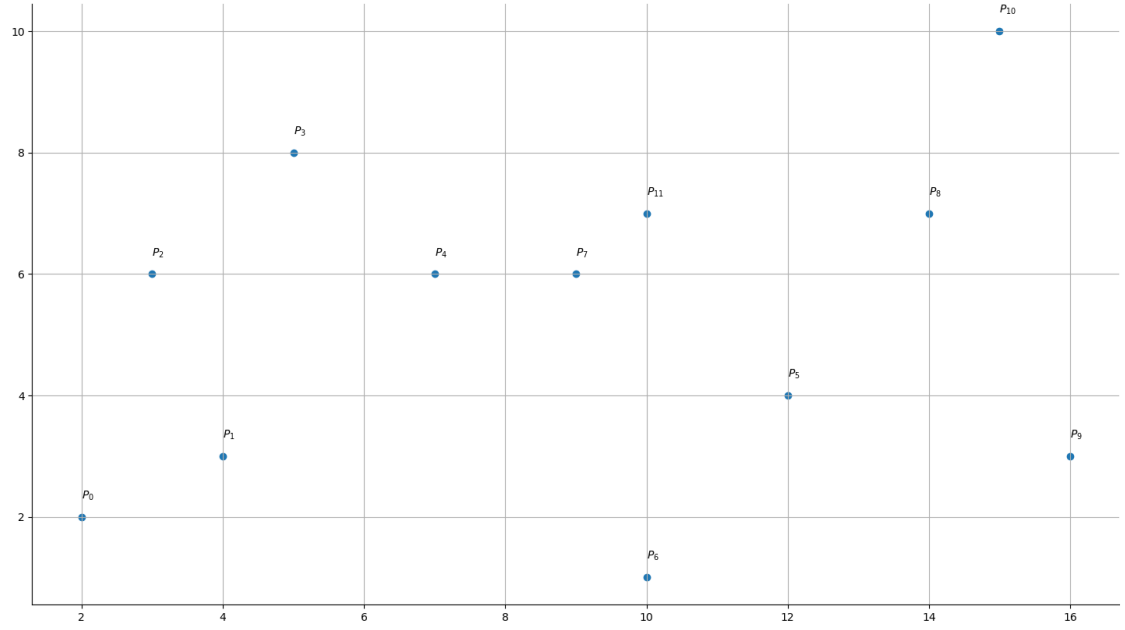
TP PC* : Points les plus proches dans un nuage

September 17, 2024

Considérons un nuage de $n \geq 3$ points du plan définis par leurs coordonnées cartésiennes dans un repère orthonormé. La distance entre deux points deux points P, Q est la norme euclidienne de $\overrightarrow{PQ}$. Le problème des *deux plus proches points du plan* consiste à identifier les deux points les plus proches dans le nuage.



Une solution serait de calculer les distances entre toute paire de sommets et de choisir la meilleure. On aurait alors une complexité de l'ordre de $\dfrac{n(n-1)}{2}$. On peut cependant faire beaucoup mieux.

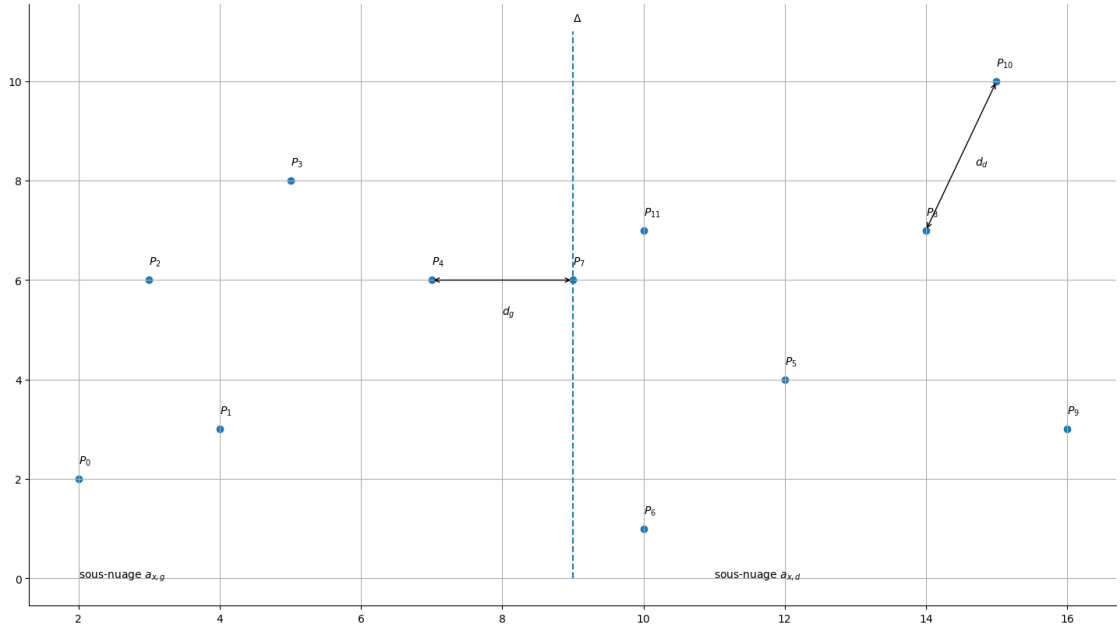
Notons a le tableau qui rassemble les coordonnées du nuage. On construit deux tableau a_x et a_y qui contiennent respectivement les éléments de a triés par abscisses croissantes et par ordonnées croissantes.

Si $n \leq 3$, un algorithme direct détermine les deux points les plus proches parmi les deux ou trois points, ce qui constitue un cas de base pour la suite de l'algorithme. Si $n \geq 4$, l'algorithme procède comme suit :

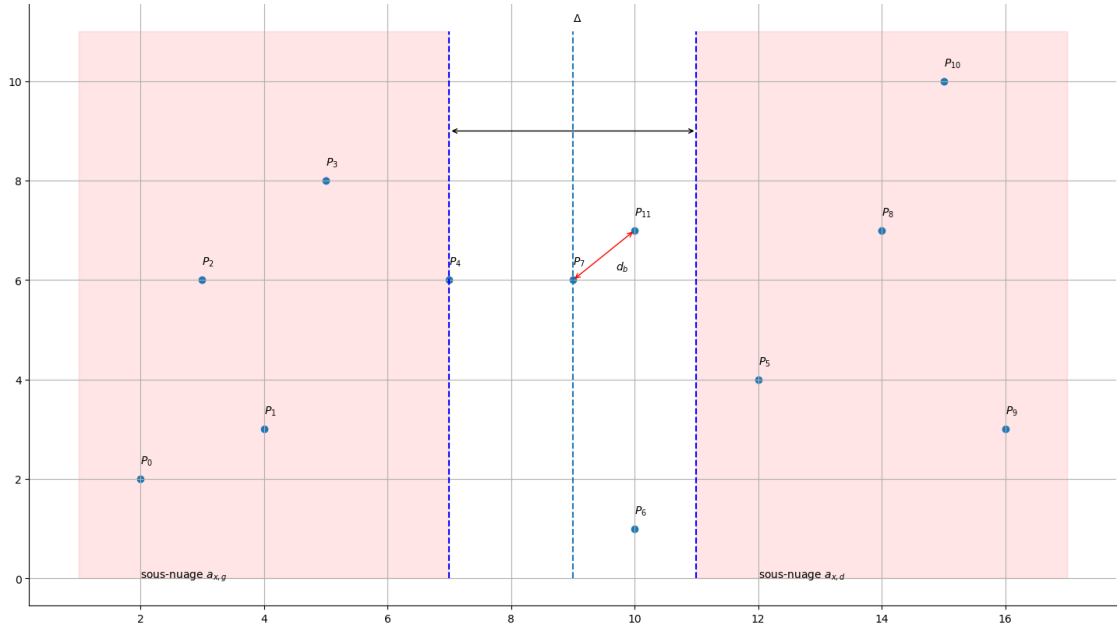
- **Diviser** Le tableau a_x est divisé en deux sous-tableaux $a_{x,g}$ et a_{x_d} de tailles $\lfloor n/2 \rfloor$ et $\lceil n/2 \rceil$ respectivement. les points stockés dans ces sous-tableaux sont situés de part et d'autre de la

droite verticale Δ d'équation $x = x_{\text{med}}$ ou $x_{\text{med}} = \lfloor n/2 \rfloor$. Cette valeur x_{med} est l'abscisse du point $a_x[\lfloor n/2 \rfloor - 1]$

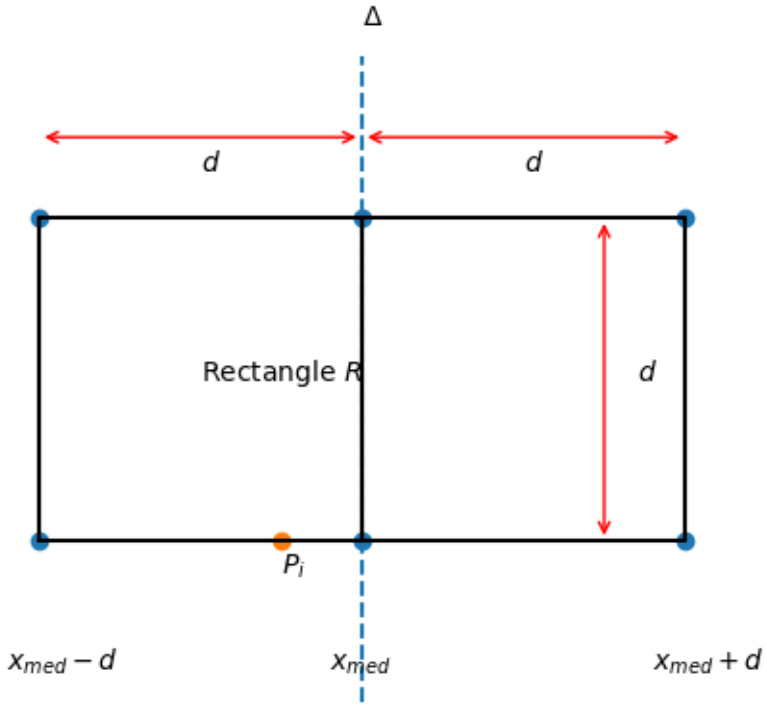
- **Régner** L'algorithme, appelé récursivement sur $a_{x,g}$ et $a_{x,d}$, détermine les deux distances minimales d_g et d_d ainsi que les deux couples de points associés à ces distances. Seul le couple de points correspondant à $d = \min(d_g, d_d)$ est conservé.



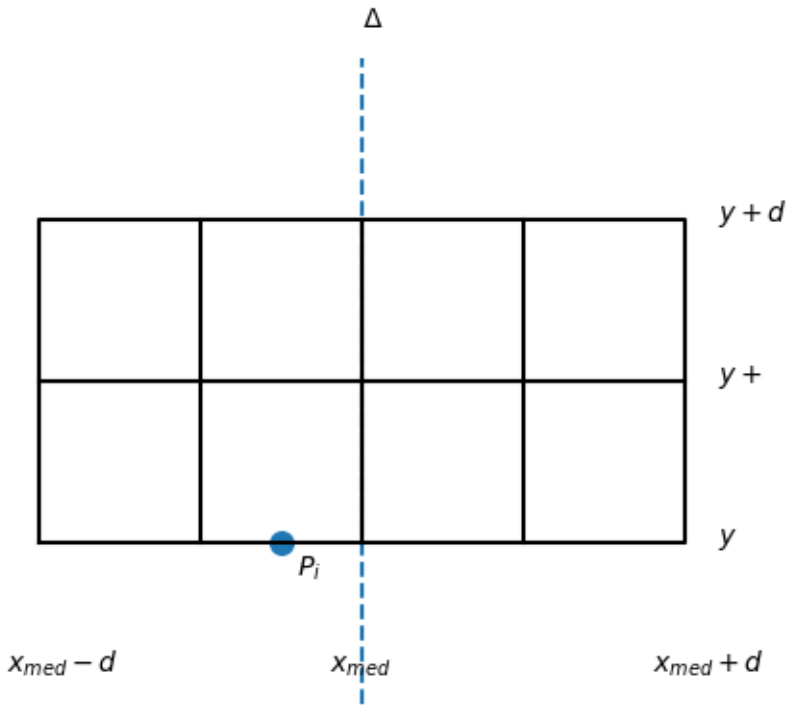
- **Rassembler** Une fois un couple de plus proches points identifiés dans $a_{x,g}$ ou $a_{x,d}$, on étudie une bande verticale de largeur $2d$ centrée sur Δ. Il est en effet possible que deux points de cette bande soient plus proches que ceux déjà trouvés. Il faut donc une étape de plus pour examiner les points de la bande. Cette recherche se fait en considérant le sous-tableau a_{bande} de a_y qui ne contient que les points dont les abscisses sont comprises entre $x_{\text{med}} - d$ et $x_{\text{med}} + d$.



- **Rassembler (suite)** On montre que pour chaque point de la bande verticale, il suffit d'examiner uniquement les 7 points qui le suivent dans le tableau a_{bande}. Pour cela, considérons un point $P_i(x, y)$ et cherchons les points P de la bande verticale tels que $P_iP < d$. Ils sont nécessairement dans le rectangle R de hauteur d et de largeur $2d$ centré sur Δ dont P_i est sur le côté inférieur :



- **Rassembler (suite)** Découpons notre rectangle en 8 carrés de de côtés $\frac{d}{2}$ comme ci-dessous. Chacun des petits carrés contient au plus un point du nuage initial. Prenons en effet deux points : s'ils sont du même côté de Δ, leur distance est supérieure à $\frac{d}{\sqrt{2}}$ et ils ne peuvent appartenir au même carré; et s'ils sont de part et d'autre de Δ, ils ne peuvent être dans le même carré car aucun n'est traversé par Δ. En conclusion, il y a au plus 8 points de la bande verticale dans le rectangle R. Au total, il y au maximum 7 points en plus de P_i dans le rectangle !



- **Rassembler (fin)** Il suffit donc de parcourir tous les points de la bande et d'examiner les 7 points suivants pour déterminer s'il existe une paire de points dont la distance est inférieure à d. Ce sera notre paire optimale si on en trouve une, sinon la paire optimale est dans un des deux sous-nuages a_g ou a_d.

Q1

Ecrire la fonction `dist(p1,p2)` qui calcule la distance euclidienne entre deux points du plan donnés par leurs tuples de coordonnées.

[7]:
```
dist((1,2),(-2,3))
```

[7]: 3.1622776601683795

Q2

Ecrire la fonction `sort_abs(a)` qui retourne une version triée par abscisse croissante d'un tableau de points. Le tableau n'est pas modifié. Faire de même avec `sort_ord(a)` qui trie selon les ordonnées croissantes.

[9]:
```
lesx = [2,4,3,5,7,12,10,9,14,16,15,10]
lesy = [2,3,6,8,6,4,1,6,7,3,10,7]
a = list(zip(lesx,lesy))
print(a)
print(sort_abs(a))
print(sort_ord(a))
```

```
[(2, 2), (4, 3), (3, 6), (5, 8), (7, 6), (12, 4), (10, 1), (9, 6), (14, 7), (16,
3), (15, 10), (10, 7)]
[(2, 2), (3, 6), (4, 3), (5, 8), (7, 6), (9, 6), (10, 1), (10, 7), (12, 4), (14,
7), (15, 10), (16, 3)]
[(10, 1), (2, 2), (4, 3), (16, 3), (12, 4), (3, 6), (7, 6), (9, 6), (14, 7),
(10, 7), (5, 8), (15, 10)]
```

Q2bis

Ecrire une fonction `make_pos(a)` qui prend en paramètre un nuage de points et retourne le dictionnaire (point, position dans le nuage).

[11]:
```
lesx = [2,4,3,5,7,12,10,9,14,16,15,10]
lesy = [2,3,6,8,6,4,1,6,7,3,10,7]
a = list(zip(lesx,lesy))
d = make_pos(a)
print(d)
```

```
{(2, 2): 0, (4, 3): 1, (3, 6): 2, (5, 8): 3, (7, 6): 4, (12, 4): 5, (10, 1): 6,
(9, 6): 7, (14, 7): 8, (16, 3): 9, (15, 10): 10, (10, 7): 11}
```

Q2ter

Ecrire une fonction `make_new_ay(ay,d,lo,mid,hi)` qui prend en paramètre un tableau (censé être ordonné par ordonnées croissantes), un dictionnaire des positions des points dans un autre tableau (en fait, les positions dans a_x), une borne inférieure des positions étudiées, une borne médiane et une borne supérieure.

La fonction retourne deux sous-tableaux de a_y :

- le sous-tableau des points de a_y qui occupent des positions entre `lo` et `mid-1` dans a_x.
- le sous-tableau des points de a_y qui occupent des positions entre `mid` et `hi-1` dans a_x.

[13]:
```
lesx = [2,4,3,5,7,12,10,9,14,16,15,10]
lesy = [2,3,6,8,6,4,1,6,7,3,10,7]
a = list(zip(lesx,lesy))
print(a)
ax, ay = sort_abs(a), sort_ord(a)
d = make_pos(ax)
print(ax)
print(ay)
print(d)
make_new_ay(ay,d,3,5,8)
```

```
[(2, 2), (4, 3), (3, 6), (5, 8), (7, 6), (12, 4), (10, 1), (9, 6), (14, 7), (16,
3), (15, 10), (10, 7)]
[(2, 2), (3, 6), (4, 3), (5, 8), (7, 6), (9, 6), (10, 1), (10, 7), (12, 4), (14,
7), (15, 10), (16, 3)]
[(10, 1), (2, 2), (4, 3), (16, 3), (12, 4), (3, 6), (7, 6), (9, 6), (14, 7),
(10, 7), (5, 8), (15, 10)]
{(2, 2): 0, (3, 6): 1, (4, 3): 2, (5, 8): 3, (7, 6): 4, (9, 6): 5, (10, 1): 6,
(10, 7): 7, (12, 4): 8, (14, 7): 9, (15, 10): 10, (16, 3): 11}
```

[13]:
```
([(7, 6), (5, 8)], [(10, 1), (9, 6), (10, 7)])
```

Q3

Ecrire la fonction `closest_pair_small(a)` qui prend en paramètre un tableau d'au plus 3 points et retourne le couple de points les plus proches. Une exception est soulevée si le tableau est trop long.

[15]:
```
a = [(5,2),(4,6),(2,3)]
closest_pair_small(a)
```

[15]:
```
((5, 2), (2, 3))
```

Q4

Ecrire la fonction `make_strip(a,xmin,xmax)` qui construit la bande verticale à partir d'un tableau de points a rangés par ordre croissant d'ordonnées, d'une abscisse gauche $x_{\min}$ et d'une abscisse droite $x_{\max}$.

[17]:
```
lesx = [2,4,3,5,7,12,10,9,14,16,15,10]
lesy = [2,3,6,8,6,4,1,6,7,3,10,7]
a = list(zip(lesx,lesy))
print(a)
print(sort_ord(a))
make_strip(a,1.5,7.5)
```

```
[(2, 2), (4, 3), (3, 6), (5, 8), (7, 6), (12, 4), (10, 1), (9, 6), (14, 7), (16,
3), (15, 10), (10, 7)]
```

```
[(10, 1), (2, 2), (4, 3), (16, 3), (12, 4), (3, 6), (7, 6), (9, 6), (14, 7),
(10, 7), (5, 8), (15, 10)]
```

[17]:
```
[(2, 2), (4, 3), (3, 6), (5, 8), (7, 6)]
```

Q5

Ecrire la fonction `closest_pair_in_strip(a)` qui prend en paramètres une bande verticale comme calculée par la fonction précédente. La fonction retourne les deux points les plus proches dans le tableau a en examinant à chaque tour les 7 points qui suivent le point courant.

[19]:
```
lesx = [2,4,3,5,7,12,10,9,14,16,15,10]
lesy = [2,3,6,8,6,4,1,6,7,3,10,7]
a = sort_ord(list(zip(lesx,lesy)))
closest_pair_in_strip(a)
```

[19]:
```
((9, 6), (10, 7))
```

Q6

Ecrire la fonction `closest_pair(a)` qui calcule les deux points les plus proches du nuage a en appliquant l'algorithme

[21]:
```
lesx = [2,4,3,5,7,12,10,9,14,16,15,10,12.5]
lesy = [2,3,6,8,6,4,1,6,7,3,10,7,4.2]
a = list(zip(lesx,lesy))
closest_pair(a)
```

[21]:
```
((12, 4), (12.5, 4.2))
```

[22]:
```
lesx = [2,4,3,5,7,12,10,9,14,16,15,10]
lesy = [2,3,6,8,6,4,1,6,7,3,10,7]
a = list(zip(lesx,lesy))
closest_pair(a)
```

[22]:
```
((9, 6), (10, 7))
```

0.0.1 Q7

Donner une relation de récurrence suivie par la complexité temporelle au pire $C(n)$ de `closest_pair(a)` si $|a| = n$.