UFR Sciences
M2 Calcul Formel M. WEIMANN

Polynômes à une indéterminée

1 Evaluation

1.1 Schéma d'évaluation de Hörner

Lemme 1 Soit A un anneau commutatif. Soient $P \in A[X]$ de degré n et $a \in A$.

1. Le calcul naïf de $P(a)$ est de complexité $O(n^2)$.
2. Le calcul de $P(a)$ avec exponentiation rapide est de complexité $O(n\log(n))$.

On peut faire mieux que ces deux approches naïves simplement en remarquant que calculer la suite $1, a, a^2, \ldots, a^n$ coûte en fait $n-1$ multiplications, et donc calculer $P(a)$ coûte $2n-1$ multiplications et n additions.

Le schéma d'évaluation de Hörner permet de faire encore un peu mieux. Soit $P = c_0 + c_1X + \cdots + c_nX^n$. Le principe de Hörner est basé sur la décomposition

$$P = c_0 + X(c_1 + X(c_2 + X(\cdots + X(c_{n-1} + c_nX)\cdots))),$$

où l'on évalue les termes entre parenthèses. On obtient en passant les quantités

$$b_n = c_n,\ b_{n-1} = c_{n-1} + ab_n,\ \ldots,\ b_0 = c_0 + ab_1,$$

permettant d'obtenir en bonus le quotient de la division euclidienne de P par $X - a$:

$$P = (b_1 + b_2X + \cdots + b_nX^{n-1})(X - a) + b_0, \quad b_0 = P(a).$$

En itérant ce procédé pour le quotient, puis pour tous les quotients successifs, on obtient une procédure calculant le développement de Taylor de P au point a

$$P = P_0 + P_1(X - a) + \cdots + P_n(X - a)^n$$

sans avoir à calculer les dérivées successives de P au point a.

Exercice 1 Montrer que l'évaluation de Hörner coûte n additions et seulement n multiplications. Justifier les assertions qui ne le sont pas. Ecrire en pseudo-code l'algorithme du développement de Taylor correspondant.

1.2 Evaluation multipoints

Soit A un anneau commutatif. Soient $P \in A[X]$ de degré $< n$ et $a_1, \ldots, a_n \in A$ n éléments distincts de A. En utilisant le Schéma de Hörner, l'évaluation multipoint $P(a_1), \ldots, P(a_n)$ coûte donc $O(n^2)$ opérations dans A. Il existe un algorithme d'évaluation multipoint plus rapide de type « diviser pour régner », déjà esquissé dans le CM1.

Supposons que $n = 2^k$ pour simplifier. L'algorithme part du constat que $P(a_i)$ est le reste de la division euclidienne de P par $X - a_i$, et donc se réduit à un problème de modules multiples. Il procède en deux étapes :

• Etape 1. Calculer l'arbre $\mathcal{B}$ des "sous produits" :

$$(X - a_1), \ldots, (X - a_n)$$

$$(X - a_1)(X - a_2), \ldots, (X - a_{n-1})(X - a_n)$$

$$\ldots$$

$$(X - a_1)(X - a_2) \cdots (X - a_{n/2}) \qquad \text{et} \qquad (X - a_{n/2+1}) \cdots (X - a_{n-1})(X - a_n)$$

• Etape 2. Etant donné $\mathcal{B}$ comme ci-dessus, on applique la méthode diviser pour régner usuelle.

1. Si $n = 1$, retourner $P(a_1) = P \mod X - a_1$.
2. Si $n > 1$, calculer

$$P_1 = P \mod (X - a_1) \cdots (X - a_{n/2}) \qquad \text{et} \qquad P_2 = P \mod (X - a_{n/2+1}) \cdots (X - a_n)$$

3. Calculer récursivement

$$L_1 = [P_1(a_1), \ldots, P_1(a_{n/2})] \qquad \text{et} \qquad L_2 = [P_2(a_{n/2+1}), \ldots, P_2(a_n)]$$

4. Retourner $L_1 + L_2$.

Complexité. Notons $M(n)$ le coût de la multiplication de deux polynômes de degrés au plus n. On rappelle que la division euclidienne rapide de deux polynômes de degrés n coûte $O(M(n))$ et que la multiplication rapide (Transformée de Fourier discrète) coûte $M(n) = O(n \log(n))$, donc quasi-linéaire (cf Chapitre 1).

Lemme 2 L'algorithme d'évaluation multi-points rapide coûte $O(\log(n)M(n))$, soit une complexité quasi-linéaire $\tilde{O}(n)$ si l'on utilise la multiplication rapide des polynômes.

Exercice 2 Prouver ce lemme.

2 Interpolation

Soit K un corps commutatif. Rappelons le théorème bien connu d'interpolation de Lagrange :

Théorème 1 (Lagrange) Etant donnés des points $a_1, \ldots, a_n \in K$ distincts et des valeurs $b_1, \ldots, b_n \in K$, il existe un unique polynôme $P \in K[X]$ de degré $< n$ tel que $P(a_i) = b_i$ pour tout i. On appelle P *le polynôme d'interpolation* associés aux instances a_i et b_i.

Partant du constat que $P(a_i) = b_i$ équivaut à $P \equiv b_i \mod (X - a_i)$, une preuve rapide non constructive consiste à dire que P est uniquement déterminé comme étant l'image de $(b_1, \ldots, b_n)$ par l'isomorphisme du théorème des restes chinois

$$\frac{K[X]}{(X - a_1)} \times \cdots \times \frac{K[X]}{(X - a_n)} \simeq \frac{K[X]}{(X - a_1) \cdots (X - a_n)}.$$

2.1 Formule d'interpolation de Lagrange

L'existence du polynôme d'interpolation P se déduit aussi de la formule explicite d'interpolation de Lagrange :

$$P = \sum_{i=1}^{n} b_i \prod_{j \neq i} \frac{X - a_j}{a_i - a_j}$$

L'unicité du polynôme d'interpolation découle alors du fait qu'un polynôme de degré $< n$ ayant plus de n racines est nul (K est un corps). On note que les polynômes de Lagrange

$$L_i := \prod_{j \neq i} \frac{X - a_j}{a_i - a_j}, \quad i = 1, \ldots, n$$

sont unitaires de degrés $n-1$ et vérifient $L_i(a_j) = \delta_{ij}$ (symbole de Kronecker).

Exercice 3 Montrer que $L_1, \ldots, L_n$ forme une base de $K[X]_{<n}$. Montrer que l'on peut calculer les L_i, et donc P, avec $O(n^2)$ opérations dans K.

2.2 Formule de Newton

Il existe d'autres formules d'interpolation pour le polynôme interpolateur P. L'une d'entre elle est la formule de Newton qui consiste à écrire P dans la base

$$N_0 = 1, \quad N_1 = (X - a_1), \quad N_2 = (X - a_1)(X - a_2), \quad \ldots \quad , N_{n-1} = (X - a_1) \cdots (X - a_{n-1})$$

de $K[X]_{<n}$. L'avantage est que si l'on rajoute un noeud a_{n+1}, il suffit de calculer N_n sans avoir à recalculer tous les éléments de la base, contrairement à l'interpolation de Lagrange. La difficulté réside cette fois dans le calcul des coefficients. On cherche $u_0, \ldots, u_{n-1} \in K$ tels que

$$\begin{aligned} P &= u_0 N_0 + u_1 N_1 + \cdots + u_{n-1} N_{n-1} \\ &= u_0 + u_1 (X - a_1) + \cdots + u_{n-1} (X - a_1) \cdots (X - a_{n-1}). \end{aligned}$$

On peut montrer par récurrence que les u_i se calculent à l'aide des *puissances divisées* que l'on définit ainsi. On pose $d[a_i] := b_i$ pour $i = 1, \ldots, n$ puis on définit récursivement

$$d[a_1, \ldots, a_n] = \frac{d[a_2, \ldots, a_n] - d[a_1, \ldots, a_{n-1}]}{a_n - a_1}$$

Lemme 3 Avec les notations ci-dessus, on a l'égalité $u_i = d[a_1, \ldots, a_{i+1}]$ pour tout $i = 0, \ldots, n-1$.

Exercice 4 Prouver ce Lemme et étudier la complexité de l'algorithme d'interpolation sous-jacent.

2.3 Diviser pour régner

Là encore, il existe un algorithme d'interpolation plus rapide, de type diviser pour régner. Puisque la condition $P(a_i) = b_i$ est équivalente à $P \mod (X - a_i) = b_i$, il suffit de résoudre le système de congruences

$$\begin{cases} P(X) & = b_1 \mod X - a_1 \\ P(X) & = b_2 \mod X - a_2 \\ & \vdots \\ P(X) & = b_n \mod X - a_n \end{cases}$$

dans l'anneau euclidien $K[X]$. Ce système de congruence peut se résoudre avec une méthode diviser pour régner, en pré-calculant l'arbre des sous-produits $\mathcal{B}$ comme pour l'évaluation multi-points :

1. Si $n = 1$, retourner $P = b_1$
2. Si $n > 1$, calculer récursivement P_1 et P_2 de degrés $< n/2$ tels que $P_1(a_i) = b_i$ pour $i \leq n/2$ et $P_2(a_i) = b_i$ pour $i > n/2$.
3. Retourner la solution P de degré $< n$ du système de deux congruence

$$P \equiv P_1 \mod (X - a_1) \cdots (X - a_{n/2}) \qquad \text{et} \qquad P \equiv P_2 \mod (X - a_{n/2+1}) \cdots (X - a_n)$$

que l'on résoud à l'aide de l'algorithme d'Euclide étendu (relations de Bezout).

Exercice 5 Montrer que cet algorithme d'interpolation est correct et écrire proprement les détails de l'étape 3. Montrer que la complexité est $O(M(n)\log(n))$. Cf aussi TP2.

3 Multiplication rapide dans $K[X]$ par FFT

3.1 La transformée de Fourier discrète

L'algorithme de FFT (Fast Fourier Transform) pour la multiplication de deux polynômes $G, H \in K[X]$ repose sur le principe d'évaluation multipoints - interpolation. On s'inspire ici de la Section 2.4 de [BCGLLSS]. On considère n une puissance de 2 telle que $n > \deg(GH)$ et on calcule $F = GH$ selon le schéma suivant :

1. Evaluation multi-points : Calculer $G(a_i)$ et $H(a_i)$ en des points $a_1, \ldots, a_n \in K$ distincts.
2. Produits dans K : Calculer $b_i = G(a_i)H(a_i)$ pour $i = 1, \ldots, n$.
3. Interpolation : Calculer F tel que $F(a_i) = b_i$ pour $i = 1, \ldots, n$.

On a nécessairement $F = GH$ par unicité du polynôme d'interpolation. Toute l'idée est d'utiliser pour les a_i des racines n-ème de l'unité pour avoir un schéma d'évaluation-interpolation qui n'utilise pas la multiplication dans $K[X]$.

Proposition 1 Soit $\alpha \in K$ une racine primitive n-ème de l'unité dans K (en supposant qu'une telle racine existe).

1. L'évaluation multi-points en $1, \alpha, \ldots, \alpha^{n-1}$ coûte $O(n\log(n))$ opérations dans K. On appelle cette opération la transformée de Fourier discrète (DFT).
2. L'interpolation rapide aux points $1, \alpha, \ldots, \alpha^{n-1}$ coûte $O(n\log(n))$ opérations dans K.

Preuve. On suppose que n est une puissance de 2 pour simplifier, et on pose $k = n/2$. On remarque que $X^n - 1 = (X^k - 1)(X^k + 1)$ et de manière similaire à l'évaluation multi-point usuelle, on calcule les deux restes

$$R_0 = G \mod (X^k - 1) \qquad \text{et} \qquad R_1 = G \mod (X^k + 1),$$

avec $\deg R_i < k$. Le point clé est qu'aucune division euclidienne n'est ici nécessaire au calcul des R_i. En effet, si $G = \sum_{i=0}^{n-1} c_i X^i$, on a simplement

$$R_0 = \sum_{i=0}^{k-1} (c_i + c_{i+k}) X^i \qquad \text{et} \qquad R_1 = \sum_{i=0}^{k-1} (c_i - c_{i+k}) X^i.$$

Les racines de $X^k - 1$ sont les puissances paires de α et les racines de $X^k + 1$ sont les puissances impaires de α. On a donc $G(\alpha^{2i}) = R_0(\alpha^{2i})$ et $G(\alpha^{2i+1}) = \tilde{R}_1(\alpha^{2i})$ où $\tilde{R}_1(X) = R_1(\alpha X)$. Il suffit donc d'évaluer R_0 et $\tilde{R}_1$ en $1, \alpha^2, \ldots, \alpha^{2(k-1)}$, ce que l'on fait récursivement en remarquant que α^2 est une racine primitive k-ème de l'unité. On montre par récurrence que le coût total est de $O(n\log(n))$.

Pour l'interpolation, on remarque que l'évaluation

$$Ev_\alpha : F \mapsto (F(1), \ldots, F(\alpha^{n-1}))$$

correspond à l'application linéaire de $K[X]_{<n} \simeq K^n \to K^n$ dont la matrice dans la base canonique $(1, X, \ldots, X^{n-1})$ est la matrice de Vandermonde $V_\alpha = (\alpha^{ij})_{0 \leq i,j \leq n-1}$. L'interpolation revient à calculer F connaissant $P = (F(1), \ldots, F(\alpha^{n-1}))$, c'est à dire à calculer $F = Ev_\alpha^{-1}(P)$. On utilise le lemme clé suivant :

Lemme 4 L'inverse de V_α est $\frac{1}{n} V_{\alpha^{-1}}$.

Il s'ensuit que l'on a l'égalité $F = \frac{1}{n} Ev_{\alpha^{-1}}(P)$. L'interpolation revient donc ici à calculer l'évaluation multi-point en $1, \alpha^{-1}, \ldots, \alpha^{-(n-1)}$, ce que l'on sait faire en $O(n \log(n))$ d'après le point 1 de la proposition, puisque α^{-1} est aussi une racine primitive n-ème de l'unité (exo).

Exercice 6 Prouver le lemme clé.

On obtient ainsi un résultat fondamental du calcul formel :

Corollaire 1 Si K contient une racine primitive n-ème de l'unité, on peut multiplier deux polynômes de $K[x]$ dont la somme des degrés est $< n$ avec $O(n \log(n))$ opérations dans K.

Exercice 7 Justifier proprement cette assertion en étudiant de manière précise la complexité de l'algorithme sous-jacent.

3.2 Racines de l'unité

Un dernier souci est que tout corps n'admet pas nécessairement de racine primitive n-ème de l'unité. Dans le cas des corps finis, on a une caractérisation facile :

Exercice 8 Montrer que le corps fini $\mathbb{F}_q$ contient une racine primitive n-ième de l'unité si et seulement si n divise $q - 1$.

Cet exercice suggère qu'un choix judicieux est de considérer $\mathbb{F}_p$ avec p premier de la forme $p = 2^e + 1$ pour avoir beaucoup d'entiers n une puissance de 2 divisant $p - 1$. Sauf qu'il n'existe que très peu de premiers p de cette forme connus. En effet, on sait que $2^e + 1$ premier implique $e = 2^k$. Donc on cherche k tel que $2^{2^k} + 1$ soit premier (nombres premiers de Fermat). On en connaît seulement 5 (correspondant à $k = 0, 1, 2, 3, 4$), et on ne sait pas à ce jour s'il en existe d'autres (le premier cas ouvert est $k = 33$). On privilégie donc plutôt p de la forme $p = r2^e + 1$ (nombres premiers de Fourier). Par exemple,

$$p = 29 \times 2^{57} + 1 = 4179340454199820289$$

est un tel nombre premier. Dans ce cas, il existe dans $\mathbb{F}_p$ des racines primitives 2^k-ème de l'unité pour tout $k \leq 57$ et on peut multiplier deux polynômes de degrés n en $O(n \log n)$ opérations dans $\mathbb{F}_p$ pour n très grand (jusqu'à un milliard de milliards dans cet exemple). A noter que chaque opération dans $\mathbb{F}_p$ ne coûte "que" $\tilde{O}(\log(p))$ opérations binaires, soit ici une soixantaine d'opérations binaires.

Les racines de l'unité peuvent se calculer à partir des éléments primitifs de $\mathbb{F}_p$ (générateurs de $\mathbb{F}_p^\times$). Quelques tirages aléatoires suffisent pour p grand. En effet, il est établi que la probabilité qu'un élément α de $\mathbb{F}_p$ soit primitif tend vers $6/\pi^2 \approx 0.6$ quand p tend vers $+\infty$.

Cette approche modulo un grand p permet aussi de multiplier des polynômes dans $\mathbb{Z}[X]$: connaissant une majoration de la taille des coefficients du produit en fonction de celle des facteurs, on peut se réduire à faire des calculs modulo p pour p suffisamment grand.

Si K ne contient pas de racines primitives n-èmes de l'unité (n une puissance de 2), on travaille dans une extension de K qui en contient, ce qui est possible dès lors que K n'est pas de caractéristique 2. C'est le principe de l'algorithme de Schönhage-Strassen (voir par exemple Section 2.5 dans [BC-GLLSS]). Ces algorithmes se généralisent au cas où K est un anneau dès lors que 2 est inversible. Si K est de caractéristique 2, il faut développer d'autres techniques.

3.3 Le cas des entiers

La multiplication rapide par FFT s'étend aux entiers de la façon suivante. On veut multiplier deux entiers $a, b \in \mathbb{Z}$. On associe aux listes des représentations binaires de a et b les polynômes correspondants $G, H \in \mathbb{Z}[X]$ à coefficients 0 ou 1. On a alors $ab = G(2)H(2)$. Il suffit donc de calculer $F = GH \mod p$ avec $p > ab$. Il existe un tel premier p tel que $p < 2ab$. On en déduit que le coût total en la taille $n = log(ab)$ de la sortie est $O(n \log n \log\log n)$.

La multiplication rapide par FFT est aujourd'hui implémentée (pour des grands degrés ou des grands entiers) dans la plupart des logiciels de calcul formel, en particulier `SAGE`.

4 Localisation des racines

On cherche à estimer le nombre (en particulier la présence) de racines d'un polynôme $P \in \mathbb{R}[X]$ dans un intervalle donné. Paragraphe inspiré de notes de cours de Denis Simon.

4.1 Théorème de Sturm

Soit $P \in \mathbb{R}[X]$ un polynôme non nul. On définit une suite de polynômes (P_k) en modifiant légèrement l'algorithme d'Euclide du calcul du PGCD de P et de sa dérivée P'.

1. On pose $P_0 = P$ et $P_1 = P'$.
2. Tant que P_k est non nul, on définit $P_{k+1} = -P_{k-1} \mod P_k$ (attention au signe).

Autrement dit le polynôme P_{k+1} est l'opposé du reste de la division euclidienne de P_{k-1} par P_k. La suite (P_k) est finie. On l'appelle *la suite de Sturm* de P.

Si on note P_m le dernier reste non nul, alors on remarque que P_m est le PGCD des polynômes P et P'. En particulier, P_m est constant si et seulement si P n'a pas de racines multiples.

Etant donné un réel $x \in \mathbb{R}$, on note $V(x)$ le nombre de changements de signes dans la suite $P_0(x), P_1(x), \ldots, P_m(x)$ (à laquelle on a retiré les valeurs nulles).

Théorème 2 (Sturm) Soient $a, b \in \mathbb{R}$ avec $a < b$. Si P est sans racines multiples, alors le nombre de racines réelles de P dans l'intervalle $]a, b]$ est égal à $V(a) - V(b)$.

Preuve (esquisse). Par le théorème des valeurs intermédiaires, $V(x)$ est constant tant que x ne traverse aucune racine des P_k. Il s'agit de montrer que V diminue de 1 quand x traverse une racine de P et que V ne change pas quand x traverse une racine de P_k qui n'est pas une racine de P. On remarque que $P_m = pgcd(P_k, P_{k+1})$ pour tout $k < m$. Comme P n'a pas de racine multiple, P_m est un polynôme constant (non nul) et deux polynômes consécutifs P_k et P_{k+1} n'ont pas de racine commune. Si α est une racine de P_k pour un certain $1 \leq k \leq m-1$, on a donc $P_{k-1}(\alpha)P_{k+1}(\alpha) < 0$ par définition de la suite de Sturm. Par continuité $P_{k-1}(x)P_{k+1}(x) < 0$ pour tout x voisin de α. Ainsi, le nombre de changements de signe dans la suite $P_1(x), P_2(x), \ldots, P_m(x)$ est constant au voisinage de α. Si α n'est pas racine de P, on en déduit que V est constant au voisinage de α. Si α est une racine de P, alors P^2 a un minimum en α, donc $2P'(\alpha + h)P(\alpha + h)$ est du signe de h lorsque h est assez petit. Donc il y a un changement de signe de plus pour x à gauche de α que pour x à droite de α dans la suite $P_0(x), P_1(x), \ldots, P_m(x)$. En faisant la somme, on en déduit que $V(a) - V(b)$ est exactement égal au nombre de racines de P. □

Exercice 9 Comment traiter le cas où P a des racines multiples ?

4.2 Nombres de racines réelles

Pour calculer le nombre total de racines, il faut considérer $a = -\infty$ et $b = +\infty$. Pour appliquer l'algorithme de Sturm avec $b = +\infty$, il suffit, lors du calcul de $V(b)$, de remplacer $P_k(b)$ par le coefficient dominant de P_k (qui donnera bien le signe de $P_k(b)$ pour $b \to +\infty$). Si on veut l'appliquer avec $a = -\infty$, on remplace cette fois $P_k(a)$ par $(-1)^{\deg(P_k)}$ fois son coefficient dominant.

Une autre possibilité est de trouver une borne $R > 0$, simple à calculer, et telle que toutes les racines réelles de P soient dans l'intervalle $[-R, R]$. Plusieurs bornes ont cette propriété. Donnons l'une d'entre elles.

Lemme 5 Soit $P \in \mathbb{C}[X]$ un polynôme unitaire $P = a_0 + a_1 X + \cdots + a_{n-1}X^{n-1} + X^n$. Alors toute racine α de P vérifie

$$|\alpha| < 1 + \max_{i=0,\ldots,n-1} |a_i|$$

Preuve. On pose

$$f(X) = \left(1 - \frac{|a_{n-1}|}{X} - \cdots - \frac{|a_0|}{X^n}\right).$$

Cette fonction est une bijection continue strictement décroissante de $]0, +\infty[$ sur $]-\infty, 1[$. Elle admet donc une unique racine réelle $r > 0$. Puisque $P(\alpha) = 0$, on a par l'inégalité triangulaire

$$|\alpha|^n \leq |a_{n-1}||\alpha|^{n-1} + \cdots + |a_0|.$$

Il s'ensuit que $|\alpha^n| f(|\alpha|) \leq 0$ et donc $|\alpha| \leq r$. De la même manière, si R est tel que

$$R^n > |a_{n-1}|R^{n-1} + \cdots + |a_0|$$

alors on déduit que $R > r \geq |\alpha|$ est une majoration de $|\alpha|$ pour toute racine α de P. Soit $R = 1 + \max |a_i|$. On a alors

$$R^n = (R-1)(R^{n-1} + \cdots + 1) + 1 > (\max |a_i|)\,(R^{n-1} + \cdots + 1) \geq |a_{n-1}|R^{n-1} + \cdots + |a_0|$$

d'où le résultat. □

Exercice 10 Montrer de la même manière que $R = \max\{1, \sum_{i=0}^{n-1} |a_i|\}$ est aussi une borne convenable pour la taille des racines de P.

4.3 Calcul approché des racines

Plutôt que de chercher des valeurs décimales ou rationnelles proches des racines d'un polynôme, on peut chercher des intervalles de plus en plus petits contenant ces racines. Cela revient à faire un algorithme de type dichotomie. Dans un premier temps, si on sait que l'intervalle $]a, b]$ contient $N \geq 2$ racines réelles distinctes, on peut chercher à isoler ces racines, c'est-à-dire à trouver exactement N intervalles contenant chacun une unique racine. Pour cela, on coupe cet intervalle en deux en posant $m = (a+b)/2$ et on cherche le nombre N_1 de racines dans $]a, m]$ et N_2 dans $]m, b]$. Si N_1 ou N_2 est nul, on remplace l'intervalle $]a, b]$ par celui parmi $]a, m]$ et $]m, b]$ qui contient toutes les racines. Si N_1 et N_2 sont non nuls, alors on poursuit l'algorithme séparément avec chacun des deux intervalles. On recommence cette procédure jusqu'à ce que chaque intervalle contienne une unique racine. Dans un deuxième temps (ou en parallèle), pour chaque intervalle $]a, b]$ contenant une unique racine réelle, on peut appliquer la dichotomie comme précédemment jusqu'à obtenir un intervalle d'amplitude aussi petite que voulue.

Itération de Newton. Une fois que l'on a approché suffisamment bien une racine (simple) α de P par un réel x_0, l'algorithme basé sur l'itération de Newton

$$x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}$$

peut prendre le relais. En supposant x_0 suffisamment proche de α, cet algorithme a une convergence quadratique, i.e. le nombre de chiffres significatifs est doublé à chaque étape. C'est extrêmement performant, à la base de nombreux calculs numériques. Quelques 6 ou 7 itérations suffisent en général pour atteindre la précision machine.

Il pourrait sembler ici que l'on éloigne du calcul formel, pour rentrer dans le monde du calcul numérique. Mais l'itération de Newton permet d'approcher des racines de fonction de classe $\mathcal{C}^2$ beaucoup plus générales que les polynômes, avec des coefficients (donc des racines) dans des anneaux beaucoup plus généraux que $\mathbb{R}$. Ainsi, l'itération de Newton est utilisée dans de nombreux autres algorithmes de Calcul Formel. Citons : calcul des sommes de Newton, inversions des séries formelles cf [Section 3.3, BCGLLSS], division euclidienne rapide [Section 4.2, BCGLLSS], factorisation dans $\mathbb{Z}[X]$, etc.). Tout ceci mériterait quelques chapitres supplémentaires.

Exercice 11 Montrer à l'aide de la formule de Taylor à l'ordre 2 avec reste que l'itération de Newton, si elle converge, a une convergence quadratique. Donner une condition sur P' et P'' assurant que l'itération de Newton avec condition initiale x_0 converge.