Algorithmique
M2 Data Science
Convex hull

AGBOHOUTO Omraam Olivier YOKAM TATEU Muriel AKROUT
Leyth SUN Yannick

Section 1

Introduction de la problématique

Introduction de la problématique (1)

On appelle **enveloppe convexe** d'un ensemble fini de points, le plus petit ensemble convexe qui contient tous les points considérés. L'enveloppe convexe revient donc à un polyèdre qui a pour sommet certains ou tous les points considérés.

Introduction de la problématique (2)

Il existe plusieurs méthodes de différentes complexités qui permettent de retrouver l'enveloppe convexe d'un ensemble de points. Nos objectifs principales sont:

- Présenter une solution naïve dans un premier temps, et ensuite présenter deux autres méthodes améliorées (de plus petites complexités)
- Comparer les temps de calcul de ces différents algorithmes sur R et sur C++.

Section 2

Solution naïve

Solution naïve

L'algorithme naïf cherche à construire l'enveloppe convexe de points placés dans R^2

On commence par fixer un point dans notre ensemble noté p_1

- Etape 1 : On fixe un deuxième point p_2 différent de p_1 et on trace le segment joignant ces deux points noté s_1
- Etape 2 : On prend un troisième point p_3 différent de p_1 et de p_2 et on trace le segment passant par p_1 et p_3 noté s_2.

Solution naïve (1)

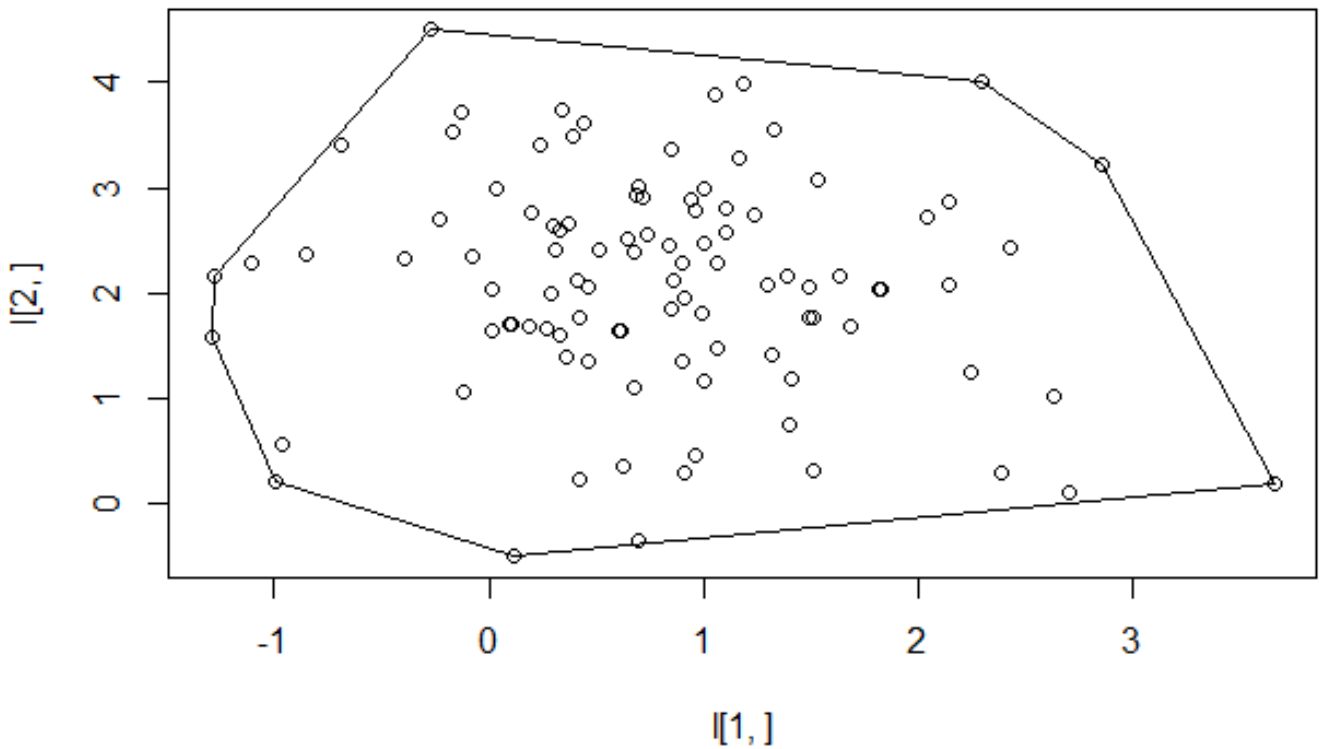
- Etape 3 : On stocke dans un vecteur noté v la position relative du segment s_2 par rapport au segment s_1. Cela est fait ici en calculant l'aire orientée du parallélogramme défini par s_1 et s_2.
- Etape 4 : On réitère les étapes 2 et 3 pour tous les points p_3 différent de p_1 et de p_2

Solution naïve (2)

- Etape 5 : On regarde si v ne contient que des valeurs positives ou que des valeurs négatives si c'est le cas cela veut dire que tous les points p_3 sont à gauche ou à droite du segment s_1 et donc que le point p_1 appartient à l'enveloppe convexe.
- Etape 6 : On reprend ensuite toutes les étapes en prenant un autre point p_2
- Etape 7 : On reprend enfin ces étapes pour tous les points p_1

Exemple de construction d'enveloppe convexe avec l'algorithme naif

```
env_convex_naif(X_transpose)
```



Etude de la complexité

Si on suit les étapes une à une on remarque que l'on a simplement 3 boucles qui s'imbriquent :

- La boucle concernant le point initial p_1 de longueur n
- A l'intérieur de cette première boucle une seconde boucle concernant le point p_2 de longueur n-1
- Enfin à l'intérieur de cette deuxième boucle une troisième boucle concernant le point p_3 qui permet de calculer les positions relatives et qui est de longueur n-2

On a donc à faire à une complexité cubique : $O(n^3)$

Section 3

Présentation des solutions nouvelles et exemples simples

La marche de Jarvis

La marche de Jarvis est un algorithme qui permets de déterminer l'enveloppe convexe d'un nombre de points donnés. La marche de Jarvis est un algorithme itératif et récursif. Il consiste en trois étapes:

- Etape 1: Recherche du premier point X_0 de l'enveloppe convexe

Ce point (X_0) est le plus à gauche et le plus en bas de l'ensemble des points. Autrement dit ce point est donc d'abcisse minimal et si plusieurs points sont d'abcisses minimal on prends le point d'ordonnée minimale.

La marche de Jarvis

- Etape 2: Recherche du point suivant X_i

Ce point est tel que l'angle polaire formé par $X_{i-1}X_i$ avec $X_{i-1}X_{i-2}$. Autrement dit on recherche le point suivant X_i parmi tous les points restants tel que tous les autres points de l'ensemble soient à droite de la droite $(X_{i-1}X_i)$.

Pour vérifier cette condition, nul besoin de calculer l'angle formé par ces segments. Pour ce faire, il suffit juste de calculer le produit vectoriel des vecteurs définis par les points X_i, X_{i-1} et X_{i-2}. Nous retiendrons donc le point X_i pour lequel le produit vectoriel des vecteurs définis par les trois points X_i, X_j et X_k est strictement supérieur à 0 (pour éviter les points qui sont alignés).

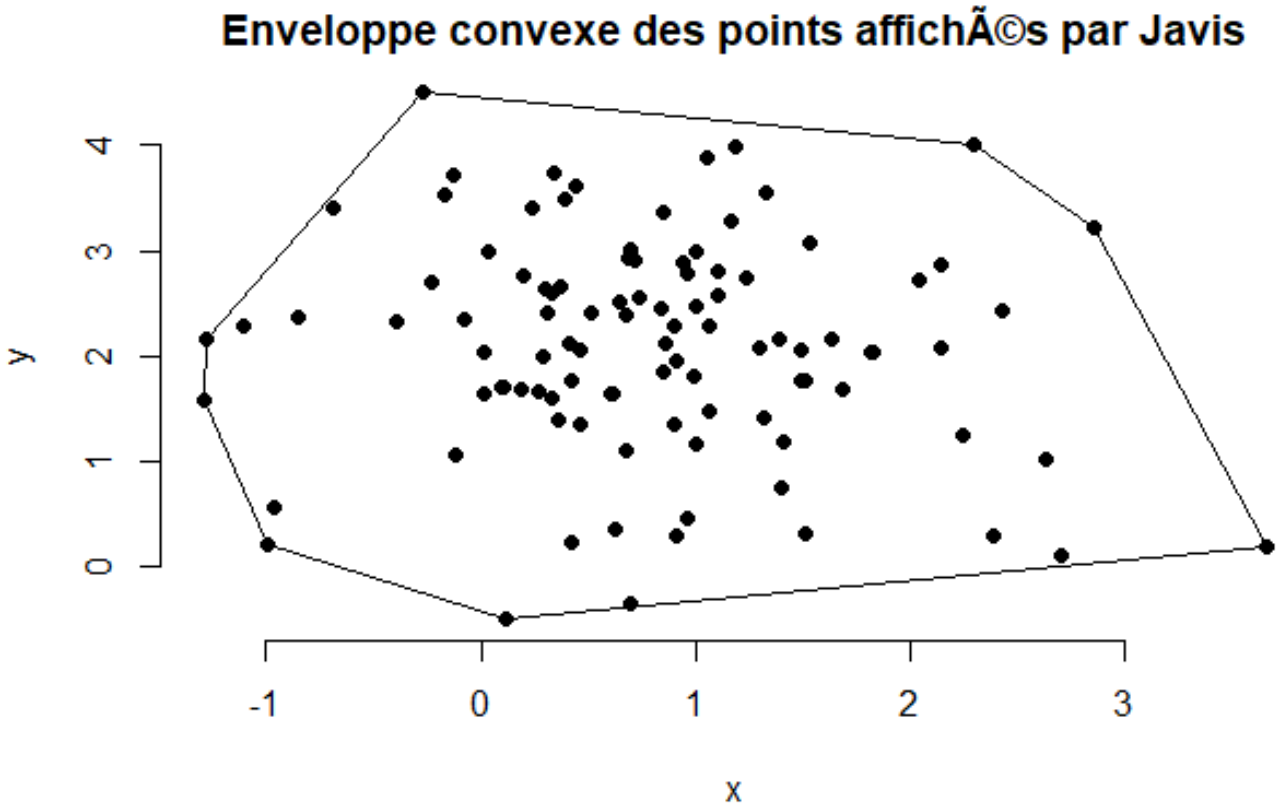
La marche de Jarvis

- Etape 3: Construction de l'enveloppe convexe

Enfin, il suffit de rechercher itérativement les points X_i qui forment l'enveloppe convexe en appelant récursivement la fonction **next_point** précédente de recherche du point suivant. On s'arrête lorsqu'on retombe sur le premier point X_0.

Exemple de construction d'enveloppe convexe avec la marche de Jarvis

```
dessin_env_jarvis(X_transpose)
```



Etude la complexité

L'algorithme de Jarvis a une complexité de l'ordre de $O(nh)$ avec n le nombre total de points considérés et h le nombre de points formant l'enveloppe convexe. En effet, la première étape de l'algorithme est de complexité $O(1)$. La seconde de recherche du point suivant connaissant le précédent est en $O(n)$. La fonction finale qui détermine l'enveloppe convexe est une fonction récursive de complexité $O(nh)$. Cet algorithme marchera très bien pour les ensembles dont l'enveloppe convexe contient très peu de point. Cependant, si tous les points de l'ensemble font partie de l'enveloppe convexe on aura une complexité théorique de l'ordre de n^2.

Le parcours de Graham

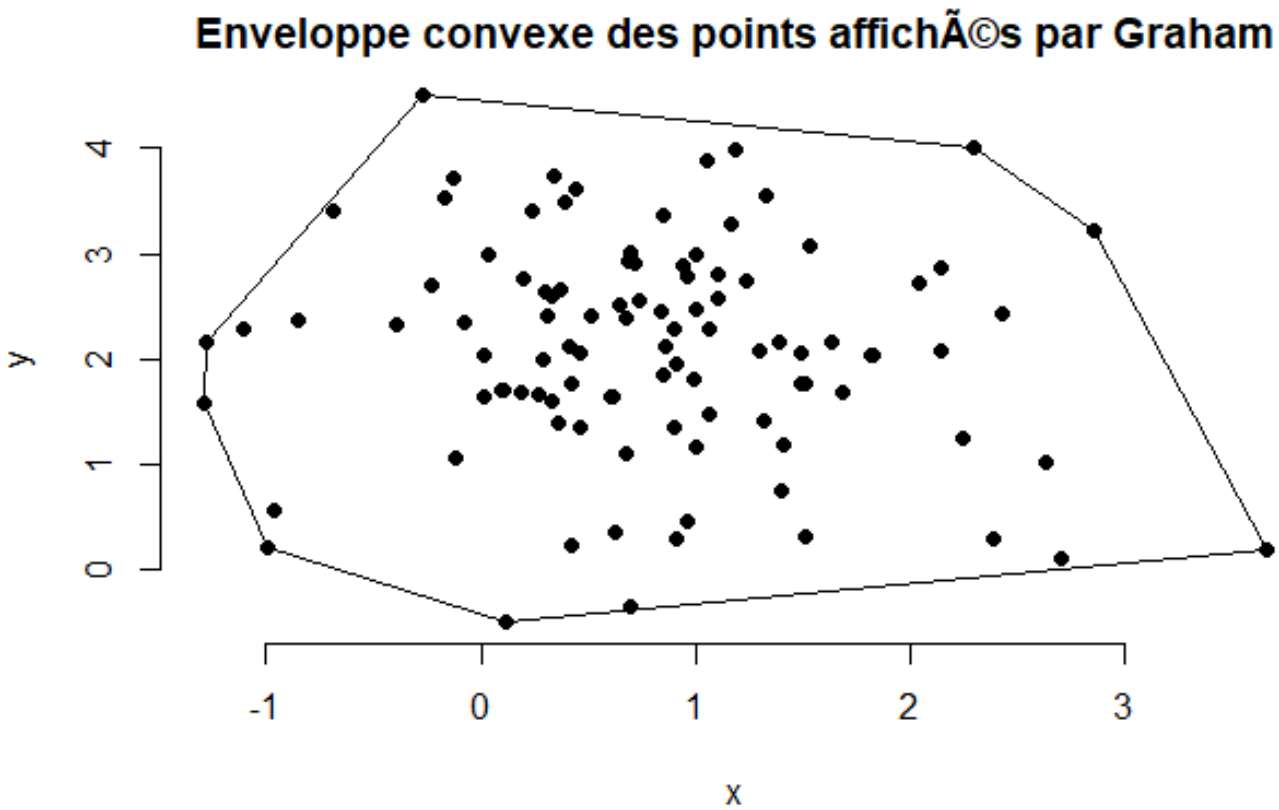
Le parcours de Graham commence par trouver un point pivot (point de plus petite ordonnée) et trie ensuite les autres points de manière croissante selon l'angle que fait l'axe des abscisses avec le segment joignant le pivot et le point courant. Il intègre alors dans une nouvelle liste A le pivot et le point suivant et va à chaque étape considérer un nouveau point courant qui sera le suivant dans la liste triée B.

Le parcours de Graham

Pour tous les points p dans la liste B tant que le segment formé par le dernier point dans la liste A et le point courant p est à droite du segment formé par les deux dernier points de la liste A on suprime de la liste A le dernier point. En sortant de la boucle on ajoute à la liste A le point courant p et on refait le même travail avec le point suivant. On obtient ainsi les points de l'enveloppe convexe.

Exemple de construction d'enveloppe convexe avec l'algorithme de Graham

```
dessin_env_graham(X_transpose)
```



Etude de la complexité

L'algorithme **angle_2dim** permet de trouver le pivot et de calculer l'angle entre ce dernier et tous les autres points. Il est de complexité $O(n)$. Les algorithmes **fusion** et **tri_fusion** sont des algorithmes de type diviser pour mieux régner et ont une complexité $O(nlog(n))$. L'algorithme **labeltri** produit à partir d'une matrice de points dans R^2 la matrice de points triée par le procédé expliqué plus haut. Il est de complexité $O(n)$. L'algorithme **dessin_ordre** permet d'afficher les points choisit labélisé selon l'ordre. Il est de complexité $O(n)$. L'algorithme principal **parcours_graham** est un algorithme dynamique qui permet de ne garder que les points appartenant au bord de l'enveloppe convexe. Il est de complexité $O(nlog(n))$.

Section 4

Simulations et comparaison des temps de calcul

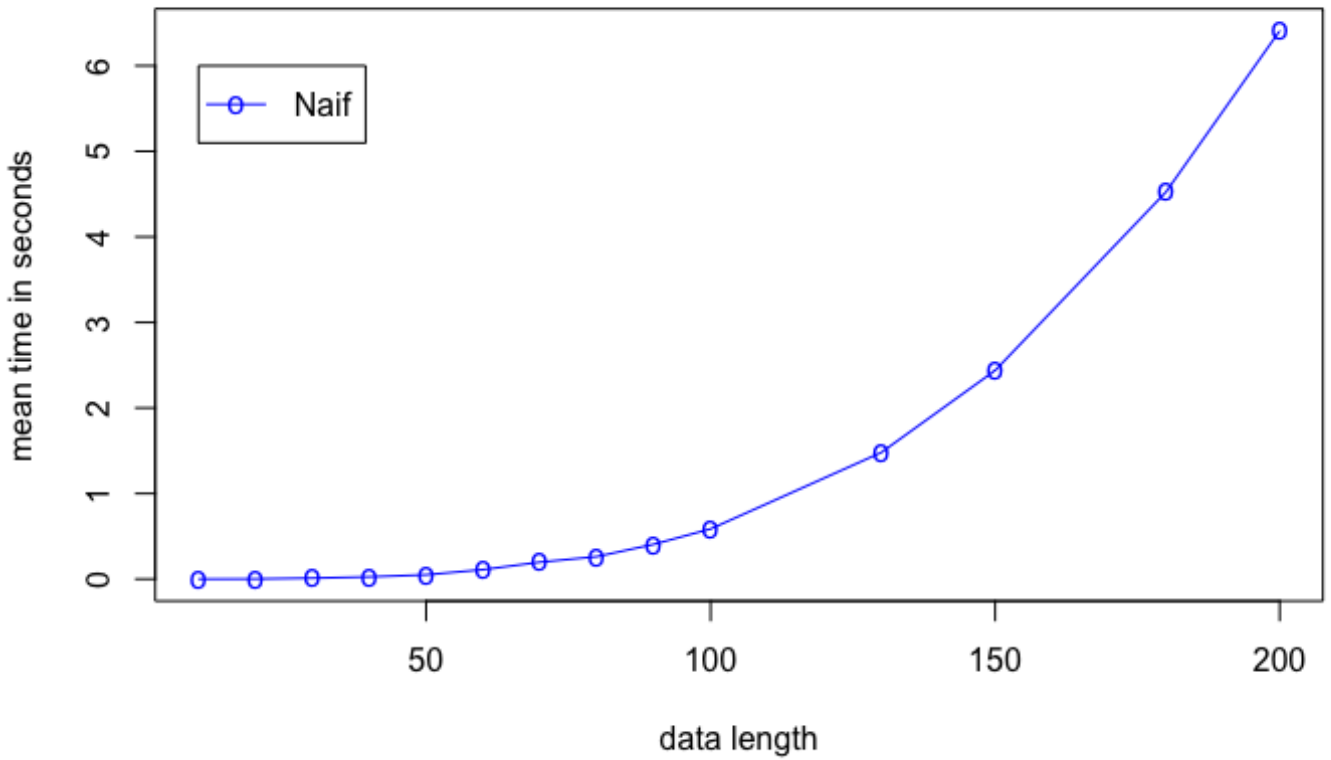
Algorithme naïf (1)

Pour évaluer le temps de calcul de l'algorithme naïf nous utiliserons les fonctions **temps_naive** pour R et **temps_naive_rcpp**. L'algorithme naïf sous R prends en moyenne environ 5 secondes contre 44 secondes sous C++ soit pratiquement 8 fois plus de temps que sous R.

```
t_naif_r=rep(0,100)
for (k in 1:100)
  t_naif_r[k]=temps_naive(X_transpose)
mean(t_naif_r)

t_naif_rcpp=rep(0,100)
for (k in 1:100)
  t_naif_rcpp[k]=temps_naive_rcpp(X_transpose)
mean(t_naif_rcpp)
```

Algorithme Naif (2)



Marche de Jarvis

Pour évaluer le temps de calcul de la marche de Jarvis nous utiliserons les fonctions **temps_javis_r** pour R et **temps_javis_rcpp**. L'algorithme de Jarvis sous R prends en moyenne environ 4.710913 millisecondes contre 0.5581596 millisecondes sous C++ soit pratiquement 8 fois moins de temps que sous R.

```
t_jarvis_r=rep(0,100)
for (k in 1:100)
  t_jarvis_r[k]=temps_javis_r(X_transpose)
mean(t_jarvis_r)

t_jarvis_rcpp=rep(0,100)
for (k in 1:100)
  t_jarvis_rcpp[k]=temps_javis_rcpp(X_transpose)
mean(t_jarvis_rcpp)
```

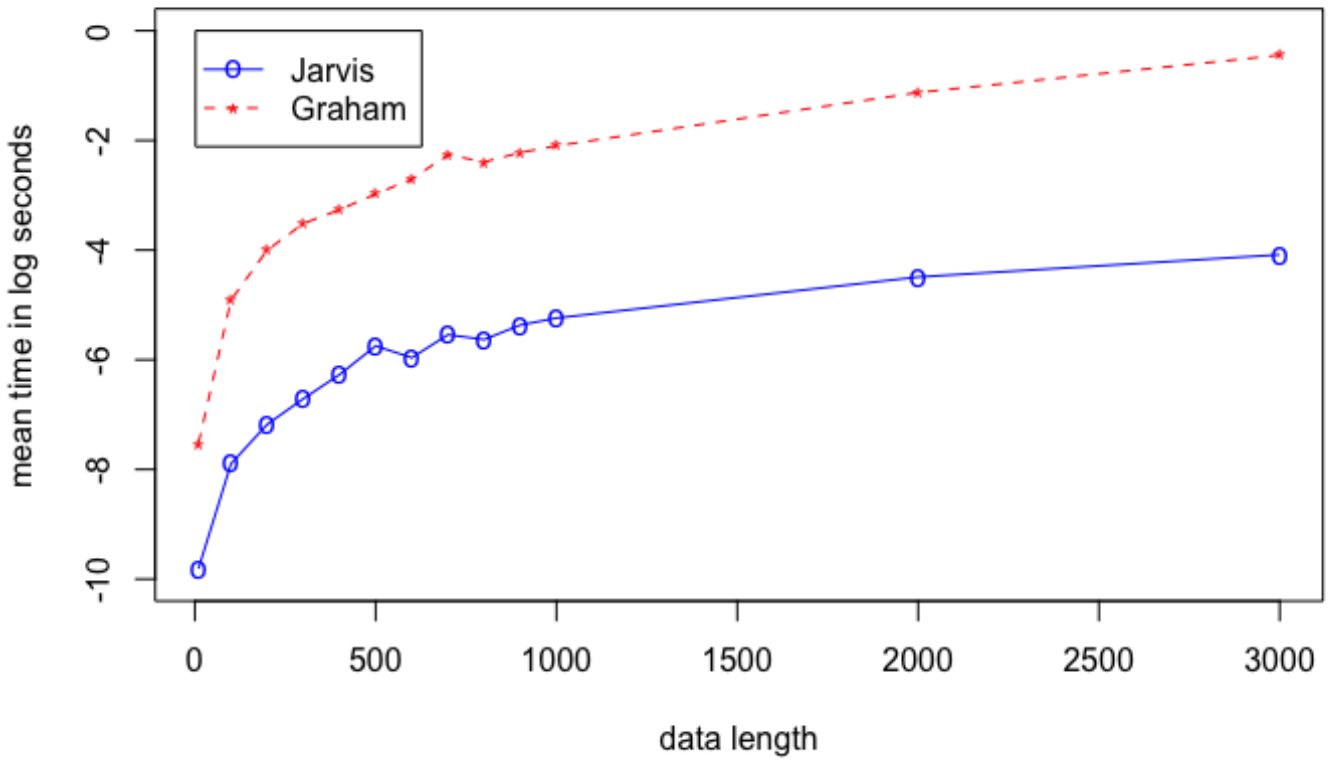
Algorithme de Graham (1)

Pour évaluer le temps de calcul de l'algorithme de Graham nous utiliserons les fonctions **temps_graham** pour R et **temps_graham_rcpp**.
L'algorithme de Graham prends en moyenne 19 millisecondes contre 8 millisecondes soit pratiquement 2 fois moins que sous R.

```
t_graham_r=rep(0,100)
for (k in 1:100)
  t_graham_r[k]=temps_graham(X_transpose)
mean(t_graham_r)

t_graham_rcpp=rep(0,100)
for (k in 1:100)
  t_graham_rcpp[k]=temps_graham_rcpp(X_transpose)
mean(t_graham_rcpp)
```

Algorithme de Graham (2)



Comparaison des temps de calcul

	Grham_r	Graham_c..	naive_r	naivs_c..	javis_r	javis_c..
1	0.0002589226	0.0001480579	0.02505684	3.600121e-05	0.0001280308	2.789497e-05
2	0.0004420280	0.0002121925	0.02152705	8.201599e-05	0.0001409054	3.099442e-05
3	0.0006649494	0.0002660751	0.02436399	5.102158e-05	0.0001571178	2.694130e-05
4	0.0005278587	0.0002920628	0.02114105	6.699562e-05	0.0002219677	3.194809e-05
5	0.0006330013	0.0003299713	0.03292894	9.417534e-05	0.0002508163	4.005432e-05
6	0.0010750294	0.0004389286	0.02466512	1.058578e-04	0.0002350807	4.196167e-05
7	0.0008900166	0.0004918575	0.02393603	1.380444e-04	0.0002760887	3.504753e-05
8	0.0010631084	0.0005791187	0.02609205	2.059937e-04	0.0003378391	5.006790e-05

Section 5

Conclusion

Conclusion

Les enveloppes convexes ont divers applications dans de nombreux domaines. Ils sont utiliser par exemple pour résourdre les problèmes d'optimisation et pour l'étude des polynômes.

Nous nous sommes limités au calcul de l'enveloppe convexe des points en dimension 2. Cependant, il existe des méthodes plus complexes de calcul de l'enveloppe convexe des points en dimension supérieur à 2.