PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

FACTORIZACIÓN DE POLINOMIOS MEDIANTE UN MÉTODO ITERATIVO

Tesis para obtener el título profesional de Licenciado en Matemática

AUTOR:

Eduardo Alexis Llamoca Palomino

ASESOR:

Alfredo Bernardo Poirier Schmitz

Lima, Junio, 2025

Informe de Similitud

Yo, Alfredo POIRIER Schmitz, docente de la Facultad de Ciencias e Ingenieria de la Pontificia Universidad Católica del Perú, asesor del trabajo de tesis titulado

FACTORIZACION DE POLINOMIOS MEDIANTE UN METODO ITERATIVO

del autor Eduardo Alexis LLAMOCA Palomino

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 12%, con ninguna referencia individual que supere el 1%, excepto trabajo preliminar del mismo autor. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 05/05/2025.
- He revisado con detalle dicho reporte y la Tesis y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: Lima, 5 de mayo del 2025

Apellidos y nombres del asesor: POIRIER SCHMITZ, Alfredo Bernardo	
DNI: 10803756	Firma
ORCID: 0000-0003-2789-3630	

Agradecimientos

Algo fundamental que aprendí en mis cinco años de estudios es que si se busca entender un objeto no se le puede separar de la manera cómo interactúa con su entorno. En este sentido, su entorno es parte de su esencia. Creo que este concepto no se limita a objetos matemáticos; por lo cual, me gustaría agradecer a todas las personas que han moldeado mi forma de entender el mundo. Entre ellas quiero resaltar a mi familia, profesores y amigos. A mi familia, por haberme apoyado incondicionalmente y sobre todo a mi mamá por haber estado desde el comienzo. A todos mis profesores, por haber compartido su forma de entender las matemáticas conmigo; muchas veces a través de simples palabras o frases que desvelaron el corazón del objeto estudiado. A mi profesor y asesor de tesis Alfredo Poirier, también por su paciencia en la elaboración de este trabajo. Finalmente, a mis amigos Aaron García, Marcelo Gallardo, Carlos Cosentino, Blas Molero y Luis Zegarra por haber hecho estos cinco años muy agradables.

Resumen

Un tema recurrente a lo largo de la historia de las matemáticas ha sido el obtener raíces de un polinomio dado. En un principio este problema fue atacado a través de fórmulas algebraicas; es decir, se presentaba una expresión cerrada sobre los coeficientes del polinomio. Sin embargo, desarrollos posteriores descartan el éxito de este método para polinomios de grado mayor a cuatro. Como consecuencia las técnicas actuales resultan ser de corte numérico. Entre ellas podemos encontrar el popular método de Newton.

El objetivo de esta tesis es aportar con otro método para factorizar polinomios que al igual que los anteriores tiene sus ventajas y desventajas. Entre las ventajas encontramos su aplicabilidad a una amplia familia de polinomios con coeficientes en los complejos. Asimismo, presentamos la implementación, en el lenguaje C++, para el método propuesto.

Nuestro método es de carácter iterativo y en cada una de estas iteraciones obtenemos un polinomio. Esta sucesión de polinomios converge y utilizaremos el polinomio límite para aislar la raíz de mayor norma del polinomio que deseamos factorizar. Posteriormente, reutilizamos la información de cada iteración para poder dar con las demás raíces del polinomio.

Contenido

Capítulo 1

Introducción

El estudio de las raíces de los polinomios ha sido una cuestión central en las matemáticas desde tiempos antiguos, con aplicaciones que se extienden desde la solución de ecuaciones algebraicas básicas hasta problemas avanzados en física, ingeniería y ciencias computacionales.

No obstante, calcular las raíces de un polinomio no es un problema trivial, especialmente cuando se trata de polinomios de grado mayor a 4, pues para estos es imposible formular una solución general en términos de radicales. Este hecho ha motivado el desarrollo de vastos métodos numéricos para la aproximación de raíces, cada cual con sus ventajas y limitaciones, en función del tipo de polinomio y las características de las raíces que se buscan.

En esta tesis, buscamos contribuir presentando un método iterativo que elude algunas de las limitaciones de enfoques existentes. Nuestro método se distingue por abordar polinomios con coeficientes en $\mathbb{Q}[i]$ y por su efectividad computacional en una amplia gama de escenarios.

1.1 Nuestros resultados

A lo largo de este trabajo se demostrará que el algoritmo presentado es especialmente eficaz cuando el polinomio a factorizar no presenta raíces distintas de un mismo módulo. En efecto, en tal circunstancia, la complejidad del método es de $O(Ln^2)$, donde n denota el grado del polinomio y L la cantidad de iteraciones que utilizaremos para aproximar las raíces del polinomio.

Asimismo, abordaremos el caso no favorable. Para ello proponemos un enfoque probabilístico, en el cual trasladamos repetidamente el polinomio dado hasta que eventualmente el polinomio obtenido caiga dentro del primer patrón.

1.2 Nuestras técnicas

En la primera parte de este trabajo lidiaremos con polinomios cuyas raíces son simples y de norma distinta. Este tipo de polinomios recibirá el nombre de **polinomio solar** debido a que la distribución de sus raíces en el plano complejo se asemeja a un sistema planetario. Posteriormente extenderemos nuestros resultados a polinomios con todas sus raíces simples. Por último, atacaremos polinomios arbitrarios con coeficientes en $\mathbb{Q}[i]$, nuestro objetivo.

Comencemos considerando un polinomio solar de grado n y coeficientes complejos $P \in \mathbb{C}[z]$ (no necesariamente solo en $\mathbb{Q}[i]$) con raíces $\alpha_1, \ldots, \alpha_n$. Asumiremos que n es mayor que 1, pues de no serlo la factorización es trivial. Si dividimos z^k entre P, en el esquema de división de Euclides, tendremos un cociente Q_k y un residuo R_k relacionados vía

$$z^k = Q_k(z)P(z) + R_k(z). \tag{1.1}$$

En particular, si reemplazamos los distintos α_i, obtendremos explícitamente

$$\alpha_i^k = R_k(\alpha_i). \tag{1.2}$$

Si los α_i son distintos, como los R_k son de grado a lo sumo $n-1$, obtendremos un sistema lineal soluble sobre los coeficientes de R_k. Este trabajo estudia tal sistema y lo explota; de este modo, se logra aislar las raíces de P siguiendo un orden decreciente en norma. A continuación las raíces aisladas podrán ser recuperadas mediante las relaciones de Peano.

Capítulo 2

Desgranando polinomios solares

Sea P nuestro polinomio solar de grado mayor a 1 con raíces $\alpha_1, \alpha_2, \ldots, \alpha_n$ ordenadas de mayor a menor norma. De las relaciones en (1.2) tendremos una presentación matricial

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \ldots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \ldots & \alpha_n^{n-1} \end{pmatrix} \begin{pmatrix} \mathrm{coef}_0(R_k) \\ \mathrm{coef}_1(R_k) \\ \mathrm{coef}_2(R_k) \\ \vdots \\ \mathrm{coef}_{n-1}(R_k) \end{pmatrix} = \begin{pmatrix} \alpha_1^k \\ \alpha_2^k \\ \alpha_3^k \\ \vdots \\ \alpha_n^k \end{pmatrix}, \tag{2.1}$$

donde $\mathrm{coef}_j(R_k)$ claramente hace referencia al coeficiente de grado j de R_k. De esta igualdad, podemos despejar los $\mathrm{coef}_j(R_k)$ invirtiendo la matriz de Vandermonde. Antes nos detendremos para apreciar las relaciones obtenidas.

2.1 Convergencia en el primer nivel

Utilizaremos la notación $V(a_1, a_2, \ldots, a_n)$ para referirnos a la matriz

$$\begin{pmatrix} 1 & a_1 & a_1^2 & \ldots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \ldots & a_2^{n-1} \\ 1 & a_3 & a_3^2 & \ldots & a_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \ldots & a_n^{n-1} \end{pmatrix};$$

asimismo, utilizaremos $W(a_1, a_2, \ldots, a_n)$ para denotar su inversa.

Lema 2.1. *Dados $a_1, a_2, \ldots, a_n$ distintos entre sí, se tiene*

$$W(a_1, a_2, \ldots, a_n)_{ij} = \mathrm{coef}_{i-1}\left(\prod_{l \neq j} \frac{z - a_l}{a_j - a_l}\right),$$

donde i e j son índices entre 1 y n.

Prueba. Por definición sabemos que se cumple

$$\begin{pmatrix} 1 & a_1 & a_1^2 & \dots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \dots & a_2^{n-1} \\ 1 & a_3 & a_3^2 & \dots & a_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \dots & a_n^{n-1} \end{pmatrix} W = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}.$$

Para cada $l \neq j$, al multiplicar la fila l por la columna j, se obtiene

$$\sum_{r=1}^{n} a_l^{r-1} W_{rj} = 0;$$

ello implica que el polinomio

$$W_j(z) = \sum_{r=1}^{n} W_{rj} z^{r-1}$$

tiene como raíces aquellos a_l distintos de a_j. No obstante, para $l = j$, se logra

$$W_j(a_j) = \sum_{r=1}^{n} a_j^{r-1} W_{rj} = 1;$$

y W_j resulta un polinomio no nulo. Por lo mismo, W_j tiene que ser un reescalamiento de

$$\prod_{l \neq j} (z - a_l).$$

Utilizamos la condición $W_j(a_j) = 1$ para confirmar que este reescalamiento debe ser

$$\prod_{l \neq j} \frac{z - a_l}{a_j - a_l}.$$

Al ser W_{ij} el coeficiente de grado $i-1$ de W_j, obtenemos lo deseado. □

Al pasar la matriz de Vandermonde en (2.1) a la izquierda, se logra

$$\begin{pmatrix} \text{coef}_0(R_k) \\ \text{coef}_1(R_k) \\ \text{coef}_2(R_k) \\ \vdots \\ \text{coef}_{n-1}(R_k) \end{pmatrix} = W(\alpha_1, \alpha_2, \dots, \alpha_n) \begin{pmatrix} \alpha_1^k \\ \alpha_2^k \\ \alpha_3^k \\ \vdots \\ \alpha_n^k \end{pmatrix}.$$

En esta última relación, conforme k avanza, el vector de la izquierda será cada vez más "parecido" a la primera columna de W; ello se debe a la preponderancia de α_1, en norma, sobre las demás raíces. Esto se hace explícito en el siguiente teorema.

Teorema 2.2. *Para $n \geq 1$, sea T_k una sucesión de polinomios de grado menor a n y $a_1, a_2, \ldots, a_n$ números complejos, de normas distintas entre sí, ordenados de mayor a menor norma. Supongamos que $T_k(a_j)$ puede expresarse como $C_{jk}a_j^k$ y los C_{jk} (vistos como sucesiones indexadas por k) converjan a un c_j. Si adicionalmente c_1 es no nulo, las formas mónicas de los T_k convergen al polinomio mónico de raíces $a_2, a_3, \ldots, a_n$.*

Prueba. Podemos trabajar con los T_k como si tuvieran n coeficientes (los completamos con ceros de ser necesario). Las relaciones asumidas se dejan leer en forma matricial cual

$$\begin{pmatrix} 1 & a_1 & a_1^2 & \ldots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \ldots & a_2^{n-1} \\ 1 & a_3 & a_3^2 & \ldots & a_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \ldots & a_n^{n-1} \end{pmatrix} \begin{pmatrix} \operatorname{coef}_0(T_k) \\ \operatorname{coef}_1(T_k) \\ \operatorname{coef}_2(T_k) \\ \vdots \\ \operatorname{coef}_{n-1}(T_k) \end{pmatrix} = \begin{pmatrix} a_1^k C_{1k} \\ a_2^k C_{2k} \\ a_3^k C_{3k} \\ \vdots \\ a_n^k C_{nk} \end{pmatrix}.$$

Si pasamos $V(a_1, a_2, \ldots, a_n)$ a derecha tenemos

$$\begin{pmatrix} \operatorname{coef}_0(T_k) \\ \operatorname{coef}_1(T_k) \\ \operatorname{coef}_2(T_k) \\ \vdots \\ \operatorname{coef}_{n-1}(T_k) \end{pmatrix} = W(a_1, a_2, \ldots, a_n) \begin{pmatrix} a_1^k C_{1k} \\ a_2^k C_{2k} \\ a_3^k C_{3k} \\ \vdots \\ a_n^k C_{nk} \end{pmatrix}.$$

Al dividir ambos miembros entre $a_1^k C_{1k}$, pues son distintos de 0 a partir de cierto K, conseguimos

$$\frac{1}{a_1^k C_{1k}} \begin{pmatrix} \operatorname{coef}_0(T_k) \\ \operatorname{coef}_1(T_k) \\ \operatorname{coef}_2(T_k) \\ \vdots \\ \operatorname{coef}_{n-1}(T_k) \end{pmatrix} = W(a_1, a_2, \ldots, a_n) \begin{pmatrix} 1 \\ (a_2/a_1)^k (C_{2k}/C_{1k}) \\ (a_3/a_1)^k (C_{3k}/C_{1k}) \\ \vdots \\ (a_n/a_1)^k (C_{nk}/C_{1k}) \end{pmatrix}. \quad (2.2)$$

Al tomar límite, obtenemos

$$\lim_{k\to\infty} \frac{1}{a_1^k C_{1k}} \begin{pmatrix} \operatorname{coef}_0(T_k) \\ \operatorname{coef}_1(T_k) \\ \operatorname{coef}_2(T_k) \\ \vdots \\ \operatorname{coef}_{n-1}(T_k) \end{pmatrix} = W(a_1, a_2, \ldots, a_n) \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

pues a_1 tiene norma máxima. Entonces para cada i desde 0 hasta $n-1$ se cumple

$$\lim_{k\to\infty} \frac{\operatorname{coef}_i(T_k)/a_1^k C_{1k}}{\operatorname{coef}_{n-1}(T_k)/a_1^k C_{1k}} = \frac{W_{i1}}{W_{(n-1)1}},$$

o, lo que es lo mismo,

$$\lim_{k\to\infty} \frac{\operatorname{coef}_i(T_k)}{\operatorname{coef}_{n-1}(T_k)} = \frac{W_{i1}}{W_{(n-1)1}}.$$

Esto es equivalente a decir que la forma mónica de los T_k converge a la forma mónica del polinomio ligado a la primera columna de W y este último, gracias al lema 2.1, es el polinomio deseado. □

Observación. Obsérvese que el polinomio ligado a la primera columna de $W(a_1, a_2, \ldots, a_n)$ es de grado $n-1$ y por ende a partir de cierto K, los T_k también tendrán grado $n-1$.

Definición 2.3. Diremos que una sucesión x_k converge a x geométricamente, si existe un δ menor que 1 tal que a partir de cierto índice K se tiene

$$|x_k - x| \leq \delta^k.$$

Corolario 2.4. *Bajo las condiciones del teorema 2.2, si los C_{jk} converjen a c_j geométricamente, entonces los coeficientes de las formas mónicas de los polinomios T_k también lo hacen.*

Prueba. En referencia a la prueba anterior, debemos notar que el índice K a partir del cual podemos dividir entre $a_1^k C_{1k}$ es aquel cuando C_{1k} se despega de 0. Sabemos que C_{1k} converge a c_1, distinto de 0, con una velocidad no menor a θ^k (donde θ es menor que 1) y por tal motivo se despegará de 0 a tal velocidad. En consonancia, de la relación (2.2) se observa que la convergencia de

$$\frac{1}{a_1^k C_{1k}} \begin{pmatrix} \operatorname{coef}_0(T_k) \\ \operatorname{coef}_1(T_k) \\ \operatorname{coef}_2(T_k) \\ \vdots \\ \operatorname{coef}_{n-1}(T_k) \end{pmatrix}$$

a la primera columna de $W(a_1, a_2, \ldots, a_n)$ es tan rápida como la convergencia de

$$\begin{pmatrix} 1 \\ (a_2/a_1)^k (C_{2k}/C_{1k}) \\ (a_3/a_1)^k (C_{3k}/C_{1k}) \\ \vdots \\ (a_n/a_1)^k (C_{nk}/C_{1k}) \end{pmatrix},$$

entrada por entrada, a

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

La primera entrada se queda estática en 1 y no hay nada que probar. Para los demás j, distintos de 1, la sucesión C_{jk}/C_{1k} converge a c_j/c_1 y por ende está acotada. De este modo,

$$\left(\frac{a_j}{a_1}\right)^k \cdot \frac{C_{jk}}{C_{1k}}$$

converge a 0 tan rápido como

$$\left(\frac{a_j}{a_1}\right)^k$$

lo hace. Por tales motivos, si denotamos con δ al máximo entre θ y los $|a_2/a_1|, |a_3/a_1|, \ldots, |a_n/a_1|$, tendremos que la convergencia de

$$\frac{\text{coef}_j(T_k)}{\text{coef}_{n-1}(T_k)}$$

a

$$\frac{W_{j1}}{W_{(n-1)1}}$$

es al menos tan rápida como la de δ^k a 0 y claramente δ es menor que 1. □

Observación. Aquí obsérvese que el corolario implica que el grado de T_k se estabiliza en $n-1$ a velocidad geométrica.

Nuestra primera aplicación del teorema 2.2 y su corolario será a la sucesión R_k, pues para esta tenemos $R_k(\alpha_j) = \alpha_j^k$; es decir, podemos tomar los C_{jk} como constantes (iguales a 1), y estos convergen trivialmente con la velocidad requerida. Concluimos que la forma mónica de los R_k converge al polinomio

ligado a la primera columna de $W(\alpha_1, \alpha_2, \ldots, \alpha_n)$ y este no es sino la forma mónica de $P(z)/(z-\alpha_1)$. Además la velocidad de convergencia en este caso será no menor a $|\alpha_2/\alpha_1|^k$. En la práctica este procedimiento permite separar α_1 de P y, mediante las relaciones de Peano, recuperar tal raíz cual

$$\alpha_1 = \lim_{k\to\infty} \frac{\text{coef}_{n-2}(R_k)}{\text{coef}_{n-1}(R_k)} - \frac{\text{coef}_{n-1}(P)}{\text{coef}_n(P)}; \tag{2.3}$$

más aún, la velocidad con la que se obtiene este límite es geométrica.

2.2 Armando el segundo nivel

Con los resultados obtenidos por el momento, podemos idear un algoritmo para calcular la raíz dominante, la de mayor norma. Podemos calcular los R_k iterativamente hasta habernos adentrado suficiente en la sucesión y luego usar la relación (2.3) para recuperar α_1. Sin embargo, hay un modo aún más rápido y esto viene de notar que podemos desplazarnos dentro de la sucesión de una forma exponencial en vez de una manera lineal. Es decir una vez computado R_k, en el siguiente paso podemos computar R_{2k} luego R_{4k} y así sucesivamente. Para materializar esta idea, observemos que R_{2k} no es nada más que el resto de R_k^2 módulo P. Además como el grado de los R's está acotado por n, la cantidad de operaciones que realizamos en cada paso no crece descomunalmente. Así, si indexamos nuestros pasos con la variable l, nuestra convergencia se dará con una velocidad no menor a $|\alpha_2/\alpha_1|^{2^l}$, exponencial (cuadrática) esta vez.

Ya en esta etapa, el sentido común dicta que mientras las condiciones nos lo permitan, podemos retirar α_1 de P mediante una división y repetir el proceso en $P(z)/(z-\alpha_1)$ para obtener la siguiente raíz. Sin embargo, incluso cuando las condiciones son las adecuadas este proceso no es el más eficiente ni el más preciso. En primer lugar, para calcular el resto de z^k módulo $P(z)/(z-\alpha_1)$ no debería ser necesario repetir todo desde un inicio; es decir los R_k deberían poder ser reutilizados para conseguir una sucesión que aísle α_2 de $P(z)/(z-\alpha_1)$ y así ahorrarnos muchos cálculos. En segundo lugar la división $P(z)/(z-\alpha_1)$ nos hará perder precisión (fundamentalmente en los coeficientes) dado que en ningún momento hemos calculado tal α_1, sino apenas aproximaciones de este.

Debido a todo ello, en aras de generalizar nuestro método, buscaremos un refinamiento de la relación (1.1) a fin de que como residuo obtengamos polinomios con grado, eventualmente, $n-2$. De esta manera, tal como R_k, por tener grado $n-1$, aisló α_1, abrigamos la esperanza de que esta nueva

sucesión de polinomios aísle α_1 y α_2. Llegado su momento veremos que relaciones tipo

$$z^k(z - a_k) \equiv S_k(z) \pmod{P}, \tag{2.4}$$

con S_k polinomio de grado no mayor a $n-2$, juegan un rol importante.

Esta relación no necesariamente se puede forzar para todo k. No obstante, a partir de cierto K, no solo existirá sino que además tanto a_k como S_k serán únicos (salvo dependencia en k).

Para construir estas relaciones es necesario fijar K_1, el índice a partir del cual R_k es honestamente de grado $n-1$; la existencia de este índice fue vista en la observación posterior al teorema 2.2. Ahora para todo índice k por encima de K_1, basta notar que

$$a_k = \frac{\text{coef}_{n-1}(R_{k+1})}{\text{coef}_{n-1}(R_k)}$$

y

$$S_k = R_{k+1} - a_k R_k$$

quedan sujetas a (2.4). Es natural preguntar si existe otro número complejo $\widehat{a_k}$ tal que la reducción de $z^k(z - \widehat{a_k})$ módulo P es de grado menor o igual a $n-2$. De existir, al restar ambas relaciones se tendrá que el resto de $z^k(\widehat{a_k} - a_k)$ módulo P será otro polinomio de grado menor igual a $n-2$. Esto es imposible ya que $\widehat{a_k} - a_k$ es distinto de 0 y el resto de z^k respecto a P es de grado $n-1$, pues k es mayor o igual a K_1.

Gracias a este último detalle, podemos analizar la sucesión a_k (comenzando en K_1) y si ver si esta nos lleva a algún lado.

Lema 2.5. *La sucesión a_k converge a α_1, la raíz de mayor norma de P.*

Prueba. Por definición de R_k como congruente a z^k se tiene

$$R_{k+1}(z) \equiv zR_k(z) \pmod{P}$$

con miembro izquierdo de grado $n-1$ y derecho de grado n. Por ende R_{k+1} puede hallarse de manera explícita como

$$R_{k+1}(z) = zR_k(z) - \frac{\text{coef}_{n-1}(R_k)}{\text{coef}_n(P)} P(z).$$

Al computar el coeficiente de grado $n-1$ en ambos lados obtenemos la igualdad

$$\text{coef}_{n-1}(R_{k+1}) = \text{coef}_{n-2}(R_k) - \text{coef}_{n-1}(R_k)\frac{\text{coef}_{n-1}(P)}{\text{coef}_n(P)},$$

o, lo que es equivalente,

$$a_k = \frac{\text{coef}_{n-2}(R_k)}{\text{coef}_{n-1}(R_k)} - \frac{\text{coef}_{n-1}(P)}{\text{coef}_n(P)}.$$

El lado derecho de esta igualdad es el mismo que en (2.3), de donde se sigue

$$\lim_{k\to\infty} a_k = \alpha_1.$$

□

Corolario 2.6. *La sucesión a_k converge con velocidad geométrica.*

Prueba. La velocidad de convergencia de esta sucesión depende de cuán rápido encontramos K_1 y de la convergencia de

$$\frac{\text{coef}_{n-2}(R_k)}{\text{coef}_{n-1}(R_k)}.$$

Este último converge geométricamente gracias al corolario 2.4. Además en la observación posterior a ese corolario, hemos indicado por qué el K_1 puede hallarse también a ese ritmo. □

2.3 Los siguientes niveles

A lo realizado en la sección anterior aún queda por justificar por qué los S_k en efecto aíslan las dos raíces superiores de P. Pero, no solo eso, sino que además es posible generalizar el trabajo a fin de aislar sucesivamente cada vez más raíces. Todos estos pendientes los condensaremos en un teorema.

Teorema 2.7. *Para cada d desde 0 hasta $n-2$ existe un índice K_d a partir del cual hay un único A_k^d mónico de grado d tal que el polinomio de la derecha en*

$$z^k A_k^d(z) \equiv S_k^d(z) \pmod{P}$$

es de grado no mayor a $n-d-1$. Adicionalmente, fijado el d, la sucesión A_k^d converge al polinomio mónico generado por las d raíces de mayor norma de P y los equivalentes mónicos de la sucesión S_k^d convergen al polinomio mónico generado por las $n-d-1$ raíces de menor norma de P. En otras palabras, esta relación aisla las $d+1$ raíces superiores de P.

Prueba. Para $d = 0$, $K_0 = 0$, todo lo que hay por demostrar o es trivial o ya fue hecho en los capítulos anteriores. Para $d = 1$, debemos constatar que los

S_k^1 aislan las dos raíces superiores de P. Demás está indicar que los K_d se asumen crecientes.

Procederemos por inducción. Supongamos que para cierto d conocemos la existencia, unicidad y convergencia de los A_k^d para los k mayores o iguales a K_d. Veamos por qué los S_k^d aislan las $d+1$ raíces superiores de P. Al reemplazar $\alpha_{d+1}, \alpha_{d+2}, \ldots, \alpha_n$ en la congruencia se obtiene

$$S_k^d(\alpha_j) = \alpha_j^k A_k^d(\alpha_j).$$

Gracias a la hipótesis inductiva se cumple

$$\lim_{k\to\infty} A_k^d(\alpha_j) = \prod_{l=1}^{d}(\alpha_j - \alpha_l),$$

lo cual implica que para los j mayores a d este límite es distinto de 0. Entonces S_k^d y las raíces $\alpha_{d+1}, \alpha_{d+2}, \ldots, \alpha_n$ satisfacen las condiciones del teorema 2.2. Gracias a este teorema, las formas mónicas de S_k^d convergen al polinomio deseado. En particular, existe un K_{d+1} tal que todo elemento de la sucesión con un índice mayor o igual a este tiene grado exactamente $n-d-1$.

Ahora construyamos el nivel $d+1$; el procedimiento será totalmente análogo a lo realizado en la sección anterior. Comencemos notando que para los k a partir de K_{d+1}, el polinomio

$$A_k^{d+1}(z) = zA_{k+1}^d(z) - \frac{\operatorname{coef}_{n-d-1}(S_{k+1}^d)}{\operatorname{coef}_{n-d-1}(S_k^d)} A_k^d(z)$$

cumple con lo que deseamos. De haber otro, digamos $\widehat{A_k}(z)$, tendríamos

$$z^k(\widehat{A_k}(z) - A_k^{d+1}(z)) \equiv T(z) \pmod{P},$$

donde T tiene un grado no mayor a $n-d-2$ al ser la diferencia de dos polinomios de a lo mucho ese grado. Si denotamos por e al grado de $\widehat{A_k} - A_k^{d+1}$, tendremos que este ha de ser menor o igual a d pues son mónicos de grado $d+1$.

De este modo, al tenerse $k \geq K_{d+1} \geq K_{e+1}$, el resto de

$$z^k(\widehat{A_k}(z) - A_k^{d+1}(z))$$

al ser dividido entre P tiene grado mayor a $n-e-1$ cuando $\widehat{A_k} - A_k^{d+1}$ no es un reescalamiento de A_k^e y exactamente $n-e-1$ cuando sí lo es. Así, por un lado tenemos que el grado de T no es mayor a $n-d-2$ y por otro es al menos $n-e-1$: de acá obtenemos $d+2 \leq e+1$, una clara contradicción.

Finalmente, para cerrar el círculo, resta probar que los A_k^{d+1} convergen al polinomio adecuado. Fijemos j en $1, 2, 3, \ldots, d+1$ y sea P_j el polinomio dado por

$$(z - \alpha_j) \cdot \frac{P(z)}{\prod_{i=1}^{d+1}(z - \alpha_i)}.$$

Ahora notemos que

$$z^k A_k^{d+1}(z) \equiv S_k^{d+1}(z) \pmod{P}$$

implica

$$z^k A_k^{d+1}(z) \equiv S_k^{d+1}(z) \pmod{P_j},$$

pues P_j divide a P. Para más comodidad expandamos A_k^{d+1} como

$$z^{d+1} + c_k^d z^d + \cdots + c_k^1 z + c_k^0.$$

Al reemplazar en la congruencia obtenemos

$$z^{k+d+1} + c_k^d z^{k+d} + \cdots + c_k^1 z^{k+1} + c_k^0 z^k \equiv S_k^{d+1}(z) \pmod{P_j}. \tag{2.5}$$

Por otro lado, a partir de cierto K, para cada i de k hasta $k+d$ existe un a_i^j tal que en

$$z^i(z - a_i^j) \equiv T_i^j(z) \pmod{P_j} \tag{2.6}$$

el polinomio T_i^j es de grado a lo sumo $n - d - 2$. Si nos fijamos en el grupo de polinomios (con la operación suma) cocientado por el subgrupo generado por los polinomios $1, z, z^2, \ldots, z^{n-d-2}$ junto al ideal (P_j), tenemos que (2.5) es equivalente a

$$z^{k+d+1} + c_k^d z^{k+d} + \cdots + c_k^1 z^{k+1} + c_k^0 z^k \sim 0$$

y las otras relaciones devienen en

$$z^i(z - a_i^j) \sim 0.$$

Ahondando un poco más en ellas tenemos

$$\begin{aligned}
z^{k+1} &\sim a_k^j z^k \\
z^{k+2} &\sim a_{k+1}^j a_k^j z^k \\
z^{k+3} &\sim a_{k+2}^j a_{k+1}^j a_k^j z^k \\
&\vdots \\
z^{k+d+1} &\sim a_{k+d}^j a_{k+d-1}^j \ldots a_k^j z^k;
\end{aligned}$$

y al reemplazarlas en (2.5) obtendremos

$$(c_k^0 + a_k^j c_k^1 + a_{k+1}^j a_k^j c_k^2 + \cdots + a_{k+d-1}^j a_{k+d-2}^j \ldots a_k^j c_k^d + a_{k+d}^j a_{k+d-1}^j \ldots a_k^j) z^k \sim 0.$$

En lenguaje cotidiano ello significa que este múltiplo de z^k, el polinomio de la izquierda, tiene un resto de grado menor o igual a $n-d-2$ cuando es dividido entre P_j. Esto es solo posible cuando el número que se obtiene al operar todo lo que está dentro de los paréntesis resulta 0, ya que el resto de dividir z^k entre P_j tiene grado $n-d-1$. Esta última afirmación es consecuncia de que la construcción de (2.6) requiere que el grado del resto de z^k ya se haya estabilizado —obligatoriamente en $n-d-1$—. En lenguaje matricial tenemos entonces

$$\begin{pmatrix} 1 & a_k^j & a_k^j a_{k+1}^j & \ldots & a_k^j a_{k+1}^j \ldots a_{k+d-1}^j \end{pmatrix} \begin{pmatrix} c_k^0 \\ c_k^1 \\ c_k^2 \\ \vdots \\ c_k^d \end{pmatrix} = -a_k^j a_{k+1}^j \ldots a_{k+d}^j.$$

Al combinar las relaciones obtenidas al variar j desde 1 hasta $d+1$ conseguimos

$$\begin{pmatrix} 1 & a_k^1 & a_k^1 a_{k+1}^1 & \ldots & a_k^1 a_{k+1}^1 \ldots a_{k+d-1}^1 \\ 1 & a_k^2 & a_k^2 a_{k+1}^2 & \ldots & a_k^2 a_{k+1}^2 \ldots a_{k+d-1}^2 \\ 1 & a_k^3 & a_k^3 a_{k+1}^3 & \ldots & a_k^3 a_{k+1}^3 \ldots a_{k+d-1}^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_k^{d+1} & a_k^{d+1} a_{k+1}^{d+1} & \ldots & a_k^{d+1} a_{k+1}^{d+1} \ldots a_{k+d-1}^{d+1} \end{pmatrix} \begin{pmatrix} c_k^0 \\ c_k^1 \\ c_k^2 \\ \vdots \\ c_k^d \end{pmatrix} = - \begin{pmatrix} a_k^1 a_{k+1}^1 \ldots a_{k+d}^1 \\ a_k^2 a_{k+1}^2 \ldots a_{k+d}^2 \\ a_k^3 a_{k+1}^3 \ldots a_{k+d}^3 \\ \vdots \\ a_k^{d+1} a_{k+1}^{d+1} \ldots a_{k+d}^{d+1} \end{pmatrix}.$$

Llamaremos M_k a la matriz de la izquierda y m_k al vector de la derecha y reformulamos la relación anterior como

$$M_k \begin{pmatrix} c_k^0 \\ c_k^1 \\ c_k^2 \\ \vdots \\ c_k^d \end{pmatrix} = -m_k.$$

Gracias al lema 2.5 los a_k^j convergen a la raíz de mayor norma de P_j, es decir a α_j. En consecuencia M_k converge a

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^d \\ 1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^d \\ 1 & \alpha_3 & \alpha_3^2 & \ldots & \alpha_3^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{d+1} & \alpha_{d+1}^2 & \ldots & \alpha_{d+1}^d \end{pmatrix}$$

y m_k converge a

$$\begin{pmatrix} \alpha_1^{d+1} \\ \alpha_2^{d+1} \\ \alpha_3^{d+1} \\ \vdots \\ \alpha_{d+1}^{d+1} \end{pmatrix}.$$

Como el límite de los M_k tiene inversa, tenemos la convergencia

$$\lim_{k\to\infty} \begin{pmatrix} c_k^0 \\ c_k^1 \\ c_k^2 \\ \vdots \\ c_k^d \end{pmatrix} = - \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^d \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^d \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{d+1} & \alpha_{d+1}^2 & \dots & \alpha_{d+1}^d \end{pmatrix}^{-1} \begin{pmatrix} \alpha_1^{d+1} \\ \alpha_2^{d+1} \\ \alpha_3^{d+1} \\ \vdots \\ \alpha_{d+1}^{d+1} \end{pmatrix}.$$

Esto caracteriza al límite de los coeficientes de A_k^{d+1} y son quienes queremos que sean, pues el resultado de la derecha solo puede ser el vector formado por los coeficientes no principales del polinomio mónico que tiene como raíces a $\alpha_1, \alpha_2, \dots, \alpha_{d+1}$. □

Corolario 2.8. *Así como en los demás resultados, la convergencia de las formas mónicas de los S_k^d se da con una velocidad geométrica.*

Prueba. Para $d = 0$, la velocidad de convergencia de los $A_k^0 = 1$ es trivial y la de los $R_k = S_k^0$ ya la hemos probado. Adicionalmente, la velocidad de convergencia de los A_k^1 está refrendada en el corolario 2.6.

Así, asumamos que para cierto d la velocidad de convergencia de los A_k^d es la deseada. Para los índices j desde $d+1$ hasta n, se tiene

$$S_k^d(\alpha_j) = \alpha_j^k A_k^d(\alpha_j)$$

y

$$\lim_{k\to\infty} A_k^d(\alpha_j) = \prod_{l=1}^{d} (\alpha_j - \alpha_l).$$

Este límite se da con una velocidad geométrica, pues por nuestra hipótesis, los A_k^d convergen con la velocidad deseada. Por tanto, los S_k^d y las raíces $\alpha_{d+1}, \alpha_{d+2}, \dots, \alpha_n$ satisfacen las condiciones del corolario 2.4 y por ende las versiones mónicas de S_k^d convergen geométricamente.

Ahora analicemos la sucesión A_k^{d+1}. Como consecuencia de lo que acabamos de probar, el índice K_{d+1} a partir del cual se podrá construir la sucesión A_k^{d+1} puede ser encontrado con una velocidad geométrica, pues este índice es en el cual los grado de los S_k^d se estabilizan en $n - d - 1$.

Recordemos que para la relación

$$\begin{pmatrix} 1 & a_k^1 & a_k^1 a_{k+1}^1 & \dots & a_k^1 a_{k+1}^1 \dots a_{k+d-1}^1 \\ 1 & a_k^2 & a_k^2 a_{k+1}^2 & \dots & a_k^2 a_{k+1}^2 \dots a_{k+d-1}^2 \\ 1 & a_k^3 & a_k^3 a_{k+1}^3 & \dots & a_k^3 a_{k+1}^3 \dots a_{k+d-1}^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_k^{d+1} & a_k^{d+1} a_{k+1}^{d+1} & \dots & a_k^{d+1} a_{k+1}^{d+1} \dots a_{k+d-1}^{d+1} \end{pmatrix} \begin{pmatrix} c_k^0 \\ c_k^1 \\ c_k^2 \\ \vdots \\ c_k^d \end{pmatrix} = - \begin{pmatrix} a_k^1 a_{k+1}^1 \dots a_{k+d}^1 \\ a_k^2 a_{k+1}^2 \dots a_{k+d}^2 \\ a_k^3 a_{k+1}^3 \dots a_{k+d}^3 \\ \vdots \\ a_k^{d+1} a_{k+1}^{d+1} \dots a_{k+d}^{d+1} \end{pmatrix}.$$

fue necesario trabajar no solo después de K_{d+1}, sino también a partir del K donde las relaciones

$$z^k(z - a_k^j) \equiv T_k^j \pmod{P_j}$$

existían. Nótese que este K, como consecuencia de la observación posterior al corolario 2.4, también puede ser encontrado con una velocidad geométrica (el corolario tendrá que ser aplicado a cada P_j y la más lenta de las $d+1$ velocidades obtenidas seguirá siendo adecuada). Así, el máximo entre K_{d+1} y K será encontrado con la velocidad deseada. Ahora, como a partir de este índice la relación mencionada anteriormente es válida, la matriz

$$\begin{pmatrix} 1 & a_k^1 & a_k^1 a_{k+1}^1 & \dots & a_k^1 a_{k+1}^1 \dots a_{k+d-1}^1 \\ 1 & a_k^2 & a_k^2 a_{k+1}^2 & \dots & a_k^2 a_{k+1}^2 \dots a_{k+d-1}^2 \\ 1 & a_k^3 & a_k^3 a_{k+1}^3 & \dots & a_k^3 a_{k+1}^3 \dots a_{k+d-1}^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_k^{d+1} & a_k^{d+1} a_{k+1}^{d+1} & \dots & a_k^{d+1} a_{k+1}^{d+1} \dots a_{k+d-1}^{d+1} \end{pmatrix},$$

bautizada como M_k, converge a

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^d \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^d \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{d+1} & \alpha_{d+1}^2 & \dots & \alpha_{d+1}^d \end{pmatrix}$$

con la velocidad deseada, pues cada a_k^j converge a α_j geométricamente. Como la inversa de M_k tiene una forma explícita en función de las entradas, resulta que M_k^{-1} también converge con la velocidad deseada a la matriz

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^d \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^d \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{d+1} & \alpha_{d+1}^2 & \dots & \alpha_{d+1}^d \end{pmatrix}^{-1}.$$

Finalmente, de la relación

$$\begin{pmatrix} c_k^0 \\ c_k^1 \\ c_k^2 \\ \vdots \\ c_k^d \end{pmatrix} = -M_k^{-1} \begin{pmatrix} a_k^1 a_{k+1}^1 \dots a_{k+d}^1 \\ a_k^2 a_{k+1}^2 \dots a_{k+d}^2 \\ a_k^3 a_{k+1}^3 \dots a_{k+d}^3 \\ \vdots \\ a_k^{d+1} a_{k+1}^{d+1} \dots a_{k+d}^{d+1} \end{pmatrix}.$$

obtendremos que

$$\begin{pmatrix} c_k^0 \\ c_k^1 \\ c_k^2 \\ \vdots \\ c_k^d \end{pmatrix}$$

converge geométricamente, pues el vector

$$\begin{pmatrix} a_k^1 a_{k+1}^1 \dots a_{k+d}^1 \\ a_k^2 a_{k+1}^2 \dots a_{k+d}^2 \\ a_k^3 a_{k+1}^3 \dots a_{k+d}^3 \\ \vdots \\ a_k^{d+1} a_{k+1}^{d+1} \dots a_{k+d}^{d+1} \end{pmatrix},$$

conocido como m_k, converge a

$$\begin{pmatrix} \alpha_1^{d+1} \\ \alpha_2^{d+1} \\ \alpha_3^{d+1} \\ \vdots \\ \alpha_{d+1}^{d+1} \end{pmatrix}$$

con la velocidad deseada. De esta forma, hemos completado la inducción. □

Capítulo 3

Factorizando polinomios solares

En este capítulo, seguiremos las pautas del anterior para construir un algoritmo capaz de factorizar polinomios solares.

En caso que nuestro polinomio P sea de grado 1 damos el algoritmo por terminado. De no ser así, recordemos que podemos aproximar α_1 mediante el límite

$$\alpha_1 = \lim_{k\to\infty} \frac{\text{coef}_{n-2}(R_k)}{\text{coef}_{n-1}(R_k)} - \frac{\text{coef}_{n-1}(P)}{\text{coef}_n(P)};$$

con velocidad de aproximación geométrica (en k). Por tal motivo, la expresión

$$\frac{\text{coef}_{n-2}(R_k)}{\text{coef}_{n-1}(R_k)} - \frac{\text{coef}_{n-1}(P)}{\text{coef}_n(P)};$$

será bautizada como $\text{aprox}_k(\alpha_1)$. Del mismo modo, tendremos el límite

$$\alpha_1 + \alpha_2 = \lim_{k\to\infty} \frac{\text{coef}_{n-3}(S_k^1)}{\text{coef}_{n-2}(S_k^1)} - \frac{\text{coef}_{n-1}(P)}{\text{coef}_n(P)};$$

y llamaremos $\text{aprox}_k(\alpha_2)$ al resultado de la expresión

$$\frac{\text{coef}_{n-3}(S_k^1)}{\text{coef}_{n-2}(S_k^1)} - \frac{\text{coef}_{n-1}(P)}{\text{coef}_n(P)} - \text{aprox}_k(\alpha_1).$$

Dentro de este patrón, para los j desde 2 hasta $n-1$, denotaremos por $\text{aprox}_k(\alpha_j)$ al valor obtenido en

$$\frac{\text{coef}_{n-j-1}(S_k^{j-1})}{\text{coef}_{n-j}(S_k^{j-1})} - \frac{\text{coef}_{n-1}(P)}{\text{coef}_n(P)} - \sum_{r=1}^{j-1} \text{aprox}_k(\alpha_r)$$

y por $\text{aprox}_k(\alpha_n)$ a lo obtenido en

$$-\frac{\text{coef}_{n-1}(P)}{\text{coef}_n(P)} - \sum_{r=1}^{n-1} \text{aprox}_k(\alpha_r).$$

No es difícil ver que la convergencia de $\text{aprox}_k(\alpha_1)$ hacia α_1 tiene un efecto dominó sobre las velocidades de convergencia de los $\text{aprox}_k(\alpha_j)$ hacia α_j, pues cada una de estas depende de que tan rápido convergen

$$\frac{\text{coef}_{n-j-1}(S_k^{j-1})}{\text{coef}_{n-j}(S_k^{j-1})}$$

y $\text{aprox}_k(\alpha_r)$, para los r anteriores a j. Para $\text{aprox}(\alpha_n)$ es aún más evidente, puesto que la rapidez de convergencia de este a α_n solo depende de la de los índices anteriores. Es decir, todas estas aproximaciones convergen geométricamente.

De este modo, una vez calculados $S_K^0, S_K^1, \ldots, S_K^{n-2}$ para un K suficientemente grande, logramos aproximaciones bastante decentes de las raíces de nuestro P.

3.1 Adentrándonos en el primer nivel

Ya hemos mencionado que en el primer nivel no es necesario avanzar de uno en uno, pues tenemos a mano la relación

$$S_{2k} \equiv S_k^2 \pmod{P}.$$

Lo que queremos insinuar es que para calcular S_{2k}^0 basta multiplicar S_k^0 consigo mismo y aplicar el algoritmo de división de Euclides a este resultado y a P.

Con la misma receta arrancamos de $S_1^0 = z$ y obtenemos $S_2^0, S_4^0, S_8^0, \ldots$ en ese orden. Detenerse en la iteración L significa que hemos calculado $S_1^0, S_2^0, S_4^0, \ldots, S_{2^L}^0$; por comodidad denotaremos por K a 2^L. Ahora notemos que en nuestra forma de pasar de S_k^0 a S_{2k}^0 hemos elevado S_k^0 al cuadrado y luego hemos aplicado el algoritmo de división de Euclides. Como el grado de S_k^0 no pasa de $n-1$, para elevarlo al cuadrado efectuamos no más de $O(n^2)$ sumas y restas. Asimismo, en aplicar la división a este resultado y a P tampoco se efectúa más de $O(n^2)$ sumas y restas, pues el grado del primero es a lo sumo $2n-2$ y el del segundo es n. El efecto final es que no se efectúan más de $2n^2$ operaciones en cada paso. Por ello el costo total de calcular S_K^0 es de $O(Ln^2)$, pues hay L multiplicaciones y L divisiones.

3.2 Avanzando en niveles más profundos

Ahora queda como tarea idear una forma de calcular los $S_K^1, S_K^2, \ldots, S_K^{n-2}$ de forma eficiente. Asumiremos que a partir del índice K los grados de los

polinomios $S_k^0, S_k^1, S_k^2, \ldots, S_k^{n-2}$ ya se han estabilizado en $n-1, n-2, n-3, \ldots, 1$ respectivamente. Podemos asumir esto sin afectar la velocidad del algoritmo, ya que los K con esta propiedad son encontrados geométricamente rápido.

Recordemos que S_k^d es el resto de $z^k A_k^d(z)$ módulo P y este fue construido en base al nivel anterior cual

$$A_k^d(z) = zA_{k+1}^{d-1}(z) - \frac{\operatorname{coef}_{n-d}(S_{k+1}^{d-1})}{\operatorname{coef}_{n-d}(S_k^{d-1})} A_k^{d-1}(z).$$

Así, el residuo de $z^k A_k^d(z)$ será el mismo que el de

$$z^{k+1} A_{k+1}^{d-1}(z) - \frac{\operatorname{coef}_{n-d}(S_{k+1}^{d-1})}{\operatorname{coef}_{n-d}(S_k^{d-1})} z^k A_k^{d-1}(z).$$

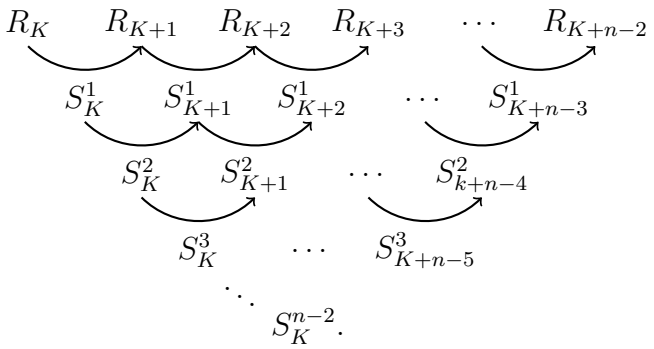
No obstante, este último es congruente módulo P a

$$S_{k+1}^{d-1}(z) - \frac{\operatorname{coef}_{n-d}(S_{k+1}^{d-1})}{\operatorname{coef}_{n-d}(S_k^{d-1})} S_k^{d-1}(z).$$

Este polinomio es de grado a lo sumo $n-d-1$, pues es el resto de dividir S_{k+1}^{d-1} entre S_k^{d-1}. La condición en el grado implica que este polinomio ha de ser S_k^d. De este modo, obtenemos la igualdad

$$S_k^d(z) = S_{k+1}^{d-1}(z) - \frac{\operatorname{coef}_{n-d}(S_{k+1}^{d-1})}{\operatorname{coef}_{n-d}(S_k^{d-1})} S_k^{d-1}(z),$$

lo cual sugiere calcular $S_K^1, S_K^2, \ldots, S_K^{n-2}$ mediante el esquema



La cantidad de operaciones de este proceso es $O(n^3)$, pues bajar del primer al segundo nivel toma $n-1$ restas de polinomios de grado $n-1$. Es decir, para bajar al segundo nivel estamos usando al menos $(n-1)^2$ operaciones.

Similarmente, para bajar al tercer nivel usaremos $(n-2)^2$, para bajar al cuarto serán $(n-3)^2$ y así sucesivamente. Esta suma da un polinomio de orden cúbico en n.

La gracia de este capítulo recae en que este resultado puede ser optimizado hasta $O(n^2)$. El truco radica en ingeniárnosla para constuir los S_K^d de una manera alternativa.

Sea T_K^1 resto de P al ser dividido por S_K^0. Este es de la forma

$$P(z) - B(z)S_K^0(z),$$

donde B es un polinomio de grado 1. Esto se debe a que P es exactamente un grado superior a S_K^0. Entonces módulo P tenemos

$$T_K^1(z) \equiv -B(z)S_K^0(z),$$

o, equivalentemente,

$$T_K^1(z) \equiv -B(z)z^K \pmod{P}.$$

Debido a que el grado de T_K^1 es a lo sumo $n-2$ por ser un residuo respecto a S_K^0, deducimos que $-B$ es un reescalamiento de A_K^1 y por ende T_K^1 lo es de S_K^1. Si remedamos esta idea logramos un análogo para los demás niveles.

Teorema 3.1. *Para los d mayores o iguales que 1 se tiene que S_K^{d+1} es un reescalamiento del residuo de dividir S_K^{d-1} entre S_K^d.*

Prueba. Comencemos observando que el grado de S_K^{d-1} es $n-d$ y el de S_K^d es $n-d-1$. Entonces el residuo, al que llamaremos T_K^{d+1}, es de la forma

$$S_K^{d-1}(z) - B(z)S_K^d(z),$$

para algún polinomio B de grado 1. Módulo P tendremos que se cumple

$$T_K^{d+1}(z) \equiv z^K A_K^{d-1}(z) - B(z)z^K A_K^d(z),$$

o, lo que es lo mismo,

$$T_K^{d+1}(z) \equiv z^K(A_K^{d-1}(z) - B(z)A_K^d(z)) \pmod{P}.$$

Sabemos que el grado de T_K^{d+1} es a lo sumo $n-d-2$ (por ser un residuo respecto a S_K^d) y que el polinomio dentro del paréntesis es de grado $d+1$, pues A_K^d y A_K^{d-1} son de grados d y $d-1$ exactamente. Ello implica que $A_K^{d-1} - BA_K^d$ ha de ser un reescalamiento de A_K^{d+1} y, consecuentemente, T_K^{d+1} lo es de S_K^{d+1}. □

Corolario 3.2. *El resto de dividir un reescalamiento de S_K^{d-1} por uno de S_K^d es un reescalamiento de S_K^{d+1}.*

Prueba. Esto es claro. □

Este corolario es muy conveniente ya que nos brinda otra manera de calcular reescalamientos de S_K^d. Para nuestros fines estos reescalamientos son idénticos al polinomio original, pues la información que extraemos de ellos viene a través de las relaciones de Peano. Es así que ahora podemos trabajar con la sucesión $T_K^0, T_K^1, T_K^2, \ldots, T_K^{n-2}$ donde T_K^0 es R_K, T_K^1 es el residuo de dividir P por R_K y para los d mayores que 1 el polinomio T_K^d es el resto de dividir T_K^{d-2} por T_K^{d-1}. Ahora notemos que la cantidad de operaciones realizadas para calcular T_K^d a partir de la división de T_K^{d-2} respecto a T_K^{d-1} es a lo sumo $O(n)$, debido a que los grados de estos son $n-d+1$ y $n-d$ respectivamente. Para llegar hasta T_K^{n-2}, este proceso se realizará menos de n veces y por ende la complejidad total no es mayor a $O(n^2)$.

Capítulo 4

Extendiendo nuestro algoritmo

Ahora extenderemos nuestros algoritmo para polinomios con coeficientes en $\mathbb{Q}[i]$. Primeramente, lo haremos para polinomios con raíces simples, sin ninguna condición en sus normas.

4.1 Polinomios con raíces simples

En esta nueva situación no podemos asegurar que el algoritmo propuesto en el capítulo anterior funcione. Por tal motivo, recurriremos a un enfoque probabilístico. Trasladaremos P por un número aleatorio τ y obtendremos el polinomio P_τ definido como

$$P_\tau(z) = P(z + \tau).$$

Las raíces de P_τ pasan a ser $\alpha_1 - \tau, \alpha_2 - \tau, \ldots, \alpha_n - \tau$. Este polinomio es solar cuando

$$|\alpha_i - \tau| \neq |\alpha_j - \tau|$$

para cualesquiera i y j distintos. Esta condición es equivalente a decir que τ se encuentra fuera de la mediatriz del segmento con extremos α_i y α_j.

Entonces P_τ será un polinomio solar si y solo si τ está fuera de las $\frac{n(n-1)}{2}$ mediatrices generadas por las n raíces de P. Es claro que una cantidad finita de rectas en el plano complejo forma un conjunto de medida nula y por ende cualquier τ que escojamos cumplirá con lo que buscamos casi seguramente.

Implementaremos esta idea de la siguiente manera. Recibimos el polinomio P con raíces simples y escogemos un τ aleatorio para trasladar P a P_τ. Luego, a este último le aplicamos el algoritmo descrito en el capítulo anterior. Finalmente, verificaremos que la evaluación de P_τ en las aproximaciones obtenidas resulte en valores muy próximos a 0. De ser este el caso,

sumamos τ a cada raíz obtenida para recuperar las raíces de P y damos por culminado el proceso. De no serlo, escogemos otro τ aleatorio y repetimos las instrucciones.

Notemos que para cada elección de τ tendremos que trasladar P, aplicar el algoritmo solar a P_τ y evaluar P_τ en los resultados obtenidos. El costo de trasladar P es $O(n^2)$. Para ver el porqué expandimos P como

$$c_n z^n + c_{n-1} z^{n-1} + \cdots + c_2 z^2 + c_1 z + c_0$$

y su traslado cual

$$c_n (z+\tau)^n + c_{n-1}(z+\tau)^{n-1} + \cdots + c_2 (z+\tau)^2 + c_1 (z+\tau) + c_0.$$

Primero, calcularemos $1, (z+\tau), (z+\tau)^2, \ldots, (z+\tau)^n$. Esto se puede hacer en $O(n^2)$ pasos, ya que cuando se obtiene $(z+\tau)^j$, el computar $(z+\tau)^{j+1}$ es apenas cuestión de multiplicarlo por $(z+\tau)$; y esto al ser la multiplicación de un polinomio de grado a lo sumo n con uno de grado 1 deviene en $O(n)$ de coste. Es así que calcular tal lista con $n+1$ elementos nos costará en total $O(n^2)$. De acá, para obtener P_τ tendremos que escalar cada $(z+\tau)^j$ por c_j y sumarlos todos. Como cada uno de estos tiene grado a lo sumo n, el coste de realizar esto será a lo sumo nuevamente $O(n^2)$. A la postre, el costo de aplicar el algoritmo a P_τ será de $O(Ln^2)$. Finalmente, evaluar P_τ en cualquier número complejo cuesta $O(n)$ operaciones, siguiendo el mismo proceso que usamos para trasladar P. Es así que evaluar P_τ en las n aproximaciones obtenidas tomará en total $O(n^2)$ operaciones.

En resumen, cada elección de τ nos cuesta $O(Ln^2)$; y si denotamos por I a la cantidad de veces que nos entregaremos al azar hasta salir exitosos, el costo total de nuestro algoritmo será $O(ILn^2)$.

Cuando obtengamos el éxito con algún τ, tendremos que corregir las aproximaciones obtenidas, sumándoles τ. Esta rutina tiene coste $O(n)$ y se realiza solo una vez; es decir, su costo es negligible. Como la elección de τ es un éxito casi seguro, se asumirá que tal I es una constante y la complejidad de nuestro algoritmo quedará en $O(Ln^2)$.

4.2 Polinomios con coeficientes gaussianos

Finalmente, atacaremos el caso en que P es cualquier polinomio con coeficientes en $\mathbb{Q}[i]$. Para completar este trabajo, basta notar que el polinomio $\frac{P}{\text{mcd}(P,P')}$ es un polinomio con raíces simples y, como conjunto, sus raíces son las mismas que las de P.

El coste de calcular $\mathrm{mcd}(P, P')$ con el algoritmo de Euclides es de $O(n^2)$ y el de dividir P por $\mathrm{mcd}(P, P')$ será también a lo sumo $O(n^2)$, pues ambos son polinomios de grado a lo sumo n.

La veracidad de nuestra primera afirmación recae en que el algoritmo de Euclides para polinomios es un proceso por etapas, donde en cada una de estas se reemplaza el polinomio de mayor grado por el resto de dividir este mismo por el de menor grado. Si denotamos con D el grado del dividendo y con d el grado del divisor, esta división realiza no más de $O(d(D - d + 1))$ operaciones básicas. Como en todo momento el grado de los polinomios envueltos no pasa de n, podemos controlar este costo por la suma de los $O(n(D - d + 1))$. Si indexamos las etapas desde 1 hasta c, y agregamos el número de etapa como subíndice tanto a D como a d tendremos que la complejidad rebasa

$$\sum_{j=1}^{c} O(n(D_j - d_j + 1)).$$

Llegado este punto, solo basta notar que se satisface $D_j = d_{j+1}$, pues el polinomio de menor grado se convierte en el dividendo de la siguiente etapa. Así esta suma se convierte en una telescópica con resultado final a lo sumo

$$O(n(D_1 + c)).$$

El número de etapas c no sobrepasa $2n$ ya que en cada una de estas la suma de grados tiene que decrecer en al menos 1 y sabemos también que D_1 no es superior a n; por ende, el coste de tomar mcd es a lo sumo $O(n^2)$.

Ahora si llamamos $\beta_1, \beta_2, \ldots, \beta_r$ a las aproximaciones obtenidas al aplicar el algoritmo de la sección anterior al polinomio

$$P/\,\mathrm{mcd}(P, P')$$

tendremos que estas aproximan también a todas las raíces de P. Sin embargo, aún nos falta conocer la multiplicidad de cada una. Esto se puede hacer en $O(n^2)$ pasos vía el siguiente proceso. Comenzaremos con el conjunto C_0 formado por $\beta_1, \beta_2, \ldots, \beta_r$ y denotaremos con C_j al subconjunto de C_0 cuyos elementos son las aproximaciones de las raíces de multiplicidad al menos j. En cada paso, avanzaremos de C_j a C_{j+1} manteniendo solo los β_j que al ser evaluados en $P^{(j+1)}$ entreguen un resultado muy cercano a 0. Llamaremos $m_1, m_2, \ldots, m_r$ a las multiplicades de las raíces asociadas a $\beta_1, \beta_2, \ldots, \beta_r$ respectivamente. En el proceso descrito tendremos que β_j será evaluado solo en $P^{(1)}, P^{(2)}, \ldots, P^{(m_j)}$ y por ende β_j está envuelto en a lo sumo $O(nm_j)$ operaciones, pues cada uno de esos polinomios es de grado menor a n. Como

cada evaluación envuelve a un β_j, la cantidad total de operaciones en estas es de orden

$$\sum_{j=1}^{r} O(nm_j).$$

Recordemos que $\beta_1, \beta_2, \ldots, \beta_r$ son las únicas raíces de P y por ende se cumple

$$\sum_{j=1}^{r} m_j = n.$$

De este modo, la cantidad de operaciones que envuelven evaluaciones será de orden $O(n^2)$. Finalmente, para poder evaluar es necesario haber precalculado $P^{(1)}, P^{(2)}, \ldots, P^{(n)}$. Esto se logra en $O(n^2)$ pasos, pues cuando ya se ha calculado $P^{(j)}$ computar $P^{(j+1)}$ requiere solo de derivarlo y esto último no cuesta más de $O(n)$.

Es así que este proceso nos da una manera de identificar la multiplicidad de la raíz asociada a cada una de nuestras aproximaciones y nuestro objetivo ha sido logrado.

Capítulo 5

Ejemplos

En este capítulo, usaremos diversos tipos de polinomios y varios L's para poner a prueba nuestro algoritmo. Apuntamos el tiempo de ejecución y el error entre las raíces originales y las estimadas. El error que presentaremos es la norma máxima que se obtiene al restar cada raíz original con su aproximación.

5.1 Ejemplos solares

Como indica el subtítulo, primero aplicaremos nuestro algoritmo a polinomios solares. Para nuestro primer ejemplo factorizaremos el polinomio de grado 8 cuyas raíces, ordenadas de mayor a menor norma, son

$$-86.89 - 86.31i, \quad 94.13 - 40.79i, \quad -38.79 + 93.66i, \quad -16.83 + 97.28i,$$
$$59.40 + 54.30i, \quad 48.14 + 51.89i, \quad 8.61 - 68.68i, \quad -24.72 - 0.29i.$$

Los datos resultantes son

L	Error	Tiempo(s)
9	2.48678496517633 e+00	2.097873
10	5.55864020303965 e-03	2.297969
11	2.85092913530717 e-08	2.602764
12	7.49912343678924 e-19	2.862336
13	5.18870120359987 e-40	3.146492
14	2.48401939244180 e-82	3.506579
15	5.69308439267136 e-167	3.769358
16	2.99042490627857 e-336	4.067766
17	8.25094058945234 e-675	4.281943

.

En nuestro segundo ejemplo abordaremos un polinomio más exigente, de grado 15 y con raíces

$$\begin{array}{cccc} 65.28+82.62i, & 65.43-78.05i, & -61.08+81.22i, & -80.59-61.44i, \\ 63.75+72.52i, & -9.01-95.36i, & -86.41+38.1i, & -84.26+29.83i, \\ -32.19-82.17i, & -86.6+7.15i, & -73.17+42.42i, & -67.29-37.98i, \\ 61.2-37.38i, & 40.11-17.87i, & -2.71+25.38i. & \end{array}$$

Los resultados son

L	Error	Tiempo(s)
9	1.71606300958666 e+02	6.432603
10	1.16870450886076 e+02	7.39478
11	1.40312690668573 e+01	8.729825
12	1.52261829617272 e-01	9.579008
13	1.85233771472376 e-05	10.270248
14	2.87602807029042 e-13	11.191145
15	6.94842659887015 e-29	12.530459
16	4.05576871972971 e-60	13.358438
17	1.38180447847317 e-122	14.519003

.

En nuestro tercer ejemplo trataremos un caso aún más ambicioso, un polinomio de grado 20 con las siguientes raíces

$$\begin{array}{cccc} 82.614+95.503i, & 73.820-85.456i, & 48.209+98.959i, & 74.396+67.443i, \\ 76.031-55.370i, & 0.585+85.116i, & -1.155-84.987i, & 71.868-35.413i, \\ -79.424-7.908i, & 19.625-75.453i, & 51.284+50.830i, & 5.910-63.016i, \\ -20.784-55.677i, & -0.006-56.818i, & 8.675-50.414i, & 47.256+7.628i, \\ 36.939-17.329i, & 33.265-19.947i, & 0.702-32.531i, & -7.853-0.752i. \end{array}$$

Los resultados obtenidos son

L	Error	Tiempo(s)
9	1.70110418945225 e+02	11.179499
10	1.70119477367685 e+02	12.886351
11	1.70174445154308 e+02	14.436636
12	1.71159608665796 e+02	16.329424
13	4.21823120164742 e+01	18.262189
14	3.95441090032439 e-04	20.19468
15	1.96705411695599 e-14	21.137639
16	4.86726914562631 e-35	22.994728

.

Nótese que en todos estos ejemplos el error converge a 0 cuadráticamente; esto se debe a que las raíces tienen normas distintas entre sí.

5.2 Ejemplos ciclotómicos

Como segunda tanda de ejemplos, atacamos polinomios con raíces distintas y todas de la misma norma. No hay mejor ejemplo para esto que los polinomios ciclotómicos. A los datos que recopilamos, le adicionaremos la cantidad de traslaciones usadas (I), la traslación exitosa (τ) y el tiempo de ejecución que tomó factorizar esta última traslación (T. parcial).

Factorizando el ciclotómico de grado 12, obtenemos los siguientes datos

L	I	τ	Error	T. parcial(s)	T. total(s)
12	2	-1.000 - 0.333 i	6.00895058487385 e-108	6.8288	7.7026
13	4	2.500 - 2.000 i	7.91955448184537 e-64	6.7164	20.6813
14	2	-1.500 + 3.000 i	3.08917329697785 e-124	7.3130	8.2123
15	2	0.400 - 2.000 i	7.10394489293256 e-235	8.1733	9.0533
16	2	1.500 - 0.500 i	9.56787208943175 e-1063	8.7043	9.6788
17	4	2.250 + 2.500 i	4.86699795255496 e-1253	9.2998	28.7574

El mismo proceso para el ciclotómico de grado 16 nos entrega

L	I	τ	Error	T. parcial(s)	T. total(s)
13	3	1.250 + 0.333 i	2.26366882030496 e-72	12.258948	25.180862
14	2	0.667 + 2.000 i	2.93112870543631 e-104	13.464402	14.542886
15	2	1.000 + 0.667 i	3.62956296365267 e-178	15.319702	16.611212
16	2	1.500 + 1.500 i	1.44986069373499 e-213	8.074469	9.199366
17	3	2.000 - 0.600 i	4.11970666685904 e-723	17.137764	34.726252

Finalmente, para el de grado 18, obtenemos

L	I	τ	Error	T. parcial(s)	T. total(s)
13	3	1.800 + 0.400	1.10304471907419 e-58	15.095463	30.527987
14	2	-0.500 - 0.800	2.20504070470069 e-49	15.751336	17.607339
15	2	1.000 + 0.250	6.02772892023472 e-204	16.854215	18.579865
16	2	2.000 + 1.333	9.59940270435357 e-289	19.138753	21.295259
17	2	3.333 + 0.333	2.89710669358940 e-666	20.878759	23.044009

Nótese que para este grupo de polinomios, tanto los errores como los tiempos son más erráticos debido al componente aleatorio que hemos agregado.

5.3 Ejemplos mixtos

Para nuestra última muestra tomaremos polinomios que tengan tanto raíces repetidas como raíces distintas con la misma norma.

En nuestro primer ejemplo afrontaremos el polinomio de grado 8, de raíces

$$-30.00 + 40.00i, \quad -30.00 + 40.00i, \quad 40.00 - 30.00i, \quad 12.34 - 23.45i,$$
$$-12.34 + 23.45i, \quad 5.67 - 8.90i, \quad -5.67 + 8.90i, \quad 0.00 + 0.00i.$$

Claramente tenemos raíces repetidas y también raíces distintas de misma norma. Para este caso, obtenemos los siguientes resultados

L	I	τ	Error	T. parcial(s)	T. total(s)
10	6	-0.667 + 2.000 i	2.24180635240550 e-63	2.097254	11.253781
11	2	-0.667 + 2.000 i	3.77840581612173 e-130	2.321483	4.162268
12	3	-1.800 + 1.667 i	5.21435996351661 e-342	2.520397	6.750634
13	2	0.200 + 2.000 i	3.96200161130099 e-358	2.790895	4.903745
14	2	-3.333 + 0.333 i	7.26126603535547 e-1439	2.976863	5.340433
15	2	-0.333 + 2.667 i	1.14026824122488 e-2408	3.420871	6.034366
16	2	0.400 - 1.333 i	2.57665022178560 e-2749	3.386469	6.154771

Nuestro segundo ejemplo será el polinomio de grado 10 dado por las raíces

$$30.12 - 12.34i, \quad 30.12 - 12.34i, \quad -10.00 + 30.00i, \quad -24.00 - 7.00i,$$
$$7.00 - 24.00i, \quad -15.00 - 20.00i, \quad 15.00 + 20.00i, \quad 3.45 + 10.23i,$$
$$-1.23 - 3.45i, \quad -1.23 - 3.45i.$$

Aquí también tenemos raíces repetidas y raíces distintas de misma norma. Los datos que obtenemos al correr nuestro algoritmo sobre este polinomio son

L	I	τ	Error	T. parcial(s)	T. total(s)
12	2	3.333 - 1.333 i	1.30775005222565 e-127	3.693511	6.908053
13	8	1.333 + 1.750 i	1.21532793593321 e-103	3.798908	28.824791
14	2	-2.500 + 0.250 i	5.89186788780851 e-183	4.003009	7.943665
15	2	1.500 - 2.000 i	9.41223420707098 e-190	4.337286	8.36966
16	2	-1.000 + 0.000 i	2.13161492824908 e-699	4.464963	8.825763
17	2	0.000 - 2.000 i	1.87535322551842 e-1681	5.060881	9.657633

Para nuestro ejemplo final, usaremos el polinomio de grado 13 dado por la raíces

$$-40.00i + 30.00i, \quad 40.00 - 30.00i, \quad -30.00 - 40.00i, \quad 25.50 + 35.50i,$$
$$-25.50 - 35.50i, \quad -20.00 + 44.00i, \quad 20.00 - 44.00i, \quad 12.34 - 12.34i,$$
$$-12.34 + 12.34i, \quad 5.00 + 5.00i, \quad -5.00 - 5.00i, \quad 1.11 - 0.99i,$$
$$1.11 - 0.99i,$$

Este también es un polinomio mixto y los resultados obtenidos para este son

L	I	τ	Error	T. parcial(s)	T. total(s)
13	3	0.000 - 3.333 i	9.31380942041943 e-92	6.822192	19.641109
14	6	-0.400 - 1.800 i	3.95714697784651 e-152	7.500962	42.83404
15	7	-0.600 - 1.250 i	1.98829324927868 e-295	9.450124	59.248432
16	4	-2.667 - 1.750 i	2.03754166321791 e-347	9.624125	37.24789

.

5.4 Implementación

La implementación en el lenguaje C++ del algoritmo sigue a continuación.

```
#include <gmpxx.h>
#include <iostream>
#include <iomanip>
#include <utility>
#include <vector>
#include <random>
#include <chrono>

std::mt19937_64 rng(
  std::chrono::steady_clock::now().time_since_epoch().count()
);

mpf_class eps_coef() {
  return mpf_class("1e-50000");
}

mpf_class eps_zero() {
  return mpf_class("1e-18");
}
```

```
int random(int a, int b) {
  return std::uniform_int_distribution<int> (a, b) (rng);
}

struct Complex_q {
  mpq_class real;
  mpq_class imag;

  Complex_q() : real(0), imag(0) {}
  Complex_q(mpq_class r, mpq_class j) : real(r), imag(j) {}

  Complex_q& operator -= (const Complex_q& w) {
    real -= w.real;
    imag -= w.imag;
    return *this;
  }

  Complex_q operator * (const Complex_q& w) const {
    mpq_class r = real * w.real - imag * w.imag;
    mpq_class j = real * w.imag + imag * w.real;
    return Complex_q(r, j);
  }

  Complex_q operator / (const Complex_q& w) const {
    mpq_class norm = w.real * w.real + w.imag * w.imag;
    mpq_class r = (real * w.real + imag * w.imag) / norm;
    mpq_class j = (w.real * imag - real * w.imag) / norm;
    return Complex_q(r, j);
  }

  bool nul() const {
    if(real == 0 and imag == 0) return true;
    return false;
  }

  friend std::istream& operator >> (
    std::istream& is, Complex_q& z
  ) {
    is >> z.real >> z.imag;
    return is;
```

```
  }
};

struct Complex_f {
  mpf_class real;
  mpf_class imag;

  Complex_f() : real(0), imag(0) {}
  Complex_f(mpf_class r, mpf_class j) : real(r), imag(j) {}
  Complex_f(Complex_q z) : real(z.real), imag(z.imag) {}

  Complex_f& operator += (const Complex_f& w) {
    real += w.real;
    imag += w.imag;
    return *this;
  }

  Complex_f& operator -= (const Complex_f& w) {
    real -= w.real;
    imag -= w.imag;
    return *this;
  }

  Complex_f operator - (const Complex_f& w) const {
    return Complex_f(real - w.real, imag - w.imag);
  }

  Complex_f operator - (const Complex_q& w) const {
    return Complex_f(real - (mpf_class) w.real,
      imag - (mpf_class) w.imag);
  }

  Complex_f operator * (const Complex_f& w) const {
    mpf_class r = real * w.real - imag * w.imag;
    mpf_class j = real * w.imag + imag * w.real;
    return Complex_f(r, j);
  }

  Complex_f operator / (const Complex_f& w) const {
    mpf_class norm = w.real * w.real + w.imag * w.imag;
    mpf_class r = (real * w.real + imag * w.imag) / norm;
```

```
    mpf_class j = (w.real * imag - real * w.imag) / norm;
    return Complex_f(r, j);
  }

  bool close_to_zero() {
    if(abs(real) < eps_zero() and abs(imag) < eps_zero())
      return true;
    return false;
  }

  bool lead_zero() {
    if(abs(real) < eps_coef() and abs(imag) < eps_coef())
      return true;
    return false;
  }

  friend std::ostream& operator << (
  std::ostream& os, const Complex_f& z
  ) {
    os << z.real << " & " << z.imag;
    return os;
  }
};

struct Poly_q{
  std::vector<Complex_q> pol;
  int deg;

  Poly_q() : pol({}), deg(-1) {}
  Poly_q(std::vector<Complex_q> p) :
    pol(p), deg((int) p.size() - 1) {}

  Poly_q operator / (const Poly_q& q) const {
    Poly_q copy = *this;
    std::vector<Complex_q> p(0);
    while(copy.deg >= q.deg) {
      p.push_back(copy.pol.back() / q.pol.back());
      for(int i = 0; i <= q.deg; i++) {
        copy.pol[copy.deg - i] -= p.back() * q.pol[q.deg - i];
      }
      copy.pol.pop_back();
```

```
      copy.deg -= 1;
    }
    std::reverse(p.begin(), p.end());
    return Poly_q(p);
  }

  Poly_q& operator %= (const Poly_q& q) {
    while(deg >= q.deg) {
      Complex_q cte = pol.back() / q.pol.back();
      for(int i = 0; i <= q.deg; i++) {
        pol[deg - i] -= cte * q.pol[q.deg - i];
      }
      pol.pop_back();
      deg -= 1;
    }
    while(deg >= 0 and pol.back().nul()) {
      pol.pop_back();
      deg -= 1;
    }
    return *this;
  }

  bool nul_in(Complex_f z) {
    Complex_f r(pol.back());
    for(int i = deg - 1; i >= 0; i--) {
      r = r * z;
      r += Complex_f(pol[i]);
    }
    r = r / Complex_f(pol.back());
    return r.close_to_zero();
  }

  friend std::istream& operator >> (
    std::istream& is, Poly_q& p
  ) {
    is >> p.deg;
    p.pol = std::vector<Complex_q> (p.deg + 1);
    for(int i = p.deg; i >= 0; i--) {
      is >> p.pol[i];
    }
    return is;
```

```
  }

  friend std::ostream& operator << (
    std::ostream& os, const Poly_q& p
  ) {
    os << p.deg << std::endl;
    for(int i = p.deg; i >= 0; i--) {
      os << p.pol[i] << std::endl;
    }
    return os;
  }

};

struct Poly_f{
  std::vector<Complex_f> pol;
  int deg;

  Poly_f() : pol({}), deg(-1) {}
  Poly_f(std::vector<Complex_f> p) :
    pol(p), deg((int) p.size() - 1) {}
  Poly_f(Poly_q P) : deg(P.deg) {
    pol = std::vector<Complex_f> (0);
    for(Complex_q& z: P.pol) pol.push_back(Complex_f(z));
  }

  Poly_f operator * (const Poly_f& q) const {
    if(deg == -1 or q.deg == -1) return Poly_f();
    int n = deg + q.deg;
    std::vector<Complex_f> p(n + 1);
    for(int i = 0; i <= deg; i++) {
      for(int j = 0; j <= q.deg; j++) {
        p[i + j] += pol[i] * q.pol[j];
      }
    }
    return Poly_f(p);
  }

  Poly_f& operator %= (const Poly_f& q) {
    while(deg >= q.deg) {
      Complex_f cte = pol.back() / q.pol.back();
```

```
      for(int i = 0; i <= q.deg; i++) {
        pol[deg - i] -= cte * q.pol[q.deg - i];
      }
      pol.pop_back();
      deg -= 1;
    }
    while(deg >= 0 and pol.back().lead_zero()) {
      pol.pop_back();
      deg -= 1;
    }
    return *this;
  }

  bool nul_in(Complex_f z) {
    Complex_f r(pol.back());
    for(int i = deg - 1; i >= 0; i--) {
      r = r * z;
      r += Complex_f(pol[i]);
    }
    r = r / pol.back();
    return r.close_to_zero();
  }

  Complex_f roots_sum() const {
    return Complex_f(-1, 0) * pol[deg - 1] / pol[deg];
  }

  friend std::ostream& operator << (
    std::ostream& os, const Poly_f& p
  ) {
    os << p.deg << std::endl;
    for(int i = p.deg; i >= 0; i--) {
      os << p.pol[i] << std::endl;
    }
    return os;
  }
};

Poly_q gcd(Poly_q P, Poly_q Q) {
  while(Q.deg != -1) {
    P %= Q;
```

```
    std::swap(P, Q);
  }
  return P;
}

Poly_q der(Poly_q P) {
  for(int i = 0; i < P.deg; i++) {
    P.pol[i] = P.pol[i + 1] * Complex_q(i + 1, 0);
  }
  P.pol.pop_back();
  P.deg -= 1;
  return P;
}

Complex_f get_root(Poly_f& T, Poly_f& T_aux) {
  return T.roots_sum() - T_aux.roots_sum();
}

std::vector<Complex_f> algoritmo_solar(
  const Poly_f& P, int L, bool& finished
) {
  Poly_f R({Complex_f(), Complex_f(1, 0)});
  for(int i = 0; i < L; i++) {
    R = R * R;
    R %= P;
  }
  std::vector<Complex_f> roots_0(0);
  Poly_f T = P;
  Poly_f T_aux = R;
  while(T_aux.deg >= 1) {
    if(T.deg - T_aux.deg != 1) {
      finished = false;
      return {};
    }
    roots_0.push_back(get_root(T, T_aux));
    T %= T_aux;
    std::swap(T, T_aux);
  }
  if((int) roots_0.size() != P.deg - 1) {
    finished = false;
    return {};
```

```
  }
  roots_0.push_back(T.roots_sum());
  return roots_0;
}

Poly_f translate(const Poly_f& P, Complex_f r) {
  Poly_f z({r, Complex_f(1, 0)});
  Poly_f P_r({P.pol.back()});
  for(int i = P.deg - 1; i >= 0; i--) {
    P_r = P_r * z;
    P_r.pol[0] += P.pol[i];
  }
  return P_r;
}

mpq_class aleat() {
  int num = random(-10, 10);
  int den = random(3, 5);
  return mpq_class(num, den);
}

int main() {
  mpf_set_default_prec(200000);
  int L;
  Poly_q P;
  std::cin >> P >> L;

  Poly_f P_base = P / gcd(P, der(P));

  bool finished = false;
  std::vector<std::vector<Complex_f>> roots(1);
  Complex_f r;
  while(!finished) {
    Poly_f P_r = translate(P_base, r);
    finished = true;
    roots[0] = algoritmo_solar(P_r, L, finished);
    for(Complex_f& z: roots[0]) {
      if(!P_r.nul_in(z)) {
        finished = false;
        break;
      }
```

```
    }
    if(!finished) r = Complex_f(aleat(), aleat());
  }

  for(Complex_f& z: roots[0]) z += r;
  while(P.deg >= 1) {
    P = der(P);
    std::vector<Complex_f> deeper_roots;
    for(Complex_f& z: roots.back()) {
      if(P.nul_in(z)) {
        deeper_roots.push_back(z);
      }
    }
    roots.push_back(deeper_roots);
  }

  for(auto v: roots) for(auto z: v) std::cout << z << std::endl;

  return 0;
}
```

Referencias

J. Torres. *Factorización de polinomios con dinámica compleja.* Trabajo de Bachillerato, Pontificia Universidad Católica del Perú, 2020.

J. Torres. Approximating roots of polynomials. Tesis de Licenciatura, Pontificia Universidad Católica del Perú, 2021.

Poirier, A. y Torres, J. (2023). Approximating roots by quadratic iteration. Proyecciones (Antofagasta, On line), 42 (2), pp. 407-431.

Poirier, Alfredo. *Iteración de polinomios y funciones racionales.* Fondo Editorial PUCP, 2016.