# Algorithme de Weyl

L'objectif de ce projet est d'étudier un algorithme d'approximation des racines complexes d'un polynôme.

Nous considérerons donc un polynôme $P = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0$ à coefficients réels. Nous supposerons que $a_n \neq 0$ et $a_0 \neq 0$.

## 1 Résultats théoriques

1. Soit $z_0$ une racine de $P$. Montrer que $|z_0| \leqslant 1 + \dfrac{\max_{i<n} |a_i|}{|a_n|}$.
   Qu'en déduit-on sur l'emplacement des racines de $P$ dans le plan complexe ?
2. Soit $z_1$ dans $\mathbf{C}$. On peut développer l'expression de $P(z_1, Y)$ et regrouper les termes de même degré pour l'écrire sous la forme $P(z_1 + Y) = b_0 + b_1 Y + b_2 Y^2 + \cdots + b_n Y^n$. Soit $r > 0$. Montrer que si
   $$|b_0| = |P(z_1)| > |b_1| r + |b_2| r^2 + \cdots + |b_n| r^n, (1)$$
   alors $P$ n'a pas de racine dans le disque de centre $z_1$, de rayon $r$.
3. Un exemple : trouver une valeur de $r$ telle que $X^5 - X^4 + X^3 + X^2 + X - 1$ n'a pas de racinsudo apt installe dans le disque de centre $O$, de rayon $r$.
4. Rappeler la formule de Taylor qui permet d'obtenir facilement le développement de l'expression $P(z_1 + Y)$.

## 2 Complexes et polynômes en Python

Nous définirons en Python le nombre complexe $z = a + \mathrm{i}b$ par le couple de nombres réels *z=(a,b)*. (Pour rappel, les commandes *z[0]* et *z[1]* renvoient les valeurs de $a$ et $b$).

1. Écrire des programmes qui réalisent l'addition et la multiplication des nombres complexes écrits sous cette forme. Ces opérations prennent donc deux couples en entrée et renvoient un couple en sortie.
2. Écrire un programme qui pour un nombre complexe $z$ et un entier positif $n$ permet de calculer $z^n$.
3. Écrire un programme qui renvoie le module d'un nombre complexe.

   En Python, un polynôme $a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0$ est représenté par la liste des coefficients $[a_0, a_1, \ldots, a_n]$.
4. Écrire un programme *Eval(P,z)* qui calcule la valeur de $P(z)$.
5. Écrire un programme *Deriv(P,n)* qui renvoie le polynôme dérivé $n$ fois de $P$.

# 3 Algorithme

L'idée est la suivante : on sait d'après la question 1-1 que les racines de $P$ sont toutes situées dans une certaines région du plan. On considère ainsi pour débuter l'algorithme un carré de centre $O$ qui contient cette région.

On divise ensuite ce carré en quatre carrés. On va tester, à l'aide du résultat de la question 1-2, si ces carrés peuvent contenir ou non des racines de $P$. On considère pour cela le centre $z_1$ de chaque carré et on teste l'inégalité (1) avec $r$ le rayon du carré (c'est-à-dire la moitié d'une diagonale). Si l'inégalité est satisfaite, cela signifie que ce carré ne contient pas de racine et qu'on peut donc l'exclure de notre recherche. Sinon, ce carré contient éventuellement une racine et nous allons y faire des recherches plus poussées.

On considère ensuite les carrés qui n'ont pas été exclus. On divise chacun en quatre et on continue l'algorithme récursivement.

1. Écrire un programme *Test(P,z,r)* qui, pour un vecteur $P$, un nombre complexe $z$ et un réel positif $r$, teste l'inégalité (1).

   Dans la suite, un carré sera représenté par une paire $[c, a]$ où $c \in \mathbf{C}$ est le centre du carré (donc $c$ est u couple de nombre) et $a \in \mathbf{R}_+$ la longueur de ses côtés.
2. Écrire un programme *TestCarres(P, c, a)* qui découpe le carré $(c, a)$ en quatre sous-carrés, qui teste chacun d'entre eux, et qui renvoie la liste des carrés (donc une liste de listes) qui n'ont pas été exclus.
3. Écrire un programme *Weyl(P, n)* qui, partant du carré initial, effectue $n$ étapes de découpages et tests et qui renvoie la liste de tous les petits carrés restant à la fin.
4. Écrire un programme *Carre(c,a)* qui représente le carré de centre $c$ et de côté $a$.
5. Écrire un programme qui représente, dans le grand carré initial, les carrés restants à la fin de l'algorithme. *On pourra utiliser la commande* fill.
6. Appliquer vos programmes pour localiser, avec une précision d'au moins $10^{-2}$, les racines de nombreux polynômes de votre choix (de degrés variés).

# 4 Commentaires

1. Combien d'étapes sont nécessaires pour obtenir une précision $\varepsilon$ voulue ? Combien cela représente-t-il de tests ?
2. La méthode de Newton étudiée l'an dernier fonctionne encore avec des fonctions à valeurs complexes.
   Laquelle des deux méthodes est la plus rapide ? Quels sont les avantages et les inconvénients de la méthode Weyl ?