INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE FÍSICA Y MATEMÁTICAS

SERVICIO SOCIAL

RODRIGO RIVERA ESTRADA

NOMBRE DEL PROYECTO DE INVESTIGACIÓN:
PROPIEDADES ESPECTRALES DE MATRICES Y OPERADORES DE TOEPLITZ

DIRECTOR DE PROYECTO:
DR. EGOR MAXIMENKO

MÉXICO, D.F.

JUNIO 2012

Índice general

Reporte Global

Justificación

Los problemas computacionales científicos o de ingeniería por lo general se reducen a cálculos con matrices, donde, eventualmente se tiene que resolver un sistema de ecuaciones lineales. La estructura del problema original por lo general resulta en una estructura matricial del sistema de ecuaciones lineales, por lo que se busca diseñar algoritmos que exploten estas estructuras con el objeto de que sean más rápidos que los algoritmos convencionales.

Objetivos

- Estudiar y programar la multiplicación de polinomios vía la Transformada Rápida de Fourier.
- Estudiar y programar los algoritmos asintóticamente rápidos para resolver sistemas de ecuaciones lineales con matrices de Toeplitz.

Marco Teórico

Una aplicación fundamental de la matemática aplicada moderna es la Transformada Discreta de Fourier (TDF). Esta transformación se deduce a partir de la llamada matriz de Fourier. Se implementa mediante un algoritmo llamado Transformada Rápida de Fourier y se aplica en campos tan diversos como por ejemplo el tratamiento de señales en telecomunicación, la solución de ecuacione en derivadas parciales, la multiplicación de números grandes en computación. Entre sus aplicaciones se incluyen métodos numéricos de interpolación y aproximación.

La eliminación de Gauss-Jordan es un algoritmo numérico usado para una gran cantidad de casos específicos, aunque posteriormente se han desarrollado algoritmos alternativos mucho más eficientes. La mayoría de estos algoritmos mejorados tienen una complejidad computacional de $O(n^2)$ (donde n es el número de ecuaciones del sistema). Existen algoritmos nuevos, llamados asintóticamente rápidos o a menudo algoritmos de Toeplitz

superrápidos, que pueden resolver con un costo de $(n \log^p n)$ para varios p. Para los problemas de la forma $\mathbf{Ax} = \mathbf{b}$, donde $\mathbf{A}$ es una matriz de Toeplitz simétrica, se puede utilizar un método derivado de la recursión de Levinson que es la recursión de Schur.

Desarrollo

Se sabe que la Transformada Discreta de Fourier está basada en términos de las raíces de la unidad que se puede ver como funciones de seno y coseno, por lo que se da un repaso a la forma polar de números complejos, además de algunas de sus propiedades.

Se estudió así la Transformada Discreta de Fourier y algunas de sus propiedades, para después estudiar y programar el algoritmo de la Transformada Rápida de Fourier, usando como tema auxiliar el algoritmo de ordenamiento por mezcla con la técnica "divide y vencerás".

Una de las aplicaciones que se estudió de este algoritmo fue la multiplicación rápida de polinomios, al expresar la transformada discreta de Fourier como valores de un polinomio en las raíces de la unidad y después calcular los coeficientes de un polinomio si están dados sus valores en las raíces de la unidad

Por último se estudió el algoritmo de Schur que es utilizado principalmente para resolver sistemas de Toeplitz, ya que produce la factorización LU de la matriz con una complejidad significativamente menor.

Se escribieron apuntes en LaTeX de todos los temas estudiados.

Conlusiones

Al participar en un proyecto de investigación se requiere profesionalismo que se va adquiriendo con la experiencia. Formar parte de un programa de servicio social en éste ámbito, permite trabajar directamente con un profesor investigador lo que brinda experiencia y estándares de calidad sobre los productos finales de una investigación.

La revisión bibliográfica fue parte fundamental en éstas actividades, por lo que queda como un aprendizaje práctico importante para ir desarrollando la habilidad para filtrar las referencias y usarlas como una buena herramienta.

Fue necesaria una amplia gama de conocimientos en el ámbito de las matemáticas para ir cubriendo los objetivos planteados, éstos conocimientos sirvieron como base para desarrollar temas y aprender nuevas cosas a través del planteamiento y solución de ejercicios auxiliares que fueron construyendo temas más complejos.

No sólo conocimientos abstractos fueron necesarios, también fue necesario conocer un lenguaje de programación para implementar los algoritmos estudiados y además comparar su eficacia contra funciones internas de dichos paquetes.

Para la elaboración de los apuntes se utilizó LaTeX, por lo que el uso de éste paquete también contribuye a mi formación profesional como una herramienta útil no sólo en matemáticas sino en una diversidad de áreas.

Ya que el proyecto de investigación Propiedades espectrales de matrices y operadores de Toeplitz es teórico, el beneficio social del trabajo que se desarrolló es apoyar la investigación básica en su nivel teórico, así como la educación de nivel superior al desarrollar apuntes que pueden servir de guía a otros estudiantes y la formación de recursos humanos.

Los apuntes elaborados están publicados en la página del director de proyecto[1] y pueden servir a estudiantes a estudiar la transformada discreta de Fourier, y temas relacionados como el algoritmo de la transformada rápida de Fourier, así como otras aplicaciones.

[1] Dr. Egor Maximenko, `http://esfm.egormaximenko.com/discrete_fourier_transform.html`

Introducción

Una aplicación fundamental de la matemática aplicada moderna es la Transformada Discreta de Fourier (TDF). Esta transformación se deduce a partir de la llamada matriz de Fourier que se estudia en el Capítulo 2. Se implementa mediante un algoritmo llamado Transformada Rápida de Fourier que se estudia en el Capítulo 3, y se aplica en campos tan diversos como por ejemplo el tratamiento de señales en telecomunicación, la solución de ecuaciones en derivadas parciales, la multiplicación de números grandes en computación. Entre sus aplicaciones se incluyen métodos numéricos de interpolación y aproximación.

En el Capítulo 4 se presenta una aplicación de la Transformada Rápida de Fourier en la multiplicación de polinomios. La ventaja principal que se tiene al aplicar esta herramienta es que se reduce el costo computacional de n^2 operaciones a $n \log n$.

Por otra parte, en el Capítulo 5 se estudia un método rápido para resolver sistemas de ecuaciones lineales con matrices de Toeplitz. Una matriz de Toeplitz, denominada así en honor a Otto Toeplitz es una matriz cuadrada tal que todas sus diagonales paralelas a la diagonal principal son constantes. Es decir, es una matriz de la forma $T_n = [t_{k,j}]_{k,j=1}^{n}$ donde $t_{k,j} = t_{k-j}$:

$$T_n = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \dots & t_{-(n-1)} \\ t_1 & t_0 & t_{-1} & & \\ t_2 & t_1 & t_0 & & \vdots \\ \vdots & & & \ddots & \\ t_{n-1} & & & \dots & t_0 \end{bmatrix}$$

El método que se presenta es el algoritmo de Schur que proporciona la factorización LU de la matriz T_n en un sistema que incluye este tipo de matrices.

Capítulo 1

Preliminares

En este capítulo se presentan algunos conceptos básicos que son necesarios para el desarrollo del Capítulo 2 Transformada Discreta de Fourier y del Capítulo 3 Transformada Rápida de Fourier.

1.1. Forma polar de números complejos

1.1.1. Fórmula de Euler

Para todo $\theta \in \mathbb{R}$,

$$\mathrm{e}^{\mathrm{i}\,\theta} = \cos(\theta) + \mathrm{i}\,\mathrm{sen}(\theta). \tag{1.1}$$

para todo número real θ. Donde, e es la base del logaritmo natural, $i = \sqrt{-1}$.

La fórmula puede interpretarse geométricamente como una circunferencia unitaria en el plano complejo, dibujada por la función $\mathrm{e}^{i\theta}$ al variar θ sobre los números reales. Así, θ es el ángulo de una recta que conecta el origen del plano y un punto sobre la circunferencia unitaria, con el eje positivo real, medido en sentido contrario a las manecillas del reloj y en radianes.

1.1.2. Forma polar de números complejos

Para todo $z \in \mathbb{C}$ existe un $r \geq 0$ y un $\theta \in \mathbb{R}$ tales que

$$z = r\,\mathrm{e}^{\mathrm{i}\,\theta}\,.$$

En esta representación, r es el módulo del número complejo y el ángulo θ es el argumento del número complejo. La relación se obtiene de

$$\theta = \arctan\left(\frac{y}{x}\right) = \arctan\left(\frac{\mathrm{Im}(z)}{\mathrm{Re}(z)}\right) = -\arctan\left(\frac{-\,\mathrm{Im}(z)}{\mathrm{Re}(z)}\right)$$

$$\cos\theta = \frac{x}{r}, \qquad \operatorname{sen}\theta = \frac{y}{r} \tag{1.2}$$

Despejando x y y de (1.2) y utilizando la representación binomial $z = a + \mathrm{i}\, b$, tenemos

$$\begin{aligned} z &= r\cos\theta + \mathrm{i}\, r \operatorname{sen}\theta \\ &= r(\cos\theta + \mathrm{i} \operatorname{sen}\theta) \end{aligned}$$

Finalmente, usando (1.1) tenemos que

$$z = r\, \mathrm{e}^{\mathrm{i}\,\theta}\,.$$

1.1.3. Valor absoluto de un número complejo escrito en la forma polar

El valor absoluto, módulo o magnitud de un número complejo z, está dado por la siguiente expresión

$$|z| = \sqrt{zz^*} = \sqrt{\mathrm{Re}^2(z) + \mathrm{Im}^2(z)}.$$

Considerando las coordenadas cartesianas del número complejo z como algún punto en el plano, por el teorema de Pitágoras se puede ver que el valor absoluto de un número complejo coincide con la distancia euclidiana desde el origen del plano a dicho punto.

Si el número complejo está escrito en forma polar $z = r\, \mathrm{e}^{\mathrm{i}\,\theta}$, entonces $|z| = r$.

Se tienen las siguientes propiedades

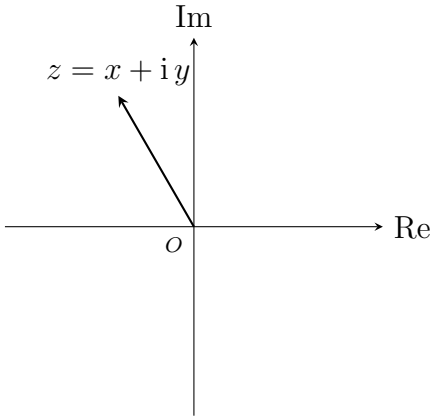
- $|z| = 0 \iff z = 0$
- $|z + \omega| \leq |z| + |\omega|$
- $|z\omega| = |z||\omega|$
- $|z - \omega| \geq |z| - |\omega|$

para cualquier número complejo z y ω.

1.1.4. Sentido geométrico de la forma algebraica de números complejos

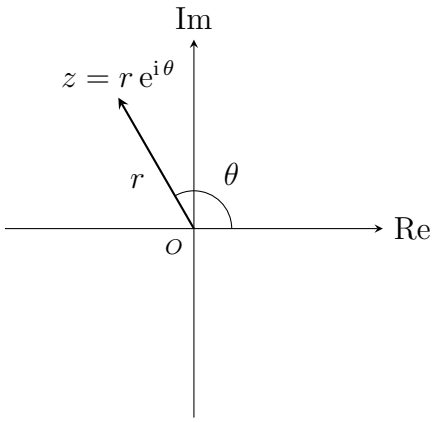
El concepto de plano complejo permite interpretar geométricamente los números complejos. La suma de números complejos se puede relacionar con la suma de vectores, y la multiplicación de números complejos puede expresarse simplemente usando coordenadas polares, donde la magnitud del producto es el producto de las magnitudes de los términos, y el ángulo contado desde el eje real del producto es la suma de los ángulos de los términos.

Recordamos que el número complejo $z = x + \mathrm{i}\, y$ se identifica con el punto $P = (x, y)$ del plano cartesiano y también con el vector $\overrightarrow{OP}$, donde $O = (0, 0)$ es el origen.



1.1.5. Sentido geométrico de la forma polar de números complejos

Sea $z = r\, \mathrm{e}^{\mathrm{i}\,\theta}$, donde $r \geq 0$ y $\theta \in \mathbb{R}$. Entonces r es el módulo del vector determinado por el origen de coordenadas y su afijo, y θ el ángulo que forma el vector con el eje real.



Definición 1.1.1. *Sean $\alpha, \beta \in \mathbb{R}$. Decimos que α* divide *a β y escribimos $\alpha \mid \beta$ si existe un número entero k tal que $\beta = k\alpha$:*

$$\alpha \mid \beta \qquad \overset{def}{\Longleftrightarrow} \qquad \exists k \in \mathbb{Z} \quad \beta = k\alpha.$$

Ejemplos

$$\frac{\pi}{2} \mid \frac{3\pi}{2}, \qquad 5 \nmid 7, \qquad \sqrt{2} \mid \sqrt{8}, \qquad \sqrt{2} \nmid \sqrt{5}.$$

Lema 1.1.1. *Sea $\theta \in \mathbb{R}$. Entonces*

$$\mathrm{e}^{\mathrm{i}\,\theta} = 1 \qquad \Longleftrightarrow \qquad 2\pi \mid \theta.$$

1.1.6. Criterio de la igualdad de números complejos escritos en la forma polar

Teorema 1.1.1. *Sean $r_1, r_2 > 0$ y sean $\theta_1, \theta_2 \in \mathbb{R}$. Entonces*

$$r_1 \, \mathrm{e}^{\mathrm{i}\,\theta_1} = r_2 \, \mathrm{e}^{\mathrm{i}\,\theta_2} \qquad \Longleftrightarrow \qquad \begin{cases} r_1 = r_2, \\ 2\pi \mid \theta_1 - \theta_2. \end{cases}$$

Demostración. $\Longrightarrow$) Por demostrar que si $r_1 \, \mathrm{e}^{\mathrm{i}\,\theta_1} = r_2 \, \mathrm{e}^{\mathrm{i}\,\theta_2}$ entonces, $r_1 = r_2$ ó $2\pi \mid \theta_1 - \theta_2$. Tenemos que: $r_1 \, \mathrm{e}^{\mathrm{i}\,\theta_1} = r_2 \, \mathrm{e}^{\mathrm{i}\,\theta_2}$, como $r_1, r_2 > 0$, entonces

$$\frac{r_1 \, \mathrm{e}^{\mathrm{i}\,\theta_1}}{r_2 \, \mathrm{e}^{\mathrm{i}\,\theta_2}} = 1$$

Es decir:

$$\frac{r_1}{r_2} \frac{\mathrm{e}^{\mathrm{i}\,\theta_1}}{\mathrm{e}^{\mathrm{i}\,\theta_2}} = 1$$

Lo cual se cumple sólo si se tiene que:

1. $\frac{r_1}{r_2} = 1$.
2. $\frac{\mathrm{e}^{\mathrm{i}\,\theta_1}}{\mathrm{e}^{\mathrm{i}\,\theta_2}} = 1$.

□

1.1.7. Multiplicación de números complejos escritos en la forma polar

Sean $r_1, r_2 \geq 0$ y sean $\theta_1, \theta_2 \in \mathbb{R}$. Entonces:

$$\left(r_1 \, \mathrm{e}^{\mathrm{i}\,\theta_1}\right)\left(r_2 \, \mathrm{e}^{\mathrm{i}\,\theta_2}\right) = r_1 r_2 \, \mathrm{e}^{\mathrm{i}(\theta_1+\theta_2)}$$

Fórmula de Moivre

La fórmula de Moivre (en honor a Abraham de Moivre) afirma que para cualquier $\theta \in \mathbb{R}$ y $n \in \{1, 2, 3, \dots\}$, se verifica que:

$$(\cos\theta + \mathrm{i}\,\mathrm{sen}\,\theta)^n = \cos(n\theta) + \mathrm{i}\,\mathrm{sen}(n\theta)$$

Al expandir la parte izquierda la igualdad y comparando la parte real con la imaginaria, es posible deducir expresiones muy útiles para $\cos(n\theta)$ y $\mathrm{sen}(n\theta)$ en términos de $\cos(\theta)$ y $\mathrm{sen}(\theta)$. Además, esta fórmula puede ser utilizada para encontrar expresiones explícitas para la n-ésima raíz de la unidad.

Partiendo de (1.1), con $\theta = \pi$, se tiene que

$$e^{i\pi} = \cos\pi + i\,\mathrm{sen}\,\pi = -1 + 0 = -1$$

Es decir

$$e^{i\pi} = -1$$

Además de las siguientes igualdades

$$\begin{aligned} e^{i\theta} &= \cos\theta + i\,\mathrm{sen}\,\theta \\ e^{-i\theta} &= \cos\theta - i\,\mathrm{sen}\,\theta \end{aligned}$$

se deduce que

$$\begin{aligned} \cos\theta &= \frac{e^{i\theta} + e^{-i\theta}}{2} \\ \mathrm{sen}\,\theta &= \frac{e^{i\theta} - e^{-i\theta}}{2i} \end{aligned}$$

Esta fórmula puede ser utilizada para encontrar la potencia de las raíces n-ésimas de un número complejo escrito en forma polar

$$z^n = [r(\cos\theta + i\,\mathrm{sen}\,\theta)]^n = r^n[\cos(n\theta) + i\,\mathrm{sen}(n\theta)]$$

1.2. Raíces de Unidad

Las raíces n-ésimas de la unidad, son todos los números complejos que resultan 1 cuando son elevados a una potencia dada n. Están localizados en el círculo unitario del plano complejo y en ese plano forman los vértices de un polígono regular de n lados con un vértice sobre 1.

En todos los siguientes ejercicios se supone que $n \in \{1, 2, 3, \ldots\}$.

Notación: ω_n

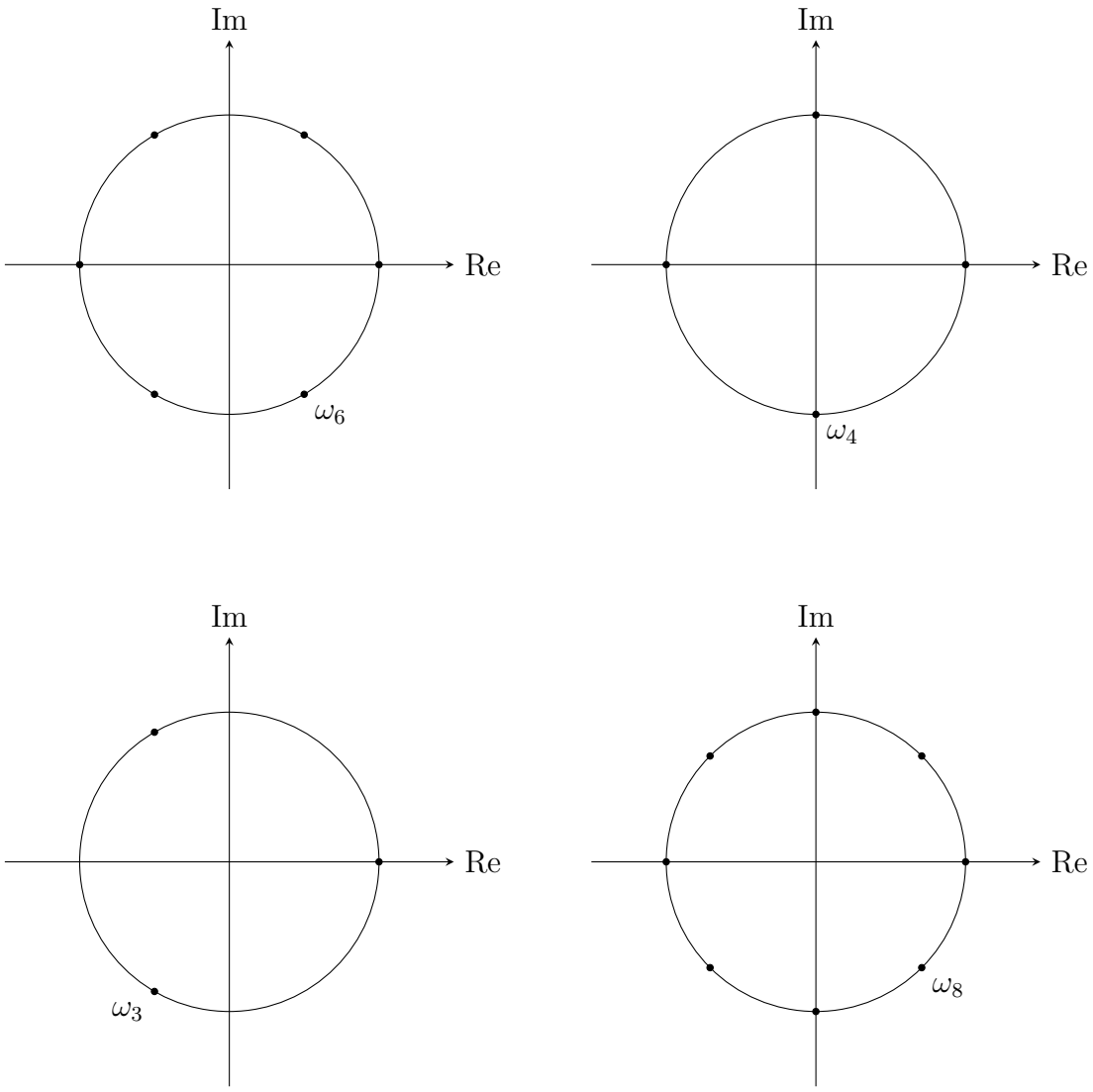
$$\omega_n := e^{-i\frac{2\pi}{n}}\,.$$

1.2.1. Valor absoluto de ω_n

Recuerde el valor de $|\,e^{i\theta}\,|$, donde $\theta \in \mathbb{R}$. Entonces $|\omega_n| = 1$.

Ejemplos

Para cada uno de los números $n = 6, 4, 3, 8$ divida la circunferencia unitaria en n partes iguales (empezando con el ángulo cero) e indique el número ω_n.



Notación: $\alpha\mathbb{Z}$

Sea $\alpha \in \mathbb{R}$. Se denota por $\alpha\mathbb{Z}$ el conjunto de los números enteros que son múltiplos de α:

$$\alpha\mathbb{Z} \coloneqq \big\{\beta \in \mathbb{R}\colon \quad \alpha \mid \beta\big\} = \big\{\beta \in \mathbb{R}\colon \quad \exists k \in \mathbb{Z} \quad \beta = k\alpha\big\}.$$

Ejemplos

Escriba los conjuntos $\alpha\mathbb{Z}$:

$$\sqrt{3}\,\mathbb{Z} = \{\ldots,\ -3\sqrt{3},\ -2\sqrt{3},\ -\sqrt{3},\ 0,\ \sqrt{3},\ 2\sqrt{3},\ 3\sqrt{3},\ \ldots\},$$

$$7\,\mathbb{Z} = \{\ldots,\ -21,\ -14,\ -7,\ 0,\ 7,\ 14,\ 21,\ \ldots\},$$

$$\frac{\pi}{2}\,\mathbb{Z} = \{\ldots,\ -\frac{3\pi}{2},\ -\pi,\ -\frac{\pi}{2},\ 0,\ \pi,\ \frac{\pi}{2},\ \frac{3\pi}{2},\ \ldots\},$$

$$2\pi\,\mathbb{Z} = \{\ldots,\ -6\pi,\ -4\pi,\ -2\pi,\ 0,\ 2\pi,\ 4\pi,\ 6\pi,\ \ldots\}.$$

Lema 1.2.1. *Sea $\theta \in \mathbb{R}$. Entonces*

$$\mathrm{e}^{\mathrm{i}\,\theta} = 1 \qquad \Longleftrightarrow \qquad 2\pi \mid \theta \qquad \Longleftrightarrow \qquad \theta \in 2\pi\mathbb{Z}.$$

1.2.2. Criterio de la igualdad de potencias de ω_n

Sean $m, k \in \mathbb{Z}$. Demuestre que

$$\omega_n^m = \omega_n^k \qquad \Longleftrightarrow \qquad m - k \in n\mathbb{Z}.$$

Demostración.
Supongamos que $\omega_n^m = \omega_n^k$, entonces

$$\frac{\omega_n^m}{\omega_n^k} = 1 \quad \Longrightarrow \quad \omega_n^{m-k} = 1$$

$$\left(e^{-\frac{i2\pi}{N}}\right)^{m-k} = 1$$

$$e^{-\frac{i2\pi(m-k)}{N}} = 1$$

$$\Longrightarrow\ 2\pi|\frac{2\pi(m-k}{N}\ \Longrightarrow \exists n \in \mathbb{Z} \text{ tal que } \frac{2\pi(m-k)}{N} = n2\pi$$

es decir $m - k = nN$, finalmente $m - k \in \mathbb{Z}$. □

n-ésima potencia de ω_n

Se puede demostrar que $\omega_n^n = 1$.

Demostración. Tenemos que

$$\omega_n^n = \left(\mathrm{e}^{-i\frac{2\pi}{n}}\right)^n = \mathrm{e}^{-i2\pi} = 1$$

pues $2\pi| - 2\pi$. □

Lema 1.2.2. *Sean $j, k \in \mathbb{Z}$ tales que*

$$0 \leq j < n, \qquad 0 \leq k < n \qquad y \qquad j - k \in n\,\mathbb{Z}.$$

Demuestre que $j = k$.

Solución.

$$j - k \in n\,\mathbb{Z} \Longleftrightarrow n \mid j - k$$

$$j - k \in \{\ -n+1,\ -n+2, \ldots\ 0,\ 1,\ 2, \ldots\ , n-2,\ n-1\}$$

Es decir

$$l - r = 0 \Longleftrightarrow l = r$$

□

Teorema 1.2.1. *Demuestre que los números ω_n^k, donde $k \in \mathbb{Z}$ y $0 \leq k < n$, son distintos.*

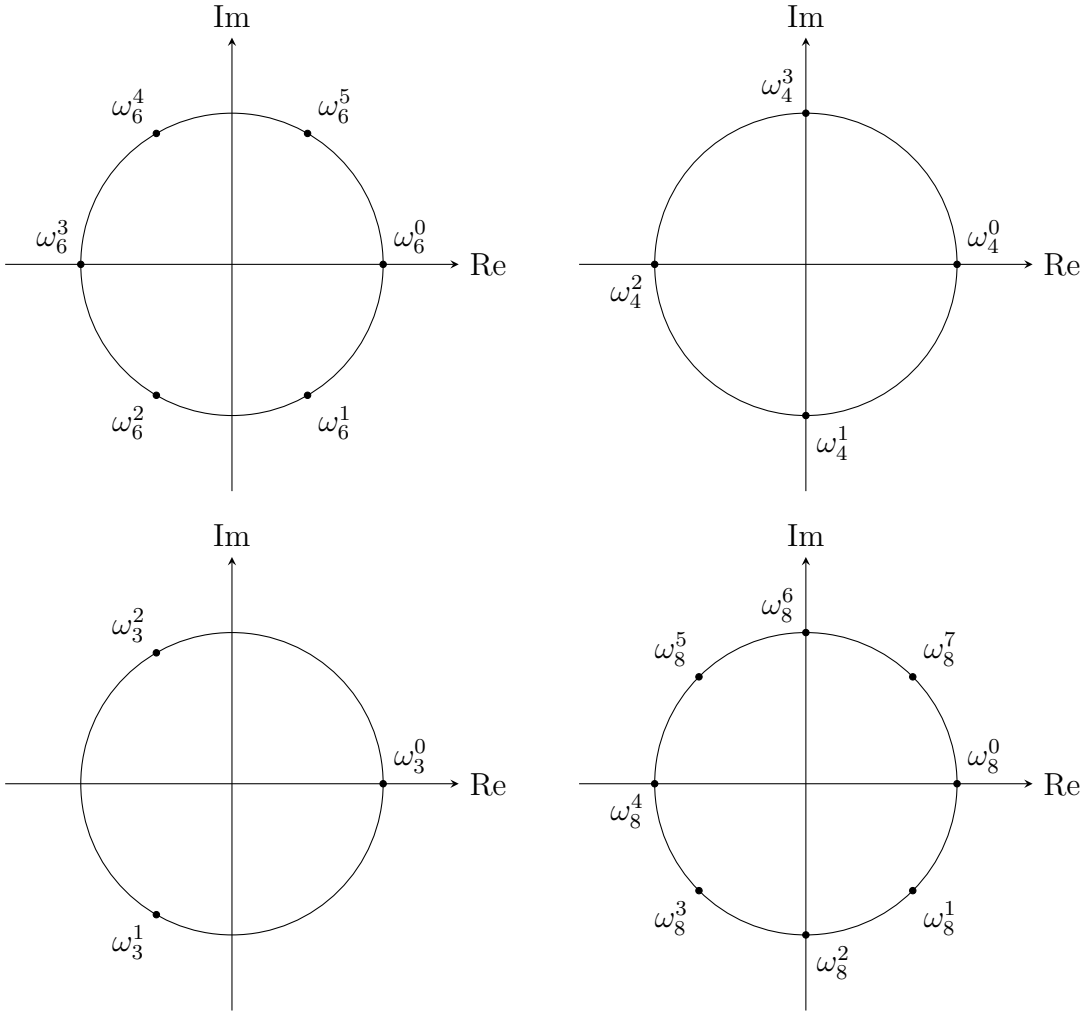
Demostración.
Sea r tal que $r = k + n\ell$ con $\ell \in \mathbb{Z}$. Entonces

$$\begin{aligned} \mathrm{e}^{-\mathrm{i}\,\frac{2r\pi}{n}} &= \mathrm{e}^{-\mathrm{i}\left(\frac{2k\pi}{n} + 2\pi\ell\right)} \\ &= \mathrm{e}^{-\mathrm{i}\,\frac{2k\pi}{n}}\,\mathrm{e}^{-\mathrm{i}\,2\pi\ell} \\ &= \mathrm{e}^{-\mathrm{i}\,\frac{2k\pi}{n}} \cdot 1 = \mathrm{e}^{-\mathrm{i}\,\frac{2k\pi}{n}} \end{aligned}$$

Así, todos los valores ω_n^k, $k \in \mathbb{Z}$ y $0 \leq k < n$ son diferentes. □

1.2.3. Sentido geométrico de las raíces de la unidad

Para cada uno de los números $n = 6, 4, 3, 8$ marque en el plano complejo las raíces de la ecuación $z^n = 1$.



1.3. Suma de las raíces de unidad

Fórmula para las raíces de la unidad

Usando la notación ω_n escriba el conjunto solución de la ecuación $z^n = 1$.

Solución.

$$z^n = \{\omega_n^k : k = 0, \ldots, n-1\}$$

□

1.3.1. Suma de la progresión geométrica (repaso)

Hacia la fórmula de la suma de la progresión geométrica, ejemplo con tres sumandos

Sea $q \in \mathbb{C}$. Calcule el siguiente producto:

$$(1-q)(1+q+q^2).$$

Solución.

$$\begin{array}{ccccccccc} 1 & + & q & + & q^2 & & & & \\ & - & q & - & q^2 & - & q^3 & = & 1-q^3 \end{array}$$

□

Hacia la fórmula de la suma de la progresión geométrica, ejemplo con cuatro sumandos

Sea $q \in \mathbb{C}$. Calcule el siguiente producto:

$$(1-q)(1+q+q^2+q^3) = 1-q^4$$

Hacia la fórmula de la suma de la progresión geométrica

Sea $q \in \mathbb{C}$. Calcule el siguiente producto:

$$(1-q)\sum_{k=0}^{n-1} q^k = (1-q)(1+q+q^2+\ldots+q^{n-1}) = 1-q^n$$

Fórmula de la suma de la progresión geométrica, caso $q \neq 1$

Sea $q \neq 1$. Calcule la siguiente suma:

$$\sum_{k=0}^{n-1} q^k = \frac{1-q^n}{1-q}$$

Fórmula de la suma de la progresión geométrica, caso $q = 1$

Sea $q = 1$. Calcule la siguiente suma:

$$\sum_{k=0}^{n-1} q^k = 1+\ldots+1 = n$$

1.3.2. Fórmula general de la suma de la progresión geométrica

Sea $q \in \mathbb{C}$. Entonces

$$\sum_{k=0}^{n-1} q^k = \begin{cases} \frac{1-q^n}{1-q} & , \text{ si } \quad q \neq 1; \\ 1 & , \text{ si } \quad q = 1. \end{cases}$$

Sumas de las raíces de la unidad

En esta subsección se supone que $n \in \{2, 3, \ldots\}$. El caso $n = 1$ es trivial y se excluye.

Problema 1.3.1. *Suma de todas las raíces de la unidad. Calcular la suma* $\sum_{k=0}^{n-1} \omega_n^k$

Solución.

$$\sum_{k=0}^{n-1} \omega_n^k = \frac{1-\omega_n^n}{1-\omega_n} = \frac{1-1}{1-\omega_n} = \frac{0}{1-\omega_n} = 0$$

□

Problema 1.3.2. *Suma de las potencias de las raíces de la unidad, primer caso Sea* $m \in \mathbb{Z}$ *tal que* $n \mid m$. *Calcule la suma* $\sum_{k=0}^{n-1} \omega_n^{km}$

Solución.

$$\sum_{k=0}^{n-1} \omega_n^{km} = \sum_{k=0}^{n-1} (\omega_n^m)^k = \sum_{k=0}^{n-1} 1^k = n$$

□

Problema 1.3.3. *Suma de las potencias de las raíces de la unidad, segundo caso. Sea* $m \in \mathbb{Z}$ *tal que* $n \nmid m$. *Calcule la suma* $\sum_{k=0}^{n-1} \omega_n^{km}$

Solución.

$$\begin{aligned} \sum_{k=0}^{n-1} \omega_n^{km} = \sum_{k=0}^{n-1} (\omega_n^m)^k &= \frac{1-\omega_n^{mn}}{1-\omega_n^m} \\ &= \frac{1-(\omega_n^n)^m}{1-\omega_n^m} \\ &= \frac{1-1^m}{1-\omega_n^m} \\ &= 0 \end{aligned}$$

□

1.3.3. Suma de las potencias de las raíces de la unidad

Sea $m \in \mathbb{Z}$.

$$\sum_{k=0}^{n-1} \omega_n^{km} = \begin{cases} n, & \text{si} \quad n \mid m \\ 0, & \text{si} \quad n \nmid m \end{cases}$$

1.3.4. Ortogonalidad de las raíces de la unidad

Problema 1.3.4. *Sean $p, q \in \{0, 1, \ldots, n-1\}$, $p \neq q$. Calcule la suma* $\sum_{k=0}^{n-1} \omega_n^{pk} \omega_n^{-qk}$

Solución.

$$\begin{aligned} \sum_{k=0}^{n-1} \omega_n^{pk} \omega_n^{-qk} &= \sum_{k=0}^{n-1} \left(\omega_n^p \omega_n^{-q}\right)^k \\ &= \sum_{k=0}^{n-1} \left(\omega_n^{(p-q)}\right)^k \\ &= \frac{1 - \left(\omega_n^{(p-q)}\right)^n}{1 - \omega_n^{(p-q)}} \\ &= \frac{1 - \left(\omega_n^n\right)^{p-q}}{1 - \omega_n^{(p-q)}} \\ &= \frac{1 - \left(\omega_n^n\right)^{p-q}}{1 - \omega_n^{(p-q)}} \end{aligned}$$

Notemos que:

$$p - q \in \{-(n-1), \ldots, -1, 1, \ldots, n-1\} \Rightarrow n \nmid p - q \Rightarrow \omega_n^{p-q} \neq 1$$

Por lo tanto:

$$\frac{1 - 1^{p-q}}{1 - \omega_n^{(p-q)}} = 0$$

Finalmente:

$$\sum_{k=0}^{n-1} \omega_n^{pk} \omega_n^{-qk} = 0.$$

□

Problema 1.3.5. *Calcule también la suma* $\sum_{k=0}^{n-1} \omega_n^{pk} \omega_n^{-pk}$

Solución.

$$\sum_{k=0}^{n-1} \omega_n^{pk} \omega_n^{-pk} = \sum_{k=0}^{n-1} \left(\omega_n^{p-p}\right)^k = \sum_{k=0}^{n-1} 1^k = n.$$

□

Escriba la fórmula general usando la delta de Kronecker

Solución.

$$\sum_{k=0}^{n-1} \omega_n^{pk} \omega_n^{-qk} = n\delta_{p,q}$$

Donde:

$$\delta_{p,q} = \begin{cases} 1, & \text{si} \quad p = q \\ 0, & \text{si} \quad p \neq q \end{cases}$$

□

1.4. Ordenamiento por mezcla

El algoritmo de ordenamiento por mezcla es un algoritmo de ordenamiento externo estable basado en la técnica divide y vencerás. Es de complejidad $O(n \log n)$.

Conceptualmente el ordenamiento por mezcla funciona de la siguiente manera:

1. Si la longitud de la lista es 0 ó 1, entonces ya está ordenada. En otro caso
   a) Dividir la lista desordenada en dos sublistas de la mitad del tamaño.
   b) Ordenar cada sublista recursivamente aplicando ordenamiento por mezcla.
   c) Mezclar las dos sublistas en una sola lista ordenada.

El ordenamiento por mezcla incorpora dos ideas principales para mejorar su tiempo de ejecución:

1. Una lista pequeña necesitará menos pasos para ordenarse que una lista grande.
2. Se necesitan menos pasos para construir una lista ordenada a partir de dos listas también ordenadas que a partir de dos listas desordenadas. Por ejemplo, sólo será necesario entrelazar cada lista una vez que estén ordenadas.

Primero repasamos el algoritmo trivial de ordenamiento y la búsqueda binaria.

1.4.1. Algoritmo trivial de ordenamiento

Búsqueda del elemento máximo

La función `indmax` busca el índice del elemento máximo en una lista. Si hay varios elementos máximos, entonces debe regresar el índice mayor entre estos.

```
indmax[list_]:=Module[
{im = 1, i},
```

```
For[i = 2, i <= Length[list], i++,
If[list[[im]] <= list[[i]], im = i(*Compara elemento a elemento y guarda el
índice del elemento más alto*) ]
];
im]
```

Algoritmo trivial de ordenamiento

La función `trivialsort` ordena la lista dada de la siguiente manera: primero encuentra el elemento máximo y lo intercambia con el último (n-ésimo) elemento, luego encuentra el elemento máximo entre los primeros $n-1$ elementos y lo intercambia con el elemento en la posición $n-1$, etc.

```
swap[list_, i_, j_] := Module[{list2 = list},
  list2[[i]] = list[[j]]; list2[[j]] = list[[i]]; list2](*Intercambia elementos entr

trivialsort[list_] := Module[
  {im, i, list2 = list, n = Length[list]},
  For[i = 0, i < n, i++,
   im = indmax[list2[[1 ;; n - i]]];(*Busca el índice máximo*)
   list2 = swap[list2, im, n - i];(*Ordena elemento a elemento*)
   ];

  list2]
```

1.4.2. Búsqueda binaria

El algoritmo de búsqueda binaria es un método para buscar datos dentro de una estructura que generalmente es un arreglo unidimensional. Se le da el nombre de búsqueda binaria porque el algoritmo divide en dos el arreglo, aludiendo al concepto de bit, el cual puede tener dos estados.

La única condición para usar este algoritmo es que los datos dentro del arreglo ya estén ordenados de menor a mayor.

La función `binarysearch` de dos argumentos a y b elemento b en una lista ordenada a compara b con el elemento que está en la mitad de la lista, y dependiendo del resultado de la comparación busca b en la primera o en la segunda parte de la lista.

Entradas: una lista ordenada a y un elemento b.

Salida: el índice máximo i tal que $a[i] \leq b$; si $b < a[i]$ para todo i, entonces debe regresar 0.

```
binarysearch[list_, b_] := Module[
  {n = Length[list], i, m, k},
  m = Quotient[n, 2];(*Encuentra el elemento a la mitad del arreglo*)
  If[list[[m]] > b, i = 0, i = m];(*Revisa en que lado del arreglo está el elemento*)
  If[n != 2,
   i += binarysearch[list[[i + 1 ;; m + i]], b],(*Se aplica recursión*)
   ];
  i]
```

1.4.3. Ordenamiento por mezcla

Mezcla de dos listas ordenadas

La función `merge` construye una lista ordenada de dos listas ordenadas dadas.

Entradas: dos listas ordenadas, A y B.

Salida: una lista ordenada C que consiste en los elementos de A y B (tomando en cuenta las repiticiones).

```
merge[l1_, l2_] := Module[
  {n1 = Length[l1], n2 = Length[l2], list, i1 = 1, i2 = 1,
   ind = 1},(*Entrada, dos listas ordenadas*)
  list = Table[0, {n1 + n2}];(*Se construye el arreglo de salida*)
  While[i1 <= n1 && i2 <= n2,
   If[l1[[i1]] <= l2[[i2]],(*Se compara elemento a elemento*)
    list[[ind]] = l1[[i1]];
    i1++,(*Se actualizan indices*)
    list[[ind]] = l2[[i2]];
    i2++
    ];
   ind++;
   ];
  If[i1 - 1 < n1,(*Agrega elementos restantes*)
   list[[ind ;; n1 + n2]] = l1[[i1 ;; n1]],
   list[[ind ;; n1 + n2]] = l2[[i2 ;; n2]]
   ];
  list]
```

Ordenamiento por mezcla, versión recursiva

La función recursiva `mergesort` ordena la lista dada de la siguiente manera: primero llama a si misma dando como argumentos la primera y la segunda mitad de la lista original, y luego mezcla los resultados usando la función `merge`.

```
mergesortrec[list_] := Module[
  {n = Length[list], l1, l2, l3, m},
  m = Quotient[n, 2];
  If[n != 1,
  (*Aplica recursión aplicando el algoritmo en la mitad de los arreglos*)
   l1 = mergesortrec[list[[1 ;; m]]];
   l2 = mergesortrec[list[[m + 1 ;; n]]];
   l3 = merge[l1, l2],
   l3 = list;(* Caso base, si list tiene un elemento ya estta ordenado*)
   ];
  l3]
```

Ordenamiento por mezcla, versión no recursiva

```
mergesortIt[lista_] := Module[
  {n = Length[lista], p, k, j, LIS},
  LIS = Table[{lista[[i]]}, {i, n}];
  p = Ceiling[Log[2, n]];(*Control del numero de operaciones*)
  For[k = p, k >= 1, k--,
   n = Length[LIS];
   For[j = 0, j < Quotient[n, 2], j++,(*Se divide el arreglo en dos*)

    LIS[[j + 1]] =
      merge[LIS[[1 + 2*j]], LIS[[2*(j + 1)]]];(*Ordena por mitades*)
    ];
   If[Mod[n, 2] != 0,
    LIS[[Quotient[n, 2] + 1]] = LIS[[n]]];(*Agrega el último elemento*)

   LIS = Take[LIS, Mod[n, 2] + Quotient[n, 2]];
   ];
  LIS[[1]]]
```

Capítulo 2

Transformada discreta de Fourier

En este capítulo se presenta la transformada discreta de Fourier. Esta transformación se implemente mediante un algoritmo llamado transformada rápida de Fourier que se presenta en el siguiente capítulo, y se aplica en campos tales como el tratamiento de señales en telecomunicación, la resolución de ecuaciones en derivadas parciales o la multiplicación de números muy grandes en computación. Entre sus numerosas aplicaciones se incluyen métodos numéricos de interpolación y aproximación.

En ciertas operaciones importantes entre vectores $\mathbf{x}$, tienen una formulación mucho más simple cuando se expresan en términos de sus transformadas. Un caso análogo se plantea con los logaritmos de los números positivos, multiplicar x y y corresponde, en lenguaje logarítmico a sumar $\log x$ y $\log y$, una operación mucho más sencilla en tiempos anteriores a las calculadoras o mucho más rápida de calcular en computadoras actuales. El proceso en este ejemplo sería

1. Se transforma $x \to X = \log x$, $y \to Y = \log y$.
2. Se opera con las transformaciones $Z = X + Y$.
3. Se deshace la transformación, pasando del número Z a otro z tal que $\log z = Z$.

2.1. Matriz de Fourier

En ocasiones es necesario plantear lo que se conoce como análisis de Fourier discreto, es decir, relaciones discretas de vectores de dimensión finita con vectores de dimensión finita. Puesto que una de las propiedades más importantes del análisis de Fourier continuo es la linealidad, se debe tratar de conservarla en el caso discreto, lo que lleva a plantear dicha relación entre vectores como una aplicación lineal dada por una matriz.

Hay varios problemas cuya resolución acaba en la misma cuestión, por ejemplo, en el caso de procesamiento de señales una relación entre vectores de valores en los dominios del

tiempo y de la frecuencia definida por una matriz conocida como matriz de Fourier. Dicha relación y las propiedades de dicha matriz son la base del análisis discreto de Fourier.

Definición 2.1.1. *La matriz de Fourier se define como*

$$\Omega_n = \left[\omega_n^{mk}\right]_{m,k=0}^{n-1}$$

Por ejemplo, para $n = 2$

$$\Omega_2 = \begin{bmatrix} \omega_2^0 & \omega_2^0 \\ \omega_2^0 & \omega_2^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Para $n = 3$

$$\Omega_3 = \begin{bmatrix} \omega_3^0 & \omega_3^0 & \omega_3^0 \\ \omega_3^0 & \omega_3^1 & \omega_3^2 \\ \omega_3^0 & \omega_3^2 & \omega_3^4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{-i\frac{2\pi}{3}} & e^{-i\frac{4\pi}{3}} \\ 1 & e^{-i\frac{4\pi}{3}} & e^{-i\frac{8\pi}{3}} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 \\ 1 & -\frac{1}{2} - i\frac{\sqrt{3}}{2} & -\frac{1}{2} + i\frac{\sqrt{3}}{2} \\ 1 & -\frac{1}{2} + i\frac{\sqrt{3}}{2} & -\frac{1}{2} - i\frac{\sqrt{3}}{2} \end{bmatrix}$$

Luego, tenemos que

$$\Omega_n^{-1} = \frac{1}{n}\left[\omega_n^{-mk}\right]_{m,k=0}^{n-1}$$

Teorema 2.1.1. *Sea Ω_n la matriz de Fourier de orden n, es decir la matriz cuadrada cuyo elemento (k, n) es ω^{nk} para $n, k = 0, 1, \ldots n-1$ siendo ω^n la raíz n-ésima de la unidad. Entonces*

1. *Ω_n es simétrica.*
2. *Las columnas de Ω_n son ortogonales dos a dos y tienen norma igual a $\sqrt{n}$, es decir, la matriz $\dfrac{1}{\sqrt{n}}\Omega_n$ es unitaria.*

2.2. Transformada de Fourier

La transformada de Fourier discreta (n-dimensional) es la aplicación lineal de $\mathbb{C}^n$ en sí mismo dada por $\mathbf{x} \longrightarrow \mathbf{X} = \Omega_n \mathbf{x}$.

Definición 2.2.1. *La transformada de Fourier $\mathcal{F}_n \colon \mathbb{C}^n \to \mathbb{C}^n$ se define mediante la siguiente fórmula:*

$$\mathcal{F}_n(a) = \left[\sum_{k=0}^{n-1} a_k \omega_n^{kj}\right]_{j=0}^{n-1}.$$

2.3. Transformada inversa a la transformada discreta de Fourier

Teorema 2.3.1. *La matriz Ω_n es invertible y $\Omega_n^{-1} = \frac{1}{n}\overline{\Omega_n}$, donde $\overline{\Omega_n}$ es la matriz que resulta al conjugar cada elemento de Ω_n*

Demostración. Basta ver que $\Omega_n\overline{\Omega_n} = nI$. El elemento (j,k) de dicha matriz es

$$\begin{aligned}(\Omega_n\overline{\Omega_n})_{jk} &= \sum_{l=0}^{n-1}(\Omega_n)_{jl}\overline{(\Omega_n)_{lk}} \\ &= \sum_{l=0}^{n-1}\omega_n^{jl}\overline{\omega_n^{lk}} \\ &= \sum_{l=0}^{n-1}\omega_n^{jl}\omega_n^{-lk} \\ &= \sum_{l=0}^{n-1}\left(\omega_n^{j-k}\right)^l.\end{aligned}$$

Si $j = k$

$$(\Omega_n\overline{\Omega_n})_{jk} = \sum_{l=0}^{n-1} 1^l = n,$$

y si $j \neq k$, hay que sumar una progresión geométrica con los términos $\omega_n^{j-k} \neq 1$, entonces se tiene

$$\begin{aligned}(\Omega_n\overline{\Omega_n})_{jk} &= \frac{(\omega_n^{j-k})^n - 1}{\omega_n^{j-k} - 1} \\ &= \frac{(\omega_n^n)^{j-k} - 1}{\omega_n^{j-k} - 1} = \frac{1^{j-k} - 1}{\omega_n^{j-k} - 1} = 0.\end{aligned}$$

Es decir

$$(\Omega_n\overline{\Omega_n})_{jk} = \begin{cases}1, & \text{si j=k} \\ 0, & \text{si } \jmath \neq k\end{cases}$$

□

Por lo tanto para tener la transformada inversa a la transformada discreta de Fourier basta con multiplicar la inversa de la matriz de Fourier por un vector. Es decir

$$\mathcal{F}_n^{-1}(b) = \frac{1}{n}\left[\sum_{j=0}^{n-1} b_j\omega_n^{-kj}\right]_{k=0}^{n-1}.$$

2.4. Implementación en lenguaje Mathematica

```
TDF[v_] := Module[
  {n = Length[v], wn, tdf, j},
  tdf = Table[0, {n}];
  wn = N[Exp[-(2*Pi*\[ImaginaryI])/
      n]];(*Calcula las n raíces de la unidad*)
  For[j = 0, j <= n - 1, j++,
   tdf[[j + 1]] =
    Sum[v[[k + 1]]*wn^(k*j), {k, 0,
      n - 1}]]; (*Calcula la transformada como multiplicación de la \
matriz de Fourier por el vector*)
  tdf]
```

Mientras que para la TDF inversa:

```
ITDF[v_] := Module[(*Calcula la TDF Inversa*)
  {n = Length[v], wn, tdf, j},
  tdf = Table[0, {n}];(*Vector de salida*)
  wn = N[Exp[-(2*Pi*\[ImaginaryI])/n]];(*Raíces de la unidad*)
  For[j = 0, j <= n - 1, j++,
   tdf[[j + 1]] = (1/n)*
     Sum[v[[k + 1]]*wn^(-k*j), {k, 0,
       n - 1}]];(*Se calcula con el conjugado de la matriz de fourier*)

  tdf]
```

Capítulo 3

Transformada rápida de Fourier

La evaluación de la Transformada Discreta de Fourier requiere n^2 operaciones aritméticas. Mediante el algorimo de la Transformada Rápida de Fourier se obtiene el mismo resultado para calcular la Transformada Discreta y su Inversa con tan sólo $n\log(n)$ operaciones. Este algoritmo es mucho más eficiente en cuanto al tiempo de cómputo para arreglos grandes y fue publicado por Cooley y Tuckey en 1965.

La versión que vamos a estudiar es la mas habitual y corresponde al caso en que $n = 2^m$, es decir, que la dimensión del espacio es una potencia de 2.

3.1. Algoritmo de Cooley y Tuckey

La idea básica es aprovechar la gran cantidad de operaciones que se repiten en los cálculos y aplicar una técnica de diseño de algoritmos que se conoce como divide y vencerás. Se comienza por dividir el vector $\mathbf{X}$ de longitud $n = 2^m$ en dos vectores de tamaño par:

$$\mathbf{X}^P = (x_0, x_2, \ldots, x_{n-2})^t, \quad y \quad \mathbf{X}^I = (x_1, x_3, \ldots, x_{n-1})^t$$

y finalmente reconstruiremos el vector transformado $\mathbf{X}$ a partir de éstos dos

Proposición 3.1.1. *En la situación anterior las $\frac{n}{2}$ primeras y las $\frac{n}{2}$ últimas componentes de $\mathbf{X}$ están dadas por*

$$\mathbf{X}_k = \mathbf{X}_k^P + \omega_n^k \mathbf{X}_k^I, \quad k = 0, 1, \ldots \frac{n}{2} - 1 \tag{3.1}$$

$$\mathbf{X}_{\frac{n}{2}+k} = \mathbf{X}_k^P - \omega_n^k \mathbf{X}_k^I, \quad k = 0, 1, \ldots \frac{n}{2} - 1 \tag{3.2}$$

Demostración. Para $k = 0, 1, \ldots \frac{n}{2} - 1$, tenemos que:

$$\mathbf{X}_k = \sum_{j=0}^{n-1} \omega_n^{jk} x_j$$

y agrupando por separado en esta suma los términos con j par e impar, tenemos:

$$\mathbf{X}_k = \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{2jk} x_{2j} + \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{(2j+1)k} x_{2j+1}$$

y como $\omega_n^2 = \omega_{\frac{n}{2}}$, resulta que:

$$\begin{aligned}
\mathbf{X}_k &= \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{2jk} x_{2j} + \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{(2j+1)k} x_{2j+1} \\
&= \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{2jk} x_{2j} + \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{2jk} \omega_n^k x_{2j+1} \\
\mathbf{X}_k &= \sum_{j=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{jk} x_j^P + \omega_n^k \sum_{j=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{jk} x_j^I
\end{aligned}$$

Esta igualdad prueba (1) pues las expresiones del segundo miembro son totalmente válidas para $k = 0, 1, \ldots, \frac{n}{2} - 1$ y evidentemente las sumas coinciden con las componentes $k-$ésimas de $\mathbf{X}^P$ y $\mathbf{X}^I$ respectivamente.

Para los índices $\frac{n}{2}, \frac{n}{2}+1, \ldots, n-1$, que escribiremos en la forma $\frac{n}{2}+k$ con $k = 0, 1, \ldots, \frac{n}{2}-1$, siguiendo el mismo proceso:

$$\begin{aligned}
\mathbf{X}_{\frac{n}{2}+k} &= \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{2j(\frac{n}{2}+k)} x_{2j} + \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{(2j+1)(\frac{n}{2}+k)} x_{2j+1} \\
&= \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{2j\frac{n}{2}} \omega_n^{2jk} x_{2j} + \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{2j\frac{n}{2}} \omega_n^{2jk} \omega_n^{\frac{n}{2}} \omega_n^k x_{2j+1} \\
&= \sum_{j=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{j\frac{n}{2}} \omega_{\frac{n}{2}}^{jk} x_j^P + \omega_n^{\frac{n}{2}} \omega_n^k \sum_{j=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{j\frac{n}{2}} \omega_n^{jk} x_j^I
\end{aligned}$$

pero $\omega_{\frac{n}{2}}^{\frac{n}{2}} = 1$ y $\omega_n^{\frac{n}{2}} = -1$, por lo tanto

$$\mathbf{X}_{\frac{n}{2}+k} = \sum_{j=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{jk} x_j^P - \omega_n^k \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{2jk} x_j^I$$

□

La esencia del algoritmo termina aquí. Se reemplaza la tarea de calcular una transformada $n-$dimensional (cuando n es par) por la de dos dimensiones ($\frac{n}{2}$). Como $\frac{n}{2}$ también es par, podemos calcular cada una de ellas a partir de dos transformadas $\frac{n}{4}$-dimensionales y de forma recurrente reduciremos el cálculo de Ω_n al calcular un buen número de transformadas de dos elementos que no necesitan ninguna multiplicación.

3.2. Número de operaciones en la TRF

Si escribimos como $M(m)$ el número de multiplicaciones de números complejos necesarias para realizar la transformada de Fourier discreta de $N = 2^m$ elementos mediante el algoritmo TRF y tomamos como condición inicial del cálculo recursivo que $M(1) = 0$, entonces tenemos que

$$M(m) = 2M(m-1) + 2^{m-1}, \tag{3.3}$$

teniendo en cuenta que el producto que aparece en el segundo término de ambas ecuaciones del algoritmo es el mismo, no es necesario repetirlo para el cálculo de la segunda mitad de los términos. Una sola aplicación de este proceso de división y recomposición ya reduce el costo operativo desde N^2 multiplicaciones, que se realizan con el producto de matriz por vector, hasta $2\left(\frac{N}{2}\right)^2 + \frac{N}{2} = \frac{N^2}{2} + \frac{N}{2}$, es decir poco más de la mitad cuando N es grande. Pero es la iteración del proceso lo que permite la reducción. en efecto, en base a la recurrencia, tenemos que

$$\begin{aligned} M(m) &= 2[2M(m-2) + 2^{m-2}] + 2^{m-1} = 2^2 M(m-2) + 2 \cdot 2^{m-1} \\ &= 2^k M(m-k) + k2^{m-1} \quad k = 3, \ldots, m-2 \\ &= 2^{m-1} M(1) + (m-1)2^{m-1} = (m-1)2^{m-1}, \end{aligned}$$

lo que en términos de N resulta ser $\frac{N}{2}(\log(N-1)) = \frac{N}{2}\log(\frac{N}{2})$.

3.3. Implementación en Mathematica

La función que implementa el algortimo de la FFT es:

```
FFT[v_] := Module[(*Entrada un vector v*)
  {n = Length[v], X, k, e, o},
  If[n == 1,
   X = TDF[v],(*Caso base*)
   X = Table[0, {n}];
   e = FFTrec[v[[;; ;; 2]]]; (*Aplica recursión*)
   o = FFTrec[v[[2 ;; ;; 2]]];
   For[k = 0, k <= (n/2) - 1, k++,
    X[[k + 1]] =
     e[[k + 1]] +
      N[Exp[-2 Pi \[ImaginaryI] (k/n)]]*
       o[[k + 1]];(*Actualiza elementos de salida para la primera
         mitad del arreglo que guarda la transformada*)
    X[[k + (n/2) + 1]] =
     e[[k + 1]] -
      N[Exp[-2 Pi \[ImaginaryI] (k/n)]]*
```

```
      o[[k + 1]];(*Segunda mitad del arreglo*)
   ];
  ];
 X]
```

Capítulo 4

Multiplicación rápida de polinomios vía TRF

Transformada discreta de Fourier escrita en términos de los polinomios

4.1. Transformada discreta de Fourier como valores de un polinomio en las raíces de la unidad

Dado un vector $a \in \mathbb{C}^n$ definimos el polinomio P_a:

$$P_a(z) = \sum_{k=0}^{n-1} a_k z^k.$$

Notamos que:

$$\deg(P_a) \leq n-1$$

Entonces la TDF se expresa en términos del polinomio

$$b = \Omega_N a = \left[\sum_{k=0}^{n-1} a_k \omega_n^{kj}\right]_{j=0}^{n-1} = \left[P_a(\omega_n^j)\right]_{j=0}^{n-1}$$

Calcular los coeficientes de un polinomio si están dados sus valores en las raíces de la unidad

Sea P un polinomio de grado $\leq n-1$:

$$P(z) = \sum_{k=0}^{n-1} a_k z^k.$$

Denotemos por b_j ($j \in \{0, 1, \ldots, n-1\}$) sus valores en las raíces de la unidad:

$$b_j = P(\omega_n^j).$$

Entonces, los coeficientes a a través de b, están dados por:

$$a = \Omega_n^{-1} b = \left[\sum_{k=0}^{n-1} b_k \omega_n^{-kj}\right]_{j=0}^{n-1}$$

La transformada inversa discreta de Fourier permite calcular los coeficientes de un polinomio de grado menor o igual a $n-1$ si se saben sus valores en $\omega_n^0, \ldots, \omega_n^{n-1}$.

4.2. Algoritmo de multiplicación de polinomios usando la TRF

Sean dos polinomios dados P y Q, donde:

$$P(z) = \sum_{k=0}^{n-1} a_k z^k \qquad y \qquad Q(z) = \sum_{k=0}^{n-1} b_k z^k$$

Sabemos que a través de la TDF podemos calcular las coeficientes de un polinomio de grado menor o igual a $N-1$ si conocemos sus valores en las raíces de unidad. Como tenemos dos polinomios, a traves de la TDF podemos conocer los valores de P y Q en las raíces de unidad, y al multiplicar dichos valores obtenemos el valor del producto PQ en las raíces de unidad $\omega_N^0, \ldots, \omega_n^{n-1}$, por lo que bastaría aplicar la inversa de la TDF para conocer los coeficientes del polinomio PQ. El algoritmo es el siguiente:

Entrada: $a_0, \ldots, a_{n-1}$; $b_0, \ldots, b_{n-1}$
$u := \Omega_{2n-1} a$ // Valores de P
$v := \Omega_{2n-1} b$ // Valores de Q
$w := u \cdot v$ // Valores de PQ
$c := \Omega_{2n-1}^{-1} w$ // coeficientes de PQ
Salida: c.

4.3. Implementación en Mathematica

Para calcular Ω_{2n-1} cuando a tiene n componentes, creamos vectores auxiliares dentro de la función que implementa el algorimo que sean de tamaño $2n-1$, cada una de las $n-1$ componentes que faltan en el vector a serán 0.

```
multTDF1[a_, b_] :=
 Module[(*Multiplicacion de polinomios con funcion TDF*)
```

```
  {n = Length[a], A, B, aux, u, v, w, c},
  aux = Table[0, {2 n - 1 - n}];
  A = Join[a, aux]; B = Join[b, aux];(*Crea vectores auxiliares*)
  u = Table[0, {2 n - 1}];
  v = Table[0, {2 n - 1}];
  c = u;
  u = TDF[A];
  v = TDF[B];(*Aplica transformada a los coeficientes de los \
polinomios*)
  w = u*v;(*Multiplicación de las transformaciones*)
  c = ITDF[
    w];(*Se calculan los coeficientes de la transformación calculando \
la transformada inversa*)
  Chop[c]](*Salida: Coeficientes aproximando los números muy cercanos \
a cero*)
```

Usando las funciones internas de Mathematica, el algoritmo implementado (que es más rápido) es:

```
multTDF[a_, b_] :=
 Module[(*Calcula con funciones internas de Mathematica*)
  {n = Length[a], aux, A, B, u, v},
  aux = Table[0, {2 n - 1 - n}];
  A = Join[a, aux]; B = Join[b, aux];
  u = Fourier[A, FourierParameters -> {1, 1}];
  v = Fourier[B, FourierParameters -> {1, 1}];
  Chop[InverseFourier[u*v, FourierParameters -> {1, 1}]]](*Salida con redondeos*)
```

Capítulo 5

Algoritmos rápidos para resolver sistemas de ecuaciones lineales con matrices de Toeplitz

Los problemas computacionales científicos o de ingeniería por lo general se reducen a cálculos con matrices, donde, eventualmente se tiene que resolver un sistema de ecuaciones lineales. La estructura del problema original por lo general resulta en una estructura matricial del sistema de ecuaciones lineales, por lo que se busca diseñar algoritmos que exploten estas estructuras con el objeto de que sean más rápidos que los algoritmos convencionales.

5.1. Matrices de Toeplitz

Una matriz de Toeplitz es una matriz $n \times n$ $T_n = [t_{k,j}]_{k,j=1}^{n}$ donde $t_{k,j} = t_{k-j}$, es decir una matriz de la forma

$$T_n = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \dots & t_{-(n-1)} \\ t_1 & t_0 & t_{-1} & & \\ t_2 & t_1 & t_0 & & \vdots \\ \vdots & & & \ddots & \\ t_{n-1} & & & \dots & t_0 \end{bmatrix}$$

Estas matrices tienen muchas aplicaciones. Por ejemplo, si se supone que

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

es un vector columna que denota las entrada y que t_k es cero para $k < 0$. Entonces el vector

$$\begin{aligned} y = T_n x &= \begin{bmatrix} t_0 & 0 & 0 & \cdots & 0 \\ t_1 & t_0 & 0 & & \\ t_2 & t_1 & t_0 & & \vdots \\ \vdots & & & \ddots & \\ t_{n-1} & & & \cdots & t_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} \\ &= \begin{bmatrix} x_0 t_0 \\ t_1 x_0 + t_0 x_1 \\ \sum_{i=0}^{2} t_{2-i} x_i \\ \vdots \\ \sum_{i=0}^{n-1} t_{n-1-i} x_i \end{bmatrix} \end{aligned}$$

con entradas

$$y_k = \sum_{i=0}^{k} t_{k-i} x_i$$

representa la salida de tiempo discreto en señales con un filtro h con respuesta de impulso t_k.

Como otro ejemplo se considera que $\{X_n\}$ es un proceso de tiempo discreto aleatorio con media $m_k = E(X_k)$ y función de covarianza $K_X(k, j) = E[(X_k - m_k)(X_j - m_j)]$. La teoría del procesamiento de señales como predicción, estimación, detección, clasificación, regresión, comunicaciones y teoría de información se desarrollan bajo el supuesto de que la media es constante y la covarianza es Toeplitz, es decir $K_X(k, j) = K_X(k - j)$, en cuyo caso el proceso es estacionario débil. En este caso, las matrices de covarianza $n \times n$ $K_n = [K_X(k, j)]_{k,j=0}^{n-1}$ son matrices de Toeplitz. Gran cantidad de la teoría de procesos estacionarios débiles involucran aplicaciones de matrices de Toeplitz. Las matrices de Toeplitz aparecen también en la solución de ecuaciones diferenciales e integrales, funciones splines, problemas y métodos en física, matemáticas, estadística y procesamiento de señales.

5.2. Algoritmo de Schur

El presente algoritmo principalmente es utilizado para resolver sistemas Toeplitz, ya que produce la factorización LU de la matriz. Asimismo puede ser combinado con el algoritmo Levinson reemplazando los cálculos de producto interno. El método resultante tiene una complejidad secuencial ligeramente mayor que el algoritmo de Levinson pero una complejidad significativamente menor en computación paralela.

Sea T una matriz de Toeplitz de orden n:

$$T_n = \begin{bmatrix} a_0 & \dots & a_{-n+1} \\ \vdots & \ddots & \vdots \\ a_{n-1} & \dots & a_0 \end{bmatrix}$$

Denotamos a las submatrices T_k de T_n como:

$$T_k^+ = \underbrace{\left.\begin{bmatrix} a_{k-n} & \dots & a_{1-n} \\ \vdots & & \vdots \\ a_0 & \dots & a_{1-k} \end{bmatrix}\right\} n-k+1}_{k} \qquad y \qquad T_k^- = \underbrace{\left.\begin{bmatrix} a_{k-1} & \dots & a_0 \\ \vdots & & \vdots \\ a_{n-1} & \dots & a_{n-k} \end{bmatrix}\right\} n-k+1}_{k}$$

Observamos que estas submatrices son de tamaño $(n-k+1) \times k$.

Para analizar más, tomemos $n = 5$ y $k = 2$, generando la siguiente matriz

$$T_5 = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & a_{-3} & a_{-4} \\ a_1 & a_0 & a_{-1} & a_{-2} & a_{-3} \\ a_2 & a_1 & a_0 & a_{-1} & a_{-2} \\ a_3 & a_2 & a_1 & a_0 & a_{-1} \\ a_4 & a_3 & a_2 & a_1 & a_0 \end{bmatrix}$$

y las submatrices:

$$T_2^- = \begin{bmatrix} a_{-3} & a_{-4} \\ a_{-2} & a_{-3} \\ a_{-1} & a_{-2} \\ a_0 & a_{-1} \end{bmatrix} \qquad T_2^+ = \begin{bmatrix} a_1 & a_0 \\ a_2 & a_1 \\ a_3 & a_2 \\ a_4 & a_3 \end{bmatrix}$$

Por otro lado si analizamos la matriz

$$T_2 = \begin{bmatrix} a_0 & a_{-1} \\ a_1 & a_0 \end{bmatrix}$$

Podemos notar que:

- La última fila de T_k^- es la primera fila de T_k.
- La primera fila de T_k^+ es la última fila de T_k.

Por otro lado, denotaremos:

- $\mathbf{x}_k^-$ como la primera columna de T_k^{-1}.

- $\mathbf{x}_k^+$ como la última columna de T_k^{-1}.

Se tiene que:

$$\begin{bmatrix} T_k^- \\ T_k^+ \end{bmatrix} \begin{bmatrix} \mathbf{x}_k^- & \mathbf{x}_k^+ \end{bmatrix} = \begin{bmatrix} \mathbf{s}_k^{--} & \mathbf{s}_k^{-+} \\ \mathbf{s}_k^{+-} & \mathbf{s}_k^{++} \end{bmatrix}$$

$$\left[\begin{array}{cc} a_{-3} & a_{-4} \\ a_{-2} & a_{-3} \\ a_{-1} & a_{-2} \\ \hline a_0 & a_{-1} \\ a_1 & a_0 \\ a_2 & a_1 \\ a_3 & a_2 \\ a_4 & a_3 \end{array}\right] \begin{bmatrix} \mathbf{x}_k^- & \mathbf{x}_k^+ \end{bmatrix} = \left[\begin{array}{cc|cc} [a_{-3} & a_{-4}]\mathbf{x}_k^- & [a_{-3} & a_{-4}]\mathbf{x}_k^+ \\ \multicolumn{2}{c|}{\vdots} & \multicolumn{2}{c}{\vdots} \\ \hline [a_0 & a_{-1}]\mathbf{x}_k^- & [a_0 & a_{-1}]\mathbf{x}_k^+ \\ [a_1 & a_0]\mathbf{x}_k^- & [a_1 & a_0]\mathbf{x}_k^+ \\ \multicolumn{2}{c|}{\vdots} & \multicolumn{2}{c}{\vdots} \\ [a_4 & a_3]\mathbf{x}_k^- & [a_4 & a_3]\mathbf{x}_k^+ \end{array}\right]$$

$$= \begin{bmatrix} \mathbf{s}_k^{--} & \mathbf{s}_k^{-+} \\ \mathbf{s}_k^{+-} & \mathbf{s}_k^{++} \end{bmatrix}$$

$$= \left[\begin{array}{cc} s_{1,k}^{--} & s_{1,k}^{-+} \\ \vdots & \vdots \\ s_{n-k+1,k}^{--} & s_{n-k+1,k}^{-+} \\ \hline s_{1,k}^{+-} & s_{1,k}^{++} \\ \vdots & \vdots \\ s_{n-k+1,k}^{+-} & s_{n-k+1,k}^{++} \end{array}\right]$$

De donde:

- $s_{i,k}^{--} = [a_{k+1-n-1}, \ldots, a_{i-n}]\mathbf{x}_k^-$.
- $s_{i,k}^{+-} = [a_i, \ldots, a_{i+1-k}]\mathbf{x}_k^-$.

En particular, si:

$$T_k \left[\begin{array}{c|c} \mathbf{x}_k^+ & \mathbf{x}_k^- \end{array}\right] = \begin{bmatrix} a_0 & a_{-1} \\ a_1 & a_0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_k^+ & \mathbf{x}_k^- \end{bmatrix} = \begin{bmatrix} 1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 1 \end{bmatrix}$$

Así:

- $s_{1,k}^{+-} = 0 \qquad s_{1,k}^{++} = 1.$
- $s_{n-k+1,k}^{--} = 1 \qquad s_{n-k+1,k}^{-+} = 0.$

Los vectores $\mathbf{s}_k^{\pm\pm}$ serán llamados *vectores residuales*.

Además, se definen los *vectores de reflexión*

$$\gamma_k^+ = s_{2,k}^{+-} \quad \text{y} \quad \gamma_k^- = s_{n-k,k}^{-+}$$

Tomemos como caso particular nuestro ejemplo en el que $n = 5$ y $k = 2$.

$$\left[\begin{array}{cc} a_{-3} & a_{-4} \\ a_{-2} & a_{-3} \\ a_{-1} & a_{-2} \\ \hline a_0 & a_{-1} \\ a_1 & a_0 \\ a_2 & a_1 \\ a_3 & a_2 \\ a_4 & a_3 \end{array}\right] \left[\begin{array}{cc} \mathbf{x}_k^- & \mathbf{x}_k^+ \end{array}\right] = \left[\begin{array}{cc} s_{1,k}^{--} & s_{1,k}^{-+} \\ s_{2,k}^{--} & s_{2,k}^{-+} \\ s_{3,k}^{--} & s_{3,k}^{-+} \\ s_{4,k}^{--} & s_{4,k}^{-+} \\ \hline s_{1,k}^{+-} & s_{1,k}^{++} \\ s_{2,k}^{+-} & s_{2,k}^{++} \\ s_{3,k}^{+-} & s_{3,k}^{++} \\ s_{4,k}^{+-} & s_{4,k}^{++} \end{array}\right] = \left[\begin{array}{cc} s_{1,k}^{--} & s_{1,k}^{-+} \\ s_{2,k}^{--} & s_{2,k}^{-+} \\ s_{3,k}^{--} & \gamma_k^- \\ 1 & 0 \\ \hline 0 & 1 \\ \gamma_k^+ & s_{2,k}^{++} \\ s_{3,k}^{+-} & s_{3,k}^{++} \\ s_{4,k}^{+-} & s_{4,k}^{++} \end{array}\right]$$

En el caso general:

$$\left[\begin{array}{c} T_k^- \\ T_k^+ \end{array}\right] \left[\begin{array}{cc} \mathbf{x}_k^- & \mathbf{x}_k^+ \end{array}\right] = \left[\begin{array}{cc} s_{1,k}^{--} & s_{1,k}^{-+} \\ \vdots & \vdots \\ s_{n-k,k}^{--} & s_{n-k,k}^{-+} \\ \hline s_{n-k+1,k}^{--} & s_{n-k+1,k}^{-+} \\ s_{1,k}^{+-} & s_{1,k}^{++} \\ s_{2,k}^{+-} & s_{2,k}^{++} \\ \vdots & \vdots \\ s_{n-k+1,k}^{+-} & s_{n-k+1,k}^{++} \end{array}\right] = \left[\begin{array}{cc} s_{1,k}^{--} & s_{1,k}^{-+} \\ \vdots & \vdots \\ s_{n-k,k}^{--} & \gamma_k^- \\ 1 & 0 \\ \hline 0 & 1 \\ \gamma_k^+ & s_{2,k}^{++} \\ \vdots & \vdots \\ s_{n-k+1,k}^{+-} & s_{n-k+1,k}^{++} \end{array}\right]$$

Los vectores residuales tienen un papel importante en la factorización LU de la matriz T_n. Sea V la matriz triangular superior cuya k-ésima columna es $\left[\begin{array}{c} \mathbf{x}_k^+ \\ \mathbf{0} \end{array}\right]$. Entonces $L = T_n V$ es triangular inferior y la k-ésima columna de L es $\left[\begin{array}{c} \mathbf{0} \\ \mathbf{s_k^{++}} \end{array}\right]$. Entonces $T_n = LV^{-1}$ es la factorización triangular de T_n. Es decir, la matriz L que está formada por los vectores residuales s_k^{++} es la matriz L triangular inferior de la factorización LU de la matriz T_n.

El algoritmo de Schur calcula los vectores residuales recursivamente. Para deducir como, utilizamos una estructura de Toeplitz. Para $(u_j)_{j=1}^m$, denotamos por $I_+\mathbf{u}$, $I_-\mathbf{u}$ a los vectores

$$I_+\mathbf{u} = (u_j)_{j=2}^m, \quad I_-\mathbf{u} = (u_j)_{j=1}^{m-1} \tag{5.1}$$

respectivamente. Es decir I_+, quita la primera entrada del vector $\mathbf{u}$ y I_- corta la última entrada del vector. Además

$$I_{\pm\pm}\mathbf{u} = I_\pm(I_\pm\mathbf{u}). \tag{5.2}$$

Notemos que

$$\begin{bmatrix} T_k^- \\ I_\pm T_k \\ T_k^+ \end{bmatrix} x_k^\pm = \begin{bmatrix} s_k^{-\pm} \\ \mathbf{0}_{k-2} \\ s_k^{+\pm} \end{bmatrix}$$

y el paso $k \longrightarrow k+1$ extiende el vector cero del lado derecho de la ecuación en un cero más arriba y abajo. Tenemos

$$\begin{bmatrix} T_{k+1}^- \\ T_{k+1}^+ \end{bmatrix} \begin{bmatrix} x_k^- & 0 \\ 0 & x_k^+ \end{bmatrix} = \begin{bmatrix} T_+ s_k^{--} & I_- s_k^{-+} \\ I_+ s_k + - & I_- s_k^{++} \end{bmatrix}$$

Teorema 5.2.1. *Para $k = 1, \ldots, n-1$, los vectores residuales $s_k^{\pm\pm}$ satisfacen la recursión*

$$\begin{bmatrix} s_{k+1}^{--} & s_{k+1}^{-+} \\ s_{k+1}^{+-} & s_{k+1}^{++} \end{bmatrix} = \begin{bmatrix} I_+ s_k^{--} & I_- s_k^{-+} \\ I_+ s_k^{+-} & I_- s_k^{++} \end{bmatrix} \Gamma_k^{-1},$$

donde

$$\Gamma_k = \begin{bmatrix} 1 & s_{n-k,k}^{-+} \\ s_{2,k}^{+-} & 1 \end{bmatrix}.$$

La recursión inicia con

$$s_1^{++} = s_1^{+-} = \frac{1}{a_0}(a_{i-1})_i = 1^n, \quad s_1^{-+} = s_1^{--} = \frac{1}{a_0}(a_{i-n})_{i=1}^n.$$

Para escribir la recursión de Schur en términos polinómicos, se usa la proyección P_m definida por los polinomios de Laurent

$$P_m\Big(\sum_{k=-M}^{N} u_k t^{k-1}\Big) = \sum_{k=1}^{m} u_k t^{k-1} \tag{5.3}$$

P_m quita todas las potencias negativas de t y todas las potencias mayores a $m-1$, en particular para $\mathbf{u} = (u_i)_{i=1}^m$,

$$(I_+\mathbf{u})(t) = (\mathbf{u}(t) - u_1)t^{-1} = P_{m-1}t^{-1}\mathbf{u}(t), \quad (I_-\mathbf{u})(t) = \mathbf{u}(t) - u_m t^{m-1} = P_{m-1}\mathbf{u}(t).$$

Así la recursión del Teorema 5.2.1 puede ser escrita de la siguiente forma

$$\begin{bmatrix} s_{k+1}^{--}(t) & s_{k+1}^{-+}(t) \\ s_{k+1}^{+-}(t) & s_{k+1}^{++}(t) \end{bmatrix} = P_{n-k} \begin{bmatrix} s_k^{--}(t) & s_k^{-+}(t) \\ s_k^{+-}(t) & s_k^{++}(t) \end{bmatrix} \begin{bmatrix} t^{-1} & 0 \\ 0 & 1 \end{bmatrix} \Gamma_k^{-1}.$$

Aquí la proyección P_{n-k} debe ser aplicada entrada por entrada a la matriz polinomial.

La recursión en el Teorema 5.2.1 puede ser usada para calcular los vectores $\mathbf{x}_k\pm$, sin necesidad de operaciones de producto interno.

5.3. Solución de sistemas de Toeplitz con el algoritmo de Schur

El algoritmo de Schur produce la factorización LU de una matriz de Toeplitz $T_n = LDU$. Esta factorización puede ser usada para resolver el sistema $T_n\mathbf{z} = \mathbf{b}$ por sustitución hacia atrás. Esto significa que primero se resuelve un sistema triangular inferior $LD\mathbf{y} = \mathbf{b}$ y después el sistema triangular superior $U\mathbf{z} = \mathbf{y}$.

La complejidad de resolver un sistema triangular es $\frac{1}{2}n^2(M) \cdot \frac{1}{2}n^2(A)$. Entonces la complejidad de resolver un sistema de Toeplitz aplicando el algoritmo de Schur es $3n^2(M) \cdot 3n^2(A)$.

5.4. Implementación del Algoritmo de Schur en Mathematica

```
SchurLU[vm_] :=
 Module[{t = ToeplitzMatrix[vm],
   n = Length[
     vm]},(*Factorización de Schur de una matriz para matrices de \
Topelitz T_n*)
  V = Table[0, {n}, {n}];
  For[j = 1, j <= n, j++,
   For[i = 1, i <= j, i++,
    V[[i, j]] =
     Inverse[ToeplitzMatrix[vm[[1 ;; j]]]][[i,
      j]]]];(*Se crea la matriz V con los vectores x^+*)
  L = t.V; (*Se aplican la fórmula para obtener V*)
  {L // MatrixForm, Inverse[V] // MatrixForm,
   L.Inverse[V] //
    MatrixForm}](*Muestra resultados en forma matricial*)
```

Bibliografía

[1] Ammar, G.& Gragg, W. (1987). *The Generalized Schur Algorithm for the Superfast Solution of Toeplitz Systems*, Department of Mathematics, University of Kentucky.

[2] Burden R.L. & Faires, J.D. (2001).*Numerical Analysis (7th ed.)*, Cengage Learning.

[3] Heinig, G. & Rost, K. (2011). *Fast Algorithms for Toeplitz and Hankel Matrices*, Linear Algebra and its applications.

[4] Wong, M. W. (2011). *Discrete Fourier Analysis*. Birkhäuser, Ontario