ⵜⴰⴳⵍⴷⵉⵜ ⵏ ⵍⵎⵖⵔⵉⴱ
ⵜⴰⵎⴰⵡⴰⵙⵜ ⵏ ⵜⵣⵔⴼⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵜⵉⵏⵣⵡⵉⵏ ⵜⵉⵎⵙⵙⴰⵏⵉⵏ
ⴰⵙⴳⵎⵉ ⵏ ⵍⵎⵖⵔⵉⴱ ⵜⴰⵎⴰⵡⴰⵙⵜ ⵏ ⵜⵣⵔⴼⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵜⵉⵏⵣⵡⵉⵏ ⵜⵉⵎⵙⵙⴰⵏⵉⵏ

المملكة المغربية
وزارة التربية الوطنية والتكوين المهني
والتعليم العالي والبحث العلمي

Royaume du Maroc

Ministère de l'Éducation Nationale, de la Formation Professionnelle, de l'Enseignement Supérieur et de la Recherche Scientifique

Département de l'Enseignement Supérieur et de la Recherche Scientifique

وزارة التجهيز و النقل و اللوجستيك و الماء
Ministère de l'Equipement, du Transport, de la Logistique et de l'Eau

المدرسة الحسانية للأشغال العمومية
École Hassania des Travaux Publics

CNC 2021

Concours National Commun

d'Admission dans les Établissements de Formation d'Ingénieurs et Établissements Assimilés

Épreuve d'**Informatique**

Filière : **MP**

Durée **2 heures**

Cette épreuve comporte **12 pages au format A4**, en plus de la page de garde

Épreuve d'Informatique – Session 2021 – Filière MP

Les candidats sont informés que la précision des raisonnements algorithmiques ainsi que le soin apporté à la rédaction et à la présentation des copies seront des éléments pris en compte dans la notation. Il convient en particulier de rappeler avec précision les références des questions abordées. Si, au cours de l'épreuve, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Remarques générales :

- Cette épreuve est composée d'un exercice et de trois parties tous indépendants ;
- Toutes les instructions et les fonctions demandées seront écrites en Python ;
- Les questions non traitées peuvent être admises pour aborder les questions ultérieures ;
- Toute fonction peut être décomposée, si nécessaire, en plusieurs fonctions.

≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈

Important : Le candidat doit impérativement traiter l'exercice ci-dessous, et inclure les réponses dans les premières pages du cahier de réponse.

Exercice : (4 points)

Développement limité du cosinus

Le développement limité de **cos(x)**, à l'ordre **n**, est :

$$cos(x) = \sum_{i=0}^{n} (-1)^i * \frac{x^{2i}}{(2i)!}$$

1 pt **Q1-** Écrire la fonction **factoriel (k)**, qui reçoit en paramètre un entier positif **k**, et qui retourne la valeur de factorielle **k** : **k! = 1 * 2 * 3 * ... * (k-1) * k**.

NB : La fonction **factoriel (0)** retourne **1**

1 pt **Q2-** Écrire la fonction **calcul (x, k)** qui reçoit en paramètres un réel **x** et un entier positif **k**, et qui retourne la valeur de l'expression suivante :

$$(-1)^k * \frac{x^{2k}}{(2k)!}$$

0.5 pt **Q3-** Déterminer la complexité de la fonction **calcul (x, k)**, et justifier votre réponse.

1.5 pt **Q4-** Écrire la fonction **cosinus (x, n)** qui reçoit en paramètres un réel **x** et un entier positif **n**, et qui retourne la valeur du développement limité de **cos(x)**, à l'ordre **n**.

Partie I : Calcul numérique

Modélisation de la propagation d'un virus

Le modèle 'SEIR'

La crise sanitaire mondiale du Coronavirus Covid-19 a démontré le rôle des modélisations mathématiques dans la prise de décisions politiques et sanitaires.

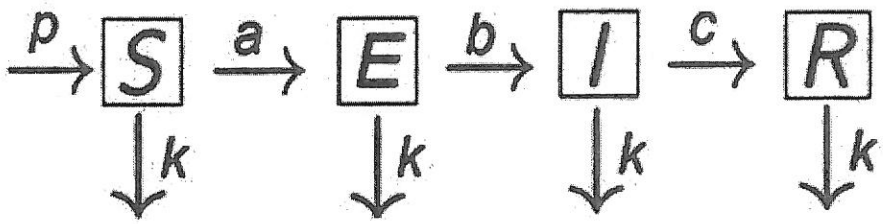
Les modèles à compartiments font partie des premiers modèles mathématiques, à avoir été utilisés en épidémiologie. L'idée est de diviser une population en plusieurs compartiments (groupes d'individus), et définir les règles d'échanges entre ces compartiments.

Le modèle SEIR

Dans ce modèle, les individus sont répartis en 4 compartiments :

- **S**, dans lequel se trouvent les individus sains et susceptibles d'être infectés ;
- **E**, pour les individus infectés qui ne sont pas contagieux ;
- **I**, pour les individus infectés qui sont contagieux ;
- **R**, pour les individus retirés du modèle, non susceptibles d'être infectés, car guéris et immunisés.

Le modèle **SEIR** décrit l'évolution dans le temps du nombre d'individus dans chaque compartiment, et part du schéma suivant :



- p : représente le taux de natalité ;
- a : représente le taux de transmission de la maladie d'un individu infecté à un individu sain ;
- b : représente le taux d'incubation de la maladie ;
- c : représente le taux de guérison ;
- k : représente le taux de mortalité.

Il paraît plus naturel de travailler avec le nombre de personnes dans chaque catégorie, mais certains calculs seront plus simples si on utilise plutôt la proportion de personnes dans chaque catégorie, ce qui nous permet de connaître tout aussi bien la progression de l'épidémie.

On note donc : $S(t)$, $E(t)$, $I(t)$ et $R(t)$ les ***proportions*** des individus dans chaque catégories à l'instant t.

L'évolution des **4** catégories de population peut alors être décrite par le système d'équations différentielles suivant :

$$(1)\quad \begin{cases} S'(t) = -a.S(t).I(t) + p.N(t) - k.S(t) \quad avec : N(t) = S(t) + E(t) + I(t) + R(t) \\ E'(t) = a.S(t).I(t) - (b+k).E(t) \\ I'(t) = b.E(t) - (c+k).I(t) \\ R'(t) = c.I(t) - k.R(t) \end{cases}$$

Les dérivées permettent de connaître la variation (c'est à dire si c'est croissant ou décroissant) des fonctions ***S(t)***, ***E(t)***, ***I(t)*** et ***R(t)*** en fonction du temps ***t***, afin d'en décrire l'évolution au cours du temps.

On suppose que les modules **numpy** et **matplotlib.pyplot** sont importés :

```
import numpy as np
import matplotlib.pyplot as pl
```

Méthodes de Runge-Kutta :

Dans cette partie, nous allons nous intéresser à une méthode d'ordre **4**, appelée **Runge-Kutta 4** (RK4), permettant de résoudre numériquement les équations différentielles du premier ordre, avec une condition initiale, sous la forme :

$$\begin{cases} y' = f(t, y(t)) \quad ; \quad \forall\, t \in [t_0, t_0 + \mathrm{T}] \\ y(t_0) = y_0 \end{cases}$$

On note $\mathrm{I} = [t_0, t_0 + \mathrm{T}]$ l'intervalle de résolution. Pour un nombre de nœuds **N** donné, soit $t_n = t_0 + n*h$, avec $n = 0, 1, 2, \ldots, \mathrm{N}$, une suite de nœuds de I induisant une discrétisation de I en sous-intervalles $\mathrm{I_n} = [t_n , t_{n+1}]$. La longueur h de ces sous-intervalles est appelée **pas** de discrétisation. Le pas de discrétisation h est donné par : $h = \frac{T}{N}$.

Soit $\mathrm{y_n}$ l'approximation de la solution exacte $y(t_n)$ au nœud t_n. Le schéma de la méthode **RK4** est donné par :

- y_0 donné
- $y_{n+1} = y_n + \frac{h}{6} * (k_1 + 2(k_2 + k_3) + k_4)$

Avec :

- $k1 = f(t_n, y_n)$
- $k2 = f(t_n + \frac{h}{2}, y_n + k1 * \frac{h}{2})$
- $k3 = f(t_n + \frac{h}{2}, y_n + k2 * \frac{h}{2})$
- $k4 = f(t_n + h, y_n + k3 * h)$

Q.1- Écrire la fonction **def RK4 (f, t_0, T, y_0, N):** qui reçoit en paramètres la fonction f : (t , y) → f (t , y), t_0 et T tels que $[t_0 , t_0 + T]$ est l'intervalle de résolution, la condition initiale y_0 et N est le nombre de nœuds. La fonction retourne **tt** = $[t_0 , t_1 , \dots , t_N]$ **et** Y = $[y_0 , y_1 , \dots , y_N]$ en utilisant le schéma de RK4.

Q.2- Le but de cette question est d'utiliser la fonction précédente **RK4**, pour résoudre numériquement le système d'équations différentielles **(1)**.

Q.2.a- En posant **Y(t) = [S(t) , E(t) , I(t) , R(t)]**, montrer que le système d'équations différentielles **(1)** peut s'écrire sous la forme **Y'(t) = F(t, Y(t))** avec :

$$F : I * IR^4 \rightarrow IR^4$$

$$(t , X = [X[0] , X[1] , X[2] , X[3]]) \rightarrow [z_0 , z_1 , z_2 , z_3]$$

z_0 , z_1, z_2 et z_3 sont à préciser en fonction de **X[0], X[1], X[2]** et **X[3]**.

Q.2.b- On suppose que **p, a, b, c** et **k** sont des variables globales initialisées par les valeurs suivantes :

p = 0.004 ; a = 0.8 ; b = 0.5 ; c = 0.09 ; k = 0.005

Écrire, en langage Python, la fonction **def F(t, X):** qui reçoit en paramètres un réel **t**, et un tableau np.array **X** de dimension **1** ligne et **4** colonnes. La fonction retourne le tableau np.array de même dimension, image de **(t, X)** par la fonction **F**.

Q.3- En utilisant la fonction précédente **RK4**, écrire un script python qui permet de tracer la représentation graphique ci-dessous (Figure 1), des fonctions **S(t), E(t), I(t), R(t)** et **N(t)**. Le nombre de points générés dans chaque courbe est $N=10^3$.

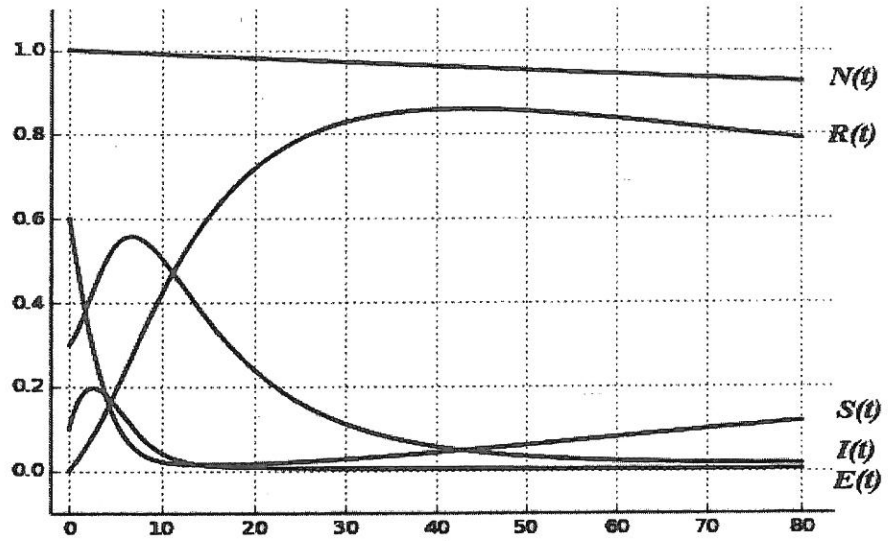


Figure 1

Partie II : Bases de données et langage SQL

Dans le but de suivre l'évolution de la pandémie du Covid-19 dans chaque pays, on propose d'utiliser une base de données composée de **3** tables : **Pays**, **Villes** et **Quotidien**.

a- Structure de la table 'Pays' :

La table **'Pays'** est composée de trois champs : Le champ '**codPays**' de type texte. Ce champ représente la clé primaire de la table 'Pays'. Le champ '**nom**' de type texte, contient le nom de chaque pays.

Exemples :

codPays	nom
DEU	Allemagne
FRA	France
ESP	Espagne
ARN	Suède
ITA	Italie
...	...

b- Structure de la table 'Villes' :

La table **'Villes'** est composée de 4 champs : Le champ '**codVille**' de type texte. Il représente la clé primaire de la table 'Ville'. Le champ '**nom**' de type texte, il contient le nom de chaque ville. Le champ '**population**' de type entier, il contient le nombre de personnes dans chaque ville. Le champ '**codP**' de type texte, il contient le code du pays de chaque ville.

Exemples :

codVille	nom	population	codP
CIA	Rome	2 844 395	ITA
NAP	Napoli	954 244	ITA
PAR	Paris	2 148 271	FRA
TXL	Berlin	3 748 148	DEU
LYS	Lyon	1 669 730	FRA
	...	...	...

c- Structure de la table 'Quotidien' :

La table 'Quotidien' est composée de 5 champs : Le champ '**jour**' de type date, il contient la date de chaque jour. Le champ '**infectés**' de type entier, il contient le nombre de nouveaux cas de personnes infectées par le virus, chaque jour dans chaque ville. Le champ '**rétablis**' de type entier, il contient le nombre de nouveaux cas de personnes guéries du virus, chaque jour dans chaque ville. Le champ '**décédés**' de type entier, il contient le nombre de nouveaux cas de personnes décédées à cause du virus, chaque jour dans chaque ville. Le champ '**codV**' de type texte, il contient les codes des villes.

Exemples :

jour	infectés	rétablis	décédés	codeV
2020-07-15	4653	3458	172	CIA
2020-07-15	6074	4892	493	TXL
2020-07-15	1762	2059	251	LYS
2020-07-16	3721	5973	85	CIA
...	...			...

Q.1- Le jour du '**17/09/2020**', la ville de code '**TXL**' a enregistré **2513** nouveaux cas de personnes infectées, **2847** nouveaux cas de personnes rétablies, et **65** nouveaux cas de personnes décédées.

Écrire, en langage **SQL**, la requête qui ajoute cet enregistrement dans la table 'Quotidien'.

Q.2- Écrire, en langage **SQL**, la requête qui donne pour résultat, le compte des villes qui ont enregistré des nombres de nouveaux cas de personnes infectées, composés d'au moins **5** chiffres, et ce le jour du '**27/08/2020**'.

Q.3- Écrire, en langage **SQL**, la requête qui donne pour résultat, le nombre de nouveaux cas de personnes infectées, le nombre nouveaux cas de personnes rétablies et le nombre de nouveaux cas de personnes décédées, chaque jour du mois de **Septembre 2020**, dans toutes les villes du pays '**Italie**'.

Q.4- Écrire, en algèbre relationnelle, la requête de la question **Q.3**

Q.5- *Le **taux d'incidence TI** pour une ville est égal au nombre de nouveaux cas de personnes infectées dans cette ville, pendant une période donnée, divisé par la population de cette ville. Le **TI** s'exprime en nombre de personnes pour **100000** personnes.*

Écrire, en langage **SQL**, la requête qui donne pour résultat, les codes des pays, les noms des villes et le **TI** de chaque ville, pendant l'année **2020**, tels que **TI** est compris entre **500** et **1000**, triés dans l'ordre croissant du **TI**.

Q.6- *Le **taux de guérison TG** pour un pays est égal au nombre total de personnes rétablies dans ce pays, divisé par le nombre total de personnes infectées dans ce même pays. Le **TG** s'exprime en pourcentage.*

Écrire, en langage **SQL**, la requête qui donne pour résultat, les noms des pays et le **TG** pour chaque pays, tels que **TG > 70%**, triés dans l'ordre décroissant du **TG**.

Q.7- *Le **taux de mortalité TM** pour une ville est égal au nombre total de personnes décédées dans cette ville, divisé par le nombre total de personnes infectées dans cette même ville. Le **TM** s'exprime en pourcentage.*

Écrire, en langage **SQL**, la requête qui donne pour résultat, les noms des pays, le plus petit **TM**, le plus grand **TM** et la moyenne des **TM** des villes dans chaque pays, triés dans l'ordre croissant du nombre de villes dans chaque pays.

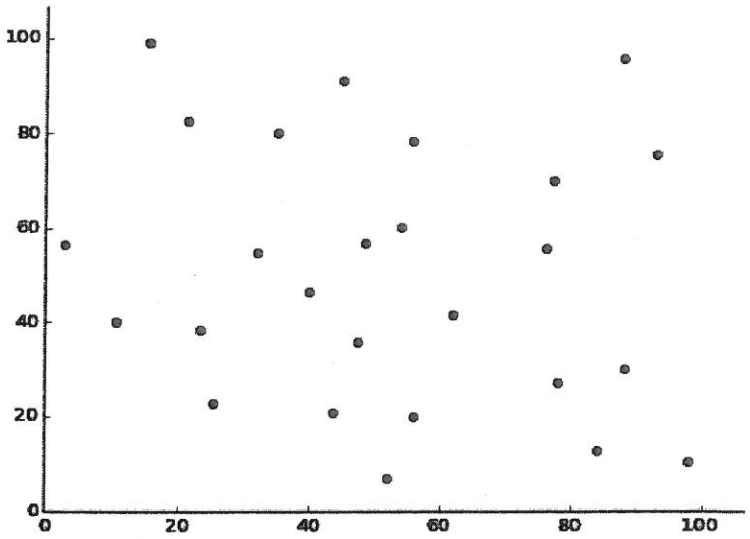
Partie III : Problème Python

Géométrie algorithmique

Recherche des deux points les plus rapprochés

La **géométrie algorithmique** est un domaine qui traite des algorithmes manipulant des concepts géométriques. La discipline qui a le plus contribué historiquement au développement de la géométrie algorithmique est l'infographie. Toutefois, à l'heure actuelle, la géométrie algorithmique se voit fréquemment impliquée dans des problèmes d'algorithmique générale. Aujourd'hui, la géométrie algorithmique est utilisée en infographie, en robotique, pour les systèmes d'information géographique, ...

En géométrie algorithmique, la **recherche des deux points les plus rapprochés** est le problème qui consiste à trouver une paire de points, d'un ensemble fini de points dans un espace métrique, dont la distance est minimale. Il fait partie des problèmes fondateurs de la géométrie algorithmique.



Ce problème de recherche voit son utilité dans les transports aériens ou maritimes par exemple. Il est utilisé par les contrôleurs aériens pour repérer les avions les plus proches les uns des autres (l'espace considéré ici est à 3 dimensions), et ainsi prévenir le risque de collision.

Dans cette partie, nous allons nous intéresser au problème suivant : étant donné un ensemble de points du plan, on cherchera à trouver le couple de points les plus rapprochés au sens de la distance euclidienne.

Un point du plan sera représenté par un tuple de réels **(x, y)** représentant ses coordonnées dans un repère orthonormé. L'ensemble des points considérés sera quant à lui stocké dans une liste **P**.

Nous supposerons que la liste **P** contient au moins deux points, et que les points de **P** sont tous différents deux à deux. Nous supposerons aussi que la liste **P** contient une seule paire de points, à distance minimale.

Exemple :

Les points de la figure précédente sont représentés par la liste **P** suivante :

P = [(84.04, 12.68), (21.23, 82.75), (35.00, 80.00), (78.00, 27.00), (52.01, 7.03), (48.59, 56.77), (88.16, 30.04), (77.3, 69.82), (40.00, 46.46), (10.72, 39.91), (97.98, 10.52), (31.86, 54.75), (54.19, 60.09), (87.97, 95.69), (23.21, 38.41), (93.19, 75.47), (55.82, 78.18), (25.21, 22.75), ...]

NB : Dans tous les exemples de cette partie, on utilisera cette liste **P**.

Distance euclidienne entre deux points :

La distance euclidienne entre deux points **A** et **B** de coordonnées (x_A, y_A) et (x_B, y_B), est calculée par la formule suivante :

$$\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

Q.1- Écrire la fonction ***distance (A, B)*** qui reçoit en paramètres deux tuples **A** et **B** qui représentent deux points. La fonction retourne la valeur de la distance euclidienne entre les deux points **A** et **B**.

Exemple :

A = (84.04, 12.68) et B = (21.23, 82.75)

La fonction **distance (A, B)** retourne le nombre : 94.10048352691925

On suppose que la complexité temporelle de la fonction racine carrée est constante. Dans la suite de cette partie, nous nous intéresserons à la mise au point de deux algorithmes différents, qui permettent de trouver les deux points les plus rapprochés, dans la liste **P**.

A- Algorithme naïf :

Le premier algorithme est un algorithme naïf qui consiste à faire une recherche exhaustive : considérer successivement tous les couples de points dans la liste **P**, ce qui peut se faire facilement à l'aide de deux boucles imbriquées.

Q.2- Écrire la fonction ***plus_proches (P)*** qui reçoit en paramètre une liste de points **P**. En utilisant le principe de l'algorithme naïf précédent, la fonction retourne deux tuples qui représentent les deux points les plus rapprochés dans la liste **P**.

Exemple :

La fonction **plus_proche (P)** retourne les deux points : **(48.59, 56.77)** , **(54.19, 60.09)**

Q.3- Déterminer la complexité de la fonction **plus_proches (P)**.

Justifier votre réponse.

B- Algorithme efficace

Le deuxième algorithme est basé sur le paradigme : *diviser pour régner*. Cet algorithme nécessite de disposer de deux copies de la liste des points **P** : l'une triée dans l'ordre croissant des abscisses des points, et l'autre triée dans l'ordre croissant des ordonnées des points.

Q.4- Écrire la fonction ***tri_points (P, k)*** qui reçoit en paramètre la liste **P**, et **k** un entier tel que : **k∈ {0, 1}**. La fonction retourne une copie de la liste **P**, triée dans l'ordre croissant du **k**ème élément de chaque tuple de la liste **P**.

Exemples :

- La fonction **tri_points (P, 0)** retourne la liste triée selon les abscisses des points :
  [(**2.99**, 56.44), (**10.72**, 39.91), (**15.33**, 99.03), (**21.23**, 82.75), (**23.21**, 38.41), …]

- La fonction **tri_points (P, 1)** retourne la liste triée selon les ordonnées des points :
  [(52.01, **7.03**), (97.98, **10.52**), (84.04, **12.68**), (56.0, **20.0**), (43.7, **20.82**), …]

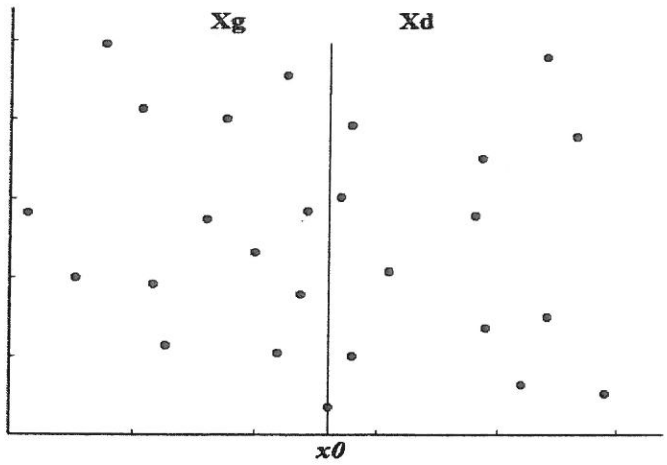
On suppose que la liste **X** est la copie de **P**, triée dans l'ordre croissant des abscisses des points.

On considère deux variables **n** et **x0**, telles que :

- **n** ← Longueur de **X** ;
- **x0** ← l'abscisse du point au milieu de la liste **X**, à l'indice **n/2**.

Ensuite, à partir des éléments de la liste triée **X**, on crée deux listes **Xg** et **Xd**, telles que :

- **Xg** contient la première moitié de la liste **X** : les éléments de **X** d'indice **i** tel que **$0 \leq i < n/2$** ;
- **Xd** contient la deuxième moitié de la liste **X** : les éléments de **X** d'indice **i** tel que **$n/2 \leq i < n$**.



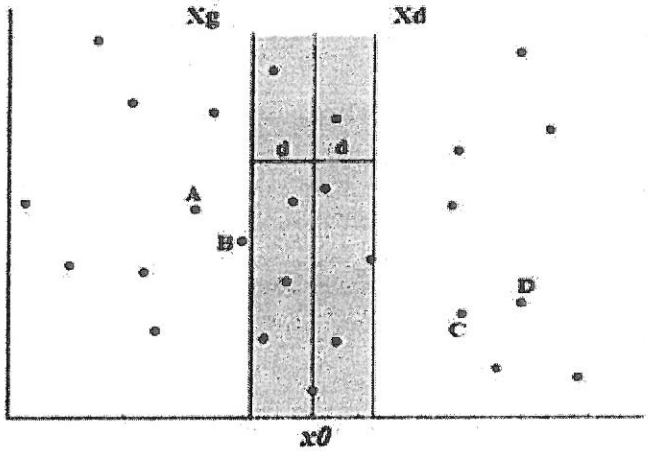
Supposons que **A** et **B** sont les deux points les plus rapprochés dans la liste **Xg**, et que **d1** est la distance entre **A** et **B**.

Supposons aussi que **C** et **D** sont les deux points les plus rapprochés dans la liste **Xd**, et que **d2** est la distance entre **C** et **D**.

On pose : **d** ← Le minimum de **d1** et **d2**

Si deux points de la liste **P** sont à une distance strictement inférieure à **d**, alors forcément :

- L'un des deux points appartient à la liste **Xg**, et l'autre appartient à la liste **Xd** ;
- Et les deux points se trouvent dans la bande verticale qui contient les points d'abscisses comprises entre **x0-d** et **x0+d**.



Q.5- Écrire la fonction ***bande_verticale (Y, x0, d)*** qui reçoit en paramètres la liste ***Y*** des points triés dans l'ordre croissant des ordonnées, et les deux réels **x0** et **d**. La fonction retourne la liste ***B*** des points de ***Y*** dont les abscisses sont comprises entre **x0-d** et **x0+d**, et triés dans l'ordre croissant des ordonnées.

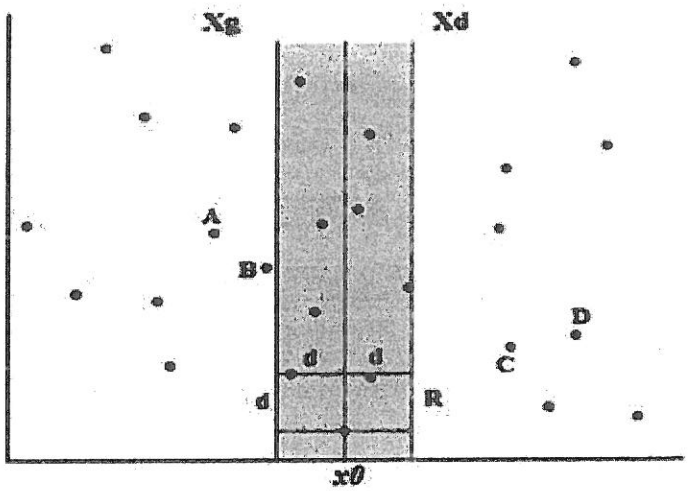
Exemple :

Y = [(52.01, **7.03**), (97.98, **10.52**), (84.04, **12.68**), (56.0, **20.0**), (43.7, **20.82**), (25.21, **22.75**), …]

On suppose que **x0 = 52.01** et **d = 10.605055398252285**

La fonction **bande_verticale (Y, x0 , d)** retourne la liste **B** suivante : [(52.01, **7.03**), (56.0, **20.0**), (43.7, **20.82**), (47.54, **35.79**), (62.0, **41.5**), (48.59, **56.77**), (54.19, **60.09**), (55.82, **78.18**), (45.0, **91.0**)]

La liste **B** contient les points situés dans la bande verticale de largeur **2*d**, et centrée en la droite verticale passante par l'abscisse **x0**.



Supposons qu'ils existent deux points B_i et B_j de la liste **B**, tels que la distance entre ces deux points est inférieure à **d**. Les points B_i et B_j sont donc dans un rectangle **R** de dimension **d*2d** centré en la droite verticale passante par **x0**. On suppose que le point B_i est sur le côté inférieur du rectangle **R**.

Montrons que le rectangle **R** contient au plus **8** points de **B**. Pour cela, on considère le carré de dimension **d*d** qui forme la moitié gauche du rectangle **R** :

- Puisque tous les points de **Xg** sont distants d'au moins **d**, il y en a au plus **4** points qui se trouvent dans ce carré : **1** point par sous-carré de dimension **(d/2)*(d/2)** de diagonale $d\sqrt{2}/2 < d$.
- De même, il y a au plus **4** points qui se trouvent dans la moitié droite de **R**.

Donc, le rectangle **R** contient au plus **8** points de **B**, qui sont classés par ordonnées croissantes, et on a bien : $i < j \leq i+7$ et la distance entre les points B_i et B_j est inférieure à **d**.

Ainsi, on déduit que si deux points de **B** sont à une distance inférieure à **d**, alors ils sont situés à au plus **7** positions l'un de l'autre dans la liste **B**.

Q.7- Écrire la fonction de complexité linéaire ***plus_proches_bande (B)*** qui reçoit en paramètre la liste **B** qui contient les points de la bande verticale. La fonction retourne les deux points les plus rapprochés dans la liste **B**.

Exemple :

B = [(52.01, 7.03), (56.0, 20.0), (43.7, 20.82), (47.54, 35.79), (62.0, 41.5), (48.59, 56.77), (54.19, 60.09), (55.82, 78.18), (45.0, 91.0)]

La fonction **plus_proches_bande (B)** retourne les deux points : (48.59 , 56.77) , (54.19 , 60.09)

Q.8- Justifier que la fonction **plus_proches_bande (B)** est de complexité linéaire.

En utilisant les fonctions précédentes, on peut mettre en point un algorithme récursif permettant de trouver les deux points les plus rapprochés à partir des deux listes **X** *(liste des points triés par abscisses croissantes)* et **Y** *(liste des points triés par ordonnées croissantes)* :

a. **n** ← longueur de la liste **X**
b. Si **n<=3** alors retourner les deux points les plus rapprochés, en utilisant l'algorithme naïf ;
c. **x0** ← l'abscisse du point au milieu de la liste **X**
d. Créer deux listes **Xg** et **Xd**, qui contiennent respectivement la première moitié et la deuxième moitié de la liste **X** ;
e. Avec un appel récursif, déterminer les deux points **A** et **B** les plus rapprochés dans **Xg** ;
f. **d1** ← la distance entre **A** et **B** ;
g. Avec un appel récursif, déterminer les deux points **C** et **D** les plus rapprochés dans **Xd** ;

h. **d2** ← la distance entre **C** et **D** ;

i. Déterminer la paire de points (**E, F**) les plus rapprochés parmi les pairs (**A, B**) et (**C, D**), ainsi que la distance **d** entre **E** et **F** ;

j. Déterminer la liste **B** qui représente la bande verticale de largeur **2*d**, centrée en la droite verticale passante par **x0** ;

k. Si la longueur de la liste **B** est inférieure strictement à **2**, alors on retourne la paire de points (**E, F**) ;

l. Si la liste **B** contient au moins deux points, alors déterminer les deux points **P** et **Q** les plus rapprochés dans **B**, ainsi que la distance **d3** entre **P** et **Q** ;

m. Retourner la paire de points à distance minimale parmi les pairs (**E, F**) et (**P, Q**)

Q.9- Écrire la fonction récursive ***plus_proches_rec (X, Y)*** qui reçoit en paramètres les deux listes **X** et **Y**, et qui retourne les deux points les plus rapprochés.

Exemple :

X = [(2.99, 56.44), (10.72, 39.91), (15.33, 99.03), (21.23, 82.75), (23.21, 38.41), …]

Y = [(52.01, 7.03), (97.98, 10.52), (84.04, 12.68), (56.0, 20.0), (43.7, 20.82), …]

La fonction **plus_proches_rec (X, Y)** retourne les points : (48.59, 56.77) , (54.19, 60.09)

Q.10- Déterminer la complexité de la fonction **plus_proches_rec (X, Y)**, et justifier votre réponse.

≈≈≈≈≈≈≈≈ **FIN DE L'ÉPREUVE** ≈≈≈≈≈≈≈≈