en blocs de longueur fixe, par exemple 7 (plus pratique pour ensuite utiliser l'ASCII : pourquoi ?). On rajoute éventuellement des zéros en bout de chaîne.
Considérons $m = 1001001111011$. On rajoute un 0 en bout de chaîne :

$$m' = 1001001\ 1110110 = \beta_1\beta_2$$

avec $\beta_1 = 1001001$ et $\beta_2 = 1110110$.

On choisit une clé de chiffrement dans $\mathfrak{S}_7$ car $K = \mathfrak{S}_7$: $\pi = \begin{pmatrix} 2 & 0 & 3 & 1 & 5 & 4 & 6 \end{pmatrix}$
Alors $E_\pi(\beta_1) = 0110001$ et $E_\pi(\beta_2) = 1101110$.



1. Chiffrez « maman » avec ce cryptosystème en transformant la chaine de lettres en chaînes de bits constituée des codes ASCII de ses caractères. Le tableau ASCII standard est-il satisfaisant ? Comment y remédier ?
2. Quelle est la fonction de déchiffrement associée à l'exemple précédent ?
3. Déchiffrez 10110111001111 sachant que la clé de chiffrement est $k = \begin{pmatrix} 1 & 2 & 0 & 6 & 5 & 4 & 3 \end{pmatrix}$
4. Déchiffrez « papa » sachant que c'est le cryptosystème du début qui a été employé.
5. Que se passe-t-il si un bloc chiffré est mal transmis ou perdu ?
6. Quelle est la principale faiblesse du mode ECB ?
7. Il reste à programmer tout ça...
   On aura d'abord besoin de la fonction `reduce` de la bibliothèque **functools** qui est en gros définie comme suit :

```
def reduit(loi, xs, neutre) :
    panier = neutre
    for x in xs :
        panier = loi(panier, x)
    return panier
```

Complétez alors le squelette suivant :

```
def decoupe_en_blocs(n, bs) :
    """découpe une liste en liste de blocs de longueur n"""
    ???????????

def ecb(cle, bs) :
    """renvoie la liste des permutations des blocs contenus dans une liste"""
    ????

def string2bin(s, n) :
    """ transforme un caractère en sa représentation sur n bits sous forme de liste"""
    ?????

def bin2string(bs) :
    """ transforme une liste de bits en l'entier décimal correspondant"""
    ????

def cats(xs) :
    """concatène une liste de listes ou de chaînes"""
    ????

def code_bloc_string(code_bloc, cle, n_bits, chaine) :
    """applique un chiffrement par bloc sur une chaîne en donnant le nom du
       chiffrement, sa clé et le nb de bits sur lequel les caractères sont codés """
    ?????
```

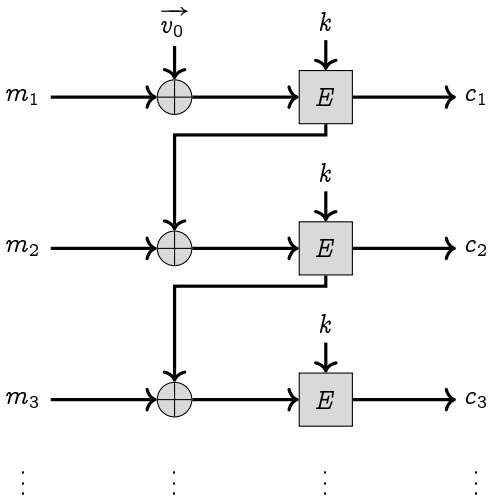
Par exemple :

```
In [5]: bs = [1,0,0,1,0,0,1,1,1,1,0,1,1]

In [6]: decoupe_en_blocs(7, bs)
Out[8]: [[1, 0, 0, 1, 0, 0, 1], [1, 1, 1, 0, 1, 1, 0]]

In [9]: ecb(Perm([2,0,3,1,5,4,6]), bs)
Out[9]: [[0, 1, 1, 0, 0, 0, 1], [1, 1, 0, 1, 1, 1, 0]]

In [10]: string2bin('(', 7)
Out[12]: [0, 1, 0, 1, 0, 0, 0]

In [13]: bin2string([0, 1, 0, 1, 0, 0, 0])
Out[20]: 40

In [21]: cats(['a', 'u', ' ', 's', 'e', 'c', 'o', 'u', 'r', 's'])
Out[21]: 'au secours'

In [22]: cats([[0,1], [2,3]])
Out[22]: [0, 1, 2, 3]

In [23]: code_bloc_string(ecb, Perm([2,0,3,1,5,4,6]), 7, 'Help')
Out[23]: '0+:h'
```

Mode CBC

CBC : *Cipher-Block Chaining*. Ce mode a été inventé par IBM en 1976. Pour éviter la faiblesse de l'ECB, les blocs sont maintenant chaînés : chaque bloc en clair est « XORé » ou « OUEXé » avec le bloc crypté précédent avant d'être crypté lui-même.
Pour le premier bloc, on utilise un *vecteur d'initialisation* public. Avec les notations habituelles, on a donc

$$c_i = E_k(m_i \oplus c_{i-1}), \quad c_0 = \overrightarrow{v_0}$$



1. Quel est le lien entre OUEX et $\mathbb{F}_2$?
2. Cryptez l' exemple traité avec ECB en prenant $\overrightarrow{v_0} = 1010101$.
3. Faites de même avec « Maman ».
4. Donnez une formule de déchiffrement. Déchiffrer le cryptogramme 101101110011111 sachant que la clé est $k = \begin{pmatrix} 1 & 0 & 2 \end{pmatrix}$ et que $\overrightarrow{v_0} = 001$.
5. Déchiffrez « Papa » sachant que c'est le cryptosystème de la question 2. qui a été employé.
6. Un petit programme ?...

Ce mode est efficace mais nécessite l'utilisation de fonctions de chiffrement et de déchiffrement différentes ce qui peut ralentir la procédure.

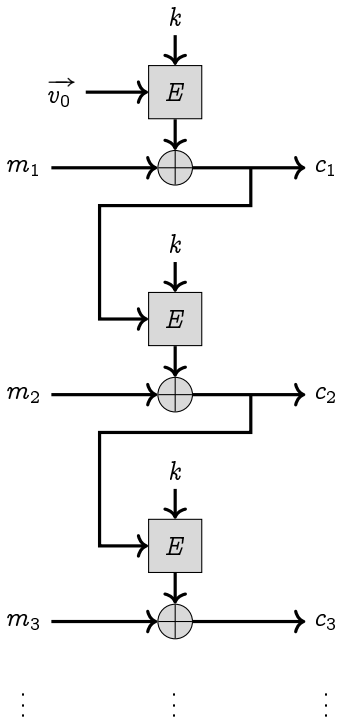
Mode CFB

CFB : *Cipher-FeedBack*. C'est un mode dérivé de CFB qui est utilisé par OpenPGP, format pour l'échange sécurisé de données (paiements sécurisés par exemple) largement utilisé actuellement mais est susceptible d'être attaqué, même s'il est très difficile de mettre en œuvre concrètement cette attaque :

http://www.cert-ist.com/fra/ressources/Publications_ArticlesBulletins/Autres/FailledanslechiffrementCFBdOpenPGP/

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.
On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;
- $s_j = E_k(t_j)$;
- g_j est la chaîne constituée des r bits les plus à gauche de s_j. Quelle est l'opération arithmétique associée ?
- $c_j = m_j \oplus g_j$;
- $t_j = (2^r t_{j-1} + c_{j-1}) \bmod 2^n$ (attention ! Il faut travailler en base 2).



Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape : démontrez-le !

1. Chiffrez « *Papa is cool* » sachant que la clé est $k = \begin{pmatrix} 3 & 0 & 2 & 1 \end{pmatrix}$, que $\overrightarrow{v_0} = 0101$ et que $r = 3$.

2. Déchiffrer le cryptogramme 101101110011111 sachant que la clé est $k = \begin{pmatrix} 1 & 0 & 2 \end{pmatrix}$, que $r = 2$ et que $\overrightarrow{v_0} = 001$.

Recherche B - 2 E3A 2015 - Mathématiques A

Des bits d'information sont transmis par l'intermédiaire d'un canal. Ce canal n'est pas complètement fiable. On observe qu'un bit envoyé peut être altéré en sortie.
On note b le bit envoyé et b' le bit de sortie. Après oservation, on modélise la transmission d'un bit de façon probabiliste :

- le bit envoyé définit une variable aléatoire b : on note α la probabilité qu'un 1 soit envoyé.
- La perturbation dans le canal est aussi modélisée de façon probabiliste :
  - on désigne par p la probabilité qu'un 1 en entrée ne soit pas altéré pendant la transmission ($\mathbb{P}(b' = 1|b = 1) = p$).
  - On désigne par q la probabilité qu'un 0 en entrée ne soit pas altéré pendant la transmission.

1. i. On a écrit ci-dessus $\mathbb{P}(b' = 1|b = 1) = p$. Exprimer de la même manière $1-p$, q et $1-q$ en terme de probabilités conditionnelles.

ii. Un bit est envoyé. Quelle est la probabilité de recevoir un 1 en sortie ?

iii. On reçoit le bit 1. Quelle est la probabilité qu'un 1 ait été envoyé en entrée ?

2. Soit n un entier supérieur ou égal à 2. On décide d'envoyer n fois le même bit b. On note b'_1, b'_2,...,b'_n les n bits obtenus en sortie et on note X la variable aléatoire qui compte le nombre de 1 en sortie.

i. Soit k un entier entre 0 et n. Exprimer $\mathbb{P}(X = k)$ en fonction des paramètres p, q et α.

ii. En déduire l'espérance de X en fonction de ces mêmes paramètres.

iii. Soit k un entier entre 0 et n. Exprimer la probabilité que le bit 1 ait été envoyé sachant que le nombre 1 en sortie vaut k.

3. Le canal est désormais supposé symétrique, i.e. chaque bit peut être altéré avec la même probabilité $1-p$. On suppose $\frac{1}{2} < p < 1$.

i. Déterminer en fonction de p et α l'ensemble des valeurs k prises par X pour lesquelles il est plus probable (au sens strict) qu'un 1 ait été envoyé plutôt qu'un 0 ?

ii. Que devient ce résultat lorsque $\alpha = 1/2$?

4. On suppose $\alpha = 1/2$. On note $f(n)$ la probabilité que l'interprétation de l'observation en sortie soit fausse.

i. Exprimer $f(n)$ en fonction des $\mathbb{P}(X = k)$ pour des entiers k entre 0 et n.

ii. Donner une expression de $f(n)$ en fonction de n et p.

iii. Écrire une fonction **`binome`** en langage Python qui prend en entrée un entier naturel N et un entier naturel k compris entre 0 et N et retourne la valeur du coefficient binomial $\binom{N}{k}$.

iv. On suppose $p = 0,95$. Écrire un programme en langage Python qui prend en entrée l'entier naturel n et donne une estimation de $f(n)$.

THÈME

C

Ingénieur, quel beau métier!

Pour en finir avec les résolutions numériques d'équas diff

Position du problème

Soit U un ouvert de $\mathbb{R}\times\mathbb{R}^n$ et $f:U\longrightarrow\mathbb{R}^n$ une application continue sur l'ouvert U.
Alors pour $(t_0,x_0)\in U$, on appelle solution du problème de Cauchy :

$$\begin{cases}x'(t)=f(t,x(t))\\x(t_0)=x_0\end{cases}\qquad C$$

tout couple (I,x) où I est un intervalle contenant t_0 et $x:I\longrightarrow\mathbb{R}^n$ une solution sur I de l'équation différentielle $x'(t)=f(t,x(t))$ telle que $x(t_0)=x_0$.
L'équation différentielle qui apparaît dans (C) sera appelée équation différentielle scalaire si $n=1$, et système différentiel sinon.
Si l'application $f:U\longrightarrow\mathbb{R}^n$ est de classe C^1 sur l'ouvert U, alors, d'après le théorème de Cauchy-Lipschitz, pour $(t_0,x_0)\in U$, il existe une unique solution maximale au problème de Cauchy (C).
Dans la plupart des cas, on ne sait pas résoudre explicitement le problème de Cauchy (C) ; d'où la nécessité de mettre au point des méthodes numériques de résolution approchée d'un tel problème.

Méthode d'Euler

La méthode d'Euler est la méthode la plus simple pour résoudre numériquement une équation différentielle. Elle présente un réel intérêt théorique, puisque elle peut être utilisée pour démontrer le théorème de Cauchy-Lipschitz. Toutefois, son intérêt pratique est limité par sa faible précision.
L'idée d'Euler consiste à utiliser l'approximation :

$$x'(t)\approx\frac{x(t+h)-x(t)}{h}\qquad\text{pour } h \text{ petit}$$

Autrement dit,

$$x(t+h)\approx x(t)+h\cdot x'(t)=x(t)+h\cdot f(t,x(t))$$

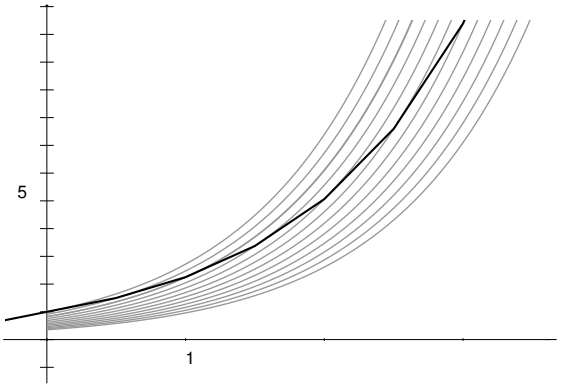
Partant du point (t_0,x_0), on suit alors la droite de pente $f(t_0,x_0)$ sur l'intervalle de temps $[t_0,t_0+h]$.
On pose alors :

$$\begin{cases}t_1=t_0+h\\x_1=x_0+h\cdot f(t_0,x_0)\end{cases}$$

De nouveau, partant du point (t_1,x_1), on suit alors la droite de pente $f(t_1,x_1)$ sur l'intervalle de temps $[t_1,t_1+h]$, et ainsi de suite.
On construit ainsi une suite de points de la manière suivante :

$$\forall k\in[\![0,N]\!],\quad\begin{cases}t_{k+1}=t_k+h\\x_{k+1}=x_k+h\cdot f(t_k,x_k)\end{cases}$$

La ligne brisée joignant les points $\{(t_k,x_k)|k\in[\![0,N]\!]\}$ donnera une solution approchée de notre équation différentielle.
Il faut bien comprendre qu'à chaque étape, on repart dans la direction d'une solution exacte de l'équation différentielle, mais qui n'est pas celle qui est solution du problème de Cauchy initial. Sur le graphique ci-contre, on trace une solution approchée pour la condition initiale $x(0)=1$ et les courbes intégrales de notre équation qui passent par les points (t_k,x_k).



Pour juger de la qualité d'une méthode (ou schéma) numérique de résolution d'équations différentielles, il faut prendre en compte plusieurs critères.

— L'erreur de consistance donne l'ordre de grandeur de l'erreur effectué à chaque pas. Par exemple, dans le cas de la méthode d'Euler, l'erreur de consistance mesure l'erreur qu'entraîne le fait d'approcher le nombre dérivé par un taux d'accroissement. La sommation des erreurs de consistance à chaque pas donnera l'ordre de grandeur de l'erreur globale (sous des hypothèses de régularité pour la fonction f). À l'aide de la formule de Taylor-Lagrange, on peut montrer que dans le cas de la méthode d'Euler, l'erreur de consistance est dominée par h^2 ; on dit alors que cette méthode est d'ordre 1. Ainsi, d'un point de vue théorique, plus le pas est petit, meilleure sera l'approximation.

Un schéma numérique est dit consistant si la somme des erreurs de consistance tend vers 0 quand le pas h tend vers 0 ; ce qui est le cas de la méthode d'Euler (en gros, car $Nh^2 \sim h \to 0$ où N désigne le nombre d'itérations).

— La stabilité contrôle la différence entre deux solutions approchées correspondant à deux conditions initiales voisines : un schéma est dit stable si un petit écart entre les conditions initiales $x(t_0) = x_0$ et $\tilde{x}(t_0) = x_0 + \varepsilon$ et de petites erreurs d'arrondi dans le calcul récurrent des $\tilde{x}_k$ provoquent une erreur finale $\left|x_k - \tilde{x}_k\right|$ contrôlable. La méthode d'Euler est une méthode stable.

— Un schéma est dit convergent lorsque l'erreur globale (qui est le maximum des écarts entre la solution exacte et la solution approchée) tend vers 0 lorsque le pas tend vers 0. En fait, pour que le schéma soit convergent, il suffit que la méthode soit consistante et stable.

— Enfin pour la mise en application du schéma, il faut aussi prendre en compte l'influence des erreurs d'arrondi. En effet, afin de minimiser l'erreur globale théorique, on pourrait être tenté d'appliquer la méthode d'Euler avec un pas très petit, par exemple de l'ordre de 10^{-16}, mais ce faisant, outre que le temps de calcul deviendrait irréaliste, les erreurs d'arrondi feraient diverger la solution approchée très rapidement, puisque pour des flottants Python de l'ordre de 10^{-16}, les calculs ne sont plus du tout exacts !

En pratique, il faut prendre h assez petit (pour que la méthode converge assez rapidement du point de vue théorique), mais pas trop petit non plus (pour que les erreurs d'arrondis ne donnent pas lieu à des résultats incohérents, et que les calculs puissent être effectués en un temps fini). La question de trouver de manière théorique le pas optimal peut s'avérer un problème épineux. Dans un premier temps, on peut se contenter de faire des tests pratiques comme dans l'exemple suivant.

Dans le script suivant, on applique la méthode d'Euler sur l'intervalle $[0, 1]$ avec la condition initiale $x(0) = 1$.

```
from math import exp


print("{:^10} | {:^12} | {:^12} | {:^13}".format('h','x(t)','exp(t)','erreur'))
print('-'*57)
for i in range(1, 8):
    h, t, x  = 10**(-i), 0, 1
    while t < 1:
        t, x = t + h, x * (h + 1)
    print("{0:>10g} | {1:>.10f} | {2:>.10f} | {3:> .10f}"
            .format(h, x, exp(t),  exp(t) - x))
```

```
    h      |     x(t)     |    exp(t)    |    erreur
---------------------------------------------------------
       0.1 | 2.8531167061 | 3.0041660239 |  0.1510493178
      0.01 | 2.7048138294 | 2.7182818285 |  0.0134679990
     0.001 | 2.7169239322 | 2.7182818285 |  0.0013578962
    0.0001 | 2.7184177414 | 2.7185536702 |  0.0001359288
     1e-05 | 2.7182954199 | 2.7183090114 |  0.0000135915
     1e-06 | 2.7182804691 | 2.7182818285 |  0.0000013594
     1e-07 | 2.7182819660 | 2.7182820996 |  0.0000001336
     1e-08 | 2.7182817983 | 2.7182818347 |  0.0000000363
     1e-09 | 2.7182820738 | 2.7182818299 | -0.0000002438
```

Pour un pas $h \lesssim 10^{-9}$, le temps de calcul devient trop important, de plus, on voit poindre l'effet des erreurs d'arrondis, puisque la solution approchée est devenu supérieure à l'exponentielle, ce qui est mathématiquement faux, en vertu de l'inégalité :

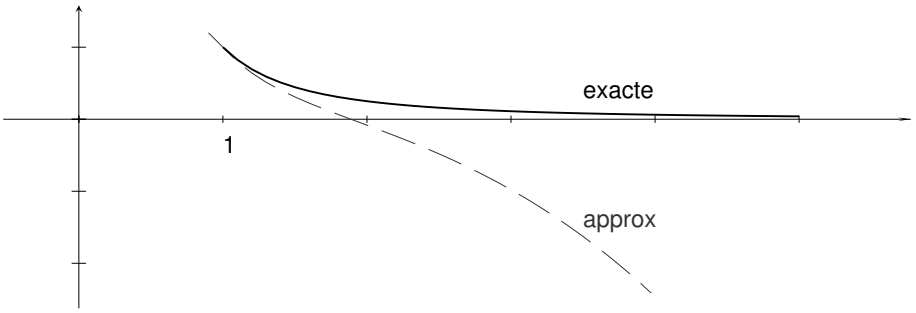
$$\forall n \in \mathbb{N}, \quad \left(1 + \frac{1}{n}\right)^n \leq e$$

En général, il ne suffit pas qu'un schéma numérique soit convergent pour qu'il donne de bons résultats sur n'importe quelle équation différentielle. Encore faut-il que le problème soit mathématiquement bien posé (en particulier, les hypothèses du théorème de Cauchy-Lipschitz doivent être vérifiées), qu'il soit numériquement bien posé (continuité suffisamment bonne par rapport aux conditions initiales) et qu'il soit bien conditionné (temps de calcul raisonnable).

Considérons par exemple le problème de CAUCHY :

$$\begin{cases} x(1) = 1 \\ x'(t) = 3\frac{x(t)}{t} - \frac{5}{t^3} \end{cases}$$

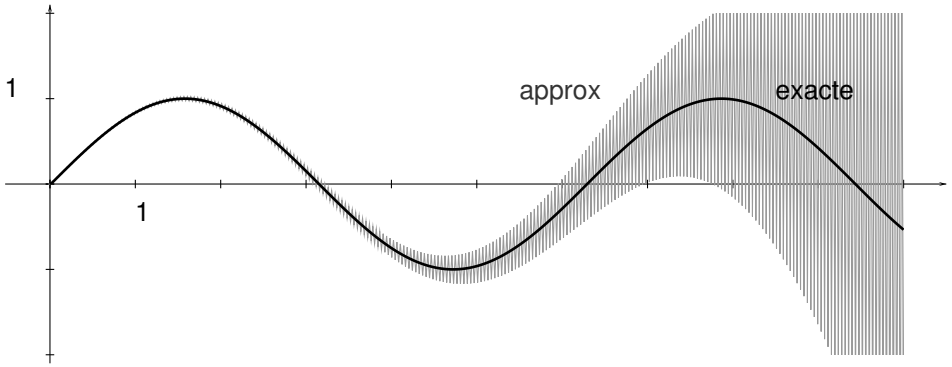
dont la solution est la fonction ? ? ?



On constate qu'ici la solution approchée s'écarte assez rapidement de la solution exacte φ. En effet, la forme générale des solutions de l'équation différentielle précédente est donnée par $t \mapsto \lambda x^3 + 1/x^2$. La condition $x(1) = 1$ impose $\lambda = 0$, mais dès qu'on s'écarte de la solution φ on suit « tangentiellement » des courbes intégrales qui comportent un terme en λx^3 et donc qui diffère notablement de la solution. Donc le problème est mal posé. Ici le schéma numérique n'est pas en cause.
Soit à résoudre à présent le problème de CAUCHY :

$$\begin{cases} x(1) = 1 \\ x'(t) = 100\,(\sin(t) - x) \end{cases}$$

dont la solution est la fonction $\varphi : t \mapsto \frac{1}{10001}(-100\cos t + 10000\sin t + 100\exp(-100\cdot t)) \approx \sin t$. Que se passe-t-il si on résout ce problème avec un pas de l'ordre de 0.02 ?



Avec un tel pas, la solution approchée oscille en s'éloignant de plus en plus de la solution exacte. En effet, le schéma numérique est donné par :

$$x_{k+1} = x_k + 100\,h\,(\sin t_k - x_k) = (1 - 100h)x_k + 100h\sin t_k$$

Donc une erreur de ε_k sur x_k aura les répercussions suivantes sur le calcul des termes ultérieurs :

$$\varepsilon_{k+1} = (1 - 100h)\varepsilon_k \implies \varepsilon_{k+n} = (1 - 100h)^n \varepsilon_k$$

Ainsi donc, tant que $1 - 100h \geq -1$, c'est-à-dire tant que $h \lesssim 0.002$, une petite erreur sur l'un des termes aura des répercussions allant en s'amplifiant.
Bien que le problème soit numériquement bien posé, il est en fait mal conditionné.

Méthode de Runge-Kutta d'ordre 4

Si, comme nous l'avons déjà dit, la méthode d'EULER présente un intérêt théorique, on préfère en pratique des méthodes d'ordre plus élevé. Parmi la multitude des schémas numériques (méthodes à un pas comme celle de TAYLOR, méthodes à pas multiples comme celles d'ADAMS-BASHFORTH, d'ADAMS-MOULTON, de prédiction-correction) celle qui présente le meilleur rapport précision/complexité est certainement celle de RUNGE-KUTTA d'ordre 4.
En voici le schéma pour $\forall k \in [\![0, N]\!]$,

$$\begin{cases} t_{k+1} &= t_k + h \\ x_{k+1} &= x_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases} \quad \text{où} \quad \begin{cases} k_1 &= h \cdot f(t_n, x_n) \\ k_2 &= h \cdot f\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}\right) \\ k_3 &= h \cdot f\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}\right) \\ k_4 &= h \cdot f(t_n + h, x_n + k_3) \end{cases}$$

Les coefficients qui apparaissent dans ces formules mystérieuses sont judicieusement ajustés pour obtenir une méthode d'ordre 4, sans pour autant avoir à calculer les dérivées successives de f (comme c'est le cas dans les méthodes de TAYLOR), ou à recourir à une formule de récurrence d'ordre au moins 2 pour définir x_k (comme c'est le cas dans les méthodes à pas multiples).
Définissez une fonction adaptée à ce schéma. Vous devriez observer pour l'exponentielle

$$\begin{cases} x'(t) = x(t) \\ x(0) = 1 \end{cases}$$

C

```
    h      |   erreur euler    |    erreur rk4
-----------------------------------------------------
       0.1 |  1.5104931784e-01 |  2.5338874514e-06
      0.01 |  1.3467999038e-02 |  2.2464341498e-10
     0.001 |  1.3578962232e-03 |  2.2204460493e-14
    0.0001 |  1.3592881559e-04 | -2.6645352591e-13
      1e-05 |  1.3591551171e-05 | -5.2180482157e-12
      1e-06 |  1.3591611072e-06 |  2.1464163780e-11
```

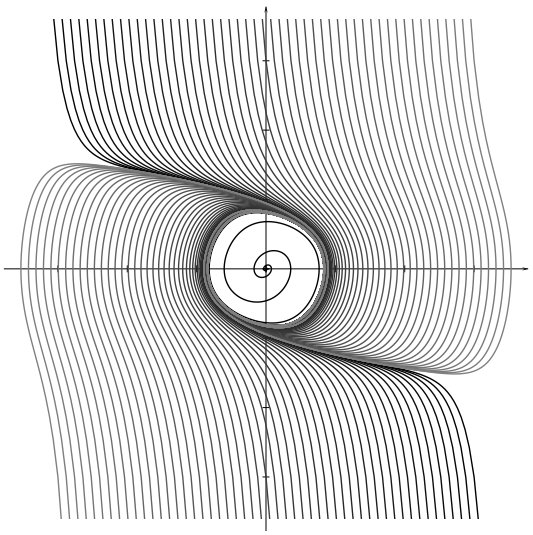
On constate que pour $h = 0.001$, la méthode d'EULER ne donne que trois décimales significatives du nombre e, alors que la méthode de RUNGE-KUTTA en donne quatorze ! On remarque également qu'il est inutile d'appliquer la méthode de RUNGE-KUTTA avec un pas $h \gtrsim 0.001$, puisque dans ce cas les erreurs d'arrondi prennent le dessus par rapport au gain théorique de précision.

Système

Soit à représenter les courbes intégrales du système autonome suivant, appelé oscillateur de van der Pol :

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}t} = y \\ \frac{\mathrm{d}y}{\mathrm{d}t} = \frac{y}{2} - x - y^3 \end{cases}$$

Les variables x, $k1$,... $k4$ représentent à présent des listes. Or une instruction comme $x + k1/2$ renverra un message d'erreur. En effet, la division d'une liste par un entier n'est pas définie ; de plus, l'addition de deux listes effectue la concaténation des listes, et non l'addition des listes composante par composante.
Il faut alors travailler non plus avec des listes, mais avec des objets supportant la vectorisation, comme c'est le cas des tableaux de la librairie de tierce partie **NumPy** ; en outre, cette option a l'avantage de réduire de manière significative les temps de calcul.



Un exemple d'équation différentielle linéaire d'ordre 2

On rappelle qu'une équation différentielle linéaire d'ordre n peut toujours s'écrire comme un système différentiel d'ordre 1. Illustrons ceci dans le cas $n = 2$: en notant :

$$A(t) = \begin{pmatrix} 0 & 1 \\ -b(t) & -a(t) \end{pmatrix}, \quad X(t) = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}, \quad C(t) = \begin{pmatrix} 0 \\ c(t) \end{pmatrix}$$

on a les équivalences

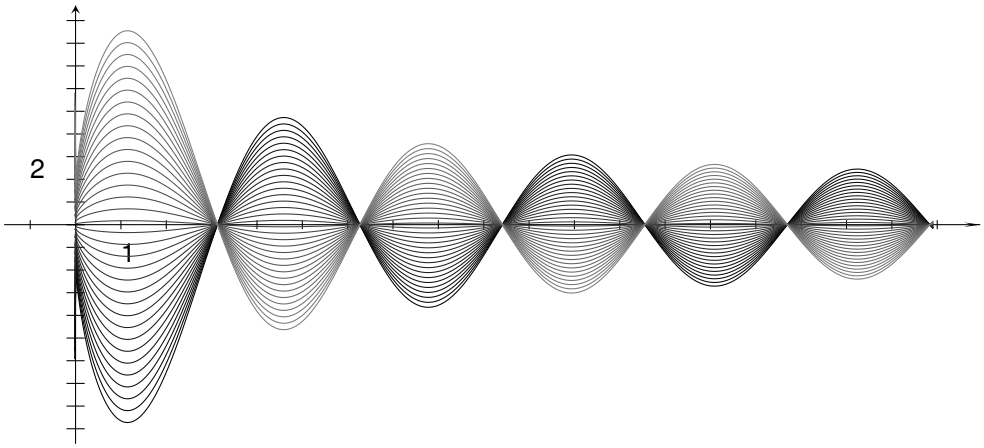
$$\forall t\in I \quad x''+a(t)x'+b(t)x=c(t) \iff \forall t\in I \quad \begin{cases} x'=x' \\ x''=-bx-ax'+c \end{cases}$$

$$\iff \forall t\in I \quad \frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ x' \end{pmatrix}=\begin{pmatrix} 0 & 1 \\ -b & -a \end{pmatrix}\cdot\begin{pmatrix} x \\ x' \end{pmatrix}+\begin{pmatrix} 0 \\ c \end{pmatrix}$$

$$\iff \forall t\in I \quad X'=A(t)\cdot X+C(t)$$

Considérons, par exemple, l'équation différentielle du 2e ordre dite de Bessel :

$$t^2x''(t)+tx'(t)+\left(t^2-\alpha^2\right)x(t)=0 \iff X'(t)=A(t)\cdot X(t) \quad \text{avec} \quad A(t)=\begin{pmatrix} 0 & 1 \\ \frac{\alpha^2-t^2}{t^2} & -\frac{1}{t} \end{pmatrix}$$



Lien avec les méthodes de quadrature

Version objet :

```python
class Integration(object):
    def __init__(self, a, b, n):
        self.a, self.b, self.n = a, b, n
        self.poids_pivots = self.quadrature()

    def quadrature(self):
        raise NotImplementedError

    def integrale(self, f):
        return sum(w * f(x) for w, x in self.poids_pivots)

class Rectangles(Integration):
    def quadrature(self):
        a, b, n = self.a, self.b, self.n
        h = (b-a) / n
        return [(h, a + i*h) for i in range(n)]

class Trapezes(Integration):
    def quadrature(self):
        a, b, n = self.a, self.b, self.n
        h = (b-a) / n
        return [(h, a + (i+0.5)*h) for i in range(n)]

class Simpson(Integration):
    def coeff(self, i, N):
        if i == 0 or i == N:
            return 1
        elif i % 2 == 1:
            return 4
        else:
```

```
            return 2

    def quadrature(self):
        a, b, n = self.a, self.b, self.n
        if n % 2 != 1:
            n += 1
        h = (b-a) * 0.5 / n
        return [(h/3*self.coeff(i, 2*n), a + i*h) for i in range(2*n+1)]

class Romberg(object):
    def __init__(self, a, b, m):
        self.a, self.b, self.m = a, b, m

    def integrale(self, f):
        a, b, m = self.a, self.b, self.m
        A = [[(b-a) * (f(a)+f(b)) * 0.5]]
        for k in range(1, m+1):
            h = (b-a) / 2**k
            Ap = h * sum([f(a + j*h) for j in range(1, 2**k, 2)])
            A.append([0.5 * A[k-1][0] + Ap])
        for t in range(1, m+1):
            for k in range(t, m+1):
                A[k].append((4**t * A[k][t-1] - A[k-1][t-1]) / (4**t-1))
        return A[m][m]
```

Représenter la spirale de Cornu définie par

$$x(t) = \int_0^t \cos\left(u^2\right)\,\mathrm{d}u \quad \text{et} \quad y(t) = \int_0^t \sin\left(u^2\right)\,\mathrm{d}u$$

Recherches

Recherche C - 1 Lynx et lapins

Alfred Lokta
US (1880-1949)

Vito Volterra
It. (1860-1940)

Vous connaissez ce problème par cœur...

$$\begin{cases} \dfrac{\mathrm{d}x(t)}{\mathrm{d}t} &= x(t)\left(\alpha - \beta y(t)\right) \\ \dfrac{\mathrm{d}y(t)}{\mathrm{d}t} &= -y(t)\left(\gamma - \delta x(t)\right) \end{cases}$$

— t est le temps ;

— $x(t)$ est l'effectif des proies en fonction du temps ;

— $y(t)$ est l'effectif des prédateurs en fonction du temps ;

— les dérivées $\mathrm{d}x(t)/\mathrm{d}t$ et $\mathrm{d}y(t)/\mathrm{d}t$ représentent la variation des populations au cours du temps.

Les paramètres suivants caractérisent les interactions entre les deux espèces :

— α, taux de reproduction des proies (constant, indépendant du nombre de prédateurs) ;

— β, taux de mortalité des proies dû aux prédateurs rencontrés ;

— γ, taux de mortalité des prédateurs (constant, indépendant du nombre de proies) ;

— δ, taux de reproduction des prédateurs en fonction des proies rencontrées et mangées ;

Vous connaissez la méthode d'Euler par cœur...
Tracez x et y en fonction de t puis y en fonction de x.

Recherche C - 2 Deux corps dans l'espace...

Bla bla Principe Fondamental de la Dynamique bla bla bla :

$$r'' - r\theta'^2 = -\frac{GMt}{r^2} + \frac{Ar}{M_0 - At} \qquad r\theta'' + 2r'\theta' = \frac{Ar\theta'}{M_0 - At}$$

Mais comment faire !? Mais comment appliquer la méthode d'Euler ? Y a deux inconnues et en plus y a des dérivées secondes...
On pose $u_1 = r$, $u_2 = r'$, $u_3 = \theta$, $u_4 = \theta'$ et tout s'arrange.

Recherche C - 3 Centrale 2015 avec une solution exacte du problème des 3 corps

Sur une idée de Philippe Roux
Nous allons terminer la résolution du problème des N corps de Centrale 2015 avec $N = 3$ et les conditions initiales suivantes :

$$x_1 = -x_2 = 0.97000436 \; y_1 = -y_2 = -0.24308753, \; x3 = y_3 = 0$$

$$vx_3 = -2vx_1 = -2vx_2 = -0.93240737 \; vy_3 = -2vy_1 = -2vy_2 = -0.86473146$$

avec des unités telles que les masses et la constante G valent 1.
On rappelle que la PFD donne

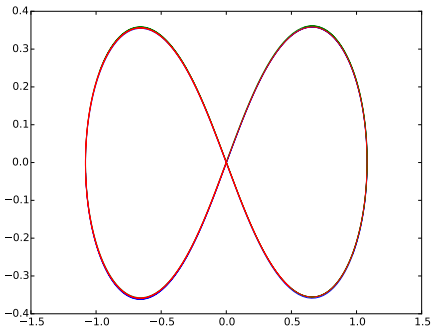
$$\vec{F}_{k/j} = G\frac{m_j m_k}{r_{jk}^3}\overrightarrow{P_jP_k}$$

avec r_{ij} la distance entre les corps j et k.
On utilisera des tableaux de **numpy** pour effectuer des opérations vectorielles.

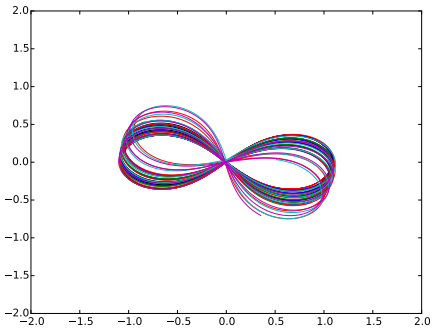
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

G = 1
m = 1
position = np.array([[0.97000436,-0.24308753],[-0.97000436,0.24308753],[0,0]])
#position2 = np.array([[0.99000436,-0.24308753],[-0.97000436,0.24308753],[0,0]])
vitesse = np.array([[ 0.46620368,  0.43236573],[ 0.46620368,  0.43236573],[-0.93240737,-0.86473146]])

def norme(v) :
    """ norme d'un vecteur"""
    return

def force2(m1,p1, m2, p2) :
    """ force exercée par le corps 2 sur le corps 1"""
    p1p2 =
    return

def force(j,m,pos) :
    """ somme des forces exercées sur le corps j
        on pourra utiliser np.sum(array,axis=0)"""
    return

def gravit(pos) :
    """renvoie le vecteur dont les composantes sont les accélérations de chaque corps pour une position
        ↪ donnée """
    return

def verlet(f, pos0, vit0, h, n) :
```

```
30      """ on utilise le schéma de Verlet vu en cours vectorialisé"""
31
32  def phase(f, pos0, vit0, h, n, no) :
33      """ trace les mouvements des trois corps"""
```

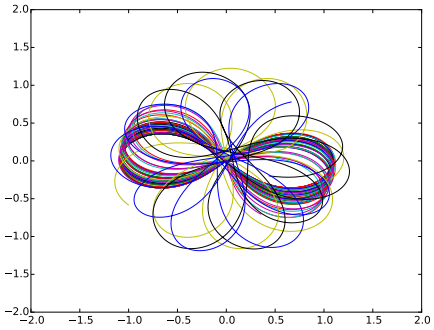
On obtient une courbe fermée :



qui ne l'est plus si une valeur initiale est légèrement perturbée (1%) :



et si une valeur initiale est un peu plus perturbée (10%) :



Recherche C - 4 CCP MP : sujet 0 de SI 2015

Après 25 questions, voici l'éternelle, l'inévitable question sur la méthode d'Euler...
Nous sommes donc à la page 14 du sujet :-) On introduit l'inévitable équa diff.
[On obtient] l'équation différentielle suivante, liant la vitesse de rotation du moteur $\omega_m(t)$ à la vitesse de consigne $\omega_{\text{const}}(t)$:

$$\frac{1}{\omega_0^2}\frac{\mathrm{d}^2\omega_m(t)}{\mathrm{d}t^2}+\frac{2m}{\omega_0}\frac{\mathrm{d}\omega_m(t)}{\mathrm{d}t}+\omega_m(t)=K\omega_{\text{const}}(t)$$

Dans la suite, on s'intéressera uniquement à la réponse indicielle unitaire du système dans les conditions de Heaviside, ce qui revient à prendre $\omega_{\text{const}}(t)=u(t)$ avec $\omega_m(0)=0$ et $\omega_m'(0)=0$.

En posant le vecteur $\boldsymbol{Y}$ tel que $\boldsymbol{Y}(t)=\begin{pmatrix}\omega_m(t)\\ \frac{\mathrm{d}\omega_m(t)}{\mathrm{d}t}\end{pmatrix}$, l'équation différentielle à résoudre peut se mettre sous la forme :

$$\frac{\mathrm{d}\boldsymbol{Y}(t)}{\mathrm{d}t} = \boldsymbol{F}(t,\boldsymbol{Y}(t)) \quad \text{avec} \quad \boldsymbol{F}(t,\boldsymbol{Y}(t)) = \begin{pmatrix} \frac{\mathrm{d}\omega_m(t)}{\mathrm{d}t} \\ K\omega_0^2 - \omega_0^2\omega_m(t) - 2m\omega_0\frac{\mathrm{d}\omega_m(t)}{\mathrm{d}t} \end{pmatrix}$$

La réponse $\omega_m(t)$ recherchée sur l'intervalle $[0, T_{\max}]$ sera obtenur par...la méthode d'Euler explicite ;-)
Le pas de temps, noté pas, sera choisi constant. L'intervalle de temps discrétisé est alors représenté par le tableau $T = [t_0 = 0, t_1, \ldots, t_{N-1} = T_{\max}]$.
Pour chaque pas de temps, une valeur approchée $\boldsymbol{Y}_i$ de la solution $\boldsymbol{Y}(t_i)$ de l'équation différentielle est recherchée. L'ensemble des $\boldsymbol{Y}_i$ représente N vecteurs de dimension 2, qui seront stockés en mémoire sous la forme du tableau :

$$\boldsymbol{SY} = \begin{pmatrix} \omega_m(0) & \omega'_m(0) \\ \omega_m(t_1) & \omega'_m(t_1) \\ \vdots & \vdots \\ \omega_m(T_{\max}) & \omega'_m(T_{\max}) \end{pmatrix}$$

Et voici enfin la première question...

1. Écrire un algorithme ou une fonction permettant de calculer le tableau T.
2. Écrire une fonction f1(ti,yi) qui prend en arguments la valeur du temps discrétisé i et la valeur du vecteur $\boldsymbol{Y}$ au temps discrétisé ti et qui retourne la valeur de $\boldsymbol{F}(t,\boldsymbol{Y}(t))$.
3. Donner la relation de récurrence qui lie $\boldsymbol{Y}_{i+1}$ à $\boldsymbol{Y}_i$ et à $\boldsymbol{F}(t,\boldsymbol{Y}(t))$ en fonction du pas de temps pas.
4. Écrire une fonction EulerExplicite(Yini,h,Tmax,F) qui prend en arguments Yini, un tableau de dimension 2 contenant la condition initiale de $\boldsymbol{Y}(t)$, h le pas de temps, Tmax l'instant final du calcul, et F la fonction du problème de Cauchy.

   Cette fonction renverra le tableau SY. L'appel à cette fonction dans le programme se fera avec la commande SY = EulerExplicite(Y0,pas,tmax,f1).
5. Si le pas de temps es divisé par un facteur 10, comment évolue l'erreur de calcul ?
6. Donner la complexité de cette méthode pour $T_{\max}$ fixé et indiquer comment évolue le temps de calcul quand le pas de temps est divisé par un facteur 10.

   On suppose que la quantité de mémoire nécessaire pour réaliser le calcul se limite au stockage de la matrice **SY** et du vecteur **T**. Ces éléments sont représentés en mémoire sous forme de tableaux de flottants en double précision.
7. Déterminer le nombre d'octets nécessaire en mémoire pour réaliser cette simulation numérique avec nombre de pas de temps $N = 10\,000$.
8. **Convergence de l'algorithme**. Avant d'appliquer l'algorithme de recherche du temps de réponse à 5%, il est nécessaire de savoir si l'algorithme de simulation converge. Après calcul, la solution ω_m est stockée dans la variable **W**.

   On suppose que la solution converge si toutes les valeurs de ω_m pour t appartenant à $[0.9T_{\max}, T_{\max}]$ ne s'écartent pas de la valeur finale calculée, $\omega_m(T_{\max})$, de plus de $0,1\%$.

   Écrire une fonction TestConvergence(t,w) qui prend en argument le tableau des temps t et la solution w et qui renvoie True si le critère est vérifié et False sinon.
9. **Détermination du temps de réponse à 5% du système**

   Écrire une fonction CalculT5 qui renvoie le temps de réponse à 5% si la convergence est vérifiée et -1 s'il n'est pas vérifié. Les entrées et sorties de cette fonction seront clairement définies.

   La programmation et la simulation de l'algorithme précédent, dans les conditions de l'étude, ont permis de déterminer une temps de réponse de $0.045s$.

Recherche C - 5 Différences divisées de Newton

Méthode de Briggs pour le calcul des logarithmes

Depuis Stiffel en 1544, on s'intéresse aux fonctions vérifiant $\ell(x\cdot y) = \ell(x) + \ell(y)$ afin de simplifier les calculs des astronomes et des navigateurs. Cependant, cela nécessite de disposer de tables *logarithmiques* ($\lambda o\gamma o\varsigma$: mot, relation, $\alpha\rho\iota\theta\mu o\varsigma$: nombre ; les logarithmes sont donc des relations entre les nombres avant d'être l'anagramme d'algorithmes...).
Pour plus de facilité, nous allons nous occuper du logarithme de base 10 que nous noterons log.
Sachant que $\log_{10}(10) = 1$ et que $\log_{10}(a^p) = p\cdot\log_{10}(a)$ pour tout rationnel p, on en déduit alors que $\log_{10}\left(\sqrt{10}\right) = \frac{1}{2}$, $\log_{10}\left(\sqrt{\sqrt{10}}\right) = \frac{1}{4}$, etc.

Nous allons travailler avec des décimaux comme les mathématiciens des XVI^{e} et XVII^{e} siècles...mais avec .
Ensuite, on utilise l'algorithme de HÉRON pour avoir une approximation décimale de $\sqrt{2}$:

```
In [1]: heron(2,1,6)
Out[2]: 1.4142135623730951
```

Déterminer une fonction `heron(a,x0,n)` qui renvoie une approximation de $\sqrt{a}$ avec n itérations
Pour plus de lisibilité, nous utiliserons dorénavant le module **`math`** de Python qui permet d'obtenir directement une approximation décimale (avec 17 chiffres affichés) des racines carrées à l'aide de **`sqrt`** : xcom]import

```
In [1]: from math import*
In [2]: sqrt(2)
Out[2]: 1.4142135623730951
In [3]: sqrt(10)
Out[3]: 3.1622776601683795
In [4]: sqrt(sqrt(10))
Out[4]: 1.7782794100389228
In [5]: sqrt(sqrt(sqrt(10)))
Out[5]: 1.333521432163324
```

On obtient donc le tableau suivant :

Nombres	10,0000	3,1623	1,7783	1,3335	1,0000
Logarithmes	1	0,5	0,25	0,125	0

Le problème, c'est que connaître le logarithme de 3,1623 n'est pas très intéressant. On aimerait plutôt connaître ceux des entiers de 1 à 10.
C'est ici qu'intervient l'anglais Henry BRIGGS qui publia en 1617 ses premières tables de logarithmes décimaux contenant 1000 valeurs avec quatorze décimales.
Voyons comment il a procédé pour le calcul de $\log(2)$.
Il a commencé par obtenir des valeurs approchées des nombres suivant :

$$\begin{array}{cc} \sqrt{10} = 10^{\frac{1}{2}} & \sqrt{2} = 2^{\frac{1}{2}} \\ \sqrt{\sqrt{10}} = 10^{\frac{1}{2^2}} & \sqrt{\sqrt{2}} = 2^{\frac{1}{2^2}} \\ \sqrt{\sqrt{\sqrt{10}}} = 10^{\frac{1}{2^3}} & \sqrt{\sqrt{\sqrt{2}}} = 2^{\frac{1}{2^3}} \\ \vdots & \vdots \\ \sqrt{\sqrt{\cdots\sqrt{10}}} = 10^{\frac{1}{2^{54}}} & \sqrt{\sqrt{\cdots\sqrt{2}}} = 2^{\frac{1}{2^{54}}} \end{array}$$

Il a finalement abouti à :

$$10^{\frac{1}{2^{54}}} \approx \underbrace{1.000\,000\,000\,000\,000\,127\,819\,149\,320\,032\,35}_{1+a}$$

$$2^{\frac{1}{2^{54}}} \approx \underbrace{1.000\,000\,000\,000\,000\,038\,477\,397\,965\,583\,10}_{1+b}$$

Or on cherche à déterminer $x = \log(2)$ mais $x = \log(2) \Leftrightarrow 10^x = 2$.
D'après les calculs de BRIGGS :

$$1 + b = 2^{\frac{1}{2^{54}}} = (10^x)^{\frac{1}{2^{54}}} = (1+a)^x$$

Un résultat classique montre que $(1+a)^x \approx 1 + ax$ pour x assez proche de zéro. On en déduit que $1 + b \approx 1 + ax$ et donc que :

$$\log(2) = x \approx \frac{b}{a} = \frac{3\,847\,739\,796\,558\,310}{12\,781\,914\,932\,003\,235}$$

```
In [1]: 3847739796558310 / 12781914932003235
Out[1]: 0.30102999566398114
```

Ainsi, $\log(2) \approx 0.301\,029\,995\,663\,981\,14$.
Cela nous permet alors d'avoir également $\log(4) = 2\cdots\log(2)$ et $\log(8) = 3\times\log(2)$ mais aussi $\log(5) = \log(10) - \log(2)$.
Il nous faudrait donc obtenir de même $\log(3)$ et $\log(7)$ pour compléter notre collection car $\log(6) = \log(3) + \log(2)$ et $\log(9) = 2\cdot\log(3)$.
On peut donc à titre d'exercice calculer ces deux logarithmes pour obtenir :

$$\log(3) \approx 0.477\,121\,254\,719\,662\,44 \qquad \log(7) \approx 0.845\,098\,040\,014\,256\,7$$

La route est longue jusqu'aux 1000 logarithmes de Briggs...

Méthode des différences de Newton et application au calcul des logarithmes

On connaît maintenant quelques valeurs du logarithme décimal et l'on voudrait connaître des valeurs intermédiaires.
Nous allons pour cela procéder par interpolation : étant données quatre valeurs, nous allons chercher un polynôme P de degré 3 tel que $P(x) = \log(x)$ pour ces 4 valeurs.
Pour cela, nous allons utiliser la méthode présentée par Isaac Newton en 1676.
Avec des notations modernes, posons $P(x) = a + bx + cx^2 + dx^3$. On connaît les valeurs du logarithmes pour 1, 2, 3 et 4.

$x = 1$	$p(x) = a + b + c + d = y_1$
$x = 2$	$p(x) = a + 2b + 4c + 8d = y_2$
$x = 3$	$p(x) = a + 3b + 9c + 27d = y_3$
$x = 4$	$p(x) = a + 4b + 16c + 64d = y_4$

En soustrayant les lignes deux par deux, on fait disparaître a :

$$\begin{aligned} y_2 - y_1 &= b + 3c + 7d = \Delta y_1 \\ y_3 - y_2 &= b + 5c + 19d = \Delta y_2 \\ y_4 - y_3 &= b + 7c + 37d = \Delta y_3 \end{aligned}$$

En soustrayant les lignes deux par deux, on fait disparaître b :

$$\begin{aligned} \Delta y_2 - \Delta y_1 &= 2c + 12d = \Delta^2 y_1 \\ \Delta y_3 - \Delta y_2 &= 2c + 18d = \Delta^2 y_2 \end{aligned}$$

En soustrayant les lignes deux par deux, on fait disparaître c :
$\Delta^2 y_2 - \Delta^2 y_1 = 6d = \Delta^3 y_1$
ce qui donne :
$d = \frac{1}{6}\Delta^3 y_1$
$c = \frac{1}{2}\Delta^2 y_1 - \Delta^3 y_1$
$b = \Delta y_1 - \frac{3}{2}\Delta^2 y_1 + 3\Delta^3 y_1 - \frac{7}{6}\Delta^3 y_1 = \Delta y_1 - \frac{3}{2}\Delta^2 y_1 + \frac{11}{6}\Delta^3 y_1$
$a = y_1 - \Delta y_1 + \frac{3}{2}\Delta^2 y_1 - \frac{11}{6}\Delta^3 y_1 - \frac{1}{2}\Delta^2 y_1 + \Delta^3 y_1 - \frac{1}{6}\Delta^3 y_1 = y_1 - \Delta y_1 + \Delta^2 y_1 - \Delta^3 y_1$
Finalement :

$$P(x) = y_1 + (x-1)\Delta y_1 + \frac{1}{2}(x-1)(x-2)\Delta^2 y_1 + \frac{1}{6}(x-1)(x-2)(x-3)\Delta^3 y_1$$

Newton avait l'habitude de réunir les résultats dans un schéma de cette forme :

$$\begin{array}{lllll} \boldsymbol{y_1} & & & & \\ & \boldsymbol{\Delta y_1} & & & \\ y_2 & & \boldsymbol{\Delta^2 y_1} & & \\ & \Delta y_2 & & \boldsymbol{\Delta^3 y_1} & \\ y_3 & & \Delta^2 y_2 & & \\ & \Delta y_3 & & & \\ y_4 & & & & \end{array}$$

avec $\Delta y_i = y_{i+1} - y_i$, $\Delta^2 y_i = \Delta y_{i+1} - \Delta y_i$ et $\Delta^3 y_i = \Delta^2 y_{i+1} - \Delta^2 y_i$.
On peut généraliser à d'autres valeurs d'entiers successifs et à un nombre quelconque de pôles.
Interpolation de Newton passant par les points (α, y_1), $(\alpha+1, y_2)$,..., $(\alpha+n-1, y_n)$:

$$p(x) = y_1 + (x-\alpha)\Delta y_1 + \frac{1}{2!}(x-\alpha)(x-\alpha-1)\Delta^2 y_1 + \cdots + \frac{1}{(n-1)!}(x-\alpha)\cdots(x-\alpha-(n-2))\Delta^{n-1} y_1$$

Et dans tout ça ?

Déterminez les fonctions suivantes :

```
def zip(xs, ys) :
    """Renvoie la liste des couples (xs[i], ys[i])"""

def reduit_liste(L) :
    """ Calcule la liste des différences 2 à 2 d'une liste de nbs """

def reduit(L) :
    """ Renvoie la liste des listes de différences : le triangle de Newton applati """

def diff_newton(L) :
    """ Renvoie les y1, Dy1, D2y1, etc. """

def base_newton(alpha, x, n) :
    """ Renvoie la liste des évaluations des (x-a)(x-a-1).../n! accumulées dans une liste """

def interpol_newton(liste, alpha, x) :
    """ renvoie l'évaluation du poly de Newton en x en partant de a avec une liste de valeurs donnée """

def liste_log_entiers(a,b) :
    """ renvoie la liste des logarithmes décimaux des entiers de a à b inclus """
```

Déterminez par exemples les logarithmes décimaux des réels entre 1 et 4 avec un pas de 0.25 en utilisant les valeurs déjà connues de $\log(2)$, $\log(3)$ et $\log(4)$:

```
Out[1]: [(1.0, 0.0),
 (1.25, 0.091005689059948658),
 (1.5, 0.17074397829308552),
 (1.75, 0.24036777729567488),
 (2.0, 0.3010299956639812),
 (2.25, 0.35388354299426877),
 (2.5, 0.4000813288828019),
 (2.75, 0.44077626292584499),
 (3.0, 0.47712125471966244),
 (3.25, 0.51026921386051849),
 (3.5, 0.54137304994467772),
 (3.75, 0.5715856725684042),
 (4.0, 0.60205999132796229)]
```

Comme Python sait calculer des logarithmes, nous pouvons comparer cette méthode vieille de 350 ans avec les résultats actuels :

Comparez par exemple les valeurs trouvées pour les logarithmes des nombres entre 44 et 47. Quel est l'ordre de l'erreur commise.

THÈME

Dessine-moi une matrice...

Aujourd'hui, des mathématiques concrètes vont nous permettre de réduire, agrandir, assombrir, éclaircir, compresser, bruiter, quantifier,... une photo. Pour cela, il existe des méthodes provenant de la théorie du signal et des mathématiques continues. Nous nous pencherons plutôt sur des méthodes plus légères basées sur l'algèbre linéaire et l'analyse matricielle. Une image sera pour nous une matrice carrée de taille 2^9 à coefficients dans $[\![0, 2^9 - 1]\!]$. Cela manque de charme ? C'est sans compter sur Lena qui depuis quarante ans rend les maths sexy (la population hantant les laboratoires mathématiques et surtout informatiques est plutôt masculine ...).

Nous étudierons pour cela la « Décomposition en Valeurs Singulières » qui apparaît de nos jours comme un couteau suisse des problèmes linéaires : en traitement de l'image et de tout signal en général, en reconnaissance des formes, en robotique, en statistique, en étude du langage naturel, en géologie, en météorologie, en dynamique des structures, en....

Dans quel domaine travaille-t-on alors : algèbre linéaire, analyse, probabilités, topologie, informatique... ?

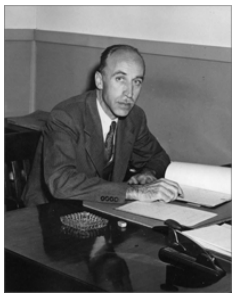
1 Lena

W. Kahan (né en 1933)

Nous allons travailler avec des images qui sont des matrices de niveaux de gris. Notre belle amie Léna sera représentée par une matrice carrée de taille 2^9 ce qui permet de reproduire Léna à l'aide de $2^{18} = 262\,144$ pixels. Léna prend alors beaucoup de place. Nous allons tenter de compresser la pauvre Léna sans pour cela qu'elle ne perde sa qualité graphique. Une des méthodes les plus abordables est d'utiliser la décomposition d'une matrice en valeurs singulières.
C'est un sujet extrêmement riche qui a de nombreuses applications. L'algorithme que nous utiliserons (mais que nous ne détaillerons pas) a été mis au point par deux très éminents chercheurs en 1965 (Gene Golub, états-unien et William Kahan, canadien, père de la norme IEEE-754). Il s'agit donc de mathématiques assez récentes, au moins en comparaison avec votre programme...
La petite histoire dit que des chercheurs américains de l'University of Southern California étaient pressés de trouver une image de taille 2^{18} pixels pour leur conférence. Passe alors un de leurs collègues avec, en bon informaticien, le dernier Playboy sous le bras. Ils décidèrent alors d'utiliser le poster central de la Playmate comme support...
La photo originale est ici : http://www.lenna.org/full/len_full.html mais nous n'utiliserons que la partie scannée par les chercheurs, de taille $5.12\text{in} \times 5.12\text{in}$...

2 La SVD

2 1 Le théorème



C. Eckart (1902-1973)

Un théorème démontré officiellement en 1936 par Carl Eckart et Gale Young affirme alors que toute matrice rectangulaire A se décompose sous la forme :

$$A = U \times S \times {}^{\mathrm{t}}V$$

avec U et V des matrices orthogonales et S une matrice nulle partout sauf sur sa diagonale principale qui contient les valeurs singulières de A rangées dans l'ordre décroissant. Sylvester s'y était déjà intéressé pour des matrices carrées réelles en 1889 mais ce résultat a semblé n'intéresser personne pendant des années.
Le court article de Eckart et Young est disponible à l'adresse suivante :
http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&handle=euclid.bams/1183501633.
Ce qui est remarquable, c'est que n'importe quelle matrice admet une telle décomposition, alors que la décomposition en valeurs propres (la diagonalisation d'une matrice) n'est pas toujours possible. Et quand on dit n'importe quelle matrice, on ne rigole pas : toute matrice, même rectangulaire. C'est vraiment très puissant et à la fois simple mais cette décomposition n'a trouvé d'applications importantes qu'assez récemment (ce qui veut dire après 1899 à l'échelle de votre programme de prépa...) ce qui explique peut-être qu'on en parle si peu en premier cycle.
Cependant, la décomposition en éléments singuliers utilise la décomposition en éléments propres. Notons r le rang de A et U_i et V_i les vecteurs colonnes de U et V. La décomposition s'écrit :

$$A = \begin{pmatrix} U_1 & U_2 & \cdots & U_r & \cdots & U_m \end{pmatrix} \times \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & \ddots & \\ & & & & 0 \end{pmatrix} \times \begin{pmatrix} {}^{\mathrm{t}}V_1 \\ \vdots \\ {}^{\mathrm{t}}V_r \\ \vdots \\ {}^{\mathrm{t}}V_n \end{pmatrix}$$

ou formulé autrement :

$$\begin{aligned} A &= \sigma_1 \cdot U_1 \times {}^{\mathrm{t}}V_1 + \sigma_2 \cdot U_2 \times {}^{\mathrm{t}}V_2 + \cdots + \sigma_r \cdot U_r \times {}^{\mathrm{t}}V_r + 0 \cdot U_{r+1} \times {}^{\mathrm{t}}V_{r+1} + \cdots \\ &= \sigma_1 \cdot U_1 \times {}^{\mathrm{t}}V_1 + \sigma_2 \cdot U_2 \times {}^{\mathrm{t}}V_2 + \cdots + \sigma_r \cdot U_r \times {}^{\mathrm{t}}V_r \end{aligned}$$

Il faut ensuite se souvenir que les σ_i sont classés dans l'ordre décroissant ce qui va avoir des applications importantes en informatique.

2 2 Interprétation géométrique

Comment interpréter géométriquement ce résultat ? Considérez une sphère de rayon 1 : que devient-elle transformée par A, i.e. par $U \times S \times {}^tV$? Faites un petit dessin en dimension 2. Que vaut $\|A\|_2 = \max_{\|x\|=1} \|Ax\|_2$?

Afin de répondre, un peu de vocabulaire d'abord. On a $AV_i = \sigma_i U_i$: on dit alors que U_i est un vecteur singulier à gauche pour la valeur singulière σ_i.

De même ${}^tAU_i = \sigma_i V_i$ et V_i est un vecteur singulier à droite pour la valeur singulière σ_i. Les bases (u_i) et (V_i) sont orthonormées : si l'on exprime la matrice de l'application linéaire associée à A dans ces bases, que pouvez-vous dire géométriquement ?

2 3 Un exemple simple

On suppose que la SVD d'une matrice A de taille $m \times n$ s'écrit $U \times S \times {}^tV$.

1. Quelles sont les dimensions de U et V ? Que pensez-vous de $A \times {}^tA$? Que représentent les colonnes de U pour $A \times {}^tA$?. Et les colonnes de V pour ${}^tA \times A$.
2. Calculez (à la main...) la SVD de $A = \begin{pmatrix} 2 & -2 \\ 1 & 1 \end{pmatrix}$.
3. Shoot again avec $\begin{pmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{pmatrix}$

Quels liens existent entre la SVD et la recherche des éléments propres d'une matrice ? Avec le théorème spectral ? Vaut-il mieux utiliser $A \times {}^tA$ ou ${}^tA \times A$?

2 4 Approximation de rang minimum d'une matrice

Dans de nombreux domaines, on doit travailler avec de grosses matrices. Il est alors très important de déterminer une matrice $\widetilde{A_s}$ de même taille qu'une matrice A mais de rang s inférieur au rang r de A et qui minimise l'erreur commise pour une certaine norme (qui est le plus souvent la norme de Frobenius, i.e. la racine carrée de la somme des carrés des coefficients). Le théorème d'Eckart-Young permet alors de dire que :

$$\min_{\mathrm{rg}(X)\leqslant s} \|A - X\|_F = \|A - \widetilde{A_s}\|_F = \sqrt{\sum_{j=s+1}^{r} \sigma_j^2(A)}$$

avec $\widetilde{A_s} = \sigma_1 \cdot U_1 \times {}^tV_1 + \sigma_2 \cdot U_2 \times {}^tV_2 + \cdots + \sigma_s \cdot U_s \times {}^tV_s$: on considère que les éléments diagonaux de A sont nuls en-dessous d'un certain seuil. Nous allons l'illustrer informatiquement à défaut de le démonter. On voit tout de suite des conséquences pratiques, que nous allons également illustrer informatiquement dans le cas de la compression des images.

3 Manipulation d'images

3 1 Quelques fonctions Python

Python contient une matrice carrée de taille 512 contenant les niveaux de gris représentant Lena. Il suffit de cherger les bonnes bibliothèques :

```
1 from pylab import *
2 from scipy import misc
3
4 l = misc.lena()
5 imshow(l,cmap=cm.gray)
```

Lena est bien une matrice :

```
1 In [8]: type(l)
2 Out[8]: numpy.ndarray
3
```

```
In [9]: shape(l)
Out[9]: (512, 512)
```

On peut préférer travailler sur une photo de format jpg (qui sera à l'envers) ou png (qui sera à l'endroit). Python la transforme en matrice avec la fonction imread. La matrice obtenue est en RVB : chaque coefficient est un vecteur (R,V,B). On la convertit en niveau de gris par exemple en calculant la moyenne des trois couleurs.
On récupère d'abord une image :

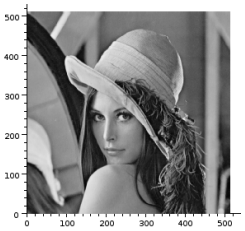
```
In [4]: from urllib.request import urlretrieve

In [5]:
        ↪ urlretrieve('http://download.tuxfamily.org/tehessinmath/les_images/marty.png','marty.pr
Out[5]: ('marty.png', <http.client.HTTPMessage at 0x7f3aeeb5fa20>)
```

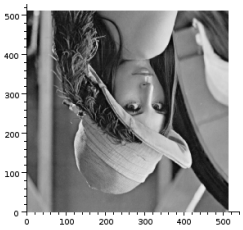
et on la transforme en matrice :

```
tm = imread('marty.png')

def rvb_to_gray(m):
    r = len(m)
    c = len(m[0])
    return(array([[m[i,j][0] for j in range(c)] for i in range(r)]))

marty = rvb_to_gray(tm)
```

3 2 Manipulations basiques

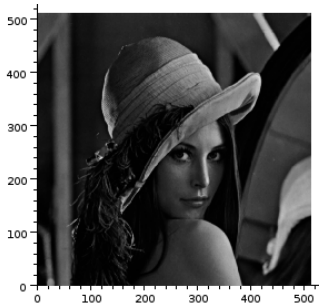
Comment obtenir les images suivantes :

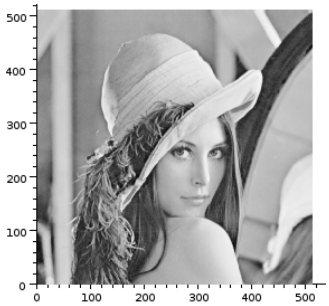


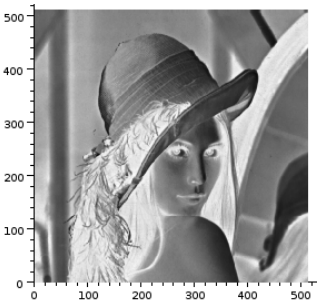


Sachant que l'échelle de gris est celle-ci :

comment obtenir les images suivantes :







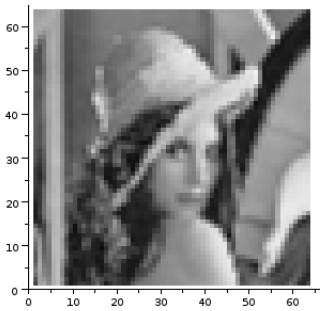
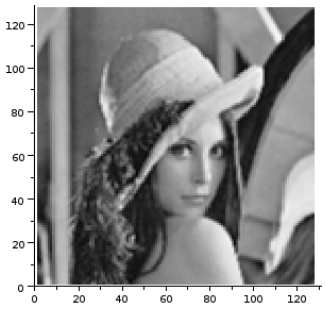
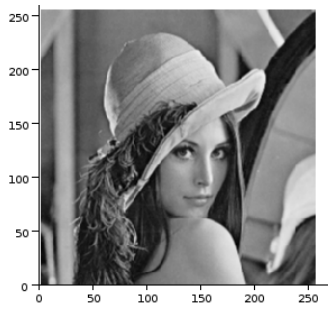
3 3 Gagner de la place

Une matrice de taille 2^9 contient 2^{18} entiers codés entre 0 et $2^8 - 1$ ce qui prend pas mal de place en mémoire. Peut-on être plus économique sans pour cela diminuer la qualité esthétique de la photo ?

3.3.1 Résolution

La première idée est de prendre de plus gros pixels, c'est-à-dire une matrice plus petite : on extrait par exemple régulièrement un pixel sur k (en choisissant k parmi une puissance de 2 inférieure à 2^9).

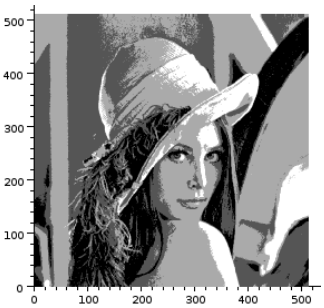
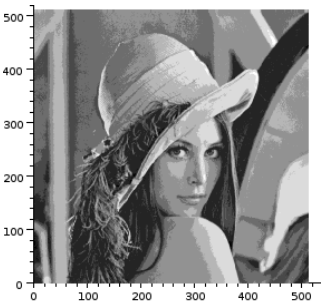
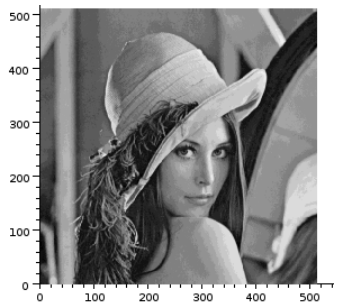
Comment obtenir par exemple ces images ?

3.3.2 Quantification

On peut également réduire le nombre de niveaux de gris en regroupant par exemple tous les niveaux entre 0 et 63 en un seul niveau 0, puis 64 à 127 en 64, 128 à 191 en 128, 192 à 256 en 192.

Par exemple, avec 4, 8 puis 16 niveaux :

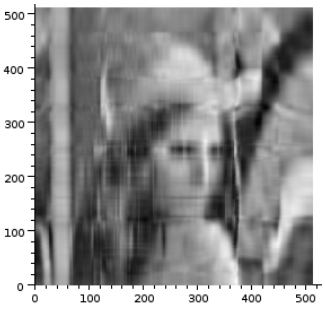
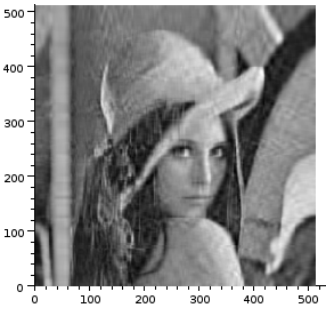
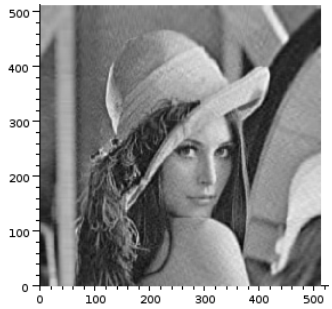




3.3.3 Compression et SVD

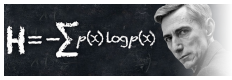
Tout langage orienté calcul comprend des fonctions liées à la décomposition en éléments simples. Python a sa fonction `linalg.svd(mat)` qui renvoie la décomposition en valeurs singulières sous la forme du triplet U, S et V.

En vous inspirant des commentaires faits dans les paragraphes précédents sur la SVD, expliquez comment cette décomposition permet de compresser une image. Quantifiez également le niveau de compression.

Voici trois niveaux de compression :

3.4 Enlever le bruit



C.E. Shannon (1916-2001)

La transmission d'informations a toujours posé des problèmes : voleurs de grands chemins, poteaux télégraphiques sciés, attaques d'indiens, etc.

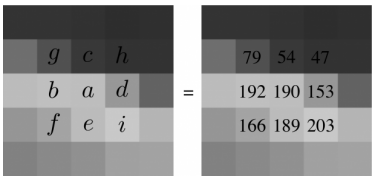
Claude Elwood Shannon (1916 - 2001) est un mathématicien-inventeur-jongleur américain qui, suite à son article « *A mathematical theory of communications* » paru en 1948, est considéré comme le fondateur de la *théorie de l'information* qui est bien sûr une des bases de...l'*informatique*.

L'idée est d'étudier et de quantifier l'« information » émise et reçue : quelle est la compression maximale de données digitales ? Quel débit choisir pour transmettre un message dans un canal « bruité » ? Quel est le niveau de sûreté d'un chiffrement ?...

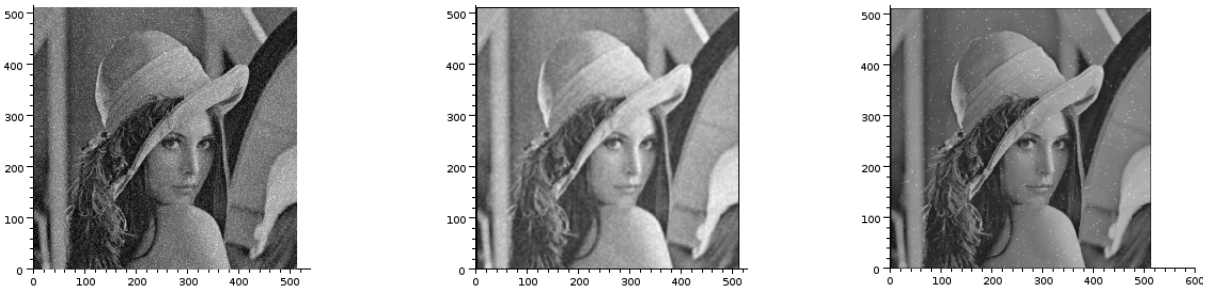
La théorie de l'information de SHANNON est fondée sur des modèles probabilistes : leur étude est donc un préalable à l'étude de problèmes de réseaux, d'intelligence artificielle, de systèmes complexes.
Par exemple, dans le schéma de communication présenté par SHANNON, la source et le destinataire d'une information étant séparés, des perturbations peuvent créer une différence entre le message émis et le message reçu. Ces perturbations (bruit de fond thermique ou accoustique, erreurs d'écriture ou de lecture, etc.) sont de nature *aléatoire* : il n'est pas possible de prévoir leur effet. De plus, le message source est par nature *imprévisible* du point de vue du destinataire (sinon, à quoi bon le transmettre).
SHANNON a également emprunté à la physique la notion d'entropie pour mesurer le désordre de cette transmission d'information, cette même notion d'entropie qui a inspiré notre héros national, Cédric VILLANI...
Mais revenons à Lena. Nous allons simuler une Lena bruitée :

```
from random import *
def ber(p):
    if random() < p:
        return 1
    else:
        return 0

def bruit(m,p):
    r = len(m)
    c = len(m[0])
    return(array([[(m[i,j] + randint(0,255)*ber(p))%256 for j in range(c)] for i in
        ↪ range(r)]))
```

Il existe de nombreuses méthodes, certaines très élaborées, permettant de minimiser ce bruit. Une des plus simples est de remplacer chaque pixel par la moyenne de lui-même et de ses 8 autres voisins directs, voire aussi ses 24 voisins directs et indirect, voire plus, ou de la médiane de ces séries de « cercles » concentriques de pixels, comme on le voit sur cette figure (extraite du site `Images des mathématiques`) :



Créez des fonctions qui feront le boulot. Vous utiliserez `mean` et `median` pour obtenir par exemple :



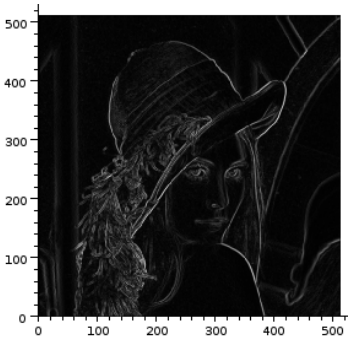
où l'on trouve de gauche à droite l'image bruitée originale puis sa correction par moyenne sur 25 pixels et par médiane sur 9 pixels.

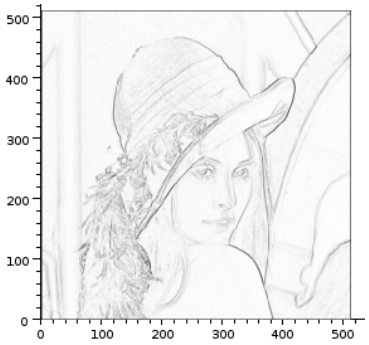
3 5 Détection de bords

Pour sélectionner des objets sur une image, on peut essayer de détecter leurs bords, i.e. des zones de brusque changement de niveaux de gris.
On remplace alors un pixel par une mesure des écarts des voisins immédiats donnée par exemple par :

$$\sqrt{(m_{i,j+1}-m_{i,j-1})^2+(m_{i+1,j}-m_{i-1,j})^2}$$

On obtient alors au choix :





4 Chiffrement de Hill

Nous allons à présent utiliser nos outils de calcul matriciel en cryptographie. En 1929, Lester Hill (1891 - 1961) proposa une méthode de chiffrement par blocs de lettres en travaillant avec des matrices à coefficients dans $\mathbb{Z}/26\mathbb{Z}$ (combien y a-t-il de lettres dans notre alphabet ?). Par la suite nous noterons $\mathbb{Z}_n$ l'ensemble $\mathbb{Z}/n\mathbb{Z}$. Cela a ouvert le domaine de la cryptographie algébrique et de nos jours, c'est l'Aes (*Advanced Encrypting Standard*), né en 2000, qui assure la majorité de la sécurité informatique.
Nous allons explorer cette méthode en travaillant sur tous les caractères affichables en ASCII.

4 0 1 Codage

Chaque caractère est associé à un codage binaire, en général sur 8 bits (un octet) ce qui donne $2^8 = 256$ caractères possibles.
En fait, un américain a mis au point en 1961 l'ASCII (on prononce « aski ») pour « American Standard Code for Information Interchange » qui codait 128 caractères sur 7 bits ce qui est suffisant pour les américains qui n'ont pas de lettres accentuées.
Il a ensuite fallu étendre ce code pour les langues comportant des caractères spéciaux et là, les normes diffèrent, ce qui crée des problèmes de compatibilité.
Depuis quelques années, le codage Unicode tente de remédier à ce problème en codant tous les caractères existants.
La plupart des langages de programmation, et bien sûr , ont une commande qui associe le code ASCII (éventuellement étendu) à un caractère. On pourra donc l'utiliser pour résoudre notre problème. Il s'agit de la commande `ord(caratère)` qui renvoie le code ASCII du caractère `caractère`.

Caractère		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2
Code	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Caractère	3	4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E
Code	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
Caractère	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Code	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88
Caractère	Y	Z	[	\	]	^	_	‘	a	b	c	d	e	f	g	h	i	j	k
Code	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107
Caractère	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~
Code	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126

Table des caractères ASCII affichables

Par exemple :

```
>>> ord('@')
64
```

Mais attention ! Cela ne fonctionne par pour les chaînes de caractères :

```
>>> ord('AB')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: ord() expected a character, but string of length 2 found
```

Il faudra donc considérer les caractères d'une chaîne un par un :

```
def ascii(chaine):
    return [ord(c)-ord((' ')) for c in chaine]
```

Le `-ord((''))` permet d'obtenir des caractères codés entre 0 et 95.

```
>>> ascii("3h de retard !!#@!")
[19, 72, 0, 68, 69, 0, 82, 69, 84, 65, 82, 68, 0, 1, 1, 3, 32, 1]
```

Tant qu'on y est, créons une fonction `deascii` qui effectuera l'opération inverse. On utilisera alors la commande chr qui renvoie le caractère associé à un code ASCII. On utilise join pour concaténer plus vite des chaînes.

```
def deascii(chaine):
    return ''.join(chr(x + ord((' '))) for x in chaine)
```

Par exemple :

```
>>> deascii([19, 72, 0, 68, 69, 0, 82, 69, 84, 65, 82, 68, 0, 1, 1, 3, 32, 1])
'3h de retard !!#@!'
```

Choisissons à présent un message à coder, assez long :

Arghh !! Ce 1000 pattes aurait 314 pattes ? ?

```
>>> m = "Arghh !! Ce 1000 pattes aurait 314 pattes ??"
```

Le principe consiste alors à regrouper les nombres par paquets de longueur constante p et donc à construire à partir de la liste `L` une matrice de p (lignes ou colonnes ?) complétée éventuellement de zéros.
Nous allons pour cela créer une fonction `to_Hill(liste,p)` qui transforme une liste en une matrice de p (lignes ou colonnes ?) en complétant par des zéros si nécessaire :

```
def to_Hill(m,p):
  ???
```

Le message en clair devient une matrice. Choisissons par exemple un 3-chiffrement de Hill ; la matrice `clair` aura donc 3 lignes :

```
>>> clair = to_Hill(m,3)
>>> clair
 33  72   1  35  17  16  65  69  65  65   0  20  65  69  31
 82  72   1  69  16   0  84  83  85  73  19   0  84  83  31
 71   0   0   0  16  80  84   0  82  84  17  80  84   0   0
```

Il faut choisir à présent une clé de chiffrement : ce sera une matrice carrée de taille 3, à coefficients dans $\mathbb{Z}_{95}$ et inversible dans $\mathbb{Z}_{95}$. Nous allons vérifier que la matrice $\begin{pmatrix} 1 & 23 & 29 \\ 0 & 31 & 1 \\ 47 & 3 & 1 \end{pmatrix}$ convient.
Nous voudrions utiliser `inverse` mais on y effectue une division réelle ce qui n'est pas autorisée dans $\mathbb{Z}_{95}$. Nous allons donc un peu transformer `inverse` pour donner la méthode d'inversion des nombres comme argument. Seule la dernière ligne de `diago_triangle` change en fait :

```
    def diago_triangle(self,inv):
        .
        .
        .
            T = T.mult_ligne(inv(T[k,r-1]),k)
        return T
```

puis :

```
def inverse(self,inv):
    return self.cat_carre_droite().triangle().diago_triangle(inv).extrait_carre_droite()
```

Oui mais quel est l'inverse d'un nombre dans $\mathbb{Z}_{95}$?
C'est un problème d'arithmétique qui se résout à l'aide de l'algorithme d'Euclide étendu :
Petit rappel sur cet algorithme qui permet de prouver le théorème de Bézout et d'obtenir des coefficients vérifiant $au + bv = a \wedge b$:

k	u_k	v_k	r_k	q_k
0	1	0	$r_0 = a$	/
1	0	1	$r_1 = b$	q_1
2	$u_0 - u_1 q_1$	$v_0 - v_1 q_1$	$r_2 = r_0 - r_1 q_1$	q_2
3	$u_1 - u_2 q_2$	$v_1 - v_2 q_2$	$r_3 = r_1 - r_2 q_2$	q_3
$\vdots$			$\vdots$	$\vdots$
$p-1$			$r_{p-1} = a \wedge b$	q_{p-1}
p			$r_p = 0$	

La démonstration de l'algorithme découle de la relation :

$$u_k - \left(u_{k+1} \times q_{k+1} \right) = u_{k+2}$$

et de la relation similaire pour les v_k et les r_k.
Par exemple, pour 19 et 15 :

k	u_k	v_k	r_k	q_k	
0	1	0	19	/	L_0
1	0	1	15	1	L_1
2	1	−1	4	3	$L_2 \leftarrow L_0 - 1 \times L_1$
3	−3	4	3	1	$L_3 \leftarrow L_1 - 3 \times L_2$
4	4	−5	1	3	$L_4 \leftarrow L_2 - 1 \times L_3$
5			0		$L_5 \leftarrow L_3 - 3 \times L_4$

C'est-à-dire $4 \times 19 - 5 \times 15 = 1$ et $19 \wedge 15 = 1$.
En version récursive, on peut procéder en deux étapes en suivant la méthode rappelée ci-dessus :

```
def aee(u,v,r,U,V,R):
    if R == 0: return [u,v,r]
    else:
        q = r // R
        return ???
```

```
6
7  def AEE(a,b):
8      return aee(1,0,a,0,1,b)
```

Soit $\dot{a}$ la classe de l'entier a modulo n. Démontrons que $\dot{a}$ est inversible dans $\mathbb{Z}_n$ si et seulement si a et n sont premiers entre eux :
$\dot{a} \times \dot{b} = \dot{1} \Leftrightarrow ab \equiv 1[n] \Leftrightarrow \exists k \in \mathbb{Z}\ ab = 1 + kn \Leftrightarrow \exists k \in \mathbb{Z}\ ba - kn = 1 \Leftrightarrow a \wedge n = 1$, la dernière équivalence étant obtenue grâce au théorème de Bézout.
Comme 11 et 13 sont premiers, chaque élément non nul de $\mathbb{Z}_{11}$ et $\mathbb{Z}_{13}$ est inversible. Mais par exemple 6 n'est pas premier avec 9, donc $\dot{6}$ n'est pas inversible dans $\mathbb{Z}_9$, c'est pourquoi la résolution de l'équation $6x \equiv b[9]$ pose problème.
Nous allons donc utiliser la fonction **AEE** créée précédemment pour déterminer l'inverse d'un nombre dans $\mathbb{Z}_n$:

```
1  def inv_mod(p,n):
2      B = AEE(p,n)
3      assert B[2] == 1, str(p) + ' non iversible modulo ' + str(n)
4      return ???
```

On peut alors obtenir directement l'inverse de la matrice `cle` :

```
1  >>> cle = list2mat([[1,23,29],[0,31,1],[47,3,1]])
2  >>> inv_cle = cle.inverse(lambda x: inv_mod(x, 95))
3  >>> inv_cle
4  -1.82136E+09 -4.16311E+09  5.69826E+10
5         23312       -675552         -496
6        -30597         22638          651
```

Oups ! Il serait plus lisible d'obtenir les représentants canoniques de $\mathbb{Z}_{95}$. Il suffit donc de calculer le reste de chaque coefficient dans la division par 95 donc d'appliquer la fonction `lambda coeff : coeff %95`.
Comme ce genre d'action est standard, on créer une méthode `map` dans notre classe `Mat` (on parle alors de *foncteur* en programmation fonctionnelle qui rejoint la notion homonyme en théorie des catégories).

```
1      def map(self,f):
2          return Mat(self.D, lambda i,j: f(self.F(i,j)))
```

Alors :

```
1  >>> inv_cle = cle.inverse(lambda x: inv_mod(x, 95)).map(lambda coeff : coeff % 95)
2  >>> inv_cle
3   18  14  34
4   37  88  74
5   88  28  81
```

On vérifie :

```
1  >>> (cle * inv_cle).map(lambda coeff : coeff % 95)
2   1  0  0
3   0  1  0
4   0  0  1
```

Nous pouvons multiplier `cle` par `clair` pour coder notre message. Pour le décoder, il suffira de multiplier le message codé par `inv_cle`.
La matrice `cle×clair` est une matrice de 3 lignes. Il faudra la transformer ensuite en liste pour lui appliquer `decode`. On utilise la fonction iter définie dans notre classe `Mat`.

```
>>> crypte = deascii(iter((cle * clair).map(lambda n : n % 95)))

>>> crypte
"sP_2Ou8?R'QOyE(Xph_<bn(h<Yc cAkDj\\pf_<bn(ho+>"
```

Si on le compare au message initial :
"Arghh !! Ce 1000 pattes aurait 314 pattes ??"
on remarque que le premier « pattes » est codé par `h_<be(` alors que le second est codé par `pf_<bn` : ce n'est donc pas un codage lettre par lettre et il est donc apparemment plus compliqué que le chiffrement de César.

4 0 2 Décodage

Celui qui reçoit ce message (l'éternel Bob...) possède la clé de décodage qui est `inv_cle`.

```
>>> decrypte = deascii(???)
>>> decrypte
'Arghh !! Ce 1000 pattes aurait 314 pattes ?? '
```

Ça marche ! On remarque les deux espaces (neutres) rajoutés à la fin pour obtenir une chaîne de longueur un multiple de 3.

5 Codes correcteurs d'erreurs linéaires

Un très rapide tour d'horizon...Vous utiliserez la classe Mat pour illustrer les notions suivantes...

5 1 Quelques rappels

On veut transmettre 2^r messages distincts : on les code sur r bits. On rajoute des bits de contrôle pour vérifier que le message a été bien transmis.
Le message tient donc en fait sur n bits.
Si une transformation linéaire permet de passer du code initial de $\mathbb{F}_2^r$ au code émis de $\mathbb{F}_2^n$, on dit que le code est linéaire.

Recherche D - 1
Par exemple, on veut transmettre des mots de 1 bit. On émet le message en le triplant (1 est émis en 111). Le code est-il linéaire ? Quelle est sa matrice G de codage ?

Notons G la matrice du code linéaire.

Recherche D - 2
Il doit y avoir au moins autant de mots codés que de mots au départ : comment cela se traduit-il quant au rang de G ?

Notons C la matrice de contrôle telle que $C \cdot m_r = 0$ si, et seulement si, m_r (le message reçu) est un mot de code (qui a donc été transmis sans erreur).

Recherche D - 3
Que dire du rang de C ?

On notera *syndrome* la fonction associée à s.
On appellera matrice de décodage la matrice D telle que $x = Dy \Leftrightarrow y = Gx$.
Par la méthode de la ℓ-réduite échelonnée de Gauss-Jordan on peut déterminer D et C : suivez les explications de votre chargé de TD préféré...

Recherche D - 4
Que donne $R \times (G|m_r)$?

Notons ε l'erreur commise que le message codé émis m_c. Alors $m_c = m_r + \varepsilon$.

Recherche D - 5
Montrez que $s(m_r) = s(\varepsilon)$ et que l'on peut répartir les vecteurs de $\mathbb{F}_2^n$ selon des classes d'équivalences par syndromes.

Montrez que si u est un message reçu quelconque, classe$(u) = u + C$ avec C l'ensemble des mots codés.

Un code de HAMMING est un code tel que tous les vecteurs de $\mathbb{F}_2^{n-r}$, sauf le vecteur nul, apparaissent dans C.

5 2 Exemples

Recherche D - 6

Déterminez une matrice de contrôle du code défini par la matrice génératrice :

$$G = \begin{pmatrix} 1&0&0&1 \\ 1&1&1&1 \\ 0&1&1&0 \\ 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix}$$

Après contrôle d'erreur, décodez les messages reçus suivants : 0111101, 0100110, 1010010. Quel est le défaut de ce code ?

Recherche D - 7

Même style de questions avec :

$$G = \begin{pmatrix} 1&1&0&1 \\ 1&0&1&1 \\ 1&0&0&0 \\ 0&1&1&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix}$$

On reçoit 1010110, 1100110, 0100001, 0010001. Quelle est la nature du code ? Quelle est la matrice de décodage D ?

Recherche D - 8

Même style de questions avec :

$$G = \begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 1&0&1&0 \\ 1&1&0&1 \\ 0&1&1&0 \\ 0&0&1&1 \\ 0&0&0&1 \end{pmatrix}$$

On reçoit 1100101. Quelle est la nature du code ? Quelle est la matrice de décodage D ?

Recherche D - 9

Si p est la probabilité d'erreur sur un bit et si les erreurs par bit sont indépendantes, exprimez la probabilité en fnctin de p qu'un message erroné devienne un mot différent du mot émis lorsque le code est de HAMMING. Donnez une valeur approchée pour $p = 0,1$. (On rappelle que le code de HAMMING ne corrige que les erreurs de poids 1).

6 Fibonacci : E3A 2015

Recherche D - 1

On s'intéresse aux nombres de FIBONACCI définis par récurrence par $F_0 = 0$, $F_1 = 1$ et

$$\forall n \in \mathbb{N}, \quad F_{n+2} = F_{n+1} + F_n$$

1. i. Écrire en langage Python une fonction **fib(n)** permettant le calcul de F_n.

ii. On peut démontrer (et on l'admettra en e3a mais pas vous, l'élite de la France) que ces nombres vérifient les relations suivantes pour tout entier naturel non nul :

$$F_n = 2F_{\frac{n}{2}-1}F_{\frac{n}{2}} + (F_{\frac{n}{2}})^2 \text{ si } n \text{ est pair}, \quad F_n = (F_{\frac{n-1}{2}+1})^2 + (F_{\frac{n-1}{2}})^2 \text{ si } n \text{ est impair}$$

Compléter l'écriture de la fonction **fibb(n)** :

```
1 def fibb(n) :
2     if n <= 1 :
3         return n
```

```
    else :
        if n%2 == 0 :
            a = fibb(n//2)
            b = fibb(n//2 - 1)
            return a * (2 * b + a)
        else : ...
```

2. **i.** Soit $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$. Pour tout entier naturel non nul n, exprimer les coefficients de A^n au moyen des nombres de FIBONACCI F_{n+1}, F_n et F_{n-1}.

ii. Écrire en langage Python une fonction **`fibbb(n)`** permettant le calcul de F_n en utilisant le calcul de A^n (dont le calcul doit intervenir dans le programme).

iii. Montrer que pour tout entier naturel non nul n, on a $F_{n+1}F_{n-1} - (F_n)^2 = (-1)^n$.

3. **i.** Montrer la convergence de la série de terme général $\frac{(-1)^n}{F_{n+1}F_n}$ et exrimer $S = \sum_{n=1}^{+\infty} \frac{(-1)^n}{F_{n+1}F_n}$ au moyen de $L = \lim_{p\to+\infty} \frac{F_p}{F_{p+1}}$ dont l'existence résulte de la convergence de la série.

ii. Grâce à la propriété concernant les séries alternées et que l'on rappellera, étant donné $\varepsilon > 0$, écrire une fonction Python permettant de calculer une vleur approchée de L au moyen d'une somme partielle de cette série, avec une précision inférieure à ε.

4. On note $\Phi = \frac{1+\sqrt{5}}{2}$ et $\varphi = \frac{1-\sqrt{5}}{2}$. On peut montrer et on l'admettra (sauf vous l'élite de la France), que l'on a :

$$\forall n \in \mathbb{N},\ F_n = \frac{1}{\sqrt{5}}\left(\Phi^n - \varphi^n\right)$$

i. Calculer L de la question 3-a au moyen de Φ.

ii. Pour tout $x \in]\varphi, -\varphi[$, montrer la convergence de la série ainsi que l'égalité :

$$\sum_{n=1}^{+\infty} F_n x^n = \frac{x}{1-x-x^2}$$

7 Construisons nos outils

7 0 1 Création d'une classe « Matrice »

Dans cette section, nous créerons nos matrices en donnant leur dimension et la fonction définissant leurs coefficients en fonction de leurs indices...comme nous l'avons fait en Haskell...sauf que c'est moins joli et mois contrôlable...

```
class Mat:
    """ une matrice sous la forme Mat([nb lignes, nb cols], fonction(i,j))"""

    def __init__(self, dim, f):
        self.F = f   # fonction (i,j) -> coeff en position i,j
        self.D = dim # liste [nb de lignes, nb de cols]
```

Par exemple, $\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$ sera creée avec :

```
>>> M = Mat([3,2],lambda i,j : 2*i + (j + 1))
```

On a quand même l'habitude de rentrer une matrice comme une liste de lignes. On va donc créer une fonction qui permet de convertir une liste de lignes en matrice :

```
def list2mat(mat):
    r,c = len(mat),len(mat[0])
    return Mat([r,c], lambda i,j : mat[i][j])
```

La matrice précédente aurait donc pu être définie par :

```
>>> M = list2mat([[1,2],[3,4],[5,6]])
```

```
>>> M.D # dimension de la matrice
[3, 2]
>>> M.F(0,1) # coefficient à la position 0,1
2
```

Recherche

Écrivez une fonction qui renvoie une matrice nulle de taille $n \times m$ puis une autre qui renvoie la matrice identité de taille n.

Nous rajoutons quelques méthodes habituellement définies pour les objets structurés de ce type :

```
    def __getitem__(self,cle):
        """ permet d'obtenir Mij avec M[i,j] """
        return self.F(*cle)

    def __iter__(self):
        """ pour itérer sur la liste des coefficients donnés par colonnes """
        [r,c] = self.D
        for j in range(c):
            for i in range(r):
                yield self.F(i,j)
```

L'emploi du mot-clé `yield` au lieu de `return` permet de ne *retourner* `self.F(i,j)` que lorsque cela est demandé. Ainsi la liste de tous les coefficients n'est pas créée en entier systématiquement ce qui économise la mémoire.
Nous pouvons ainsi créer une méthode qui va permettre un joli affichage :

```
    def __str__(self):
        """ joli affichage d'une matrice"""
        [r,c],f = self.D, self.F
        lmax = len(str(max(iter(self)))) + 1
        s = '\n'.join( (' '.join('{0:>.{l}G}'.format(f(i,j),l=lmax) for j in range(c)))
                ↪  for i in range(r))
        return s

    def __repr__(self):
        """ représentation dans le REPL"""
        return str(self)
```

La méthode `join` est à employer systématiquement pour la concaténation des chaînes de caractères : elle est beaucoup plus efficace que son pendant sous forme de boucle..

```
>>> M
1 2
3 4
5 6
```

Recherche

Créez deux fonctions qui créent un itérateur respectivement sur les lignes et sur les colonnes. Par exemple laliste des coefficients de la première colonne de M sera donnée par :

```
>>> [i for i in M.col(0)]
[1, 3, 5]
```

702 Transposée d'une matrice

La transposée d'une matrice $(M_{ij})_{\substack{1\leqslant i\leqslant n\\1\leqslant j\leqslant m}}$ est égale à la matrice $(M_{ji})_{\substack{1\leqslant i\leqslant n\\1\leqslant j\leqslant m}}$. Attention aux indices et aux nombre de lignes et de colonnes !

Recherche

Créez une fonction qui renvoie la transposée d'une matrice donnée en argument.

```
>>> M.transpose()
1 3 5
2 4 6
```

703 Somme de matrices

On voudrait créer une méthode qui prend comme arguments deux matrices et renvoie leur somme. Il faudra vérifier que les matrices à additionner sont de bonnes tailles : on utilisera la fonction `assert` qui est suivie d'une condition à vérifier et d'un message à affichher en cas de problème.
Pour pouvoir utiliser le symbole + par la suite, on va nommer notre méthode `__add__` :

```
    def __add__(self,other):
        """ somme de deux matrices : utilisation du symbole +"""
        assert self.D == other.D, "tailles incompatibles"
        return Mat(self.D, lambda i,j : self.F(i,j) + other.F(i,j))
```

Par exemple, avec la matrice M précédemment introduite :

```
>>> M + M
  2   4
  6   8
 10  12
```

Recherche

On définit de même `__neg__` pour l'opposé et `__sub__` pour la soustraction : faites-le !

704 Produit par un scalaire

On voudrait obtenir la matrice $k \cdot M$ à partir d'une matrice M et d'un scalaire k.

```
>>> A = Mat([2,3], lambda i,j : 3*i + j)
>>> A
 0  1  2
 3  4  5
>>> A.prod_par_scal(5)
  0   5  10
 15  20  25
```

Recherche

Écrivez une implémentation de la méthode `prod_par_scal`

705 Produit de matrices

Soit $A = (a_{ij})_{\substack{1\leqslant i\leqslant n\\1\leqslant i\leqslant m}}$ et $B = (b_{ij})_{\substack{1\leqslant i\leqslant m\\1\leqslant i\leqslant p}}$. Alors $A \times B = C$ avec $C = (c_{ij})_{\substack{1\leqslant i\leqslant n\\1\leqslant i\leqslant p}}$ et

$$\forall (i,j) \in \mathbb{N}_n^* \times \mathbb{N}_p^*, \quad c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

C'est l'algorithme habituel trois boucles imbriquées).
Nous allons l'abstraire d'un petit niveau : chaque coefficient c_ij est en fait égal au produit scalaire de la ligne i de A et de la colonne j de B.

Commençons donc par construire une fonction qui calcule le produit scalaire de deux itérables. Pour cela, nous allons utiliser une fonction extrêmement importante qui vient du monde de la programmation fonctionelle : `map`.
`map(fonc, iterable(s))` renvoie un itérateur de type `map` mais où chaque élément sera remplacé par son image par la fonction (si c'est une fonction de une variable) ou par l'image combinée des itérables si c'est une fonction de plusieurs variables.
Par exemple :

```
>>> map(lambda x: x*x, [1,2,3,4])
<map at 0x7fe8307f2ef0>
>>> [i for i in m]
[1 4 9 16]
```

arghhh : l'objet crée est de type `map` et on a son adresse mais on ne sait pas ce qu'il y a dedans...

```
>>> m = map(lambda x: x*x, [1,2,3,4])
>>> [i for i in m]
[1 4 9 16]
```

Mais atention ! m a maintenant été « consommé ». Si on en redemande :

```
>>> [i for i in m]
[]
```

Si on veut en garder une trace on peut utiliser par exemple `list` :

```
>>> m = map(lambda x: x*x, [1,2,3,4])
>>> l = list(m)
>>> l
[1 4 9 16]
```

Avec une fonction de deux variables et deux itérables :

```
>>> m = map(lambda x, y: x + y, [1,2,3,4], [-1,-2,-3,-4])
>>> list(m)
[0, 0, 0, 0]
```

On peut aller plus vite en utilisant les opérateurs arithmétiques écrits en notation préfixée dans la bibliothèque `operator` :

```
from operator import mul
>>> m = map(mul, [1,2,3,4], [-1,-2,-3,-4])
>>> list(m)
[-1, -4, -9, -16]
```

Pour notre produit scalaire, nous utilisons également la classique fonction `sum`.

Recherche

Créez une fonction `prod_scal`.
Utilisez `prod_scal` pour écrire en une ligne la matrice produit de deux matrices données en arguments.
Vous créerez ainsi une méthode `prod_mat(self,other)` en ajoutant à la ligne précédente une ligne vérifiant que les formats sont corrects.

Pour utiliser le symbole `*` pour le produit d'une matrice par une autre matrice ou un scalaire, on crée dans notre classe une méthode `__mul__`. On utilisera la fonction `type` qui teste le type d'un objet.

```
  def __mul__(self,other):
        """ produit d'une matrice par un scalaire ou une matrice : utilisation du
            ↪ symbole *"""
        if Mat == type(other):
            return self.prod_mat(other)
        else:
            return self.prod_par_scal(other)
```

706 Comparaison de plusieurs produits matriciels

Notre produit n'est pas si mal : il s'écrit très simplement et est assez efficace. Comparons-le avec d'autres implémentations.

Avec les outils standards de Python

D'abord une implémentation classique en introduisant une liste Python et des matrices du type `array` de `numpy`. On travaillera par exemple avec des tableaux d'entiers (l'argument `dtype = int`) et on utilisera les commandes `shape` et `zeros` de `numpy`.

```
import numpy as np

def pymatmatprod(A, B):
    '''
    Multiplication matricielle avec les outils Python usuels
    '''
    ra, ca = A.shape
    rb, cb = B.shape
    assert ca == rb, "Tailles incompatibles"
    C = np.zeros((ra, cb),dtype = int)
    for i in range(ra):
        Ai,Ci = A[i],C[i]
        for j in range(cb):
            for k in range(ca):
                Ci[j] += Ai[k] * B[k, j]
    return C
```

Avec Cython

Cython (http://cython.org/) est un langage dont la syntaxe est très proche de celle de Python mais c'est un langage qui permet de générer des exécutables compilés et d'utiliser un style de programmation proche du C. Cela permet d'optimiser grandement Python.
Le logiciel de calcul formel Sage http://www.sagemath.org/ est principalement écrit en Cython pour gagner en efficacité.
Voyons un exemple en œuvre. L'usage conjoint de Python et Cython est grandement facilité par le travail dans l'environnement iPython.

```
In [1]: %load_ext cythonmagic

In [2]:
%%cython
cimport cython
import numpy as np
cimport numpy as np

cdef long[:, :] matprodC( long[:, :]  A, long[:, :]  B):
    '''
    multiplication matricielle via cython et les array de numpy
    '''
    cdef:
        int i, j, k
        int ra = A.shape[0]
        int ca = A.shape[1]
        int rb = B.shape[0]
```

```
        int cb = B.shape[1]
        long [:, :] C
        long [:] Ai, Ci

    assert ca == rb, 'Tailles non compatibles'

    C = np.zeros((ra, cb),dtype=int)
    for i in range(ra):
        Ai,Ci = A[i],C[i]
        for j in range(cb):
            for k in range(ca):
                Ci[j] = Ci[j] + Ai[k] * B[k, j]
    return C

def matprod(long [:, :]  A, B):
    return matprodC(A, B)
```

On remarque qu'écrire en Cython revient un peu à écrire en Python mais en déclarant des variables comme en C.
iPython permet également de comparer facilement les temps de calcul avec la commande magique `%timeit`.
On commence par créer une matrice de taille 200×200 puis on va demander son coefficient [100,100] :

```
In [3]: A = np.ones((200,200),dtype = int)
```

On compare alors le produit via Python, Cython et la multiplication de `numpy` :

```
In [4]: %timeit matprod(A,A)[100,100]
100 loops, best of 3: 16.3 ms per loop

In [5]: %timeit pymatmatprod(A,A)[100][100]
1 loops, best of 3: 8.35 s per loop

In [6]: %timeit np.dot(A,A)[100,100]
100 loops, best of 3: 8.94 ms per loop
```

Notre fonction en Cython est 500 fois plus rapide!!! Elle est très légèrement plus lente que le produit de `numpy`.
Et notre implémentation personnelle avec la classe `Mat` ?

```
In [7]: A = Mat([200,200],lambda i,j:1)

In [8]: %timeit (A*A)[100,100]
10000 loops, best of 3: 79.6 µs per loop
```

Trop fort ! Nous battons tout le monde à plate couture ! Nous sommes 100 fois plus rapides que `numpy` !!!
...et nous avons bien le bon résultat :

```
In [9]: (A*A)[100,100]
Out[9]: 200
```

Bon, ça va pour un seul produit. Pour calculer une puissance 200, ce sera moins magique...

7 0 7 Puissances d'une matrice

Par exemple, avec $J = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$:

```
>>> J = Mat([2,2], lambda i,j : 1)
>>> J ** 5
```

```
16  16
16  16
```

En effet, $J^n = 2^{n-1}J$ est un résultat classique. On utilise un algorithme d'exponentiation rapide. Par exemple, en voici une version récursive rapide à écrire qui se nomme `__pow__` afin d'utiliser le symbole `**` :

```python
def __pow__(self,n):
    r = self.D[0]
    if n == 0:
        return unite(r)
    def pui(m,k,acc):
        if k == 0:
            return acc
        return pui((m*m),k//2,acc if k % 2 == 0 else (m*acc))
    return pui(self,n,unite(r))
```

Recherche

Déterminez une version impérative de la méthode précédente.

8 Rappels sur l'inversion des matrices

801 Matrice carrée inversible

Soit $M \in \mathbb{A}^{n\times n}$. M est inversible (régulière) si, et seulement si, il existe une matrice N dans $\mathbb{A}^{n\times n}$ telle que :

$$M \times N = N \times M = \mathbb{I}_n$$

On note alors $N = M^{-1}$.
On remarque (n'est-ce pas) que, d'après cette définition, $(M^{-1})^{-1} = M$.
Lorsque nous aurons étudier les déterminants, nous pourrons démontrer que si A et B sont deux matrices carrées de taille n vérifiant $A \times B = \mathbb{I}_n$, alors elles sont régulières et $A = B^{-1}$.

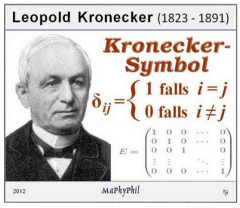
Recherche

Si A et B sont régulières et de taille n, alors comment calculer $(A\times B)^{-1}$ à partir des inverses de A te B ?

802 Opérations sur les lignes

802a Matrices élémentaires

Nous désignerons par E_n^{ij} la matrice carrée de $\mathbb{A}^{n\times n}$ dont tous les coefficients sont nuls sauf le coefficient (i,j) qui vaut $1_{\mathbb{A}}$. Par exemple, $E_3^{12} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ dans $\mathbb{Z}^{3\times 3}$.



Leoplod Kronecker (1823-1891)

Leopold Kronecker est un mathématicien prussien né dans l'actuelle Pologne. On lui doit la célèbre citation : « *Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk* ».
En son honneur, on a donné son nom à la fonction suivante :

$$\begin{array}{rccl} \delta: & \mathbb{N}\times\mathbb{N} & \to & \{0;1\} \\ & (i,j) & \mapsto & 1 \text{ si } i=j,\ 0 \text{ sinon} \end{array}$$

On condense souvent la notation en δ_{ij} et on parle alors de **symbole de Kronecker**. Par exemple, la matrice identité peut être définie par :

$$\mathbb{I}_n = (\delta_{ij})_{\substack{1\leqslant i\leqslant n \\ 1\leqslant j\leqslant n}}$$

Recherche

Étudiez le produit $E_n^{ij} \times E_n^{\ell k}$ et exprimez-le à l'aide du symbole de Kronecker.
Simplifiez ensuite le produit $(\mathbb{I}_n + \lambda E_n^{ij}) \times (\mathbb{I}_n - \lambda E_n^{ij})$: qu'en concluez-vous ?

8 0 2 b Transvections de lignes

On s'intéresse à la fonction :

$$T_\lambda^{ij}: \begin{array}{ccc} \mathbb{A}^{n\times p} & \to & \mathbb{A}^{n\times p} \\ M & \mapsto & (\mathbb{I}_n + \lambda E_n^{ij}) \times M \end{array}$$

Calculez par exemple l'image par T_λ^{23} de $\begin{pmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{pmatrix}$

Ainsi, $T_\lambda^{ij}(M)$ permet d'obtenir la matrice construite à partir de M en remplaçant la ligne i par elle-même plus λ fois la ligne j.
On note plus commodément cette transformation $L_i \leftarrow L_i \boxplus (\lambda \boxdot L_j)$.
Vous aurez bien noté que $(\mathbb{I}_n + \lambda E_n^{ij})^{-1} = \mathbb{I}_n - \lambda E_n^{ij}$.

8 0 2 c Dilatations de lignes

On veut effectuer l'opération $L_i \leftarrow \lambda \boxdot L_i$. On note

$$\Delta_n^{i,\lambda} = \mathbb{I}_n + (\lambda \boxplus (-1_{\mathbb{A}}))E_n^{ii}$$

Calculez par exemple le produit de $\Delta_3^{2,\lambda}$ par $\begin{pmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{pmatrix}$

Le produit par $\Delta_n^{i,\lambda}$ est une dilatation de ligne.

8 0 2 d Échange de lignes

On considère la matrice $S_n^{ij} = \Delta_n^{j,-1_{\mathbb{A}}} \times (\mathbb{I}_n + E_n^{ij}) \times (\mathbb{I}_n - E_n^{ji}) \times (\mathbb{I}_n + E_n^{ij})$.
Développez ce produit. Que vaut le produit de S_3^{23} par $\begin{pmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{pmatrix}$?
On note cette opération $L_i \leftrightarrow L_j$.

8 0 2 e Opérations sur les lignes

De manière plus générale, une fonction φ de $\mathbb{A}^{n\times p}$ dans lui-même est une **opération sur les lignes** si c'est la composée finie de transvections et de dilatations de lignes.
L'image d'une matrice par φ est donc le produit à gauche par des matrices de transvections ou de dilatations :

$$\varphi : M \mapsto (F_k \times F_{k-1} \cdots \times F_1) \times M$$

avec chaque F_i inversible. La fonction φ est donc elle-même totale bijective et sa réciproque est :

$$\varphi^{-1} : N \mapsto (F_1^{-1} \times F_2^{-1} \cdots \times F_k^{-1}) \times N$$

On en déduit en particulier que l'inverse de $\varphi(\mathbb{I}_n)$ existe et que c'est $\varphi^{-1}(\mathbb{I}_n)$.

8 0 2 f Lien avec les matrices régulières

Voici un théorème important qui nous sera très utile pour calculer l'inverse d'une matrice régulière.

Théorème D - 1

φ étant une opération élémentaire sur les lignes,

$$M \text{ inversible} \leftrightarrow \varphi(M) \text{ inversible}$$

En effet nous savons que $\varphi(M) = \varphi(\mathbb{I}_n) \times M$. Si M est inversible, $\varphi(M)$ s'exprime comme le produit de deux matrices inversibles et elle est donc inversible. Supposons maintenant $\varphi(M)$ inversible, comme $\varphi(\mathbb{I}_n)$ est inversible on obtient :

$$\varphi(\mathbb{I}_n)^{-1} \times \varphi(M) = \varphi(\mathbb{I}_n)^{-1} \times (\varphi(\mathbb{I}_n) \times M)$$

qui se transforme en

$$\varphi(\mathbb{I}_n)^{-1} \times \varphi(M) = \left(\varphi(\mathbb{I}_n)^{-1} \times \varphi(\mathbb{I}_n)\right) \times M = \mathbb{I}_n \times M$$

et en définitive

$$M = \varphi(\mathbb{I}_n)^{-1} \times \varphi(M) = \varphi^{-1}(\mathbb{I}_n) \times \varphi(M)$$

M s'exprimant comme le produit de deux matrices inversibles est inversible.

Théorème D - 2

Si $\varphi_1, \varphi_2, \ldots, \varphi_k$ est une suite d'opérations sur les lignes de M qui transforme M en $\mathbb{I}_n$ alors M est inversible et

$$M^{-1} = \varphi_k(\mathbb{I}_n) \times \varphi_{k-1}(\mathbb{I}_n) \times \ldots \times \varphi_1(\mathbb{I}_n) = \varphi_k \circ \varphi_{k-1} \circ \ldots \circ \varphi_1(\mathbb{I}_n)$$

En effet, si nous avons $\varphi_k \circ \varphi_{k-1} \circ \ldots \circ \varphi_1(M) = \mathbb{I}_n$ alors

$$\varphi_k \circ \varphi_{k-1} \circ \ldots \circ \varphi_1(M) = \mathbb{I}_n = (\varphi_k(\mathbb{I}_n) \times \varphi_{k-1}(\mathbb{I}_n) \times \ldots \times \varphi_1(\mathbb{I}_n)) \times M$$

Nous avons trouvé une matrice carrée qui multipliant M donne $\mathbb{I}_n$, c'est son inverse.
Le théorème précédent nous indique une méthode pour trouver l'inverse de M (s'il existe), nous allons étudier plus en détail cette technique et nous démontrerons plus loin que s'il est impossible de transformer M en $\mathbb{I}_n$ en utilisant les opérations élémentaires sur les lignes, alors M n'est pas inversible.

8 1 Rang d'une matrice

Dans tout ce qui suit nous allons utiliser les opérations élémentaires sur les lignes d'une matrice et on travaille dans $\mathbb{A}^{n\times p}$.

8 1 1 Matrices ligne-équivalentes

Nous dirons que deux matrices M et N sont **ligne-équivalentes** si, et seulement si, il existe une suite finie $(\varphi_i)_{i\in\mathbb{N}_k}$ d'opérations élémentaires sur les lignes de sorte que

$$N = \varphi_k \circ \varphi_{k-1} \circ \ldots \circ \varphi_1(M)$$

Nous écrirons alors $M \overset{\ell}{\equiv} N$. La relation $\overset{\ell}{\equiv}$ est manifestement une relation d'équivalence sur $\mathbb{A}^{n\times p}$, on démontre sans peine que

$$\begin{aligned} &M \overset{\ell}{\equiv} M \\ &M \overset{\ell}{\equiv} N \to N \overset{\ell}{\equiv} M \\ &\left(M \overset{\ell}{\equiv} N \text{ et } N \overset{\ell}{\equiv} C\right) \to M \overset{\ell}{\equiv} C \end{aligned}$$

Nous savons que $\varphi_k \circ \varphi_{k-1} \circ \ldots \circ \varphi_1(M) = \varphi_k \circ \varphi_{k-1} \circ \ldots \circ \varphi_1(\mathbb{I}_n) \times M$, par conséquent nous aurons $M \overset{\ell}{\equiv} N$ s'il existe une matrice R inversible vérifiant

$$N = R \times M$$

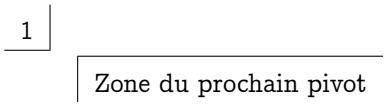
R étant le résultat du produit de matrices du type $\varphi(\mathbb{I}_n)$. Pour obtenir cette matrice R il suffit d'appliquer successivement en parallèle les opérations élémentaires qui transforment M en N sur $\mathbb{I}_n$. Pour cela on utilise un tableau où l'on juxtapose M et $\mathbb{I}_n$, M étant la partie gauche et $\mathbb{I}_n$ la partie droite de ce tableau. Toute opération élémentaire sur les lignes effectuée sur la partie gauche est simultanément effectuée sur la partie droite.

opérations φ	Partie gauche	Partie droite	Remarques
	M	$\mathbb{I}_n$	initialisation du tableau
φ_1	M_1	R_1	$M_1 = \varphi_1(M)$, $R_1 = \varphi_1(\mathbb{I}_n)$
φ_2	M_2	R_2	$M_2 = \varphi_2(M_1)$, $R_2 = \varphi_2(R_1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
φ_i	M_i	R_i	$M_i = R_i \times M$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
φ_k	N	R	$N = R \times M$

8 1 2 L réduite échelonnée

La matrice $M = (m_{ij}) \in \mathbb{A}^{n\times p}(\mathbb{K})$ est dite **ℓ-réduite** (ℓ pour « ligne ») si, et seulement si, elle satisfait aux conditions suivantes :

1. Toutes les lignes nulles (une ligne est nulle si elle ne comporte que des zéros) sont au-dessous des lignes non nulles.
2. Dans chaque ligne non nulle le premier élément non nul est $1_{\mathbb{A}}$ (on lit une ligne de la gauche vers la droite), ce $1_{\mathbb{A}}$ est appelé **pivot** ou élément pivot. La colonne où se trouve ce $1_{\mathbb{A}}$ est appelée colonne pivot et c'est le seul élément non nul de cette colonne.
3. Si, **de plus**, les pivots apparaissent en ordre croissant par numéro de ligne **et** numéro de colonne, on dit que M est **ℓ-réduite échelonnée** (en abrégé ℓré ou LRé).



Donnons quelques exemples de matrices entières :

$$\begin{pmatrix} 0 & \boxed{1} & 2 & 0 \\ \boxed{1} & 0 & 3 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ est } \ell\text{-réduite non échelonnée}$$

$$\begin{pmatrix} 0 & \boxed{1} & 0 \\ 0 & 2 & 0 \\ 0 & 0 & \boxed{1} \end{pmatrix} \text{ n'est pas } \ell\text{-réduite}$$

$$\begin{pmatrix} \boxed{1} & -2 & 0 \\ 0 & 0 & \boxed{1} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{ est } \ell\text{-réduite échelonnée}$$

Théorème D - 3

Pour toute matrice $M \in \mathbb{A}^{n\times p}$, il existe une unique matrice ℓ-réduite échelonnée $N \in \mathbb{A}^{n\times p}$ telle que $M \overset{\ell}{\equiv} N$, N est appelée la ℓ-réduite échelonnée de M que l'on peut noter par $N = \ell\text{ré}(M)$.

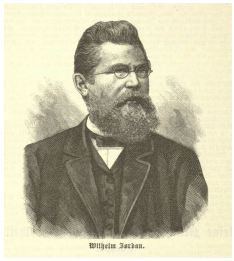
La démonstration de ce théorème se fait par récurrence sur n et elle est un peu difficile.

Le **rang** de M, noté $\text{rang}(M)$, est le nombre de lignes non nulles de sa ℓ-réduite échelonnée, c'est donc aussi le nombre de pivots da sa ℓré. M est dite de plein rang si son rang est égal à son nombre de lignes.

Nous énonçons les évidences (conséquences directes de la définition) :

1. Si $M \in \mathbb{A}^{n\times p}$ le rang de M est inférieur ou égal à n **et** à p.
2. Si $M \in \mathbb{A}^{n\times n}$ et si $\text{rang}(M) = n$ alors sa ℓ-réduite échelonnée est $\mathbb{I}_n$ et nous avons précédemment démontré que dans ce cas M est inversible.
3. Deux matrices ligne-équivalentes ont la même ℓ-réduite échelonnée et ont donc même rang.
4. Si M' est une matrice extraite de M, on a $\text{rang}(M') \leq \text{rang}(M)$.

8 2 Algorithme Fang-Tcheng



Wilhelm JORDAN
(1842-1899)

Notre eurocentrisme préfère nommer cet algorithme GAUSS-JORDAN...
Nous allons le présenter sur un exemple. Soit à chercher la ℓ-réduite échelonnée de la matrice

$$A = \begin{pmatrix} 2 & 5 & -2 & 3 \\ 3 & 6 & 3 & 6 \\ 1 & 2 & -1 & 2 \end{pmatrix}$$

Nous allons faire apparaître les pivots ordonnés par numéro de ligne et numéro de colonne, cela veut dire que si un pivot (qui sera toujours égal à 1) est obtenu à la ligne i et colonne j, le pivot suivant sera au moins en ligne $i+1$ et en colonne $j+1$ et on s'imposera de trouver la solution minimale en numéro de ligne et numéro de colonne.

— **Première étape.** On repère l'élément de la première colonne qui est le plus grand en valeur absolue (il peut y avoir plusieurs choix). Si le résultat est nul (la première colonne ne contient que des zéros), la première colonne ne peut être une colonne pivot et on passe à la colonne suivante. Ici c'est 3 qui se trouve sur la deuxième ligne première colonne. On permute alors la première ligne avec la deuxième ligne et on obtient

$$A_1 = \begin{pmatrix} 3 & 6 & 3 & 6 \\ 2 & 5 & -2 & 3 \\ 1 & 2 & -1 & 2 \end{pmatrix}$$

On divise tous les éléments de la première ligne par 3 :

$$A_2 = \begin{pmatrix} \boxed{1} & 2 & 1 & 2 \\ 2 & 5 & -2 & 3 \\ 1 & 2 & -1 & 2 \end{pmatrix}$$

On fait apparaître ensuite des zéros sous le premier 1 de la première colonne en utilisant les opérations $L_2 \leftarrow L_2 - 2L_1$ et $L_3 \leftarrow L_3 - 1L_1$:

$$A_3 = \begin{pmatrix} \boxed{1} & 2 & 1 & 2 \\ 0 & 1 & -4 & -1 \\ 0 & 0 & -2 & 0 \end{pmatrix}$$

La première colonne est une colonne pivot et elle ne devra pas être modifiée par la suite.

— **Deuxième étape.** On repère dans la deuxième colonne (en fait la colonne qui suit la dernière colonne pivot obtenue), à partir de la deuxième ligne (en fait à partir du numéro de ligne qui suit le numéro de la ligne qui a donné le dernier pivot), le plus grand élément en valeur absolue (si on obtient zéro on passe à la colonne suivante, etc···). Ici on obtient 1 qui se trouve sur la deuxième ligne et deuxième colonne et il n'y a pas de permutation de lignes à faire. Maintenant il faut faire apparaître des zéros dans la colonne pivot en ligne 1 et en ligne 3. On utilise les opérations $L_1 \leftarrow L_1 - 2L_2$ et $L_3 \leftarrow L_3 - 0L_2$ (qui ne sert à rien) ; on obtient :

$$A_4 = \begin{pmatrix} \boxed{1} & 0 & 9 & 4 \\ 0 & \boxed{1} & -4 & -1 \\ 0 & 0 & -2 & 0 \end{pmatrix}$$

La deuxième colonne est une colonne pivot, elle ne devra pas être modifiée dans la suite.

— **Troisième étape.** On repère dans la colonne qui suit la dernière colonne pivot obtenue et à partir de la ligne qui suit la ligne qui a donné le dernier pivot le plus grand élément en valeur asolue. Ici c'est -2 qui se trouve sur la troisième ligne et troisième colonne, la troisième colonne est alors la troisième colonne pivot. Il n'y a pas de permutation de lignes à faire, nous n'avons qu'à faire apparaître un 1 à la place de -2 puis faire apparaître des zéros

dans la colonne pivot en conservant évidemment le pivot 1. Appliquons à A_4 l'opération $L_3 \leftarrow -\frac{1}{2}L_3$

$$A_5 = \begin{pmatrix} \boxed{1} & 0 & 9 & 4 \\ 0 & \boxed{1} & -4 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

puis appliquons $L_1 \leftarrow L_1 - 9L_3$ et $L_2 \leftarrow L_2 + 4L_3$

$$A_6 = \begin{pmatrix} \boxed{1} & 0 & 0 & 4 \\ 0 & \boxed{1} & 0 & -1 \\ 0 & 0 & \boxed{1} & 0 \end{pmatrix}$$

Nous venons d'obtenir la ℓ-réduite échelonnée de A, A est de rang 3.

Nous avons utilisé, ici, la méthode dite du pivot partiel en cherchant dans chaque colonne le plus grand élément en valeur absolue. Si nous avions choisi de prendre le premier élément rencontré non nul, en vertu de l'unicité de la ℓ-réduite échelonnée, nous aurions obtenu le même résultat final. Il est conseillé, en programmation, d'utiliser la méthode dite pivot partiel pour éviter une trop grande propagation des erreurs lors des divisions par des petits nombres. Si on fait les calculs à la main il vaut mieux se contenter de choisir, tant que c'est possible, des nombres sympathiques.

Théorème D - 4 A est une matrice carrée d'ordre n inversible si, et seulement si, le rang de A est égal à n.

Nous avons en fait déjà démontré une partie de ce théorème mais, vu son importance, nous allons reprendre ce qui a été dit. Supposons donc que le rang de A est égal à n, dans ces conditions la ℓ-réduite échelonnée de A est $\mathbb{I}_n$, cela signifie que $A \overset{\ell}{\equiv} \mathbb{I}_n$ et donc qu'il existe A' vérifiant $A' \times A = \mathbb{I}_n$, A' est l'inverse de A. Supposons maintenant que la rang de A est strictement inférieur à n, il est alors sûr que la dernière ligne de la ℓ-réduite échelonnée de A est une ligne nulle. Notons B cette ℓ-réduite échelonnée, nous savons que A inversible équivaut à écrire que B est inversible puisque $A \overset{\ell}{\equiv} B$. Supposons B inversible, il existe alors B' vérifiant $B \times B' = \mathbb{I}_n$ or cette égalité est impossible car la dernière ligne de B étant nulle, la dernière ligne du produit $B \times B'$ sera aussi nulle et donc distincte de la dernière ligne de $\mathbb{I}_n$. Nous venons de démontrer

$$\text{rg}(A) < n \rightarrow A \text{ non inversible}$$

et donc A inversible $\rightarrow \text{rg}(A) = n$.

Pratique : pour rechercher la matrice inverse de A, si elle existe, on applique simultanément sur $\mathbb{I}_n$ les opérations faites pour déterminer la ℓ-réduite échelonnée de A. Pour cela on considère le tableau

$$[A|\mathbb{I}_n]$$

on applique l'algorithme Fang Tcheng tableau, et chaque opération faite sur la partie gauche est reproduite sur la partie droite. Si la ℓ-réduite de A est la matrice $\mathbb{I}_n$ (le résultat de la partie gauche est $\mathbb{I}_n$) alors A est inversible et sa matrice inverse est donnée par la partie droite. Si la partie gauche ne donne pas $\mathbb{I}_n$, A n'est pas inversible, son rang est strictement inférieur à n.

8 3 Résolution de systèmes

8 3 1 Généralités

Gabriel CRAMER
(1704-1752)

On appelle **système de n équations linéaires à p inconnues** dans $\mathbb{A}$ tout système de la forme :

$$(S):\begin{cases}\sum_{j=1}^{p} a_{i,j}x_j = b_i \\ i\in\{1,2,\cdots,n\} \\ a_{i,j} \text{ et } b_i \in \mathbb{K}\end{cases}$$

Soit aussi

$$(S):\begin{cases}a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,p}x_p = b_1 \\ \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,p}x_p = b_p\end{cases}$$

$x_1, x_2, \cdots, x_p$ sont les p inconnues, les $a_{i,j}$ sont appelés les coefficients du système (S) et les b_i sont appelés les seconds membres (ce sont aussi des coefficients du système (S)).
Notons $A = (a_{i,j}) \in \mathbb{A}^{n\times p}$ (A est appelée la matrice du système).

$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}$ est la matrice colonne des inconnues et $B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$ est la matrice colonne des seconds membres, le système (S) s'écrit matriciellement :

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,p} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,p} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

soit aussi

$$A \times X = B \text{ ou } {}^tX \times {}^tA = {}^tB$$

Résoudre le système (S) c'est chercher tous les p-uplets $(x_1, x_2, \cdots, x_p)$ de $\mathbb{A}^p$ qui vérifient simultanément les n équations. C'est aussi chercher toutes les matrices $X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \in \mathbb{A}^{p\times 1}$ vérifiant l'égalité matricielle $A \times X = B$. C'est pourquoi, dans ce qui suit, nous serons souvent amenés à confondre le p-uplet $(x_1, x_2, \cdots, x_p)$ avec la matrice colonne $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}$. Attention, en informatique et plus particulièrement en réseaux, le p-uplet $(x, x_2, \cdots, x_p)$ est aussi noté matriciellement par

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_p \end{pmatrix}$$

qui n'est autre que la transposée de X.
Le système est dit **homogène** si, et seulement si, tous les seconds membres (les coefficients b_i) sont nuls. Dans ce cas le système admet forcément au moins une solution : la solution nulle ou solution banale qui est le p-uplet

$$(0, 0, \cdots, 0) \in \mathbb{A}^p$$

ou bien la matrice nulle $0_{p,1}$ si on note matriciellement cette solution.
Le système $A \times X = B$ est dit **régulier** ou de **Cramer** si, et seulement si, il a autant d'équations que d'inconnues et si A est inversible (le système ayant autant d'équations que d'inconnues on a $n = p$ et A est une matrice carrée). A étant inversible, A^{-1} existe et

$$\begin{aligned} A \times X &= B \text{ se transforme en} \\ A^{-1} \times (A \times X) &= A^{-1} \times B, \text{ c'est-à-dire} \\ X &= A^{-1} \times B \end{aligned}$$

Le système admet donc au moins une solution qui est $A^{-1} \times B$. Nous allons prouver qu'il n'y en a pas d'autre. Pour cela supposons l'existence d'une autre solution X', nous avons alors simultanément

$$A \times X = B \text{ et } A \times X' = B$$

En retranchant membre à membre on obtient :

$$A \times (X - X') = 0_{n,1}$$

Multiplions maintenant les deux membres par A^{-1}

$$\begin{aligned} A^{-1} \times (A \times (X - X')) &= A^{-1} \times 0_{n,1} = 0_{n,1} \\ (A^{-1} \times A) \times (X - X') &= 0_{n,1} \\ \mathbb{I}_n \times (X - X') &= 0_{n,1} = X - X' \text{ et pour finir} \\ X &= X' \end{aligned}$$

Nous retiendrons que si le système $A \times X = B$ est tel que A est inversible (il est nécessaire que A soit carrée et donc que le système possède autant d'équations que d'inconnues) alors il admet une unique solution qui est

$$X = A^{-1} \times B$$

8 3 2 Systèmes équivalents et résolution

Deux systèmes linéaires (S) et (S') sont **équivalents** si, et seulement si, ils possèdent le même ensemble solution.
Nous allons définir trois opérations élémentaires :

1. Multiplication d'une équation par un scalaire (un élément de $\mathbb{A}$) non nul. Si E_i est la i^{e} équation, le résultat de la multiplication de E_i par $\alpha \in \mathbb{A}$ est noté αE_i et nous écrirons $E_i \leftarrow \alpha E_i$.

$$\begin{aligned} E_i &: \sum_{j=1}^{p} a_{i,j} x_j = b_i \\ \alpha E_i &: \sum_{j=1}^{p} \alpha a_{i,j} x_j = \alpha b_i \end{aligned}$$

   Pour retrouver l'équation initiale il suffit de multiplier l'équation obtenue par $\frac{1}{\alpha}$ et par conséquent le système obtenu par cette opération est équivalent au système initial.

2. Permutation de deux équations, nous noterons $E_i \longleftrightarrow E_j$ l'opération qui consiste à permuter l'équation numéro i avec l'équation numéro j. Pour retrouver le système initial il suffit d'appliquer de nouveau cette opération, cela nous permet d'affirmer que cette opération transforme un système en un système équivalent.

3. Ajout à une équation une autre équation multipliée par un scalaire : si on ajoute à l'équation E_i l'équation $E_{h \neq i}$ multipliée par β nous noterons $E_i \leftarrow E_i + \beta E_h$

$$\begin{cases} E_i : \sum_{j=1}^{p} a_{i,j} x_j = b_i \\ E_h : \sum_{j=1}^{p} a_{h,j} x_j = b_h \\ E_i \leftarrow E_i + \beta E_h : \sum_{j=1}^{p} (a_{i,j} + \beta a_{h,j}) x_j = b_i + \beta b_h \end{cases}$$

   Pour retrouver l'équation de départ il suffit d'appliquer au résultat l'opération $E_i \leftarrow E_i - \beta E_h$ et cette opération, comme les précédentes, transforme un système en un système équivalent.

Nous allons maintenant passer à la résolution : nous pouvons écrire le système

$$(S):\begin{cases}\sum_{j=1}^{p} a_{i,j}x_j = b_i \\ i \in \{1,2,\cdots,n\} \\ a_{i,j} \text{ et } b_i \in \mathbb{A}\end{cases} \quad \text{sous la forme } [A \mid B]$$

où A est la matrice du système et B la matrice colonne des seconds membres. Nous notons T cette matrice (ou ce tableau), le nombre de lignes de T est égal au nombre de lignes de A soit aussi le nombre d'équations de (S), le nombre de colonnes de T est égal au nombre de colonnes de $A+1$ soit aussi le nombre d'inconnues +1. Les opérations élémentaires sur les équations sont alors des opérations élémentaires sur les lignes de ce tableau, les opérations se faisant simultanément sur la partie gauche et sur la partie droite, c'est-à-dire sur le tableau T. Appliquons à ce tableau l'algorithme de Gauss-Jordan pour obtenir la ℓ-réduite échelonnée de A (on ne cherchera pas pour le moment à faire apparaître un pivot dans la dernière colonne du tableau) et notons R cette ℓ-réduite échelonnée. Nous notons le résultat

$$T' = [R \mid H], R = \ell\text{ré}(A)$$

Le système est alors devenu $R \times X = H$. Comme on a $A \overset{\ell}{\equiv} R$ et $B \overset{\ell}{\equiv} H$, il existe P (rappel, P est inversible) telle que

$$R = P \times A \qquad H = P \times B$$

Si nous notons U une solution du système (S), U vérifie

$$A \times U = B$$

et en multipliant les deux membres à gauhe par P on obtient

$$\begin{aligned} P \times A \times U &= P \times B \text{ soit} \\ R \times U &= H \end{aligned}$$

De même toute solution de $R \times X = H$ est solution de $A \times X = B$, en effet notons de même U une solution de $R \times X = H$, on a

$$\begin{aligned} R \times U &= H \text{ donc} \\ P \times A \times U &= P \times B \end{aligned}$$

et en multipliant les deux membres à gauche par P^{-1} on obtient

$$A \times U = B$$

On a bien prouvé que les deux systèmes $A \times X = B$ et $R \times X = H$ sont équivalents. Il nous reste à résoudre le système $R \times X = H$.
Continuons, si besoin, le calcul de la ℓ-réduite échelonnée de T et notons T'' le résultat :

$$T' = [R \mid H], T'' = [R \mid H']$$

Si on arrive à faire apparaître un pivot dans la dernière colonne, les opérations sur les lignes ne modifieront pas la partie gauche car la partie gauche de cette ligne pivot est constituée de zéros.

— Si le système est homogène (tous les seconds membres sont nuls) on a forcément $T' = T''$ et le système admet, rappelons-le, au moins la solution nulle.

— Si le rang de T est supérieur au rang de R (c'est donc que le rang de T est d'une unité supérieure au rang de A, on ne peut créer qu'un seul nouveau pivot au maximum) cela signifie qu'il y a plus de lignes nulles dans R que dans T'' et par conséquent que le système $R \times X = H'$ contient l'équation du type

$$0x_1 + 0x_2 + \cdots + 0x_p = 1$$

ou que le système $R \times X = H$ contient au moins une équation du type

$$0x_1 + 0x_2 + \cdots + 0x_p = h \text{ avec } h \neq 0$$

ce qui est impossible et le système n'admet pas de solution.

— Si $\text{rang}(T) = \text{rang}(R) = r$, c'est que $T' = T''$ et la résolution de $A \times X = B$ équivaut à la résolution des r équations non nulles de $R \times X = H$. Deux cas peuvent se présenter :

1. $r = p$, le système à résoudre est alors de la forme :

$$\begin{cases} x_1 & & & = h_1 \\ & x_2 & & = h_2 \\ & & \vdots & \vdots \\ & & x_p & = h_p \end{cases}$$

et il est résolu, il y a unicité de la solution.

2. $r < p$. Nous appelons inconnues pivots ou inconnues principales (IP) les r inconnues $x_{j_1}, x_{j_2}, \cdots, x_{j_r}$ correspondant aux r colonnes pivots $j_1, j_2, \cdots, j_r$ de R. Les $p-r$ autres inconnues sont appelées inconnues non principales (INP), certains les appellent aussi des inconnues ou des variables libres ou encore des paramètres. Pour résoudre le système la méthode consiste à faire passer dans les seconds membres les inconnues non principales, le système devient alors un sytème de Cramer résolu dont les solutions dépendent des $p-r$ inconnues non principales (ces inconnues non principales sont, à ce stade, considérées comme des paramètres), **il y a donc une infinité de solutions** ; si l'on désire une solution particulière, il suffit de donner des valeurs arbitraires aux inconnues non principales.

$$\begin{cases} x_{j_1} & & & = h_{j_1} + e_{j_1} \\ & x_{j_2} & & = h_{j_2} + e_{j_2} \\ & & \vdots & \vdots \\ & & x_{j_r} & = h_{j_r} + e_{j_r} \end{cases}$$

où les e_{j_i} sont des expressions linéaires des $(p-r)$ INP.

Il faut remarquer que si l'on modifie l'ordre d'écriture des inconnues nous n'obtiendrons pas forcément les mêmes inconnues principales et non principales mais l'ensemble solution sera évidemment le même.

9 Back to code

9.1 Inverse d'une matrice par la méthode de Gauss-Jordan

9.1.1 Opérations élémentaires

Pour effectuer une combinaison linéaire des lignes, on va créer une fonction :
`comb_ligne(ki,kj,M,i,j)`
qui renverra la matrice construite à partir de M en remplaçant la ligne L_j par $k_j \times L_j + k_i \times L_i$.

```
1    def comb_lignes(self,ki,kj,i,j):
2        """Li <- ki*Li + kj * Lj"""
3        f = self.F
4        g = lambda r,c : ki*f(i,c) + kj*f(j,c) if r == i else f(r,c)
5        return Mat(self.D,g)
```

Recherche

On crée également une fonction `mult_ligne(k,M,j)`: qui renverra la matrice construite à partir de M en remplaçant la ligne L_j par $k \times L_j$.

9.1.2 Calcul de l'inverse étape par étape

On commence par créer une fonction qui renvoie une matrice où se juxtaposent la matrice initiale et la matrice identité :
Créons une matrice :

```
>>> M = list2mat([[1,0,1],[0,1,1],[1,1,0]])
```

$$M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Complétons-la par la matrice identité de même taille :

```
>>> T = M.cat_carre_droite()
>>> T
 1  0  1  1  0  0
 0  1  1  0  1  0
 1  1  0  0  0  1
```

Recherche

Complétez...

```
    def cat_carre_droite(self):
        """
        Colle l'identité à droite pour la méthode de GJ

        """
        ???
```

On effectue $L_3 \leftarrow L_3 - L_1$:

```
>>> T = T.comb_lignes(1,-1,2,0)
>>> T
 1  0  1  1  0  0
 0  1  1  0  1  0
 0  1 -1 -1  0  1
```

puis $L_3 \leftarrow L_3 - L_2$:

```
>>> T = T.comb_lignes(1,-1,2,1)
>>> T
 1  0  1  1  0  0
 0  1  1  0  1  0
 0  0 -2 -1 -1  1
```

puis $L_3 \leftarrow \frac{1}{2}L_3$:

```
>>> T = T.mult_ligne(-0.5,2)
>>> T
 1  0  1  1   0   0
 0  1  1  0   1   0
 0  0  1  0.5 0.5 -0.5
```

On effectue $L_1 \leftarrow L_1 - L_3$:

```
>>> T = T = T.comb_lignes(1,-1,1,2)
>>> T
 1  0  1  1   0   0
 0  1  0 -0.5 0.5 0.5
 0  0  1  0.5 0.5 -0.5
```

et enfin $L_2 \leftarrow L_2 - L_3$:

```
>>> T = T.comb_lignes(1,-1,0,2)
>>> T
   1    0    0  0.5 -0.5  0.5
   0    1    0 -0.5  0.5  0.5
  -0   -0    1  0.5  0.5 -0.5
```

Il ne reste plus qu'à extraire la moitié droite du tableau qui sera l'inverse cherchée à l'aide d'une petite fonction :

Complétez...

Recherche

```
def extrait_carre_droite(self):
    """
    Extrait le carré de droite d'un tableau de GJ

    """
    ???
```

alors :

```
>>> iM = T.extrait_carre_droite()
>>> iM
 0.5 -0.5  0.5
-0.5  0.5  0.5
 0.5  0.5 -0.5
```

On vérifie que c'est bien la matrice inverse de M :

```
>>> iM * M
   1    0    0
   0    1    0
   0    0    1
```

9 1 3 Réduction sous-diagonale d'une matrice

On pourrait calculer l'inverse d'une matrice en généralisant la méthode précédente mais cela s'avèrerait beaucoup trop gourmand en temps.
Nous allons dans un premier temps trigonaliser la matrice en effectuant des opérations élémentaires.
Pour éviter de faire trop de transformations, nous allons petit à petit réduire la taille de la matrice à trigonaliser en ne considérant que le carré inférieur droit situé sous le pivot courant et mémoriser chaque ligne obtenue dans une matrice.
Cela permet également de généraliser l'emploi de la méthode de Gauss à des matrices non carrées pour la résolution de systèmes par exemple. Il faut donc faire attention maintenant à utiliser le minimum entre le nombre de lignes et le nombre de colonnes de la matrice.
On place des « mouchards » pour comprendre l'évolution du code.

```
    def triangle(self):
        """ renvoie la triangulation d'une matrice de haut en bas """
        [r,c] = self.D
        m     = min(r,c)
        S     = self
        T     = zeros(r,c)
        while m > 0:
            NoLigne = 0
            while S[NoLigne, 0] == 0 and (NoLigne < m - 1):
                    NoLigne += 1
            if S[NoLigne, 0] != 0:
```

```
                pivot = S[NoLigne,0]
                for k in range(1,m):
                    if S[k,0] != 0:
                        S = S.comb_lignes(pivot, -S[k,0],k,0)
                        print("pivot = "+str(pivot))
                        print("S dans for :")
                        print(S)
            T = T.remplace_ligned(r - m,S.F)
            print("Évolution de T :")
            print(T)
            S = S.dswap(NoLigne)
            m -= 1
        return T
```

Par exemple :

```
In [1]: M
Out[1]:
 1  2  3
 4  5  6
 7  8  8

In [2]: M.triangle()
pivot = 1
S dans for :
 1  2  3
 0 -3 -6
 7  8  8
pivot = 1
S dans for :
 1  2  3
 0 -3 -6
 0 -6 -13
Évolution de T :
 1  2  3
 0  0  0
 0  0  0
pivot = -3
S dans for :
-3 -6
 0  3
Évolution de T :
 1  2  3
 0 -3 -6
 0  0  0
Évolution de T :
 1  2  3
 0 -3 -6
 0  0  3
```

Nous avons besoin de deux fonctions intermédiaires, une remplissant le tableau T et l'autre réduisant S.

```
    def decoupe(self,i):
        """
        Fonction interne à triangle qui retire la 1ère ligne et la 1ère colonne

        """
        [lig, col], f = self.D, self.F
        return Mat([lig-1,col-1],lambda r,c : f(r+1,c+1))

    def remplace_ligned(self,i,g):
        """
        Fonction interne à triangle qui remplace dans la ligne i
```

```
        les coefficients à partir de la colonne i par ceux du tableau S
        """
        [lig, col], f = self.D, self.F
        h = lambda r,c: g(r-i,c-i) if r == i and c >= i else  f(r,c)
        return Mat([lig,col],h)
```

Recherche

Déterminez une fonction qui calcule rapidement le rang d'une matrice.

9 1 4 Calcul du déterminant

On triangularise la matrice et le déterminant est égal au produit des éléments diagonaux à un détail près : il ne faut pas oublier de tenir compte de la parité du nombre d'échanges de lignes ainsi que des multiplications des lignes modifiées par combinaisons.

Créez une fonction qui calcule le déterminant :

Recherche

```
    def det(self):
        """ renvoie le déterminant de self par Gauss-Jordan """
        ????
```

C'est assez efficace pour du Python basique : 555 ms pour un déterminant d'une matrice de taille 200.

```
In [1]: M = unite(200)
In [2]: %timeit M.det()
1 loops, best of 3: 555 ms per loop
```

9 1 5 Calcul de l'inverse d'une matrice

La trigonalisation est assez rapide du fait de la réduction progressive de S. L'idée est alors de trigonaliser de bas en haut puis de haut en bas pour arriver à une matrice diagonale ce qui sera plus efficace qu'un traitement de tout le tableau en continu.
On peut utiliser la fonction `det` qui est rapide et permet de ne pas prendre trop de précautions ensuite sachant que la matrice est inversible.

```
def inverse(self):
    return self.cat_carre_droite().triangle().diago_triangle().extrait_carre_droite()
```

Le tout est de construire cette méthode `diago_triangle` qui va trigonaliser le triangle en le parcourant de bas en haut.
On transforme légèrement les fonctions intermédiaires précedentes pour les adapter au nouveau parcours :

```
    def decoupe_bas(self):
        """
        Fonction interne à diago_triangle qui retire la dernière ligne et la
        colonne de même numéro de S

        """
        [lig, col], f = self.D, self.F
        #g = lambda r,c: f(lig-1,c) if r == lig else f(i,c) if r == lig-1 else f(r,c)
        g = lambda r,c: f(r,c) if c < lig - 1 else f(r,c+1)
        return Mat([lig-1,col-1],lambda r,c : g(r,c))

    def remplace_ligneg(self,i,g):
        """
        Fonction interne à diago_triangle qui remplace dans la ligne i
        les coefficients à partir de la colonne i par ceux du tableau S
```

```
        """
        [lig, col], f = self.D, self.F
        h = lambda r,c: g(r,c - (lig - 1) + i) if r == i and c >= i else  f(r,c)
        return Mat([lig,col],h)
```

Recherche

Construisez `diago_triangle`

```
def diago_triangle(self):
    ????
```

Recherche

On considère la matrice

$$A = \begin{pmatrix} 10^{-20} & 0 & 1 \\ 1 & 10^{20} & 1 \\ 0 & 1 & -1 \end{pmatrix}$$

Quel est son rang ? Quel est son déterminant ? Utilisez les fonctions précédentes puis du papier et un crayon...

10 Manipulation d'images

Dans cette section, des mathématiques concrètes vont nous permettre de réduire, agrandir, assombrir, éclaircir, compresser, bruiter, quantifier,... une photo. Pour cela, il existe des méthodes provenant de la théorie du signal et des mathématiques continues. Nous nous pencherons plutôt sur des méthodes plus légères basées sur l'algèbre linéaire et l'analyse matricielle. Une image sera pour nous une matrice carrée de taille 2^9 à coefficients dans $[\![0, 2^9 - 1]\!]$. Cela manque de charme ? C'est sans compter sur Lena qui depuis quarante ans rend les maths sexy (la population hantant les laboratoires mathématiques et surtout informatiques est plutôt masculine ...).

Nous étudierons pour cela la « Décomposition en Valeurs Singulières » qui apparaît de nos jours comme un couteau suisse des problèmes linéaires : en traitement de l'image et de tout signal en général, en reconnaissance des formes, en robotique, en statistique, en étude du langage naturel, en géologie, en météorologie, en dynamique des structures, etc.

Dans quel domaine travaille-t-on alors : algèbre linéaire, analyse, probabilités, topologie,... ? Un peu de tout cela et d'autres choses encore : cela s'appelle...la mathématique.

10.1 Lena

Nous allons travailler avec des images qui sont des matrices de niveaux de gris. Notre belle amie Léna sera représentée par une matrice carrée de taille 2^9 ce qui permet de reproduire Léna à l'aide de $2^{18} = 262\,144$ pixels. Léna prend alors beaucoup de place. Nous allons tenter de compresser la pauvre Léna sans pour cela qu'elle ne perde sa qualité graphique. Une des méthodes les plus abordables est d'utiliser la décomposition d'une matrice en valeurs singulières.

C'est un sujet extrêmement riche qui a de nombreuses applications. L'algorithme que nous utiliserons (mais que nous ne détaillerons pas) a été mis au point par deux très éminents chercheurs en 1965 (Gene Golub, états-unien et William Kahan, canadien, père de la norme IEEE-754). Il s'agit donc de mathématiques assez récentes, au moins en comparaison avec votre programme...

La petite histoire dit que des chercheurs américains de l'University of Southern California étaient pressés de trouver une image de taille 2^{18} pixels pour leur conférence. Passe alors un de leurs collègues avec, en bon informaticien, le dernier Playboy sous le bras. Ils décidèrent alors d'utiliser le poster central de la Playmate comme support...

La photo originale est ici : http://www.lenna.org/full/len_full.html mais nous n'utiliserons que la partie scannée par les chercheurs, de taille $5.12\text{in} \times 5.12\text{in}$...

10 2 Environnement de travail

Nous travaillerons dans Pylab avec l'option `pylab` qui charge les bibliothèques nécessaires à la manipulation d'images (en fait, pylab charge les bibliothèques pour se retrouver dans un environnement proche de celui de logiciels comme MatLab ou Scilab).

```
$ ipython --pylab
```

Les images sont alors sous forme d'`array` de la bibliothèque numpy.
Nous allons créer quelques outils pour continuer à travailler malgré tout avec notre classe `Mat`.

```
def array2mat(tab):
    """ convertit un array de numpy en objet de type Mat"""
    dim = list(tab.shape)
    return Mat(dim,  lambda i,j : tab[i,j])

def mat2array(mat):
    """ convertir un objet de type Mat en array de numpy """
    return np.fromfunction(mat.F,tuple(mat.D),dtype = int)

def montre(mat):
    """ permet de visualiser les images dans un terminal """
    return imshow(mat2array(mat), cmap = cm.gray)
```

Python contient un tableau carré de taille 512 contenant les niveaux de gris représentant Lena. Il suffit de cherger les bonnes bibliothèques :

```
from scipy import misc

# lena sous forme d'un objet de type Mat
matlena = array2mat(misc.lena())
```

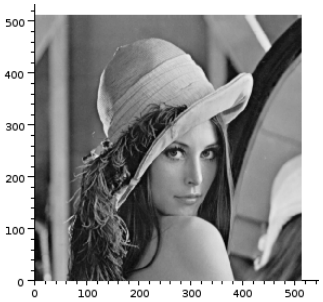
Lena est bien un carré de 512×512 pixels :

```
In [9]: matlena.D
Out[9]: [512, 512]
```

On peut voir Lena :

```
In [3]: montre(matlena)
Out[3]: <matplotlib.image.AxesImage at 0x7f06c6d0c7b8>
```

et on obtient :



On peut préférer travailler sur une photo de format jpg (qui sera à l'envers) ou png (qui sera à l'endroit). Python la transforme en matrice avec la fonction imread. La matrice obtenue est en RVB : chaque coefficient est un vecteur (R,V,B). On la convertit en niveau de gris par exemple en calculant la moyenne des trois couleurs.
On récupère d'abord une image :

```
In [4]: from urllib.request import urlretrieve

In [5]: urlretrieve('addresse de la photo en ligne','photo.png')
Out[5]: ('photo.png', <http.client.HTTPMessage at 0x7f3aeeb5fa20>)
```

et on la transforme en matrice :

```
lena_face_couleur = imread('lena_face.jpeg')

def rvb_to_gray(m):
    r = len(m)
    c = len(m[0])
    return(np.array([[m[i,j][0] for j in range(c)] for i in range(r)]))

lena_face_array = rvb_to_gray(lena_face_couleur)

lena_face = array2mat(lena_face_array)
```

En exclusivité mondiale, voici Lena de face :

```
In [3]: montre(lena_face)
Out[3]: <matplotlib.image.AxesImage at 0x7f6d8c68cc88>
```

et on obtient :

10 3 Manipulations basiques

Comment obtenir les deux images de gauche sachant que l'image originale est à droite :

Sachant que l'échelle de gris varie entre 0 et 255, comment obtenir les images suivantes :

On peut modifier le contraste en « mappant » par une fonction croissant plus rapidement ou plus lentement que la fonction identité sur [0 ;255] vers [0 ;255].
Comment obtenir l'image en négatif de Lena ?

10 4 Résolution

Une matrice de taille 2^9 contient 2^{18} entiers codés entre 0 et 2^8-1 ce qui prend pas mal de place en mémoire. Peut-on être plus économique sans pour cela diminuer la qualité esthétique de la photo ?
La première idée est de prendre de plus gros pixels, c'est-à-dire une matrice plus petite : on extrait par exemple régulièrement un pixel sur k (en choisissant k parmi une puissance de 2 inférieure à 2^9).
Comment obtenir par exemple ces images ?

10 5 Quantification

On peut également réduire le nombre de niveaux de gris en regroupant par exemple tous les niveaux entre 0 et 63 en un seul niveau 0, puis 64 à 127 en 64, 128 à 191 en 128, 192 à 256 en 192.
Par exemple, avec 4, 8 puis 16 niveaux :

Pour cela, on divise chaque coefficient de la matrice par une puissance de 2.
Mais une division d'un nombre par une puissance de 2 revient à faire un décalage de la chaîne de bits représentant ce nombre vers la droite Ce décalage vers la droite est effectué en Python par l'opérateur >>xcom]decalage !gauche@>>. L'image précedente peut donc être obtenue par ? ? ?

10 6 Enlever le bruit

La transmission d'informations a toujours posé des problèmes : voleurs de grands chemins, poteaux télégraphiques sciés, attaques d'indiens, etc.
Claude Elwood SHANNON (1916 - 2001) est un mathématicien-inventeur-jongleur américain qui, suite à son article « *A mathematical theory of communications* » paru en 1948, est considéré comme le fondateur de la *théorie de l'information* qui est bien sûr une des bases de... l'*informatique*.
L'idée est d'étudier et de quantifier l'« information » émise et reçue : quelle est la compression maximale de données digitales ? Quel débit choisir pour transmettre un message dans un canal « bruité » ? Quel est le niveau de sûreté d'un chiffrement ?...
La théorie de l'information de SHANNON est fondée sur des modèles probabilistes : leur étude est donc un préalable à l'étude de problèmes de réseaux, d'intelligence artificielle, de systèmes complexes.
Par exemple, dans le schéma de communication présenté par SHANNON, la source et le destinataire d'une information étant séparés, des perturbations peuvent créer une différence entre le message émis et le message reçu. Ces perturbations (bruit de fond thermique ou accoustique, erreurs d'écriture ou de lecture, etc.) sont de nature *aléatoire* : il n'est pas possible de prévoir leur effet. De plus, le message source est par nature *imprévisible* du point de vue du destinataire (sinon, à quoi bon le transmettre).
SHANNON a également emprunté à la physique la notion d'entropie pour mesurer le désordre de cette transmission d'information, cette même notion d'entropie qui a inspiré notre héros national, Cédric VILLANI...
Mais revenons à Lena. Nous allons simuler une Lena bruitée. Pour cela, nous allons ajouter ajouter des pixels aléatoirement selon une loi de Bernoulli de paramètre p.

```
1  from random import randint
2
3  def ber(p):
4      return 1 if random() < p else 0
5
6  def rand_image(dim,p):
7      return Mat(dim, lambda i,j : randint(0,255)*ber(p))
8
9  def bruit(mat,p):
10      return (mat + rand_image(mat.D,p)).map(lambda x : x % 256)
```

Par exemple, voici Lena bruitée à 0.1 :

Il existe de nombreuses méthodes, certaines très élaborées, permettant de minimiser ce bruit. Une des plus simples est de remplacer chaque pixel par la moyenne de lui-même et de ses 8 autres voisins directs, voire aussi ses 24 voisins directs et indirect, voire plus, ou de la médiane de ces séries de « cercles » concentriques de pixels.
Voici les résultats avec respectivement la moyenne sur des 9 pixels puis 25 pixels et à doite la médiane de carrés de 9 pixels. La médiane est plus efficace que la moyenne !

Do it !!

10 7 Compression par SVD

On en a déjà parlé.
Le problème est d'obtenir cette décomposition. Python heureusement s'en charge grâce à la fonction `linalg.svd(mat)` qui renvoie la décomposition en valeurs singulières sous la forme de trois « array » de `numpy` U, S et tV.
Par souci d'efficacité, nous travaillerons ici uniquement avec les « array » de `numpy`.
Voici les trois niveaux de compression 10, 50 et 100 :

obtenus par :

```
In [144]: imshow(c_svd(lena_face_array,100),cmap = cm.gray)
```

On peut observer la décroissance extrêmement rapide des valeurs singulières :

```
In [1]: U,s,tV = linalg.svd(misc.lena())

In [2]: x = np.arange(len(s))

In [3]: plot(x[:10],s[:10])

In [4]: plot(x[10:50],s[10:50])

In [5]: plot(x[50:],s[50:])
```

On peut également faire une petite animation des images compressées :

```
def anim(im,n):
    for k in range(n):
        plt.imsave("lena_face_svd" + str(1000 + k), c_svd(lena_face_array,k), cmap =
            ↪ cm.gray)
```

Puis ensuite dans un terminal :

```
$ apngasm anim_lena_face.png lena_face_svd*.png 1 5
```

Il faut installer au préalable le petit logiciel apngasm (disponible dans les dépôts Linux ou sur http://sourceforge.net/projects/apngasm/).
On obtient l'image animée suivante :
http://download.tuxfamily.org/tehessinmath/les_images/anim_lena_face.png

Recherche

Détection des bords

Pour sélectionner des objets sur une image, on peut essayer de détecter leurs bords, i.e. des zones de brusque changement de niveaux de gris.
On remplace alors un pixel par une mesure des écarts des voisins immédiats donnée par exemple par :

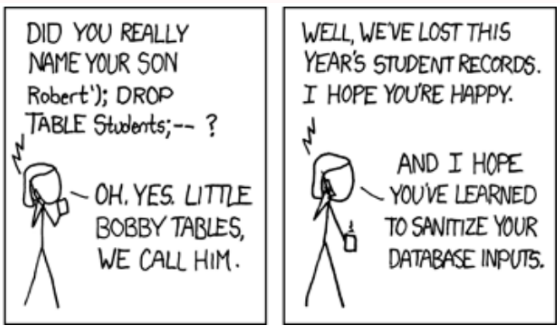
$$\sqrt{(m_{i,j+1}-m_{i,j-1})^2+(m_{i+1,j}-m_{i-1,j})^2}$$

Écrivez une fonction qui trace le contour défini ci-dessus.

THÈME

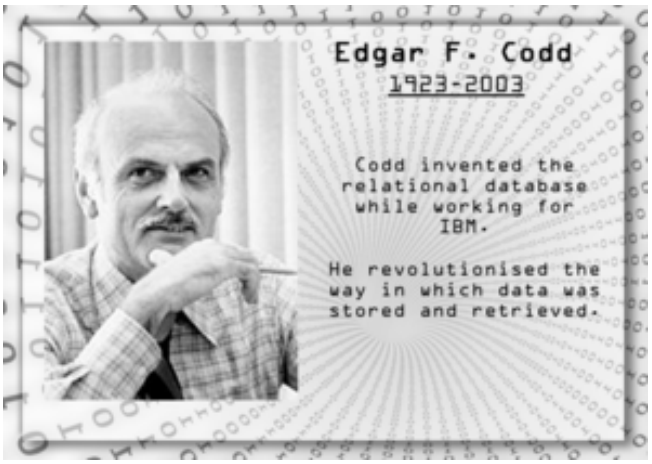
E

SQL : la révision



1 Modèle relationnel

Tout a commencé avec cet homme qui reçut le prix TURING en 1981, notamment suite à son travail initié par un court article publié en 1970 : http ://www.seas.upenn.edu/ zives/03f/cis550/codd.pdf



1.1 Table, relation

Une relation (une table) peut être représentée par un tableau à deux entrées où les lignes sont des n-uplets et les colonnes des attributs.
Considérons une table ALIEN constituée des attributs Nom, Sexe, Planète, NoCabine.
Par exemple :

Nom	Sexe	Planète	NoCabine
Zorglub	M	Trantor	1
Blorx	M	Euterpe	2
Urxiz	F	Aurora	3
Zbleurdite	F	Trantor	4
Darneurane	M	Trantor	5
Mulzo	M	Helicon	6
Zzzzz	F	Aurora	7
Arggh	M	Nexon	8
Joranum	F	Euterpe	9

NoCabine	NoAllee
1	1
2	1
3	1
4	1
5	1
6	2
7	2
8	2
9	2

La relation ALIEN est de degré 4 et de cardinalité 9. On peut choisir comme clé primaire le nom de l'Alien. On aurait pu choisir le numéro de la cabine qui est une autre clé candidate. La relation CABINE a pour clé primaire NoCabine qui est donc une clé étrangère de ALIEN.
Point de vue vocabulaire selon les différents domaines :

Maths	BDD	SGBD
Relation	Table	Fichier
n-uplet	Ligne	Enregistrement
Attribut	Colonne	Champ

Il y a de nombreuses régles d'intégrité assurant la sécurité et l'efficacité d'une BDD mais nous ne les étudierons pas.

1 2 opérateurs relationnels

Opérateurs ensemblistes classiques Union, Intersection, Différence, Produit cartésien ;

Opérateurs relationnels spécifiques Sélection (Restriction), Projection, Jointure ;

Opérateurs dérivés Division, jointure externe, etc.

La description de la construction d'une relation avec les opérateurs relationnels est appelée *requête*.
Quelles sont les contraintes sur les tables sur lesquelles l'union, l'intersection ou la différence peuvent opérer ?
La **sélection** permet d'extraire d'une relation les n-uplets satisfaisant une condition donnée :

$$\mathtt{T} = \mathtt{S(critere)R}$$

La nouvelle relation T a le même en-tête que R. Par exemple :

$$\mathtt{ALIEN\_TRANTOR} = \mathtt{S(Planete = "Trantor")ALIEN}$$

donne :

Nom	Sexe	Planète	NoCabine
Zorglub	M	Trantor	1
Zbleurdite	F	Trantor	4
Darneurane	M	Trantor	5

La **projection** permet de définir une nouvelle relation consistant en l'ensemble de tous les n-uplets de la relation de départ dans laquelle seuls les attributs de projection sont conservés.

$$\mathtt{ALIEN\_PLANETE} = \mathtt{[NomAlien, Planete]ALIEN}$$

donne ?
Une **jointure** consiste à construire une nouvelle relation dont les n-uplets sont la concaténation d'un n-uplet d'une première relation et d'un n-uplet d'une seconde relation vérifiant une condition de jointure.
Les deux relations doivent avoir au moins une colonne commune sémantiquement.
En fait, une jointure est un produit cartésien suivi d'une sélection sur la condition de jointure.

$$\mathtt{R[Att = Att']R'} = \left\{\mathtt{ligne} \in \mathtt{R} \times \mathtt{R'} \mid \mathtt{ligne.Att} = \mathtt{ligne.Att'}\right\} = \mathtt{S[Att = Att'](R \times R')}$$

Considérons la table RESPONSABLE qui donne le nom du responsable de chaque allée :

NoAllee	Nom
1	Seldon
2	Pelorat

Nous allons créer la jointure symétrique entre la table CABINE et la table RESPONSABLE avec comme condition de jointure CABINE.NoAllee = RESPONSABLE.NoAllee :

$$\mathtt{CAB\_RES} = \mathtt{CABINE[CABINE.NoAllee = RESPONSABLE.NoAllee]RESPONSABLE}$$

NoCabine	NoAllee	NoAllee	Nom
1	1	1	Seldon
2	1	1	Seldon
3	1	1	Seldon
4	1	1	Seldon
5	1	1	Seldon
6	2	2	Pelorat
7	2	2	Pelorat
8	2	2	Pelorat
9	2	2	Pelorat

Il y a redondance de l'attribut de jointure. On règle le problème en effectuant une projection. Par exemple : `[NoCabine, Nom] CAB_RES`.

NoCabine	Nom
1	Seldon
2	Seldon
3	Seldon
4	Seldon
5	Seldon
6	Pelorat
7	Pelorat
8	Pelorat
9	Pelorat

La **division cartésienne** entre deux relations est le pendant pour le produit cartésien de la division euclidienne pour le produit des entiers.
Si `R` et `S` sont deux relations, alors `Q=R Div S` est la plus grande relation au sens de l'inclusion vérifiant $Q \times S \subseteq R$: *tous les n-uplets de Q concaténés à ceux de S apparaissent dans R.*
C'est l'opération qui permet de répondre à des questions du type *quelque soit x, trouver y.*
Par exemple, supposons que l'on dispose de la table `PLANETE_ALLEE` associant planète et numéro d'allée de l'Alien orifginaire de la planète.
Comment obtient-on cette table ?

Planete	NoAllee
Trantor	1
Euterpe	1
Aurora	1
Helicon	2
Aurora	2
Nexon	2
Euterpe	2

si on cherche le nom des planètes apparaissant dans toutes les allées :

```
PLANETE_ALLEE[PLANETE_ALLEE.NoAllee/CABINE.NoAllee]CABINE
```

Planete
Euterpe
Aurora

2 Le langage SQL

SQL signifie *Structered Query Language* et a été conçu dans les années 1970 par IBM. La version étudiée est celle introduite en 1992 : SQL2.
L'obtention de données se fait exclusivement avec la clause SELECT qui possède six clauses différentes dont seules les deux premières sont obligatoires :

```
SELECT Liste des attributs à extraire
FROM Liste des tables utilisées
WHERE critère de sélection des n-uplets
GROUP BY liste des attributs qui définissent le regroupement
HAVING critère qui détermine les groupes qui sont sélectionnés
ORDER BY liste des attributs permettant d'ordonner la présentation des résultats
```

2.1 SELECT

SELECT correspond à l'opérateur de projection.
Par exemple $ALIEN\_PLANETE = [NomAlien, Planete]ALIEN$ donne en SQL :

```
SELECT ALIEN.NomAlien, ALIEN.Planete
FROM ALIEN
```

On peut omettre le nom de la table lorsqu'il n'y a pas ambiguité :

```
SELECT NomAlien, Planete
FROM ALIEN
```

Il y a une différence cependant entre la requête SQL et la projection relationnelle. Ainsi

```
SELECT Planete
FROM ALIEN
```

donnera :
Trantor, Euterpe, Aurora, Trantor, Trantor, Helicon, Aurora, Nexon, Euterpe.
En effet les doublons ne sont pas automatiquement éliminés. Pour cela, il faut utiliser le mot réservé DISTINCT :

```
SELECT DISTINCT Planete
FROM ALIEN
```

qui donnera Trantor, Euterpe, Aurora, Helicon, Nexon.
Il ne faut cependant pas en abuser car cela ralentit le traitement des données !
Ce n'est pas explicitement au programme des concours mais c'était nécessaire pour répondre à une question de l'épreuve de Centrale 2015...
Le mot réservé * permet de sélectionner tous les attributs d'une table.

```
SELECT *
FROM ALIEN
```

retourne toutes les données de la table ALIEN.
Le mot réservé AS permet de renommer les résultats.

```
SELECT Planete AS pl
FROM ALIEN
```

Pour réaliser des produits cartésiens, on met la liste des tables après FROM :

```
SELECT *
FROM RESPONSABLE, ALIEN
```

Parfois, on doit effectuer le produit cartésien d'une table par elle-même : on utilise alors des alias pour éviter les confusions :

```
SELECT A1.Nom, A2.Planete
FROM ALIEN AS A1, ALIEN AS A2
```

2 2 WHERE

WHERE permet de spécifier des critères de sélection.

```
SELECT Nom
FROM ALIEN
WHERE planete = 'Trantor'
```

Les critères peuvent s'exprimer en utilisant :

- des opérateurs de comparaison : <, >, <=, >=, =, <>
- des opérateurs logiques : AND, NOT, OR
- des prédicats définissant des ensembles : IN, BETWEEN ... AND ..., LIKE
- la comparaison avec la valeur NULL : IS NULL, IS NOT NULL
- des caractères génériques : _ qui remplace un caractère et % qui remplace une chaîne de caractères.

Pour sélectionner les Aliens de Trantor ou Euterpe, on peut écrire :

```
SELECT Nom
FROM ALIEN
WHERE planete = 'Trantor' OR planete = 'Euterpe'
```

ou bien

```
SELECT Nom
FROM ALIEN
WHERE planete IN ('Trantor', 'Euterpe')
```

2 3 Les jointures

On peut utiliser la clause WHERE ou l'opérateur JOIN.
Reprenons la jointure CABINE[CABINE.NoAllee = RESPONSABLE.NoAllee]RESPONSABLE.

```
SELECT *
FROM CABINE, RESPONSABLE
WHERE CABINE.NoAllee = RESPONSABLE.NoAllee
```

ou bien

```
SELECT *
FROM CABINE JOIN RESPONSABLE ON CABINE.NoAllee = RESPONSABLE.NoAllee
```

2 4 Les fonctions de groupes (ou d'agrégation)

Les fonctions de groupe permettent d'obtenir des informations sur un ensemble de n-uplets en travaillant sur les **colonnes** et non pas sur les lignes comme avec `WHERE`.
Les fonctions disponibles sont :

- `AVG` : calcule la moyenne d'une colonne
- `SUM` : calcule la somme d'une colonne
- `MIN` : calcule le minimum d'un colonne
- `MAX` : le max
- **`VARIANCE`** :...
- **`STDDEV`** : l'écart-type
- `COUNT` : nombre de lignes de la colonne
- `EVERY` : vrai si toutes les occurrences de la colonne sont vraies
- `ANY | SOME` : vrai si au moins une occurrence de la colonne est vraie

Que donne :

```
SELECT COUNT(Planete) AS NbPlanetes
FROM ALIEN
WHERE Planete = 'Trantor'
```

Que donne :

```
SELECT COUNT(DISTINCT Planete) AS NbPetitesPlanetes
FROM ALIEN
WHERE NoCabine < 5
```

Que donne :

```
SELECT COUNT(*)
FROM ALIEN
```

2 5 GROUP BY

Attention : c'est une clause souvent utilisée aux concours mais dangereuse...
Elle permet de subdiviser une table en groupes mais les valeurs des attributs de regroupement peuvent ne plus être atomiques ce qui limitera les opérations réalisables.
Par exemple :

```
SELECT Planete, Nom
FROM ALIEN
GROUP BY Planete
```

donne :

Planète	Nom
Trantor	Zorglub
	Zbleurdite
	Darneurane
Euterpe	Blorx
	Joranum
Aurora	Urxiz
	Zzzzz
Helicon	Mulzo
Nexon	Arggh

On peut appliquer des fonctions de groupe. Par exemple :

```
SELECT Planete, COUNT(Nom) as NbPlanetes
FROM ALIEN
GROUP BY Planete
```

donne :

Planète	NbPlanètes
Trantor	3
Euterpe	2
Aurora	2
Helicon	1
Nexon	1

2.6 HAVING

HAVING permet de faire des sélection a posteriori, en fait après un GROUP BY.

```
SELECT Planete
FROM ALIEN
GROUP BY Planete
HAVING COUNT(Nom) > 1
```

donne :

Planète
Trantor
Euterpe
Aurora

2.7 ORDER BY

La clause ORDER permet de préciser dans quel ordre les n-uplets sélectionnés seront donnés. Cette clause ne modifie pas l'ensemble des données retournées par la requête, mais simplement leur ordre de présentation. Les expressions sont généralement des noms d'attributs. On peut trier par ordre croissant par défaut, ou par ordre décroissant, en utilisant dans ce cas le mot réservé DESC.

```
SELECT Planete, COUNT(Nom) AS NbPlanetes
FROM ALIEN
GROUP BY Planete
ORDER BY Planete
```

Planète	NbPlanètes
Aurora	2
Euterpe	2
Helicon	1
Nexon	1
Trantor	3

IV Exploitation d'une base de données

À partir de mesures régulièrement effectuées par différents observatoires, une base de données des caractéristiques et des états des corps célestes de notre Système solaire est maintenue à jour. L'objectif de cette partie est d'extraire de cette base de données les informations nécessaires à la mise en œuvre des fonctions développées dans les parties précédentes, puis de les utiliser pour prévoir les positions futures des différentes planètes. Les données à extraire sont les masses des corps étudiés et leurs états (position et vitesse) à l'instant $t_{\min}$ du début de la simulation.

Une version simplifiée, réduite à deux tables, de la base de données du Système solaire est donnée figure 3. Les masses sont exprimées en kilogrammes, les distances en unités astronomiques ($1\,\text{au} = 1{,}5 \times 10^{11}\,\text{m}$) et les vitesses en kilomètres par seconde. Le référentiel utilisé pour exprimer les composantes des positions et des vitesses est galiléen, orthonormé et son centre est situé à proximité du Soleil.

CORPS

id_corps	nom	masse
⋮	⋮	⋮

ETAT

id_corps	datem	x	y	z	vx	vy	vz
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 3 Schéma de la base de données

La table `CORPS` répertorie les corps étudiés, elle contient les colonnes
- `id_corps` (clé primaire) entier identifiant chaque corps ;
- `nom`, chaine de caractères, désigne le nom usuel du corps ;
- `masse` de type flottant, contient la masse du corps.

La table `ETAT` rassemble l'historique des états successifs (positions et vitesses) des corps étudiés. Elle est constituée de huit colonnes :
- `id_corps` de type entier, identifie le corps concerné ;
- `datem` est la date de la mesure, sous forme d'un entier donnant le nombre de secondes écoulées depuis un instant d'origine ;
- trois colonnes de type flottant pour les composantes de la position `x`, `y`, `z` ;
- trois colonnes de type flottant pour les composantes de la vitesse `vx`, `vy`, `vz`.

IV.A – Écrire une requête SQL qui renvoie la liste des masses de tous les corps étudiés.

IV.B – Les états des différents corps ne sont pas forcément tous déterminés exactement au même instant. Nous allons assimiler l'état initial (à la date $t_{\min}$) de chaque corps à son dernier état connu antérieur à $t_{\min}$.

Dans toute la suite, on supposera que la valeur de $t_{\min}$, sous le format utilisé dans la table `ETAT`, est accessible à toute requête SQL via l'expression `tmin()`.

IV.B.1) On souhaite d'abord vérifier que tous les corps étudiés disposent d'un état connu antérieur à `tmin()`. Le nombre de corps présents dans la base est obtenu grâce à la requête `SELECT count(*) FROM corps`. Écrire une requête SQL qui renvoie le nombre de corps qui ont au moins un état connu antérieur à `tmin()`.

IV.B.2) Écrire une requête SQL qui renvoie, pour chaque corps, son identifiant et la date de son dernier état antérieur à `tmin()`.

IV.B.3) Le résultat de la requête précédente est stocké dans une nouvelle table `date_mesure` à deux colonnes :
- `id_corps` de type entier, contient l'identifiant du corps considéré ;
- `date_der` de type entier, correspond à la date du dernier état connu du corps considéré, antérieur à `tmin()`.

Pour simplifier la simulation, on décide de négliger l'influence des corps ayant une masse strictement inférieure à une valeur fixée `masse_min()` et de ne s'intéresser qu'aux corps situés dans un cube, centré sur l'origine du référentiel de référence et d'arête `arete()` donnée. Les faces de ce cube sont parallèles aux plans formés par les axes du référentiel de référence.

Écrire une requête SQL qui renvoie la masse et l'état initial (sous la forme `masse, x, y, z, vx, vy, vz`) de chaque corps retenu pour participer à la simulation. Classez les corps dans l'ordre croissant par rapport à leur distance à l'origine du référentiel.

IV.C – On dispose des variables python `t0`, `p0`, `v0` et `masse` initialisées à partir du résultat de la requête précédente. `t0` est un entier qui donne la date des conditions initiales : il correspond à $t_{\min}$ et à `tmin()`. `p0` est une liste de longueur N, chaque élément de `p0` est une liste à 3 éléments de la forme `[x, y, z]` représentant la position initiale d'un corps, en unité astronomique. `v0` a une structure identique mais indique les vitesses initiales des corps considérés, en km/s. `masse` est décrite en partie III.

Écrire la fonction python `simulation_verlet(deltat, n)` qui prend en paramètre un incrément de temps en secondes (`deltat > 0`) et un nombre d'itérations (`n > 0`) et qui renvoie la liste des positions des corps considérés pour chaque instant `t0`, `t0 + deltat`, ..., `t0 + n*deltat` (cf variable `position` définie en partie III). Les calculs seront menés en utilisant le schéma d'intégration de Verlet, le résultat sera fourni en unité astronomique.

• • • FIN • • •

```
SELECT masse FROM CORPS
```

```
SELECT COUNT(DISTINCT id_corps)
FROM ETAT
WHERE datem < tmin()
```

```
SELECT id_corps, MAX(datem)
FROM ETAT
WHERE datem < tmin()
GROUP BY id_corps
```

```
SELECT masse, x, y, z, vx, vy, vz
FROM (CORPS JOIN ETAT ON CORPS.id_corps = ETAT.id_corps)
     JOIN date_mesure
          ON (datem = date_der AND ETAT.id_corps = date_mesure.id_corps)
WHERE masse > masse_min() AND ABS(x) < arete()/2 AND ABS(y) < arete()/2
                          AND ABS(z) < arete()/2
ORDER BY x*x + y*y + z*z
```

EXERCICES

Recherche E - 1 Aliens : BDD avec Python

On vous demande de gérer le centre d'acceuil terrestre des aliens. Vous disposez de 6 tableaux de données. Vous devez fournir les expressions qui permettent de répondre aux questions posées. Par exemple, pour avoir la liste des aliments :

```
In [50]: { m.aliment for m in baseMiam }
Out[50]: {'Bortsch', 'Kashpir', 'Schwanstucke', 'Zoumise'}
```

La liste des aliments mangés par les Aliens de Trantor :

```
lesAlimTrantor = { m.aliment
                   for m in baseMiam
                   for a in baseAlien
                   if m.nom_alien == a.nom
                   if a.planete == "Trantor" }

In [54]: lesAlimTrantor
Out[56]: {'Bortsch', 'Schwanstucke'}
```

Voici les données :

```
class Alien:
    def __init__(self, Nom, Sexe, Planete, NoCabine):
        "Constructeur d'Aliens"
        self.nom       = Nom
        self.sexe      = Sexe
        self.planete   = Planete
        self.no_cabine = NoCabine

class Cabine:
    def __init__(self,NoCabine, NoAllee):
        "Constructeur de Cabine"
        self.no_allee  = NoAllee
        self.no_cabine = NoCabine

class Miam:
    def __init__(self, NomAlien, Aliment):
        "Constructeur d'Aliments"
        self.nom_alien  = NomAlien
        self.aliment    = Aliment

class Agent:
    def __init__(self, Nom, Ville):
        "Constructeur d'Agents"
        self.nom   = Nom
        self.ville = Ville

class Responsable:
    def __init__(self, NoAllee, Nom):
        "Constructeur de Responsables"
        self.no_allee = NoAllee
        self.nom      = Nom

class Gardien:
    def __init__(self, Nom, NoCabine):
        "Constructeur de Gardiens"
        self.nom       = Nom
        self.no_cabine = NoCabine
```

```
baseAlien = {
  Alien ( "Zorglub",    "M", "Trantor", 1 ),
  Alien ( "Blorx",      "M", "Euterpe", 2 ),
  Alien ( "Urxiz",      "F", "Aurora",  3 ),
  Alien ( "ZBleurdite", "F", "Trantor", 4 ),
  Alien ( "Darneurane","M", "Trantor", 5 ),
  Alien ( "Mulzo",      "M", "Helicon", 6 ),
  Alien ( "Zzzzz",      "F", "Aurora",  7 ),
  Alien ( "Arggh",      "M", "Nexon",   8 ),
  Alien ( "Joranum",    "F", "Euterpe", 9 )
  }

baseCabine = {
  Cabine ( 1 ,  1),
  Cabine ( 2 ,  1),
  Cabine ( 3 ,  1),
  Cabine ( 4 ,  1),
  Cabine ( 5 ,  1),
  Cabine ( 6 ,  2),
  Cabine ( 7 ,  2),
  Cabine ( 8 ,  2),
  Cabine ( 9 ,  2)
  }

baseMiam = {
  Miam ( "Zorglub",    "Bortsch"),
  Miam ( "Blorx",      "Bortsch"),
  Miam ( "Urxiz",      "Zoumise"),
  Miam ( "ZBleurdite", "Bortsch"),
  Miam ( "Darneurane", "Schwanstucke"),
  Miam ( "Mulzo",      "Kashpir"),
  Miam ( "Zzzzz",      "Kashpir"),
  Miam ( "Arggh",      "Zoumise"),
  Miam ( "Joranum",    "Bortsch")
  }

baseAgent = {
  Agent ( "Branno",   "Terminus"),
  Agent ( "Darell",   "Terminus"),
  Agent ( "Demerzel", "Reze"),
  Agent ( "Seldon",   "Terminus"),
  Agent ( "Dornick",  "Kalgan"),
  Agent ( "Hardin",   "Terminus"),
  Agent ( "Trevize",  "Hesperos"),
  Agent ( "Pelorat",  "Kalgan"),
  Agent ( "Riose",    "Terminus")
  }

baseResponsable = {
  Responsable ( 1, "Seldon"),
  Responsable ( 2, "Pelorat")
  }


baseGardien = {
  Gardien ( "Branno",   1),
  Gardien ( "Darell",   2),
  Gardien ( "Demerzel", 3),
  Gardien ( "Seldon",   4),
  Gardien ( "Dornick",  5),
  Gardien ( "Hardin",   6),
  Gardien ( "Trevize",  7),
  Gardien ( "Pelorat",  8),
  Gardien ( "Riose",    9)
  }
```

1. Formez :
   - i. l'ensemble des gardiens ;
   - ii. l'ensemble des villes où habitent les agents ;
   - iii. l'ensemble des triplets (no de cabine,alien,gardien) pour chaque cabine ;
   - iv. l'ensemble des couples (alien,allée) pour chaque alien ;
   - v. l'ensemble de tous les aliens de l'allée 2 ;
   - vi. l'ensemble de toutes les planètes dont sont originaires les aliens habitant une cellule de numéro pair ;
   - vii. l'ensemble des aliens dont les gardiens sont originaires de Terminus ;
   - viii. l'ensemble des gardiens des aliens féminins qui mangent du bortsch ;
   - ix. l'ensemble des cabines dont les gardiens sont originaires de Terminus ou dont les aliens sont des filles
2. Y a-t-il ?
   - i. des aliments qui commencent par la même lettre que le nom du gardien qui surveille l'alien qui les mange ?
   - ii. des aliens originaires d'Euterpe ?
3. Est-ce que :
   - i. tous les aliens ont un 'x' dans leur nom ?
   - ii. tous les aliens qui ont un 'x' dans leur nom ont un gardien qui vient de Terminus ?
   - iii. il existe un alien masculin originaire de Trantor qui mange du Bortsch ou dont le gardien vient de Terminus
4. Répondez aux mêmes questions à l'aide de requêtes SQL.

Recherche E - 2 Charges de voitures

Afin de comparer les charges totales de plusieurs modèles de voitures électriques, les données de plusieurs expériences ont été stockées dans une base de données.

VOITURE
id_voiture
Marque
Modele
Annee

EXPERIENCE
id_exp
id_voiture
Date

DONNEES
id_exp
temps
intensite

Celle-ci contient trois tables dont les structures sont les suivantes :

Tous les attributs sont de type numérique sauf la marque et le modèle, qui sont des chaînes de caractères.
La table VOITURE contient un attribut id_voiture qui est une clé primaire pour une voiture donnée, un nom de marque et de modèle et l'année de fabrication de la voiture.
La table EXPERIENCE contient attribut id_exp qui est une clé primaire pour une expérience, l'identifiant correspondant à la voiture testée et la date du test.
La table DONNEES contient l'identifiant de l'expérience, le temps et l'intensité mesurées.
Voici des exemples d'enregistrements pour cette base de donnée :

VOITURE			
id_voiture	Marque	Modele	Annee
1	Peugeot	Electrica	2008
2	Peugeot	Electrix	2015

EXPERIENCE		
id_exp	id_voiture	Date
1	1	01/09/2009
2	1	01/02/2015
3	1	01/02/2015

DONNEES		
id_exp	temps	intensite
1	0	0
1	3	0.025
1	10	0.02

1. Écrire une requête SQL qui affiche les modèles, marques et années des voitures dont l'année est supérieure ou égale à 2010, rangés par ordre croissant d'année.
2. Écrire une requête SQL qui affiche les modèles de voitures et les dates des expériences pour les voitures de la marque 'Renault'
3. On suppose que la fonction `resultat=interroge_bdd(requete)` permet de se connecter à la base de données, d'envoyer la requête et de retourner la réponse sous la forme d'un tableau `resultat` dont les colonnes correspondront aux colonnes sélectionnées par la commande SELECT et les lignes correspondent aux différents résultats de la requête.
   - i. Écrire une fonction `Charge(i)` qui prend en argument un identifiant d'expérience, récupère les données d'une expérience, triées par temps croissant (on utilisera la syntaxe SORT de SQL) et calcule à l'aide de la méthode des trapèzes la charge totale pour cette expérience.
   - ii. Écrire une suite de commandes qui permet d'afficher tous les modèles, années et charges totales de toutes les expériences menées sur des voitures de la marque 'Toyota'

Recherche E - 3 Cinéma

On a 4 tables :

CINÉMA
NuméroCinéma
NomCinema
AdresseCinema
VilleCinema
CodePostalCinema
TelephoneCinema

AFFICHE
NuméroCinéma
Titre

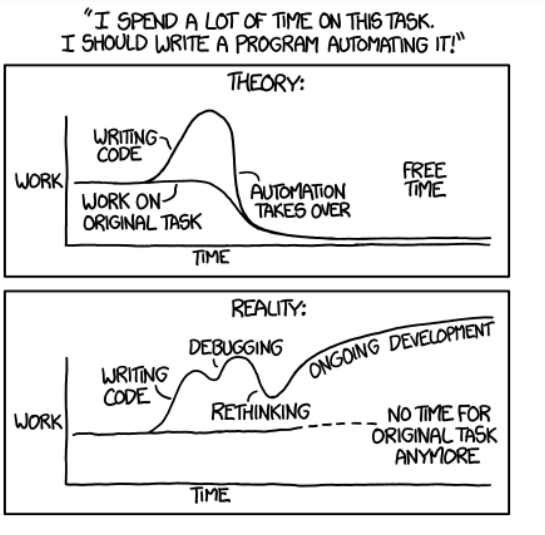
FILM
Titre
MetteurEnScene
Genre

DISTRIBUTION
Titre
NomActeur

1. Le nom et l'adresse des cinémas projetant le film *Brigadoon*
2. Le nom des metteurs en scène des films à l'affiche du *Katorza*
3. Le nom des acteurs apparaissant dans la distribution des films à l'affiche à *Nantes*.
4. Le titre des films avec *Fred* ASTAIRE à l'affiche à *Nantes*.
5. Le nom des cinémas de *Saint Mars la Jaille* ayant à l'affiche une *comédie musicale* et un *documentaire*.
6. Le nom des cinémas de *Chavagne en Paillé* n'ayant pas de film de *science fiction* à l'affiche.
7. Le nombre de cinémas de *Kermeursac'h*.
8. Pour chaque cinéma, donnez le nombre de films à l'affiche.
9. Le numéro des cinémas ayant au moins deux films à l'affiche.
10. Le numéro des cinémas ayant autant de films à l'affiche que le cinéma numéro 1.
11. Numéro des cinémas ayant tous les genres de films à l'affiche (on utilisera **EXISTS**).

THÈME

Mines-Ponts 2015



Cette session des concours 2015 voyait l'introduction d'une nouvelle épreuve d'informatique commune à toutes les filières dans le CCMP. De façon très inattendue après une phase d'échanges que nous jugions très constructive avec les représentants du concours, le sujet proposé aux candidats était de qualité très médiocre et a fait l'objet de critiques justifiées par la majorité des collègues impliqués dans cet enseignement.

Ces critiques ont suscité la détermination d'un groupe de collègues à montrer par l'exemple qu'il était tout à fait possible de produire, sur le thème choisi, un sujet conforme au programme, bien écrit, et donc susceptible de correctement déceler les qualités des candidats.

Ceci a nécessité bien entendu un certain volume de travail et des relectures patientes. C'est le résultat de ce travail que nous présentons ici. Les contraintes de l'exercice, à savoir le choix initial de la contextualisation, le cadre du programme, la progressivité du sujet, le niveau de difficulté, ont été respectées, mais il s'est avéré que tous les thèmes traités ne pouvaient être abordés dans le volume confiné d'une épreuve de 1 h 30.

Mode d'emploi du sujet sur le codage de Huffman

Choix du langage de programmation

Le programme officiel d'informatique des classes préparatoires impose les règles suivantes :

— les candidats doivent obligatoirement traiter les questions d'algorithmique en Python ;

— pour les parties du sujet consacrées à l'ingénierie numérique, les candidats doivent avoir le choix entre Python et Scilab.

Durée

Le sujet d'origine était bien trop long pour la durée prévue de 1 h 30. Notre énoncé réécrit convient pour une épreuve de 4 h. Il semble difficile d'aborder sérieusement l'ensemble des parties du programme officiel d'informatique dans un sujet de 1 h 30 ; il faudra faire des choix. Il convient toutefois de ne pas négliger l'évaluation du programme de deuxième année (piles, récursivité, tris).

Introduction du sujet

Les listes sont au programme ; il n'y a donc pas lieu de rappeler leur syntaxe. Les dictionnaires ne sont pas au programme ; il est hors de question de s'en servir. Les tuples ne sont pas au programme non plus ; il se trouve qu'ils sont présents lorsqu'une fonction renvoie plusieurs objets (les élèves savent faire) mais il n'y a pas lieu de s'y attarder.

Structure du sujet

— Les questions de calcul numérique sont logiquement regroupées dans une même partie.

— Nous plaçons à la fin de l'énoncé la partie la plus difficile (compression de texte).

— L'introduction de notre sujet précise les liens thématiques entre les quatre parties de l'énoncé ; celles-ci peuvent néanmoins être traitées de façon indépendante.

— Les trois premières parties d'une part, la quatrième partie d'autre part, pourraient être écrites de manière à constituer deux sujets autonomes, le premier relativement facile, le deuxième plus ambitieux.

Partie I. Réception des données

— **Q1**. Le programme prévoit l'enseignement du « principe de la représentation des nombres entiers en mémoire » mais n'impose pas un mode de codage précis. Nous accepterons donc à cette question plusieurs réponses (codage par complément à 2, codage par bit de signe et valeur absolue, codage par translation). Les candidats doivent indiquer une plage d'entiers cohérente avec le mode de codage choisi.

— **Lecture des caractères transmis** : pour un début de sujet qui se doit d'être progressif, il nous a paru pertinent de séparer le travail à effectuer en deux fonctions : `lect_trame` (qui lit une trame et reconstitue des nombres entiers à partir des caractères reçus) et `lect_intensite` (qui doit attendre de recevoir un caractère d'en-tête via une boucle while).

— **« checksum »** : voilà une allusion à l'un des thèmes applicatifs mentionnés à la dernière page du programme officiel [a]. On rappelle qu'aucune connaissance n'est exigible des candidats sur ces thèmes ; le vocabulaire correspondant doit être redéfini lorsqu'il intervient dans l'énoncé.

— **Tracé d'un graphe** : nous n'avons pas repris cette question qui nous a semblé sans intérêt dans une épreuve écrite. [b]

Partie II. Analyse des mesures

La question 23 du sujet d'origine demandait aux candidats d'écrire une fonction servant à « réaliser une validation ou une invalidation de la défectuosité du moteur suivant un critère à proposer ». Le choix d'un tel critère est une compétence relevant de l'enseignement des sciences industrielles mais pas de l'informatique. Rappelons que les étudiants de la filière PC et ceux de la filière MP option informatique n'étudient pas de SI. Le présent sujet n'ayant pas vocation à évaluer des compétences de SI, le critère à employer doit être fourni par l'énoncé. [c]

Partie III. Bases de données

Nous proposons quatre questions d'écriture de requêtes SQL de difficulté graduée :

— une requête simple (restriction et projection dans une seule table) ;

— une jointure ;

a. Pour mémoire, ces thèmes sont : traitement d'images, cryptographie, codes correcteurs, graphes, POO.

b. Ajoutons que conformément au programme, une telle question devrait pouvoir être traitée en Python (avec matplotlib) ou en Scilab par les candidats, elle n'a donc pas sa place dans une partie du sujet consacrée à l'algorithmique (en Python obligatoirement).

c. Le programme officiel indique clairement l'esprit dans lequel sont enseignées les simulations numériques dans le cours d'informatique : « Seules la mise en œuvre constructive des algorithmes et l'analyse empirique des résultats sont concernées. [...] On met l'accent sur les aspects pratiques comme l'impact des erreurs d'arrondi sur les résultats, les conditions d'arrêt, la complexité en temps de calcul ou le stockage en mémoire. »

— une requête avec agrégation de données ;

— pour finir une requête plus difficile afin de mettre en valeur les meilleurs candidats : trouver les imprimantes non validées (Q10 du sujet d'origine). Cette question possède des solutions variées (à l'aide d'un **`EXCEPT`** ou bien avec un **`NOT IN`** et une sous-requête).

Partie IV. Compression de Huffman

— La notion d'arbre est entièrement hors programme. Les sujets portant sur les arbres sont bien adaptés pour l'option informatique en MP, mais pas pour le tronc commun d'informatique MP/PC/PSI.

Nous avons donc choisi de ne pas aborder du tout les arbres de Huffman. La section F.4.3 propose de programmer le codage de Huffman en utilisant uniquement des listes. Dans la section F.4.4, on effectue le décodage par dichotomie sur un tableau.

— La section F.4.2 de notre énoncé porte sur le calcul de la liste des fréquences des caractères puis leur tri par fréquence croissante. Ceci permet d'évaluer le programme d'informatique de deuxième année (tri, récursivité).

— Nous reprenons le principe consistant à fournir du code dans l'énoncé. Il s'agit du tri rapide, légèrement modifié afin de ne pas tomber dans une simple récitation de cours. Exploitation du code fourni : les candidats doivent d'abord décrire l'exécution de la fonction sur un exemple simple, puis identifier le travail général effectué par cette fonction.

— Les deux dernières sections proposent des questions plus ambitieuses, à même de trier les meilleurs candidats. Nous avons repris le principe d'exposer des algorithmes d'accès délicat, mais en tentant de les rendre compréhensibles avec les outils

L'énoncé proprement dit

Introduction

Les imprimantes sont des systèmes mécatroniques (combinaison synergique et systémique de la mécanique, de l'électronique et de l'informatique en temps réel) fabriqués en grande série dans des usines robotisées. Pour améliorer la qualité des produits vendus, il a été mis en place différents tests de fin de chaîne pour valider l'assemblage des produits. Pour un de ces tests, un opérateur connecte l'outil de test sur la commande du moteur de déplacement de la tête d'impression et sur la commande du moteur d'avance papier. Une autre connexion permet de récupérer les signaux issus des capteurs de position. Différentes commandes et mesures sont alors exécutées. Ces mesures sont transmises sous la forme d'une suite de caractères vers un ordinateur. Cet ordinateur va effectuer différentes mesures pour valider le fonctionnement de l'électromécanique de l'imprimante. L'ensemble des mesures et des analyses est sauvegardé dans un fichier texte. Afin de minimiser l'espace occupé, les fichiers sont compressés. Une base de données stocke les informations concernant les mesures et les imprimantes sur lesquelles elles portent, et permet à l'entreprise d'améliorer la qualité de la production après diverses études statistiques.

Le problème comporte quatre grandes parties indépendantes, sauf par le thème traité.

— Dans la partie F.1, on s'intéresse au processus de réception par l'ordinateur des données transmises par le capteur.

— Dans la partie F.2, on procède au traitement des données numériques récupérées à fin de validation de l'imprimante assemblée.

— Dans la partie F.3, on étudie quelques aspects de la base de données constituée.

— Dans la partie F.4, on propose une introduction à la problématique de la compression de données.

Le sujet comporte des questions de programmation. Le langage à utiliser est Python sauf dans la partie F.2 où le candidat est libre de choisir le langage Python ou le langage Scilab.
Le candidat indique en tête de sa première copie s'il choisit le langage Python ou le langage Scilab pour les questions de programmation de la deuxième partie.

Réception des données issues de la carte d'acquisition

Le capteur

Le capteur utilisé est analogique, et comporte un convertisseur (appelé convertisseur analogique numérique) qui traduit les mesures effectuées sous forme numérique. Chaque donnée est ainsi codée comme un entier.

Q1. Rappeler un principe de représentation en machine des entiers signés sur 10 bits. Préciser la plage des valeurs entières représentables selon ce principe.

Q2. On considère que les valeurs analogiques s'étendent en pleine échelle de -5 V à 5 V et sont converties en entiers signés représentés sur 10 bits. Donner une valeur approchée de la résolution de la mesure en volts.

Dans la suite du problème, la question du codage des entiers en binaire n'intervient plus.

Liaison avec l'ordinateur

Une liaison série asynchrone permet la communication entre la carte de commande/acquisition et le PC. Les informations correspondant à une mesure sont envoyées par la carte électronique sous la forme d'une suite de caractères, appelée trame. Plusieurs trames consécutives correspondant à différents jeux de mesures peuvent être transmises sans interruption, le format même des trames devant permettre de les délimiter. Un exemple de trame est :

'U'	'0'	'0'	'3'	'+'	'0'	'1'	'2'	'+'	'0'	'0'	'4'	'-'	'0'	'2'	'3'	'9'	'9'	'9'	'3'

Une trame est donc constituée de la manière suivante :

- un caractère d'en-tête, qui est une des trois lettres `'U'`, `'I'`, `'P'`, permettant d'identifier la nature de la mesure effectuée (`'U'` : tension moteur, `'I'` : courant moteur, `'P'` : position absolue) ;
- trois chiffres constituant une valeur entière précisant le nombre N de mesures qui constituent la suite de la trame ;
- un ensemble de N blocs consécutifs de quatre caractères. Chacun des blocs est constitué d'un caractère de signe `'+'` ou `'-'`, suivi de trois chiffres donnant une valeur absolue. L'entier signé ainsi codé sur quatre caractères donne une valeur issue de la conversion analogique/numérique d'une mesure ;
- un dernier bloc de quatre chiffres constituant ce qu'on appelle une somme de contrôle (*checksum*). Celle-ci est calculée en formant la somme des valeurs des données précédentes de la trame, et en prenant le reste de la division euclidienne par `10000` de cette somme.

La somme de contrôle (*checksum*) permet de vérifier si la trame a été correctement transmise. On peut, à réception d'une trame, calculer la somme de contrôle associée aux données reçues, et la comparer à la somme de contrôle contenue dans la trame. Si ces deux valeurs ne coïncident pas, les données ont été altérées lors de la transmission, et il convient de ne pas les exploiter.

La fonction `car_read(nbre_carac)` permet d'obtenir des caractères consécutifs issus de la liaison asynchrone. Elle prend en argument un entier `nbre_carac`, et renvoie une chaîne de caractères constituée de `nbre_carac` caractères. Ainsi, dans l'exemple ci-dessus, deux appels consécutifs à `car_read(5)` puis `car_read(3)`, renverront respectivement les chaînes `'U003+'` et `'012'`.

Q3. Toujours à partir de l'exemple donné plus haut, on suppose que les deux appels

`car_read(5)` puis `car_read(3)`

ont été effectués. Quelle valeur est alors renvoyée par l'appel `car_read(4)` ?

Le programme de lecture des données transmises à l'ordinateur lit des caractères jusqu'à obtention d'un caractère `x` d'en-tête égal à `'U'`, `'I'` ou `'P'`. Il fait alors appel à une fonction de lecture de trame `lecture_trame(x)` lisant les données numériques contenues dans la trame qui commence par `x`.

Q4. Écrire la fonction `lecture_trame(x)` prenant en argument le caractère d'en-tête `x` qui vient d'être lu, et renvoyant une liste `[x,L,S]`, où `L` est une liste d'entiers représentant les mesures transmises dans la trame commençant par `x`, et `S` est la valeur de la somme de contrôle transmise dans la trame commençant par `x`.

Dans la question suivante, on utilise une valeur issue d'un appel à la fonction

`lecture_trame(x)`.

Une telle valeur sera stockée dans une variable `trame` *via* une affectation

`trame = lecture_trame(x)`.

Q5. Écrire une fonction `checkSum(trame)` prenant en argument une liste `trame`, et renvoyant un booléen indiquant si le troisième élément de la liste `trame` est égal à la somme de contrôle calculée à partir des valeurs stockées dans le deuxième élément de cette liste.

Q6. Écrire une fonction `lecture_intensite()`, sans argument, et procédant au traitement suivant :

① obtenir un caractère jusqu'à ce que ce soit le caractère `'I'` ;

② lire la trame dont on vient de lire le caractère d'en-tête et stocker les informations dans une variable `trame` ;

③ si la variable `trame` ne présente pas une somme de contrôle valide, recommencer à ① ;

④ renvoyer la liste des mesures numériques alors contenues dans la variable `trame`.

On fait l'hypothèse que le canal de transmission est de qualité suffisante pour assurer qu'on finira par obtenir une trame dont la somme de contrôle est correcte.

Analyse des mesures

Les fonctions demandées pourront être écrites dans le langage `Python` (éventuellement à l'aide de sa bibliothèque `numpy`) ou dans le langage `Scilab`.
On exclut dans cette partie tout appel à une fonction de calcul approché d'intégrale ou de résolution approchée d'équations différentielles qui serait préprogrammée dans le langage choisi ou dans l'une de ses bibliothèques.

Traitement numérique

Q7. Rappeler le principe de la méthode des trapèzes permettant de calculer une valeur approchée de l'intégrale d'une fonction f sur un intervalle $[a,b]$. On ne demande pas ici de coder de fonction informatique permettant ce calcul.

Pour valider le fonctionnement de l'imprimante, on considère une suite de valeurs de mesures de l'intensité du courant (en ampères) du moteur de la tête d'impression, stockée dans un tableau `mesures`. On s'intéresse alors à la valeur moyenne et à l'écart-type de cette distribution de valeurs.
Pour une fonction f définie (et continue) sur un segment $[0,t_{\text{final}}]$, la valeur moyenne de la fonction sur ce segment est définie par :

$$f_{\text{moy}} = \frac{1}{t_{\text{final}}}\int_0^{t_{\text{final}}} f(t)\mathrm{d}t.$$

On ne dispose ici des valeurs de l'intensité qu'en des instants prédéfinis : les mesures ont été effectuées toutes les 2 millisecondes.

Q8. Écrire une fonction `trapezes(L)` prenant en argument un tableau `L` de valeurs numériques, et renvoyant une valeur approchée, calculée par la méthode des trapèzes, de la valeur moyenne d'une grandeur variant durant un certain intervalle de temps $[0,t_{\text{final}}]$, en supposant que le tableau `L` contient les valeurs de cette grandeur mesurées toutes les 2 millisecondes.

On définit l'écart-type d'une fonction sur un segment $[0,t_{\text{final}}]$ par :

$$f_{\text{ec}} = \sqrt{\frac{1}{t_{\text{final}}}\int_0^{t_{\text{final}}} (f(t)-f_{\text{moy}})^2\mathrm{d}t}.$$

Q9. Écrire une fonction `indicateurs(mesures)` prenant en argument un tableau `mesures` de flottants codant les valeurs de l'intensité mesurées toutes les 2 millisecondes sur un intervalle de temps $[0,t_{\text{final}}]$, et renvoyant la valeur de la moyenne et de l'écart-type des valeurs de ce tableau. On privilégiera l'utilisation de la fonction `trapezes`.

Validation des mesures

On sait que chaque moteur de l'imprimante, soumis à un signal d'entrée e, doit fournir un signal de sortie s lié au signal d'entrée par l'équation différentielle linéaire du premier ordre :

$$\frac{\mathrm{d}s}{\mathrm{d}t} = -\frac{k}{10}(s-e), \tag{F.1}$$

où k est un nombre strictement positif. Le bon fonctionnement d'un tel moteur est testé en vérifiant l'adéquation des valeurs mesurées avec des valeurs issues d'une résolution numérique de cette équation.

Q10. On considère le signal d'entrée $e(t) = \sin(\omega t)$ sur l'intervalle de temps $[0,1]$ (en secondes), où ω est une constante numérique stockée dans une variable `omega`.

Écrire une fonction `simulation(...)` renvoyant un tableau de valeurs approchées, calculées à l'aide de la méthode d'Euler, d'une solution de l'équation (F.1) aux instants $[0, 0.002, 0.004, \ldots, 1]$ (en secondes). Le candidat sera amené à choisir un ou des arguments pour la fonction programmée, et commentera le choix effectué.

On rappelle que tout appel à une fonction de résolution approchée d'équations différentielles préprogrammée est exclu.

On suppose désormais qu'à l'instant $t=0$, la valeur $s(0)$ du signal de sortie doit être nul. Le signal de sortie réel, acquis par le capteur, est stocké dans une variable `mesures` de type tableau, contenant la valeur du signal aux instants $[0, 0.002, 0.004, \ldots, 1]$ (en secondes). L'imprimante doit avoir un comportement conforme à la solution de l'équation (F.1) avec $s(0)=0$. En raison de différences de fabrication, k peut prendre les valeurs 0.5, 1.1 ou 2, et seulement celles-ci. Le comportement du moteur est considéré valide si l'écart maximal entre valeurs observées et valeurs simulées est inférieur à un certain flottant positif `eps`, pour au moins une valeur de k parmi 0.5, 1.1 ou 2.

Q11. Écrire une fonction `validation(mesures,eps)` prenant en arguments un tableau `mesures` de valeurs observées du signal de sortie et un flottant positif `eps`, et renvoyant un booléen indiquant si le comportement du moteur peut être validé au seuil `eps`.

Bases de données

On suppose que les résultats des analyses précédentes ont été stockés dans une base de données, constituée de deux tables, dont on donne une représentation simplifiée dans l'annexe de la page 130.
Après son assemblage et avant les différents tests de validation, un numéro de série unique est attribué à chaque imprimante. À la fin des tests de chaque imprimante, les résultats d'analyse ainsi que le nom du fichier contenant l'ensemble des mesures réalisées sur l'imprimante sont stockés dans la table `testfin`. Lorsqu'une imprimante satisfait les critères de validation, elle est enregistrée dans la table `production` avec son numéro de série, la date et l'heure de sortie de production, ainsi que le nom du modèle.

Q12. Écrire une requête SQL permettant d'obtenir les numéros de série des imprimantes ayant une valeur de `Imoy` comprise strictement entre 0.4 et 0.5.

Q13. Écrire une requête SQL permettant d'obtenir les numéros de série, les valeurs de `Imoy` et `Iec` ainsi que le modèle des imprimantes ayant été validées et dont la valeur de `Imoy` est comprise strictement entre 0.4 et 0.5.

Q14. Écrire une requête SQL permettant d'obtenir les modèles des imprimantes validées et pour chacun de ces modèles, le nombre d'imprimantes validées.

Q15. Écrire une requête SQL renvoyant le numéro de série et le nom du fichier de mesures des imprimantes qui n'ont pas été validées en sortie de production. Commenter très brièvement l'intérêt de cette information pour cette ligne de production.

Compression d'un fichier texte

Le fichier de résultat va être compressé sous la forme d'un fichier binaire. Pour cela, on code chaque caractère en une séquence de bits, de longueur variable en fonction du caractère. Le taux de compression est le rapport entre la taille en bits du fichier compressé et celle du fichier initial. Pour optimiser ce taux de compression, on réserve les séquences les plus courtes pour les caractères qui apparaissent le plus souvent dans le fichier, en laissant les séquences longues pour ceux qui sont les plus rares. Le codage du fichier est alors obtenu en juxtaposant les codes de chacun de ses caractères.

Cette partie est constituée de quatre sous-parties : on propose d'abord quelques codages explicites pour de petites chaînes de caractères afin de se familiariser avec la notion de codage par des séquences de longueur variable ; on écrit ensuite des fonctions permettant, à partir d'un texte à coder, d'extraire et d'organiser les informations concernant les caractères qui apparaissent et leurs fréquences ; on présente dans les troisième et quatrième sous-parties un algorithme de compression appelé « codage de Huffman », et un algorithme de décompression associé.

Exemples de codage

Q16. On choisit les codes suivants pour les quatre lettres E, S, A et T :

$$\text{E} : 1 \qquad \text{S} : 0 \qquad \text{A} : 10 \qquad \text{T} : 01.$$

Quels sont les codes des deux chaînes de caractères EST et ASE ? Quel inconvénient majeur possède ce codage ?

Q17. On choisit maintenant les codes suivants pour les six lettres E, R, U, O , N et T :

$$\text{E} : 11 \qquad \text{R} : 10 \qquad \text{U} : 000 \qquad \text{O} : 001 \qquad \text{N} : 010 \qquad \text{T} : 011.$$

Quel mot est codé par la séquence :

$$101101100000100101111 ?$$

Sachant qu'un caractère se code habituellement sur un octet, quel taux de compression obtient-on avec ce codage ? Que doit-on transmettre au destinataire pour qu'il puisse décoder le fichier binaire obtenu ? Commenter brièvement l'intérêt d'un tel codage.

Q18. Proposer une propriété du codage permettant d'éviter l'inconvénient de la question 16.

Tri des caractères selon leur fréquence

Q19. Écrire une fonction `caracteres_presents(donnees)` prenant en paramètre une chaîne de caractères `donnees` et retournant une liste des caractères présents dans cette chaîne, chaque caractère apparaissant dans le paramètre ne devant figurer qu'une seule fois dans la valeur de retour. Estimer la complexité temporelle de la fonction ainsi codée en fonction de la longueur de la chaîne **`donnees`**.

Q20. Modifier la fonction précédente pour qu'elle retourne deux listes `cars` et `freq` telles que :

- la liste `cars` est la liste des caractères apparaissant dans l'argument `donnees`, comme à la question précédente ;
- la liste `freq` est une liste de même longueur que la liste `cars` telle que, pour chaque position `i`, `freq[i]` est un entier donnant le nombre d'apparitions du caractère `cars[i]` dans la chaîne `donnees`.

On se propose maintenant d'ordonner les tableaux **`cars`** et **`freq`** selon la fréquence des caractères. On fournit en *annexe* F.5.2 trois fonctions nommées `echange`, `aux` et `tri`. On propose dans les questions suivantes d'analyser ces fonctions.

Q21. Décrire le comportement de l'appel à `aux(cars,freq,2,7)` lorsque :

```
freq = [ 4 ,  6 ,  3 ,  4 ,  8 ,  1 ,  7 ,  4 ]
cars = [' ', 'a', 'c', 't', 'e', 'è', 's', 'r'].
```

Q22. Préciser l'action de l'instruction `aux(cars,freq,i,j)` sur les paramètres **`cars`** et **`freq`**. On justifiera la réponse en s'aidant d'un invariant de boucle, et on prouvera la terminaison.

Q23. Écrire, à l'aide d'une ou des fonctions données en annexe, une instruction ou une suite d'instructions permettant d'obtenir les tableaux **`cars`** et **`freq`** triés par ordre croissant de fréquences et vérifiant toujours la propriété de correspondance énoncée à la question 20. Donner la complexité en temps dans le pire des cas de cette ou ces instructions, en fonction de la longueur commune des deux tableaux **`cars`** et **`freq`**.
Existe-t-il un tri ayant une complexité plus faible ? Si oui, donner son nom et sa complexité dans le pire des cas.

Codage de Huffman.

La partie F.4.2 permet d'obtenir deux tableaux `cars` et `freq` vérifiant les conditions de la question 20, et tels que, de plus, le tableau `freq` est trié par ordre croissant. On commence par stocker ces informations sous la forme d'une seule liste **`cars_ponderes`**, dont chaque élément est un tableau constitué d'un caractère et de sa fréquence dans le fichier, et qui est triée par ordre croissant des fréquences.
Par exemple, pour le texte `'abbaeccedeccedadaeefeee'`, cette variable est initialement :

```
cars_ponderes = [['f',1], ['b',2], ['d',3], ['a',4], ['c',4], ['e',9]]
```

Dans ce qui suit, on appelle « poids », les entiers figurant dans les tableaux contenus dans la variable **`cars_ponderes`**.

L'algorithme de Huffman consiste alors, jusqu'à ce que la liste **`cars_ponderes`** ne contienne qu'un élément, à sélectionner les deux tableaux $[s_1,p_1]$ et $[s_2,p_2]$ ayant les poids p_1 et p_2 les plus faibles. On calcule un nouveau tableau $[s,p]$ dans lequel s est une chaîne obtenue en concaténant s_1 et s_2 et p un poids obtenu en additionnant p_1 et p_2. Ce nouveau tableau est inséré dans la liste **`cars_ponderes`** de manière à conserver la propriété de croissance des poids, tandis que les deux tableaux $[s_1,p_1]$ et $[s_2,p_2]$ sont supprimés. Par exemple, avec les valeurs figurant ci-dessus, la première étape consiste à remplacer **`['f',1]`** et **`['b',2]`** par **`['fb',3]`** (il y a ici égalité entre le poids du tableau à insérer et le poids du tableau **`['d',3]`** déjà présent dans **`cars_ponderes`** : l'insertion se fait à gauche dans ce cas) ; puis la seconde étape consiste à remplacer **`['fb',3]`** et **`['d',3]`** par **`['fbd',6]`**.

Parallèlement, on calcule la séquence de bits codant chaque caractère. Les séquences de bits seront représentées ici par des chaînes de caractères formées de **0** et de **1** et appelées des « codes ». Au départ, chaque caractère est associé à une chaîne vide. À chaque étape de l'algorithme, on fusionne deux chaînes de caractères s_1 et s_2 comme on l'a vu précédemment. On modifie alors les codes associés aux caractères présents dans s_1 et s_2 : on ajoute un **0** dans le code de chaque caractère de s_1, et un **1** dans le code de chaque caractère de s_2. Les ajouts de **0** et de **1** se font toujours par la gauche.

Voici un exemple de déroulement de l'algorithme (fig. 1) : à gauche figure le contenu de la variable **`cars_ponderes`** et à droite le tableau des codes des différents caractères. Chaque ligne correspond à une étape de l'algorithme. La première ligne indique les valeurs initiales. Lors de la première étape, on traite les tableaux **`['f',1]`** et **`['b',2]`** et on ajoute un **0** dans le code de **`'f'`** et un **1** dans celui de **`'b'`**. Ensuite, on traite les tableaux **`['fb',3]`** et **`['d',3]`** et on ajoute un **0** dans les codes de **`'f'`** et **`'b'`** (ajout par la gauche) et un **1** dans le code de **`'d'`**, etc.

Liste cars_ponderes	Codes des caractères 'a' , 'b' , 'c' , 'd' , 'e' , 'f'
[['f',1], ['b',2], ['d',3], ['a',4], ['c',4], ['e',9]]	['' , '' , '' , '' , '' , '']
[['fb',3], ['d',3], ['a',4], ['c',4], ['e',9]]	['' , '1' , '' , '' , '' , '0']
[['a',4], ['c',4], ['fbd',6], ['e',9]]	['' , '01' , '' , '1' , '' , '00']
[['fbd',6], ['ac',8], ['e',9]]	['0' , '01' , '1' , '1' , '' , '00']
[['e',9], ['fbdac',14]]	['10' , '001' , '11' , '01' , '' , '000']
[['efbdac',23]]	['110','1001','111','101','0','1000']

Figure 1. Calcul des codes des caractères du texte **`'abbaeccedeccedadaeefeee'`**.

Q24. Écrire une fonction **`insertion(t,q,cars_ponderes)`** prenant en arguments une liste **`cars_ponderes`** telle que décrite ci-dessus, une chaîne de caractères **`t`** et un entier **`q`**, et qui insère le tableau **`[t,q]`** comme élément de la liste **`cars_ponderes`** de telle sorte que celle-ci reste triée par ordre croissant des poids. La fonction renvoie la liste obtenue après insertion.

Par exemple, l'appel
`insertion('fbd', 6, [['a',4], ['c',4], ['e',9]])`
doit renvoyer la liste
`[['a',4], ['c',4], ['fbd',6], ['e',9]]`.

Q25. Écrire une fonction **`codage(cars,freq)`** prenant en argument un tableau de caractères **`cars`** et le tableau des fréquences correspondantes **`freq`**, supposé trié en ordre croissant, et calculant le tableau des codes des caractères du tableau **`cars`** à l'aide de l'algorithme de Huffman. L'appel

```
codage(['f','b','d','a','c','e'], [1, 2, 3, 4, 4, 9])
```

devra renvoyer le tableau de codes figurant en bas à droite de la figure 1.

On supposera disposer d'une fonction **`numero(x)`** qui renvoie, pour chaque caractère **`x`** apparaissant dans le tableau **`cars`**, un entier compris entre 0 et $N-1$ où N est la longueur du tableau **`cars`**, de sorte que pour deux caractères **`x`** et **`y`** distincts, les valeurs renvoyées soient distinctes. On utilisera cette fonction pour déterminer la position, dans le tableau renvoyé, du code du caractère **`x`**. Dans l'exemple de la figure 1, la fonction **`numero`** vérifie :

`numero('a') = 0`, **`numero('b') = 1`**, ..., **`numero('f') = 5`**.

Décodage

Pour décoder un texte (variable **txt_bin** ci-dessous) encodé par l'algorithme précédent, on suppose disposer d'un tableau **cars** contenant les caractères du texte et d'un tableau **code**, contenant les codes de ces caractères, tel que rendu par la fonction **codage** de la question 25. Deux conditions sont supposées remplies par ces tableaux : la position d'un caractère dans le tableau **cars** est la même que celle de son code dans le tableau **code** ; et les éléments du tableau **code** sont rangés par ordre dit lexicographique, c'est-à-dire que pour deux codes distincts $a = a_1 \dots a_n$ et $b = b_1 \dots b_m$, a précède b dans le tableau **code** soit s'il existe un indice i inférieur à m et n tel que $a_1 \dots a_{i-1} = b_1 \dots b_{i-1}$ (condition vide lorsque $i = 1$) et $a_i < b_i$, soit si $m > n$ et $a_1 \dots a_n = b_1 \dots b_n$.

Q26. Ranger les codes `'110'`, `'1001'`, `'111'`, `'101'`, `'0'`, `'1000'` par ordre lexicographique.

En Python, les opérateurs de comparaison usuels permettent directement de comparer les codes (chaînes écrites uniquement à l'aide de 0 et de 1). Par exemple, l'expression `'01' < '11'` est évaluée à `True` tandis que l'expression `'001' < '00'` est évaluée à `False`.

Pour **txt_bin** une chaîne de **0** et de **1** obtenue par compression d'un texte par l'algorithme de Huffman, et **pos** un entier décrivant une position dans cette chaîne telle qu'en cette position commence le code d'un caractère du texte initial, les propriétés du codage de Huffman assurent l'existence d'un unique entier **m** tel que le code **code[m]** soit égal à une sous-chaîne de **txt_bin** commençant à la position **pos**. Par exemple, pour les données suivantes :

```
txt_bin = '0101000000110111011111011110'
code = ['000', '001', '010', '011', '10', '110', '1110', '1111']
cars = [ 'n' ,  'j' ,  'b' ,  ' ' , 'o' ,  '!' ,  'u' ,  'r'  ]
```

pour **pos=0**, on reconnaît que les trois premiers caractères de **txt_bin** forment un code, et on obtient **m=2**, et on trouve dans la table **cars** le premier caractère du texte initial. On passe alors la variable **pos** à la valeur 3, et on identifie le code **'10'** (**m=4**), correspondant à la lettre **'o'**, etc.

Q27. Continuer les calculs de l'exemple ci-dessus, et préciser le texte qui a été compressé. On attend que figurent les valeurs successivement obtenues pour **pos** et **m**.

Q28. Écrire une fonction **decode_car(txt_bin,pos,code)** prenant les arguments décrits ci-dessus et renvoyant l'unique entier **m** tel qu'il existe une sous-chaîne de **txt_bin** commençant à la position **pos** et égale à la chaîne **code[m]**. On demande que le nombre de comparaisons entre des chaînes de caractères effectuées par cette fonction soit $O\big(\log(\texttt{len(code)})\big)$.

Q29. Écrire une fonction **decode_txt(txt_bin,code,cars)** permettant de décoder un texte **txt_bin** compressé à l'aide des informations de compression stockées dans les variables **code** et **cars** décrites plus haut.

Annexe

Base de données

testfin				
nSerie	dateTest	Imoy	Iec	fichierMes
230-588ZX2547	2012-04-22 14-25-45	0.45	0.11	mesure31025.csv
230-588ZX2548	2012-04-22 14-26-57	0.43	0.12	mesure41026.csv
⋮	⋮	⋮	⋮	⋮

production			
Num	nSerie	dateProd	modele
20	230-588ZX2547	2012-04-22 15-52-12	JETDESK-1050
21	230-588ZX2549	2012-04-22 15-53-24	JETDESK-3050
⋮	⋮	⋮	⋮

Fonctions pour les questions 21 à 23

```
# Lors des appels aux fonctions définies dans ce fichier,
# les paramètres freq et cars sont des tableaux de même taille
# dont les éléments sont respectivement des entiers et des caractères,
# comme obtenus à l'aide de la question 20.
# Les paramètres i et j vérifient 0 <= i <= j <= len(cars) - 1.

def echange(cars,freq,i,j):
    freq[i], freq[j] = freq[j], freq[i]
    cars[i], cars[j] = cars[j], cars[i]
```

```
10
11  def aux(cars,freq,i,j):
12      k = j
13      m = j
14      while k > i:
15          k = k - 1
16          if freq[k] > freq[j]:
17              m = m - 1
18              echange(cars,freq,k,m)
19      echange(cars,freq,m,j)
20      return m
21
22  def tri(cars,freq,i,j):
23      if j >= i + 1:
24          m = aux(cars,freq,i,j)
25          tri(cars,freq,i,m-1)
26          tri(cars,freq,m+1,j)
```