PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Factorización de polinomios con dinámica compleja

TRABAJO DE INVESTIGACIÓN PARA LA OBTENCIÓN DEL GRADO DE BACHILLER EN CIENCIAS CON MENCIÓN EN MATEMÁTICAS

AUTOR

Jesús Stefano Torres Romero

ASESOR

Alfredo Bernardo Poirier Schmitz

Lima, octubre 2020

Resumen

Dentro del campo de las matemáticas, el problema de hallar las raíces de un polinomio es un problema fundamental. Este trabajo tiene como objetivo aplicar métodos de dinámica compleja e iteración de polinomios para resolver dicho problema. Partimos de ejemplos y buscamos las generalidades de los mismos con el objetivo de desarrollar un algoritmo general, que nos permita factorizar un polinomio arbitrario. Además, consideramos un análisis de los posibles limitaciones que presenta el algoritmo.

Índice general

Introducción

En matemáticas modernas, la teoría de sistemas dinámicos complejos ha sido un área muy activa por los últimos cuarenta años [Mil11].

El propósito de esta tesina es utilizar métodos elementales de iteración de funciones cuadráticas a un problema concreto: factorizar polinomios de coeficientes reales. Ver [Poi95].

Con la finalidad de motivar el objeto de estudio, supongamos que nos interesa factorizar el polinomio $p(x) = x^3 - x^2 - 2x$ en términos lineales. Por supuesto, este es el mismo problema que encontrar sus raíces; y somos conscientes de que tales raíces son 0, -1 y 2. Cada una de estas raíces α puede ser enmaracada en un proceso de iteración de la forma $z \mapsto z^2$. En cada uno de estos tres casos la iteración se hará con semilla $z_0 = \alpha$. La razón detrás de esta elección se hará clara en su momento.

De esta manera, para $\alpha = 0$ obtenemos la órbita estacionaria

$$0 \mapsto 0 \mapsto 0 \mapsto 0 \mapsto \dots.$$

Por otro lado, para $\alpha = -1$ se logra un proceso eventualmente constante

$$-1 \mapsto 1 \mapsto 1 \mapsto 1 \mapsto 1 \mapsto 1 \dots.$$

Por su parte, para $\alpha = 2$ tenemos

$$2 \mapsto 4 \mapsto 16 \mapsto 256 \mapsto \dots,$$

una iteración en apariencia un tanto menos regular. Podemos observar (y también se puede probar rigurosamente) que la órbita de $\alpha = 2$ crece mucho más rápido que las otras dos. La pregunta se convierte entonces en cómo aprovechar este hecho en nuestro beneficio.

La idea consiste en lo siguiente. Como —supuestamente— en la vida real no conocemos las raíces de antemano, llamemos α a una raíz arbitraria, con énfasis así en su caracter de incógnita. Al ponerlas todas juntas iniciamos la iteración que se mencionó previamente, la misma que ahora toma la forma $z \mapsto z^2$, igualmente con semilla α. Nótese que esta iteración, a pesar de estar escrita con un $z = \alpha$ desconocido, carga información de las tres raíces en paralelo.

De esta manera obtenemos la siguiente secuencia de polinomios

$$z \mapsto z^2 \mapsto z^4 \mapsto z^8 \mapsto z^{16} \mapsto \dots.$$

En un principio parece que el grado de estos polinomios crece sin control, pero este no es el caso. Recordemos que z representa una raíz arbitraria de $p(x)$ (o, equivalentemente, a todas las raíces de $p(x)$ a la vez), detalle inseparable de $p(x) = 0$.

En otras palabras, podemos efectuar esta iteración módulo p o, formalmente, seguir el proceso en el anillo cociente $\mathbb{R}[x]/<p(x)>$. Con ello tenemos

$$x \mapsto x^2 \mapsto 3x^2 + 2x \mapsto 43x^2 + 42x \mapsto 10923x^2 + 10922x \mapsto 715827883x^2 + 715827882x \mapsto \ldots,$$

con expresiones de grado a lo mucho 2. Por supuesto, si reemplazamos $x = 0, -1, 2$ recuperamos las sucesiones mostradas párrafos atrás.

Podemos transformar el polinomio $10923x^2 + 10922x$ en un polinomio mónico sin alterar las raíces. Al dividir entre el término líder obtenemos $x^2 + 0.99990845005951x$, un polinomio con raíces (aproximadas) 0 y -1. Estas son las dos raíces —del polinomio original— con órbita acotada (es decir, que al iterar no tienden a infinito). Al pasar al siguiente nivel de iteración obtenemos el polinomio $x^2 + 0.99999999860302x$, y observamos que conforme avanzamos en la iteración nos acercamos más y más a las raíces del polinomio original.

Podemos apreciar que el polinomio anterior mantiene exclusivamente dos raíces del polinomio original. Con ello, cabe preguntarse dónde se esfumó la tercera. La respuesta es clara, podemos aventurarnos a decir que se perdió en el proceso de iteración. Sin embargo, recuperar dicha raíz a partir del polinomio inicial y su factor perseverante es rutinario vía métodos elementales.

El motivo de esta tesina es, primero, entender lo que se esconde detrás de este fenómeno y, segundo, aplicar este método para polinomios arbitrarios, es decir, de raíces desconocidas. Lógicamente, en el camino habrá que mencionar qué ocurre con casos especiales donde el método pueda fallar o tornarse ineficiente.

Esta tesis se divide en los siguientes capítulos. En el primer capítulo definimos los conceptos de dinámica compleja necesarios para poder desarrollar el algoritmo. En el segundo capítulo explicamos cómo estas ideas se entrelazan en el algoritmo de factorización que desarrollamos vía ejemplos. En el tercer capítulo proponemos un pseudo-código de las ideas previamente discutidas. En el cuarto capítulo listamos posibles complicaciones derivadas del hecho de que solo hemos atacado polinomios genéricos. Cerramos con un último capítulo con comentarios finales.

Capítulo 1

Preliminares técnicos

En este capítulo nos centraremos en definir conceptos y presentar ejemplos que nos servirán de base para la construcción del algoritmo. Además, estas definiciones permitirán tratar temas de convergencia y velocidad de la misma. Para una explicación más a fondo de estos temas, nos referimos a [CG13], [Poi16], [Lan77].

Sea α un número complejo. En este trabajo, la **órbita** de α, denotada $\mathcal{O}_\alpha$, se define como el conjunto de elementos que son resultado de la iteración $z \mapsto z^2$ iniciada desde $z_0 = \alpha$; alguna veces se presentará este como un conjunto ordenado.

Veamos algunos ejemplos de órbitas. En el capítulo anterior exhibimos tres de ellas: $\mathcal{O}_{-1} = \{-1, 1\}$, $\mathcal{O}_0 = \{0\}$ y $\mathcal{O}_2 = \{2, 4, 16, \dots\}$. Se puede apreciar que algunas órbitas son acotadas, mientras otras, no. Esta última observación resulta extremadamente relevante para nuestro objetivo.

Con los ejemplos anteriores podemos apreciar que la órbita de 0 tiende a 0. Por otro lado, la órbita de -1 mantiene norma 1, mientras que la de -2 tiende a infinito. En realidad, el comportamiento de estas órbitas se puede generalizar a partir de los tres casos aislados ya mencionados. Esto lo enunciaremos en el siguiente teorema.

Teorema 1.1. *Si α tiene norma menor que 1, entonces $\mathcal{O}_\alpha$ se mantiene acotado por 1. Si α tiene norma 1, entonces los elementos de $\mathcal{O}_\alpha$ tienen todos norma 1. Finalmente, si $|\alpha| > 1$, entonces los elementos de $\mathcal{O}_\alpha$ tienden a infinito.*

Prueba. Sea α el número complejo en cuestión.

Si $|\alpha|$ es menor a 1, de $0 \leq |\alpha| < 1$ se pasa a $0 \leq |\alpha|^n < 1^n = 1$. En efecto, claramente se tiene $|\alpha|^n > |\alpha|^{n+1} > \dots$, y con ello obtenemos una sucesión decreciente, por tanto convergente. De la propiedad arquimideana de los reales se concluye que el límite solo puede ser 0.

Si $|\alpha| = 1$, se tiene $|\alpha|^n = 1$; esta vez, sin embargo, la secuencia no tiene por qué tener límite.

Si $|\alpha| > 1$, el proceso de iteración genera una sucesión creciente (en valor absoluto) sin cota superior. □

Enunciamos un corolario inmediato que nos será de utilidad al desarrollar el algoritmo.

Teorema 1.2. *Un número α es acotado por 1 si y solo si su órbita $\mathcal{O}_\alpha$ es acotada por 1.* □

Denotaremos como $\overline{\mathbb{D}}$ al conjunto de los $x \in \mathbb{C}$ acotados por 1.

Además de las propiedades ya mencionadas, es importante resaltar que $\overline{\mathbb{D}}$ es conexo y compacto.

Para nosotros, más importante que $\overline{\mathbb{D}}$, es su complemento. En ese sentido, el teorema 1.2 es de gran interés, ya que en este proceso de iteración no hay manera de salir de la bola de radio 1 para luego retornar.

Ahora notemos un detalle interesante. Dada la naturaleza de la iteración tenemos que el k-ésimo iterado de α tiene forma explícita α^{2^k}, y si α no pertenece a $\overline{\mathbb{D}}$ ello es síntoma de que α se aleja a ritmo exponencial del círculo unitario.

Este comentario apunta al hecho de que más que la "coordenada de escape", nos interesa la lejanía de un punto respecto a $\mathbb{D}$. Ello tendrá el efecto de que, de tener dos órbitas a mano, sabremos cuál crece más rápido que la otra.

Para evitar inconvenientes notacionales, recurriremos a $\phi(\alpha)$ para referirnos al valor absoluto (en principio desconocido) de las raíces del polinomio p cuyo meta es factorizar. (El valor absoluto explícito $|\cdot|$ será utilizado para otros fines.) De este modo, la función $\phi(\cdot)$ es estrictamente mayor que 1 para elementos fuera de $\overline{\mathbb{D}}$.

Durante este trabajo denotaremos como $f^{\circ k}(\alpha)$ a la k-ésima iteración de α. Normalmente lo usaremos en conjunción con ϕ con fines comparativos. Los tecnicismos se detallan a continuación.

Lema 1.3. *Si se cumple $\phi(\alpha) > \phi(\beta)$, entonces existe $\lambda < 1$ tal que, para k grande, el cociente $\dfrac{f^{\circ k}(\beta)}{f^{\circ k}(\alpha)}$ es menor que λ^{2^k}.*

Prueba. Si $\beta = 0$, se cumple de manera trivial la conclusión. De otro modo, de la hipótesis podemos escoger $r > 1$ sujeto a $\left|\dfrac{\phi(\alpha)}{\phi(\beta)}\right| > r > 1$. Con ello, concluimos $f^{\circ k}(\alpha)/f^{\circ k}(\beta) = |\alpha/\beta|^{2^k} > r^{2^k} > 1$. Para confirmar el lema basta elegir $\lambda = 1/r$. □

Capítulo 2

Discusión mediante ejemplos

Digamos que tenemos un polinomio arbitrario $p(x)$ de grado n que queremos factorizar. Como se hizo en la introducción, empezaremos la iteración de todas sus raíces $c_1, \ldots, c_n$ de manera simultánea. En esencia, en este proceso pueden suceder tres cosas. Puede existir una raíz que lidera la iteración (su $\phi(c)$ es mayor a las demás), en cuyo caso la llamaremos **raíz liebre**; pueden existir múltiples raíces que lideran (existe un empate en el máximo ϕ, por ejemplo en el caso de raíces complejas conjugadas) y, por último, ninguna raíz escapa (todas las raíces permanecen en $\overline{\mathbb{D}}$).

En principio, el algoritmo se centrará en hacer que se esfume la raíz líder (al final su valor se recuperará mediante métodos algebraicos elementales). En caso se desee factorizar el polinomio completamente se aplicará el algoritmo sucesivas veces, una vez por raíz o par de raíces hasta que al final quede una.

Empezaremos con el caso de una raíz líder, seguiremos con todas las raíces en $\overline{\mathbb{D}}$ y finalizaremos con el caso de varias raíces con un mismo equipotencial (con mismo $\phi(\cdot)$) mayor a 1.

2.1. Una raíz liebre

Empecemos con un ejemplo: tomemos el polinomio $p(x) = x^3 - 29.0443x^2 + 81.30831522x + 1133.2465629685$. Desplegaremos en cada estadío del proceso el polinomio $f^{\circ k}(x)$, resultado de la iteración; sus coeficientes están señalados como b_{kj}. Por temas de argumentación, exhibiremos además el múltiplo mónico de $f^{\circ k}(x)$. Este polinomio será denotado por $\hat{f^{\circ k}}(x)$; fruto de esta normalización, nos referiremos a sus coeficientes por d_{kj}.

Cantidad de iteraciones	$f^{\circ k}(x)$	$\hat{f}^{\circ k}(x)$
1	$b_{10} = 0$ $b_{11} = 1$ $b_{12} = 0$	$d_{10} = 0$ $d_{11} = 1$ $d_{12} = 0$
2	$b_{20} = 0$ $b_{21} = 0$ $b_{22} = 1$	$d_{20} = 0$ $d_{21} = 0$ $d_{22} = 1$
3	$b_{30} = -32914.353148826005$ $b_{31} = -3494.789662712746$ $b_{32} = 762.26304727$	$d_{30} = -43.1797832345$ $d_{31} = -4.5847554532$ $d_{32} = 1$
4	$b_{40} = -12003542006.017382$ $b_{41} = -1367370000.9030244$ $b_{42} = 250198792.25954336$	$d_{40} = -47.9760189792$ $d_{41} = -5.465134298$ $d_{42} = 1$
5	$b_{50} = -1.1409351502437125 * 10^{21}$ $b_{51} = -1.3031181570888432 * 10^{20}$ $b_{52} = 2.370745181713784 * 10^{19}$	$d_{50} = -48.1255918622$ $d_{51} = -5.4966605738$ $d_{52} = 1$
6	$b_{60} = -1.0195542032508078 * 10^{43}$ $b_{61} = -1.1644868627192711 * 10^{42}$ $b_{62} = 2.118519953303474 * 10^{41}$	$d_{60} = -48.1257776997$ $d_{61} = -5.4966999999$ $d_{62} = 1$

Acá los términos b_{kj} crecen de manera exponencial, detalle que no debe sorprendernos. Como todas las raíces se modifican a ritmo cuadrático, este comportamiento es totalmente estándar.

Esta forma de presentar los términos, más que darnos una sensación de convergencia, nos abruma de tal modo que despierta una ligera preocupación pues crecen sin dar señal de control. Ello representa una motivación extra para darle un vistazo a los coeficientes de $\hat{f}^{\circ k}$.

Los d_{kj} alcanzan rápidamente estabilidad. Puesto que el polinomio $\hat{f^{\circ k}}$ es mónico, el valor $-d_{k1}$ es la suma de ciertas raíces (aquellas que no se perdieron en el proceso de iteración), en contraste con 29.0443, la suma de las raíces del polinomio original. La diferencia $29.0443 - d_{k1} = 23.5476000001$ debe ser entonces el valor de la raíz de mayor escape. Y, en efecto, al reemplazar este valor en el polinomio original obtenemos una insignificancia de orden 10^{-8}.

Notemos que para que este argumento funcione debemos estar seguros de que las demás raíces no se desvanezcan. Esta será principal preocupación en el resto de la sección.

Nuestro primer paso ha sido ver cómo funciona el algoritmo en la práctica. Ahora expliquemos el porqué. El primer resultado técnico será mostrar la equivalencia entre tener $\phi(c_i) > 1$ al menos para una raíz y que los coeficientes b_{kj} no sean acotados.

Proposición 2.1. *Sea $p(x)$ un polinomio de grado n con coeficientes reales. Entonces $p(x)$ tiene al menos una raíz sujeta a $\phi(c_i) > 1$ si y sólo si los b_{kj} no son acotados.*

Prueba. Asumamos que c_1 tiene norma mayor que 1. Por contradicción, asumamos que los b_{kj} están acotados uniformemente por una constante B. Bajos estas condiciones la suma $|\sum_j b_{kj} \cdot c_i^j|$

es menor o igual que $|\sum_j B \cdot c_i^j|$. Pero esto no es el caso para los iterados de c_1: divergen en norma a ∞ y no pueden estar acotados.

Recíprocamente, asumamos que todas las raíces son, en norma, menores o iguales a 1. De ser así, se cumplirá persistentemente $|f^{\circ k}(c_i)| = |\sum_j b_{kj} \cdot c_i^j| \leq 1$. Por el absurdo, supongamos que los b_{kj} no estén acotados. Para cada k designemos por j_k a una posición con la cual se tenga $|b_{kj}| \leq |b_{kj_k}|$. Como los b_{jk} no son acotados, pasamos a subsecuencias para asumir $|b_{k_\ell j}| \leq |b_{k_\ell j_0}| \to \infty$, con un índice j_0 fijo. Tras refinar, incluso se puede asumir la convergencia $b_{k_\ell j}/b_{k_\ell j_0} \to \beta_j$ donde $|\beta_j| \leq 1$ pero $\beta_{j_0} = 1$. De este modo el polinomio $T(x) = \sum_{j=0}^{n-1} \beta_j x^j$ será no nulo y tendrá grado menor o igual a $n-1$. No obstante, todos los c_i (y son n en total) son raíces de T: en efecto, todo lo dicho anteriormente desemboca en

$$|T(c_i)| = \lim |f^{\circ k_\ell}(c_i)|/|\beta_{k_\ell j_0}| \leq \lim 1/|\beta_{k_\ell j_0}| = 0.$$

Esto está mal ya que un polinomio de grado $n-1$ tiene a lo mucho $n-1$ raíces. □

Obsérvese que en la práctica hacemos un supuesto fuerte al afirmar que tras iterar el polinomio $\hat{f}^{\circ k}$ mantiene todas las raíces —excepto la liebre (con cierto margen en la aproximación)—. ¿Cómo es esto posible y cómo es ello rentable?

Introducimos un polinomio auxiliar $F^{[k]}$ como $f^{\circ k}/w_k$, donde $w_k = \max_j |b_{kj}|$. Es decir, todos los coeficientes de $F^{[k]}$ son menores o iguales a 1 en norma. En realidad $f^{\circ k}(x)$, $\hat{f}^{\circ k}(x)$ y $F^{[k]}(x)$ son múltiplos uno de otro, por lo que comparten las mismas raíces. Por ello, su aparición tendrá en muchos casos apenas un efecto argumentativo.

Para cada una de las raíces de $p(x)$ se tiene

$$|F^{[k]}(c_i)| = \left| \sum_{0 \leq j \leq n-1} \frac{b_{kj}}{w_k} c_i^j \right| \leq \sum |c_i^j|,$$

por lo que resultan acotados (independiente de k e i) digamos por una constante Q.

Proposición 2.2. *Sea c_1 una raíz liebre de $p(x)$ y c_i una raíz que no lo es. Entonces c_i es raíz (aproximada) de $\lim F^{[k]}(x)$.*

Prueba. Por hipótesis $\phi(c_1)$ es mayor que $\phi(c_i)$. Merced al teorema 2.3 para cierto $\lambda < 1$ se tiene $|f^{\circ k}(c_i)| \leq \lambda^{2^k} |f^{\circ k}(c_1)|$, para cualquier k. Al dividir entre $|w_k|$ obtenemos

$$|F^{[k]}(c_i)| \leq \lambda^{2^k} |F^{[k]}(c_1)| \leq \lambda^{2^k} Q \to 0,$$

lo que establece el resultado. □

Una consecuencia inmediata es que el método resulta infalible en tanto exista una única raíz liebre. Es más, nada impide, una vez encontrado un factor, volver a aplicar el mismo algoritmo.

La proposición 3.1 no solo funciona cuando existe una raíz liebre, sino cuando son varias. La principal dificultad radica en que no podemos concluir nada contundente acerca del grado de $F^{[k]}$; ello lo discutiremos en secciones posteriores.

Otra cuestión importante es el tema de la velocidad de convergencia. Tal como señala la prueba, ésta depende de dos ingredientes, de Q y de λ. Por un lado, Q depende del grado del polinomio y de la magnitud de la raíz líder, mientras λ es el radio entre dos raíces. No obstante, una vez que sabemos que ella es menor que 1, la magnitud λ^{2^k} decrece exponencialmente. Es por ello que para gran parte de las aplicaciones necesitaremos un número pequeño de iteraciones.

2.2. Todas las raíces dentro del disco unitario

Existe un problema con lo discutido hasta ahora. Si arrancamos la iteración con raíces dentro del disco unitario, por más que iteremos, los coeficientes no crecerán. Y el polinomio iterado no ayudará en nada a factorizar el polinomio original. Acá ya no estamos hablando de la no unicidad de raíces liebre sino de su no existencia: este fenómeno es exclusivo de cuando las raíces caen dentro de $\overline{\mathbb{D}}$ pues, recuérdese, apenas haya una raíz fuera, por lo visto en la sección anterior, los términos se disparan a infinito.

Aparece entonces cierta duda de cómo detectar cuándo estamos en este caso. ¿Cómo poder distinguir entre coeficientes que no crecen y coeficientes que crecen a paso lento? Obsérvese que ello es una situación no del todo disparatada: es muy difícil decidir a ciencia cierta si un punto pertenece a compacto o si solo está muy cerca de él. La respuesta tajante, en este caso, es que no nos importa diferenciar esos dos casos: o bien el método no funciona, o, en su defecto, demorará tanto que no otorgará beneficios. Y en ningún caso nos conviene utilizarlo.

Tras esta discusión, cuando veamos que los coeficientes del polinomio luego de un número razonable de iteraciones (por ejemplo, 10 de ellas) permanecen acotados (la constante la decide el usuario y puede cambiar de acuerdo con la aplicación: nosotros tomaremos $3 + d$, donde d es el grado), entonces las raíces se encuentran cerca o dentro del disco unitario, y abortamos.

Para continuar adelante y lograr el objetivo trazado (es decir, encontrar al menos una raíz), desplazamos todas las raíces una cantidad k en el eje X (por ejemplo, 3 unidades reales). Esto es sencillo de lograr en un ordenador: se evalúa $p(x - k)$. Ahora todas las raíces están lejos del disco unitario, ya que en principio todas arrojaban parte real entre -1 y 1 al moverla 3 unidades escapan del disco. Con ello volvemos a recobrar la fe en nuestro algoritmo, y de tener éxito, le sumamos a las raíces así obtenidas la cantidad desplazada para recuperar las soluciones buscadas.

Veamos esto en marcha. Consideremos el polinomio $p(x) = x^3 + 1.176x^2 + 0.05521959x - 0.1681968153$. Este polinomio tiene raíces -0.579, 0.318 y -0.915, todas dentro del disco unitario. El proceso de iteración entrega los valores consignados en la siguiente tabla. En este caso mostramos exclusivamente los $f^{\circ k}(x)$ dado que tendremos iterados acotados (ver proposición 3.1).

Cantidad de iteraciones	$f^{\circ k}(x)$
1	$b_{1,0} = 0$ $b_{1,1} = 1$ $b_{1,2} = 0$
2	$b_{2,0} = 0$ $b_{2,1} = 0$ $b_{2,2} = 1$
3	$b_{3,0} = -0.1977994547928$ $b_{3,1} = 0.23313505314$ $b_{3,2} = 1.32775641$
4	$b_{4,0} = -0.20545387822885036$ $b_{4,1} = 0.2845883896304756$ $b_{4,2} = 1.1417904671642711$
5	$b_{5,0} = -0.10634912856198096$ $b_{5,1} = 0.15110906341730285$ $b_{5,2} = 0.5785371229930102$
6	$b_{6,0} = -0.025486097263225455$ $b_{6,1} = 0.036236044396915044$ $b_{6,2} = 0.13857006862130614$
7	$b_{7,0} = -0.0014594273207892132$ $b_{7,1} = 0.002075009582212664$ $b_{7,2} = 0.007935027084502593$
8	$b_{8,0} = -4.7856386320795905 * 10^{-6}$ $b_{8,1} = 6.80420729221665 * 10^{-6}$ $b_{8,2} = 2.601991316816804 * 10^{-5}$
9	$b_{9,0} = -5.1458220130117115 * 10^{-11}$ $b_{9,1} = 7.316314990998036 * 10^{-11}$ $b_{9,2} = 2.797826000899468 * 10^{-10}$
10	$b_{10,0} = -5.9495549271149536e * 10^{-21}$ $b_{10,1} = 8.459060144900177 * 10^{-21}$ $b_{10,2} = 3.2348222357421576 * 10^{-20}$

Observamos que nuestros esfuerzos por intentar factorizar este polinomio son infructuosos. Puesto que los términos son acotados, concluimos (incluso sin conocer las raíces) que debemos realizar un desplazamiento en el eje x.

Por lo tanto, en lugar de usar el polinomio original, usaremos $\tilde{p}(x) = p(x - 3)$.

Cantidad de iteraciones	$f^{\circ k}(x)$	$\hat{f}^{\circ k}(x)$
1	$b_{1,0} = 0$ $b_{1,1} = 1$ $b_{1,2} = 0$	$d_{10} = 0$ $d_{11} = 1$ $d_{12} = 0$
2	$b_{2,0} = 0$ $b_{2,1} = 0$ $b_{2,2} = 1$	$d_{20} = 0$ $d_{21} = 0$ $d_{22} = 1$
3	$b_{3,0} = 382.4289568510272$ $b_{3,1} = 309.53430859314$ $b_{3,2} = 69.43975641$	$d_{30} = 5.5073487671$ $d_{31} = 4.4575949657$ $d_{32} = 1$
4	$b_{4,0} = 374727.4490651815$ $b_{4,1} = 262914.7661931612$ $b_{4,2} = 46307.568893401476$	$d_{40} = 8.0921425594$ $d_{41} = 5.6775765275$ $d_{42} = 1$
5	$b_{5,0} = 45393487071.0396$ $b_{5,1} = 30200972710.011127$ $b_{5,2} = 4951139774.518314$	$d_{50} = 9.1682903611$ $d_{51} = 6.099802083$ $d_{52} = 1$
6	$b_{6,0} = 1.9629524111953163 * 10^{20}$ $b_{6,1} = 1.2846314870558407 * 10^{20}$ $b_{6,2} = 2.0609979267957367 * 10^{19}$	$d_{60} = 9.5242813478$ $d_{61} = 6.2330556977$ $d_{62} = 1$

Al hacer la comparación con $\tilde{p}(x) = x^3 + 10.176x^2 + 34.11121959x + 37.5814619547$, de todo lo aprendido a lo largo de este capítulo se concluye que la primera raíz es $10.176 - 6.2614951688 = 3.942945$. No obstante, a esta hay que sustraerle las 3 unidades "prestadas" a fin de recuperar la raíz original: el resultado de esto es 0.942944, una aproximación aceptable a una raíz del polinomio p. Notemos además que esta es la raíz "más expuesta" al desplazar tres unidades.

Como nota final, recordemos el protagonismo de este paso en el contexto general. El algoritmo pretende extraer las raíces, de una en una, con los $\phi(c_i)$ mayores que 1, y poco a poco las raíces faltantes están cada vez más cercanas a $\overline{\mathbb{D}}$. En fin, este paso se circunscribe exclusivamente a cuando todas las raíces caen dentro (o cerca) del disco. Por tanto, cuando se haga indispensable este desplazamiento, no habrá raíces que pretendan filtrarse en este nuestro disco problemático.

2.3. Raíces conjugadas

Ahora digamos que no hay una raíz líder, sino 2. En el caso real esto puede suceder genéricamente si el polinomio tiene raíces conjugadas. Al aplicar el método explicado en la sección 3.1 y reemplazar la supuesta raíz en el polinomio original, no obtendremos el resultado deseado. En realidad, muchas veces incluso no habrá convergencia pese a que los coeficientes crezcan desmesuradamente.

Veamos un ejemplo. Tomemos $p(x) = x^4+2x^3+11x^2-2x+33$ y arranquemos con las iteraciones.

Cantidad de iteraciones	$f^{\circ k}(x)$	$\hat{f}^{\circ k}(x)$
1	$b_{10} = 0$ $b_{11} = 1$ $b_{12} = 0$	$d_{10} = 0$ $d_{11} = 1$ $d_{12} = 0$
2	$b_{20} = 0$ $b_{21} = 0$ $b_{22} = 1$	$d_{20} = 0$ $d_{21} = 0$ $d_{22} = 1$
3	$b_{30} = -33$ $b_{30} = 2$ $b_{31} = -11$ $b_{32} = -2$	$d_{30} = 16.5$ $d_{31} = -1$ $d_{32} = 5.5$ $d_{33} = 1$
4	$b_{40} = 1188$ $b_{41} = -1326$ $b_{42} = 703$ $b_{43} = -294$	$d_{40} = -4.0408163265$ $d_{41} = 4.5102040816$ $d_{42} = -2.3911564626$ $d_{43} = 1$
5	$b_{50} = -47942565$ $b_{51} = 19186358$ $b_{52} = -17047559$ $b_{53} = 1067422$	$d_{50} = -44.9143497136$ $d_{51} = 17.9744824446$ $d_{52} = -15.9707772559$ $d_{53} = 1$
6	$b_{60} = -1.0782419372366796 * 10^{16}$ $b_{61} = 229293657627546$ $b_{62} = -2472524404136873$ $b_{63} = -1121615372556606$	$d_{60} = 9.6132949282$ $d_{61} = -0.2044316289$ $d_{62} = 2.2044316302$ $d_{63} = 1$
7	$b_{70} = 5.881610454899393 * 10^{32}$ $b_{71} = -1.335479319449419 * 10^{32}$ $b_{72} = 1.752184318683005 * 10^{32}$ $b_{73} = 2.083524996167929 * 10^{31}$	$d_{70} = 28.2291331552$ $d_{71} = -6.4097110517$ $d_{72} = 8.4097110517$ $d_{73} = 1$
8	$b_{80} = 9.860313481883566 * 10^{64}$ $b_{81} = -3.5440207277862266 * 10^{65}$ $b_{82} = 1.4004583199705992 * 10^{65}$ $b_{83} = -1.0717812039078138 * 10^{65}$	$d_{80} = -0.9199931335$ $d_{81} = 3.3066643778$ $d_{82} = -1.3066643778$ $d_{83} = 1$

Al haber un par de raíces que se escapan a infinito, deberíamos observar polinomios residuales de grado $2 = n-2$ y no de grado 3 como aparece en la tabla. Los valores tabulados sugieren que de una manera u otra se filtran falsas raíces. Estos polinomios, del grado incorrecto, son estructuralmente distintos uno del otro y, por consiguiente, sus raíces no pueden ser todas iguales. En efecto, en cada paso dos son recurrentes, lo que convierte a la tercera en esporádica. Hemos desembocado en una situación similar a la descrita a continuación.

Lema 2.3. *Sean U, V polinomios mónicos distintos de grado $r+1$. Si U, V comparten un factor*

común de grado r, entonces éste es un múltiplo de $U - V$.

Prueba. Sea $R(x)$ el divisor cómun de U, V de grado r. En $\mathbb{C}$ toda raíz de R es cero de $U(x) - V(x)$ y por tanto, al ser del mismo grado son uno múltiplo del otro. □

En nuestro caso, para deshacernos de la raíz intrusa tomamos la diferencia entre las dos últimas iteraciones. Con ello obtenemos $q(x) = -9.7163754295x^2 + 9.7163754295x - 29.149$. Al normalizar, obtenemos el polinomio $\hat{q}(x) = x^2 - x + 2.915$.

Demos un paso atrás para reparar en qué hemos obtenido. Este último polinomio es lo que nos queda después del proceso de iteración. Hemos de recuperar las raíces que hemos perdido en primer lugar. Para ello, debemos dividir el polinomio original con $\hat{q}(x)$. Esto nos da $x^2 + 3x + 11$ aproximadamente. De este polinomio podemos recuperar las raíces complejas conjugadas perdidas. Al hacer el cálculo obtenemos $-1.5 + 2.958i$ y $-1.5 - 2.958i$.

2.4. Factorización completa

Finalmente, mostraremos un ejemplo en el que se unifican las secciones previas. Mostraremos cómo se utiliza el algoritmo para factorizar un polinomio en su totalidad. Analizaremos el polinomio $p(x) = x^4 - 3.5x^3 - 1.5x^2 - 2.5x + 2$.

Procedemos mecánicamente y conseguimos una potencial raíz al restar los términos $d_{62} - a_{n-1}$, esto arroja 4. Y, efectivamente, al reemplazar este número en el polinomio original comprobamos que es raíz.

Cantidad de iteraciones	$f^{\circ k}(x)$	$\hat{f}^{\circ k}(x)$
1	$b_{10} = 0$ $b_{11} = 1$ $b_{12} = 0$	$d_{10} = 0$ $d_{11} = 1$ $d_{12} = 0$
2	$b_{20} = 0$ $b_{21} = 0$ $b_{22} = 1$	$d_{20} = 0$ $d_{21} = 0$ $d_{22} = 1$
3	$b_{30} = -2$ $b_{30} = 2.5$ $b_{31} = 1.5$ $b_{32} = 3.5$	$d_{30} = -0.5714285714$ $d_{31} = 0.7142857143$ $d_{32} = 0.4285714286$ $d_{33} = 1$
4	$b_{40} = -445.875$ $b_{41} = 445.59375$ $b_{42} = 446.59375$ $b_{43} = 891.46875$	$d_{40} = -0.5001577453$ $d_{41} = 0.4998422547$ $d_{42} = 0.500963999$ $d_{43} = 1$
5	$b_{50} = -29217464.89794922$ $b_{51} = 29217465.326538086$ $b_{52} = 29217464.326538086$ $b_{53} = 58434929.224487305$	$d_{50} = -0.5000000049$ $d_{51} = 0.5000000122$ $d_{52} = 0.4999999951$ $d_{53} = 1$
6	$b_{60} = -1.2548805492319422 * 10^{17}$ $b_{61} = 1.2548805492319422 * 10^{17}$ $b_{62} = 1.2548805492319422 * 10^{17}$ $b_{63} = 2.5097610984638845 * 10^{17}$	$d_{60} = -0.5$ $d_{61} = 0.5$ $d_{62} = 0.5$ $d_{63} = 1$

Con esto obtenemos un factor. Lógicamente, proseguimos con $\hat{f}^{\circ k}(x)$, pues —por construcción— este incluye todas las raíces restantes de $p(x)$.

En términos algorítmicos, nuestro nuevo $p(x)$ pasa a ser $x^3+0.5x^2+0.5x-0.5$ al que sometemos a la misma rutina. Al inspeccionar unos cuantos valores de $f^{\circ k}(x)$, comprobamos que estos están acotados.

Cantidad de iteraciones	$f^{\circ k}(x)$
1	$b_{10} = 0$ $b_{11} = 1$ $b_{12} = 0$
2	$b_{20} = 0$ $b_{21} = 0$ $b_{22} = 1$
3	$b_{30} = -0.25$ $b_{31} = 0.75$ $b_{32} = -0.25$
4	$b_{40} = -0.140625$ $b_{42} = -0.140625$ $b_{41} = 0.859375$
5	$b_{50} = -0.28570556640625$ $b_{52} = 0.71429443359375$ $b_{51} = -0.28570556640625$
6	$b_{60} = -0.14285714272409678$ $b_{62} = -0.14285714272409678$ $b_{61} = 0.8571428572759032$

Con observar los términos $f^{\circ k}(x)$, intuimos que hemos aterrizado en el caso 2, es decir, cuando todas las raíces se ubican en $\overline{\mathbb{D}}$. Desplazamos entonces, con lo que obtenemos $\tilde{p}(x) = p(x-3) = x^3 + 9.5x^2 + 30.5x + 32.5$. Procedemos nuevamente a iterar.

Cantidad de iteraciones	$f^{\circ k}(x)$	$\hat{f}^{\circ k}(x)$
1	$b_{10} = 0$ $b_{11} = 1$ $b_{12} = 0$	$d_{10} = 0$ $d_{11} = 1$ $d_{12} = 0$
2	$b_{20} = 0$ $b_{21} = 0$ $b_{22} = 1$	$d_{20} = 0$ $d_{21} = 0$ $d_{22} = 1$
3	$b_{30} = 308.75$ $b_{30} = 257.25$ $b_{31} = 59.75$	$d_{30} = 5.1673640167$ $d_{31} = 4.3054393305$ $d_{32} = 1$
4	$b_{40} = 198488.671875$ $b_{41} = 139638.515625$ $b_{42} = 24341.359375$	$d_{40} = 8.1543790886$ $d_{41} = 5.7366769651$ $d_{42} = 1$
5	$b_{50} = -29217464.89794922$ $b_{51} = 29217465.326538086$ $b_{52} = -16970173.750549316$	$d_{50} = -82.3941080272$ $d_{51} = -30.40276324392$ $d_{52} = 1$
6	$b_{60} = 2.6131111715699953 * 10^{18}$ $b_{61} = 2.0509946459643607 * 10^{18}$ $b_{62} = 4.02300938296283 * 10^{17}$	$d_{60} = 6.4954140615$ $d_{61} = 5.0981602346$ $d_{62} = 1$

Al pretender recuperar una raíz logramos $5.0981602346 - 9.5 = -4.4018397$. Éste candidato no solamente no resuelve ninguno de los problemas iniciales sino que, peor, al ser trasladado las tres unidades de rigor hacia la izquierda, no cae en disco unitario. Acá hay por tanto más de una raíz liebre y debemos buscar un factor cuadrático. Así que tomamos la diferencia de dos términos consecutivos para así obtener $q(x) = 35.50092347852x + 88.8895220887$. Notemos que este polinomio es el remanente que nos queda de iterar. Primero veamos que raíces perdimos en la iteración. Para esto, efectuemos $\tilde{p}(x)/\tilde{q}(x) = x^2 + 7x + 13$. De aquí obtenemos las raíces complejas conjugadas $(-7 + 1.732i)/2$ y $(-7 - 1.732i)/2$. A estas hay que sumarle 3 para terminar el cálculo, lo cual arroja $(-1 + 1.732i)/2$ y $(-1 - 1.732i)/2$.

Volviendo al polinomio residuo. El polinomio $q(x)$ al ser normalizado nos entrega $\tilde{q}(x) = x + 2.503865065$. La raíz de este polinomio es -2.5 aproximadamente, que al devolver la traslación arroja 0.5. Con ello obtenemos una raíz dentro del disco. Finalmente, hemos conseguido la factorización completa del polinomio $p(x)$.

Capítulo 3

El pseudo-código

Ahora nos centramos en reproducir a modo de pseudocódigo lo planteado anteriormente.

Durante este capítulo aludimos a los algoritmos mediante cierta convención para facilitar su puesta en contexto. Por ejemplo, en "eliminarCoeficientePrincipal(q)" empezamos con minúscula y cambiamos a mayúscula para separar palabras. De este modo confiamos quede clara su función. Los paréntesis denotan sobre quién se aplica el algoritmo.

3.1. Polinomios módulo p(x)

La idea para reducir un polinomio modulo $p(x)$ algorítmicamente es la siguiente. Primero consideremos un polinomio mónico $p(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ que genera un ideal principal. y un polinomio $q(x) = b_mx^m + b_{m-1}x^{m-1} + \cdots + b_1x + b_0$ que queremos reducir.

Si el grado de $q(x)$ es menor al de $p(x)$ no es necesario hacer nada. En caso contrario, la relación $p(x) \equiv 0$, permite escribir $x^n = -(a_{n-1}x^{n-1} + \cdots + a_1x + a_0)$. De este modo, vía el reemplazo $b_mx^m = -b_mx^{m-n} * (a_{n-1}x^{n-1} + \cdots + a_1x + a_0)$ hemos reducido el grado de $q(x)$ en al menos 1.

Evidentemente, basta aplicar este procedimiento sucesivas veces hasta lograr un grado menor que el de p. Escribamos esto de manera algorítmica.

Algoritmo `reduccionModulo(`*polinomio q, polinomio p*`)`

```
n = grado(p)
mientras grado(q) ≥ n hacer
    m = grado(q)
    coeficientePrincipal = extraerCoeficientePrincipal(q)
    q = eliminarCoeficientePrincipal(q)
    aux= coeficientePrincipal*polinomio(grado: m-n)*eliminarCoeficientePrincipal(p)
    q = q - aux
fin
devolver q
```

Algoritmo 1: Reducción módulo p(x)

3.2. Iteración de un polinomio

Recordemos que empezamos con el polinomio x, y este lo iteramos en el proceso $x \mapsto x^2$. Luego de realizar una iteración, basta usar el algoritmo de la sección anterior para tener un polinomio módulo $p(x)$.

Algoritmo `iteracionPolinomio(`*entero cantidadIteraciones, polinomio p*`)`

```
q = polinomio(grado:1)
cantidadActual = 0
mientras cantidadActual<cantidadIteraciones hacer
    q = q^2
    q = reduccionModulo(q,p)
    cantidadActual = cantidadActual + 1
fin
devolver q
```

Algoritmo 2: Iteración módulo p(x)

3.3. Traslación del polinomio

El proceso de traslación se reduce a elevar monomios a una potencia y agrupar términos por grados. Si tenemos $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, es suficiente con reemplazar $p(x-3) = a_n(x-3)^n + a_{n-1}(x-3)^{n-1} + \cdots + a_1(x-3) + a_0$.

Algoritmo `traslacionPolinomio(`*polinomio p*`)`

```
auxiliar = polinomio(grado: 1) − 3
p1 = 0
mientras grado(p)≥0 hacer
    p1 = p1 + extraerCoeficientePrincial(p) ∗ auxiliar^grado(p)
    p = eliminarCoeficientePrincipal(p)
fin
devolver p1
```

Algoritmo 3: Traslación del polinómio p(x)

3.4. Recuperación de la raíz perdida

Notemos que este caso se divide en dos. Ya que podemos recuperar raíces reales o raíces complejas.

Primero veamos la situación real. Consideremos el polinomio $p(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_0$ y el polinomio iterado normalizado $q(x) = x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_0$. Para hallar la raíz perdida

basta considerar $b_{n-2} - a_{n-1}$.

Algoritmo `recuperarRaiz(`*polinomio iterado, polinomio p*`)`

```
iterado = Normalizar(iterado)
iterado = eliminarCoeficientePrincipal(iterado)
p = eliminarCoeficientePrincipal(p)
devolver extraerCoeficientePrincipal(iterado) - extraerCoeficientePrincipal(p)
```

Algoritmo 4: Recuperación de la raíz perdida de $p(x)$

Ahora consideremos el caso complejo. Recordemos como lidiar con este caso. Digamos que el iterado tras diversos pasos queda normalizado cual $q_1 = x^{n-1} + b_{1,n-2}x^{n-2} + \cdots + b_{1,0}$, pero que en una instancia anterior es $q_2 = x^{n-1} + b_{2,n-2}x^{n-2} + \cdots + b_{2,0}$. De acuerdo con el lema 3.3 el factor buscado es $q = q_1 - q_2$.

Es posible que nos quede el polinomio 0 (en caso la iteración converja), en cuyo caso consideramos el máximo común divisor entre el polinomio iterado y $p(x)$

Algoritmo `recuperarRaizCompleja(`*polinomio iterado, polinomio p*`)`

```
q1= Normalizar(iterado)
q2 = iteracionPolinomio(9,p)
q2 = Normalizar(q2)
q = q1-q2
si q=0 entonces
    // La iteración converge, debemos usar gcd
    mcdPolinomio = mcd(p,q1)
    devolver (p/mcdPolinomio,mcdPolinomio)
en otro caso
    // Hemos conseguido una factorización
    devolver (p/q,q)
```

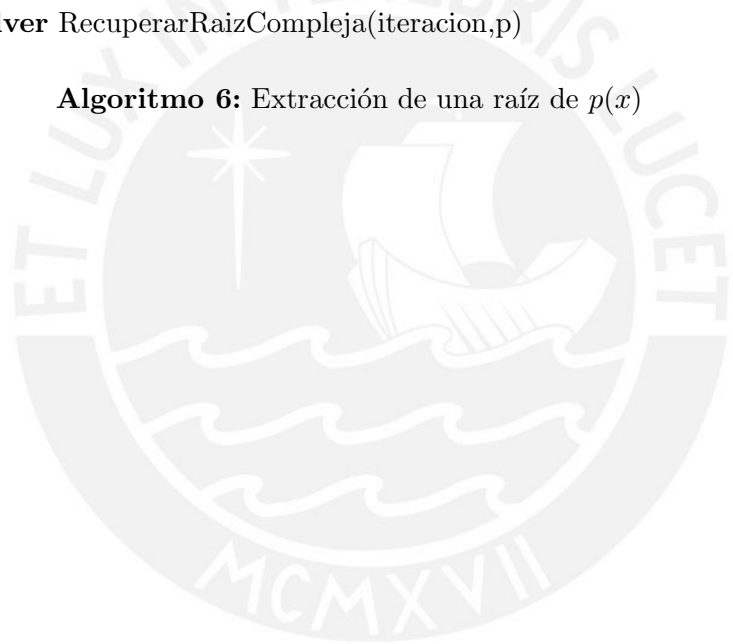
Algoritmo 5: Recuperación de la raíz perdida de $p(x)$

3.5. Extracción de una raíz de p(x)

Finalmente unificamos todo en nuestra rutina principal.

```
Algoritmo extraccionRaiz(polinomio p)
    iteracion = iteracionPolinomio(10,p)
    si (terminosAcotados(iteracion)) entonces
        // Necesitamos una traslación
        p = traslacionPolinomio(p)
        devolver extraccionRaiz(p)
    en otro caso
        c1 = recuperarRaiz(iteracion,p)
        si esRaiz(c1,p) entonces
            // Hemos encontrado una raiz
            devolver (c1,normalizar(iteracion))
        en otro caso
            // Multiplicidad en equipotencial
            devolver RecuperarRaizCompleja(iteracion,p)
    devolver p1
```

Algoritmo 6: Extracción de una raíz de $p(x)$



Capítulo 4

Limitaciones del método

En este capítulo nos centraremos en discutir potenciales problemas en el algoritmo durante su ejecución, así como posibles generalizaciones.

4.1. Multiplicidad en el mismo equipotencial

Recordemos en palabras coloquiales lo que enuncia la proposición 3.2. El polinomio iterado mantiene las raíces internas, mientras que las raíces en el mayor equipotencial —usualmente— se pierden. En esta sección ejemplificaremos cómo la coexistencia de múltiples raíces en el equipotencial dominante puede llegar a ser un problema.

El caso por defecto de esta situación ya lo hemos trabajado en el capítulo 3. Nos referimos a las raíces conjugadas. Esta situación, por lo que se vió, siempre es manejable (ya sea mediante diferencia de polinomios o vía el máximo común divisor). Sin embargo, es preciso recurrir al máximo común divisor, ya que si el polinomio iterado converge, podrían estarse filtrando raíces indeseadas.

Otra instancia del mismo fenómeno ocurre cuando tenemos multiplicidad de raíces. En este caso, el algoritmo podría converger o no. Un caso más extremo que el anterior sucede cuando tenemos múltiples raíces c_1, c_2, ... c_r de $p(x)$ tal que la norma de ellas es la misma. De igual manera que en lo discutido previamente, mientras no sean liebres, no generan ningún problema. Pero apenas lo son, no hemos presentado un análisis de cómo eludir complicaciones. Esto será discutido en la tesis de licenciatura.

4.2. Coeficientes complejos

Por ahora, hemos asumido que los polinomios $p(x)$ tienen coeficientes reales. Notemos que la generalización a coeficientes complejos mantiene la mayoría de propiedades que se cumplen con los reales. Esto debido a que en la mayoría de situaciones únicamente nos preocupamos de la norma de nuestros coeficientes y no en propiedades exclusivas de $\mathbb{R}$.

Sin embargo, hay ligeras diferencias. Por ejemplo, que la raíz liebre no sea real no significa necesariamente que tenemos un par conjugado de liebres. Esta propiedad surgió de los coeficientes reales.

En el caso general de un polinomio con coeficientes complejos $q(z)$, las raíces no necesariamente son conjugadas. Evidentemente, será necesario un análisis más profundo.

4.3. Toma de decisiones

Durante el algoritmo mostrado se ha necesitado tomar múltiples decisiones respecto a los parámetros del algoritmo. Por ejemplo, la cantidad de iteraciones, una cota para diferenciar raíces del disco, etcétera.

Estos parámetros deben ser ajustados de acuerdo con las características del polinomio en cuestión y las propiedades del algoritmo. Por ejemplo, hemos visto que debido al comportamiento cuadrático de la iteración, es suficiente con tomar 10 iteraciones para la gran mayoría de aplicaciones de polinomios con raíces relativamente pequeñas en polinomios de grado chico. Para estos ejemplos la cantidad de iteraciones nunca generaró problemas en cuanto a convergencia. Para polinomios de mayor calado, la situación podría ser otra.

4.4. Iteraciones

Elevar al cuadrado ha sido la base fundamental de nuestro proceso iterativo. En teoría, esto no genera mayor problema. Sin embargo, al momento de implementar este algoritmo, este proceso nos puede dar errores de precisión al manejar números grandes. El lector debe escoger estructuras de datos que le permitan almacenar números con gran cantidad de cifras para evitar problemas computacionales.

Lo estándar es esperar coeficientes de hasta 10^{30} (esto debiera estar ocurriendo entre la sétima y novena iteración). Sin embargo, si se necesita mayor cuidado con raíces cercanas entre sí (por ejemplo, si $p(x)$ tiene múltiples raíces distintas cercanas al 0), se puede requerir un número mayor de iteraciones y por tanto, mayor capacidad de almacenamiento.

En este sentido, operaciones básicas como la suma y multiplicación dejan de ser realizadas en tiempo constante y empiezan a tener un costo computacional que depende del tamaño del número.

Capítulo 5

Análisis y comentarios finales

Durante el desarrollo de esta tesis se ha tomado una serie de decisiones para el beneficio de la misma; sin embargo, estas decisiones no tienen por qué ser exclusivamente de esta manera. Veamos un ejemplo.

En el caso de tener todas las raíces dentro del disco unitario, se tomó la decisión de hacer una traslación en el eje real a todas las raíces. Una alternativa a esto es primero extraer todas las raíces que sean igual a 0, para luego considerar el polinomio $p(\frac{1}{x})$. Este nuevo polinomio tendrá sus raíces fuera del disco unitario. Este método es válido, pero tiene el defecto de trabajar con los recíprocos. Como se sabe, computacionalmente, no es deseable trabajar con recíprocos.

Otro ejemplo, que motiva el tema de mi tesis de licenciatura, es incluir un factor aleatorio c a la iteración. Es decir, en lugar de considerar la iteración $x \mapsto x^2$ considerar $x \mapsto x^2 + c$. Esto, como se verá en dicho texto, nos permite utilizar resultados de dinámica compleja en toda su potencia [Tor20].

Bibliografía

[CG13] Lennart Carleson y Theodore W Gamelin. *Complex dynamics*. Springer Science & Business Media, 2013.

[Lan77] S. Lang. *Complex Analysis*. Addison-Wesley, 1977.

[Mil11] John Milnor. *Dynamics in One Complex Variable.* Princeton University Press, Vol. 160, 2011.

[Poi16] A. Poirier. *Iteración de polinomios y funciones racionales*. Pontificia Universidad Católica del Perú, Fondo Editorial, 2016.

[Poi95] A. Poirier. *Approximating square roots*. ProMathematica, Vol IX Nos.11-12, pp. 95-98., 1995.

[Tor20] J. Torres. *Tesis de Licenciatura(en preparación)*. Pontificia Universidad Catolica del Perú, 2020.