# Series de Fourier y otras transformadas

Laura Herrera y Juan Esteban Duque

2025-10-24

---

### title: "Potenciación y radicación de números complejos"

### Introducción

La radicación y la potenciación de números complejos extienden las operaciones reales al plano complejo. En **forma polar**, todo número complejo puede escribirse como

$$z \;=\; r(\cos\theta + i\sin\theta) \;=\; r\,e^{i\theta},$$

donde $r = |z|$ es el **módulo** y $theta = arg(z)$ es el **argumento**.

El **Teorema de De Moivre** establece que

$$(\cos\theta + i\sin\theta)^n \;=\; \cos(n\theta) + i\sin(n\theta),$$

y en forma exponencial

$$(e^{i\theta})^n = e^{in\theta}$$

### Teorema de De Moivre (raíces cuartas de $z$ )

Sean $z = re^{i\theta}$ y $n \in \mathbb{N}$. Las $n$-ésimas raíces de $z$ están dadas por

$$w_k \;=\; r^{1/n}\,e^{i\frac{\theta+2\pi k}{n}}, \qquad k = 0, 1, \ldots, n-1.$$

Para $n = 4$:

$$w_k \;=\; r^{1/4}\,e^{i\frac{\theta+2\pi k}{4}}, \qquad k = 0, 1, 2, 3.$$

## Verificación numérica por codigo:

Cada raíz debe cumplir $w^n \approx z$

```
# Cálculo de raíces n-ésimas con De Moivre
import cmath, math #Librerias

def raices_n_esimas(z: complex, n: int):
    """
    Devuelve la lista [w_0, ..., w_{n-1}] de raíces n-ésimas de z,
    usando la forma polar y De Moivre:
       w_k = |z|^(1/n) * exp( i*(arg(z) + 2πk)/n ),  k=0..n-1
    """
    r, th = abs(z), cmath.phase(z)
    return [(r**(1/n)) * cmath.exp(1j * (th + 2*math.pi*k) / n) for k in range(n)]

# --- Ejemplo del ejercicio: z = (16 i)/(1 + i), n = 8 ---
z = (16j) / (1 + 1j) #debemos reemplazar i por j en Python
n = 8
W = raices_n_esimas(z, n)

# Verificación numérica: cada raíz debe cumplir w^n ≈ z
residuos = [abs(w**n - z) for w in W]
print("Residuos:",[f"{r:.2e}" for r in residuos])
```

```
Residuos: ['8.88e-15', '1.07e-14', '7.94e-15', '1.07e-14', '9.57e-15', '5.40e-14', '4.65e-14', '5
```

## Aplicación ejercicio 13:

### Simplificación

$$(\frac{16i}{1+i})^{1/8} = (\frac{16i(1-i)}{(1+i)(1-i)})^{1/8} = 8(1śi)^{1/8}$$

En forma polar de $z$

$$1+i = \sqrt{2}\,e^{i\pi/4} \quad \Rightarrow \quad z = 8\sqrt{2}\,e^{i\pi/4}.$$

$$|z| = 8\sqrt{2} \quad \Rightarrow \quad Modulo$$

$$z = pi/4 \quad \Rightarrow \quad Argumentoprincipal$$

Por tanto, si $z = 8\sqrt{2}\,e^{i\pi/4}$, sus **ocho raices octavas** son:

$n = 8$

$$\boxed{w_k = (8\sqrt{2})^{1/8}\,e^{\,i\left(\frac{\pi}{32}+\frac{k\pi}{4}\right)}} \qquad k = 0, 1, \ldots, 7.$$

Como $(8\sqrt{2})^{1/8} = 2^{7/16}$, el radio comun es:

$$\boxed{w_k = 2^{7/16}\Big[\cos\Big(\frac{\pi}{32} + \frac{k\pi}{4}\Big) + i\sin\Big(\frac{\pi}{32} + \frac{k\pi}{4}\Big)\Big],\ k = 0, \ldots, 7.}$$

### Raíz cuarta principal

La **raíz principal** se obtiene con el **argumento principal** $Arg(z)$ y $k = 0$:

$$\sqrt[4]{z} \; = \; r^{1/4}\, e^{\,i\ \mathrm{Arg}(z)/4}, \qquad z = re^{i\theta}.$$

**Raiz principal**

$$k = 0$$

Para el anterior ejercicio,

$$w_0 = 2^{7/16} e^{i\frac{\pi}{32}} \approx 1.349 + 0.133i.$$

Las demás raíces se obtienen rotando sucesivamente 45ř (es decir, sumando $\pi/4$ al angulo) alrededor del circulo de radio $2^{7/16}$.

### Aproximaciones numéricas

Por aritmética de punto flotante, $w_k^4$ no es **exactamente** $z$. El residuo $|w_k^4 - z|$ debe ser pequeño.

Para el ejemplo anterior, todas las raíces están sobre el círculo de centro $(0, 0)$ y radio $r = 1.3543$, formando un **octagono regular**. El primer punto esta en el angulo $\pi/32(\approx 5.625$ř) y luego vas sumando $\pi/4$

**Coordenadas aproximadas (para trazar)**

```python
def signo_md(x: float, dec: int = 4) -> str:
    """
    Devuelve el número con signo +/− y 'dec' decimales, usando el
    signo unicode '−' para negativos.
    """
    s = f"{abs(x):.{dec}f}"
    return ("+" + s) if x >= 0 else ("−" + s)  # nota: '−' U+2212

# Encabezado
print(f"z = {z}  |z|={abs(z):.6f}  arg(z)={cmath.phase(z):.6f} rad")

# Tabla Markdown
print("\n| k |  Real (Wk) | Imaginario |")
for k, w in enumerate(W):
    re = signo_md(w.real, 4)
    im = signo_md(w.imag, 4)
    print(f"| {k} | {re:>10} | {im:>10} |")
```

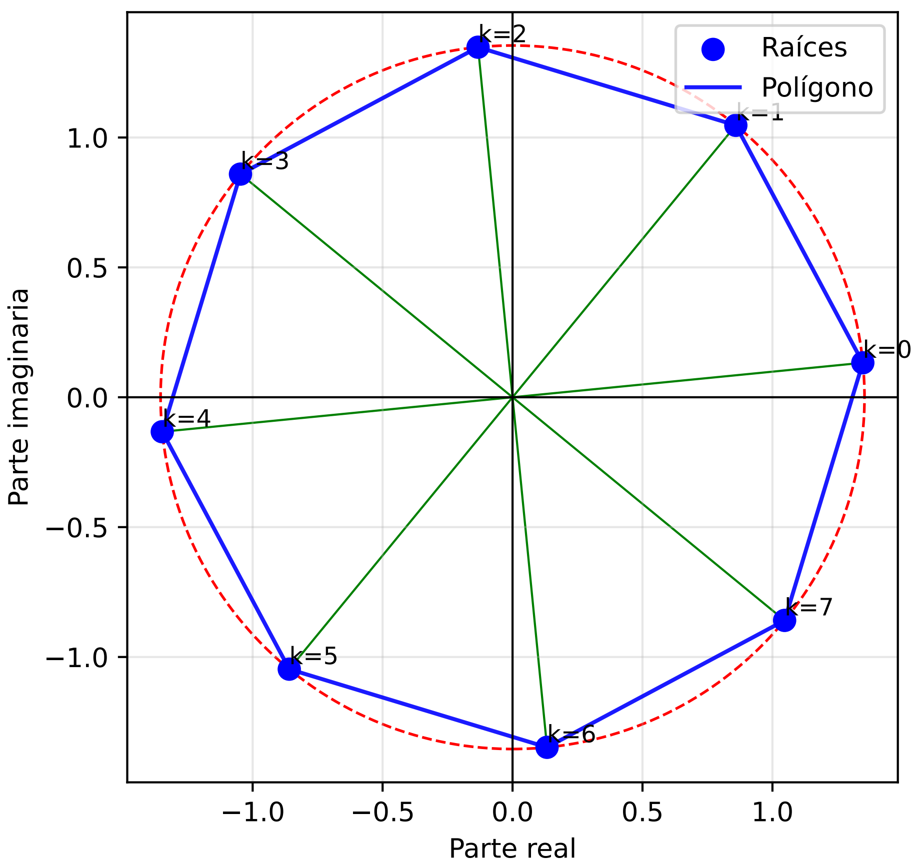
```
z = (8+8j)  |z|=11.313708  arg(z)=0.785398 rad

| k |  Real (Wk) | Imaginario |
| 0 |    +1.3477 |    +0.1327 |
| 1 |    +0.8591 |    +1.0469 |
| 2 |    −0.1327 |    +1.3477 |
| 3 |    −1.0469 |    +0.8591 |
| 4 |    −1.3477 |    −0.1327 |
```

```
| 5 |    −0.8591 |    −1.0469 |
| 6 |    +0.1327 |    −1.3477 |
| 7 |    +1.0469 |    −0.8591 |
```

### Grafica de raices

```python
import cmath
import math
import matplotlib.pyplot as plt

def graficar_raices(z: complex, n: int, conectar=True, rayos=False, circulo=True):
    """Grafica raíces n-ésimas de z y (opcional) las conecta en orden angular."""
    roots = raices_n_esimas(z, n)
    if not roots:
        return
    radio = abs(roots[0])

    # Ordenar por ángulo para conectar correctamente
    roots_sorted = sorted(roots, key=lambda w: cmath.phase(w))

    # Coordenadas
    xs = [w.real for w in roots_sorted]
    ys = [w.imag for w in roots_sorted]

    # Cerrar el polígono si se conecta
    if conectar and n > 1:
        xs.append(xs[0])
        ys.append(ys[0])

    # Gráfica
    fig, ax = plt.subplots(figsize=(5,5))

    # Círculo guía
    if circulo:
        ax.add_artist(plt.Circle((0,0), radio, fill=False, linestyle="--", color="red", lw=1))

    # Puntos
    ax.scatter([w.real for w in roots], [w.imag for w in roots],
               color="blue", s=60, zorder=3, label="Raíces")

    # Conexión (polígono)
    if conectar and n > 1:
        ax.plot(xs, ys, color="blue", lw=1.5, alpha=0.9, label="Polígono")

    # Rayos desde el origen (opcional)
    if rayos:
        for w in roots:
            ax.plot([0, w.real], [0, w.imag], color="green", lw=0.8, zorder=1)
```

```
    # Etiquetas k (según orden original de generación)
    for k, w in enumerate(roots):
        ax.text(w.real, w.imag, f"k={k}", fontsize=9, va="bottom", ha="left")

    # Ejes y estilo
    ax.axhline(0, color="black", lw=0.8)
    ax.axvline(0, color="black", lw=0.8)
    ax.set_aspect("equal", adjustable="box")
    ax.set_xlabel("Parte real")
    ax.set_ylabel("Parte imaginaria")
    ax.grid(alpha=0.3)
    ax.legend(loc="upper right")
    plt.tight_layout()
    plt.show()

# ==== Ejemplo del ejercicio ====
z = (16j)/(1+1j)  # (16 i)/(1+i)
n = 8
graficar_raices(z, n, conectar=True, rayos=True, circulo=True)
```



## Ejemplo planteado:

Sea el sistema

$$\begin{cases} -2x - 3y \;=\; 3, \\ y \;=\; 1. \end{cases}$$

Hallar la inversa de coeficientes y resolver el sistema.

Dada su forma matricial, $AX = B$:

$$A = \begin{bmatrix} -2 & -3 \\ 1 & 0 \end{bmatrix}, \quad X = \begin{bmatrix} x \\ y \end{bmatrix}, \quad B = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

Planteamos la **inversa generica** y usamos $AA^{-1} = 1$. Obtenemos:

$$\begin{bmatrix} -2\alpha - 3\gamma & -2\beta - 3\delta \\ 0\alpha + 1\gamma & 0\beta + 1\delta \end{bmatrix} \quad = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Entonces, se obtiene el sistema para $\alpha, \beta, \gamma, \delta$

$$-2\alpha - 3\gamma \;=\; 1, -2\beta - 3\delta \;=\; 0, \gamma \;=\; 0.\delta \;=\; 1.$$

Sustituimos $\gamma$ y $\delta$:

$$-2\alpha - (3)\text{ů}0 \;=\; 1, -2\beta - (3)\text{ů}1 \;=\; 0,$$

$\therefore$

$$A^{-1} = \begin{bmatrix} -\frac{1}{2} & -\frac{3}{2} \\ 0 & 1 \end{bmatrix}.$$

Comprobacion rapida

$$AA^{-1} = \begin{bmatrix} -2 & -3 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -\frac{1}{2} & -\frac{3}{2} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Resolviendo $X = A^{-1}B$:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & -\frac{3}{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}\text{ů}3 + -\frac{3}{2}\text{ů}1 \\ 0\text{ů}3 + 1\text{ů}1 \end{bmatrix} = \begin{bmatrix} -3 \\ 1 \end{bmatrix}$$

Por lo tanto,

$$\boxed{x = -3, \qquad y = 1}.$$

## Diagonalización

Ejercicio:

$$A = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix},$$

El polinomio característico de $A$ es

$$p_A(\lambda) = \det(A - \lambda I) = \det \begin{bmatrix} -\lambda & 1 \\ -2 & -3-\lambda \end{bmatrix} .p_A(\lambda) = (-\lambda)(-3-\lambda) - (-2)(1) = \lambda(\lambda+3) + 2 = \lambda^2 + 3\lambda + 2.$$

Factorizamos:

$$p_A(\lambda) = (\lambda+1)(\lambda+2).$$

**Valores propios:**

$$\lambda_1 = -1, \qquad \lambda_2 = -2.$$

**Espacio propio** de $\lambda_1 = -1$

Formamos $A - (-1)I = A + I$:

$$A + I = \begin{bmatrix} 1 & 1 \\ -2 & -2 \end{bmatrix}.$$

Resolvemos $(A+I)\vec{t} = 0$:

$$\begin{bmatrix} 1 & 1 \\ -2 & -2 \end{bmatrix} \begin{bmatrix} \vec{t_1} \\ \vec{t_2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Quiere decir que $\vec{t_1} = \vec{t_2}$,

Un **vector propio** asociado es:

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

Espacio propio de $\lambda_2 = -2$

Formamos $A - (-2)I = A + 2I$:

$$A + 2I = \begin{bmatrix} 2 & 1 \\ -2 & -1 \end{bmatrix}.$$

Resolvemos $(A+2I)\vec{t} = 0$:

$$\begin{bmatrix} 2 & 1 \\ -2 & -1 \end{bmatrix} \begin{bmatrix} \vec{t_1} \\ \vec{t_2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Esto equivale a $2\vec{t_1} + \vec{t_2} = 0$, es decir, $\vec{t_2} = -2\vec{t_1}$.

Un **vector propio** asociado es:

$$\mathbf{v}_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}.$$

**Construcción de $P$ y $D$**

Colocamos los vectores propios como columnas de $P$:

$$P = \begin{bmatrix} 1 & 1 \\ -1 & -2 \end{bmatrix}, \qquad D = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}.$$

Comprobamos que se cumple:

$$P^{-1}AP = D.$$

$\therefore$ La matriz $A$ es diagonalizable y hemos encontrado:

$$A = PDP^{-1}, D = \text{diag}(-1, -2).$$

### Ejercicio matriz $3x3$

$$A = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix},$$

### Polinomio característico

El polinomio característico de $A$ es

$$p_A(\lambda) = \det(A - \lambda I) = \det \begin{bmatrix} 2-\lambda & 1 & 0 \\ 0 & 2-\lambda & 0 \\ 0 & 0 & 2-\lambda \end{bmatrix}.$$

Como la matriz es triangular superior, el determinante es el producto de los elementos de la diagonal:

$$p_A(\lambda) = (2-\lambda)^3.$$

Por lo tanto, el único valor propio es:

$$\lambda = 2,$$

con **multiplicidad algebraica**

$$m_a = 3$$

.

## Vectores propios

**Vectores propios**: Vamos a resolver

$$(A - 2I)\vec{t} = 0,$$

es decir,

$$A - 2I = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Sea $\vec{t} = \begin{bmatrix} \vec{t_1} \\ \vec{t_2} \\ \vec{t_3} \end{bmatrix}$, el sistema resulta:

$$\vec{t_2} = 0,$$

con $\vec{t_1}, \vec{t_3}$ libres. $\therefore$

$$\vec{t} = \begin{bmatrix} \vec{t_1} \\ 0 \\ \vec{t_3} \end{bmatrix} = \vec{t_1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \vec{t_3} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

**Espacio propio**: El espacio propio asociado a $\lambda = 2$ está generado por:

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad \mathbf{v}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

$\therefore$, $A$ **no es diagonalizable** (solo tiene dos vectores propios linealmente independientes para una matriz de orden 3).

---

## Comprobación computacional en Python

```
# EDITA AQUÍ (1): MATRIZ A
A_list = [
    [2, 1, 0],
    [0, 2, 0],
    [0, 0, 2],
]

# EDITA AQUÍ (2): eigenvalor a analizar
lambda_eval = 2

import sympy as sp

A = sp.Matrix(A_list)
n = A.shape[0]

print("Matriz A =")
sp.pprint(A); print()

# 1) Valores propios y multiplicidades algebraicas
evals = A.eigenvals()
print("Valores propios (m_a):")
for lam, mult in evals.items():
    print(f"  λ = {lam}   (m_a = {mult})")
print()

# 2) Espacio propio de lambda_eval
E = A - lambda_eval*sp.eye(n)
print(f"Espacio propio para λ = {lambda_eval}:  (A - λI) v = 0")
print("A - λI =")
sp.pprint(E); print()

null_basis = E.nullspace()
print("Base del espacio propio:")
if null_basis:
    for i, v in enumerate(null_basis, 1):
        print(f"  v_{i} = "); sp.pprint(v); print()
else:
    print("  (vacío) → λ no es eigenvalor de A\n")

m_g = len(null_basis)
print(f"Multiplicidad geométrica m_g(λ={lambda_eval}) = {m_g}\n")

# 3) ¿Es A diagonalizable? (suma de m_g para todos los λ debe ser n)
eig_vects = A.eigenvects()  # [(λ, m_a, [basis]), ...]
total_geom = sum(len(basis) for _, _, basis in eig_vects)
es_diag = (total_geom == n)

print("¿A es diagonalizable?")
print(f"  Suma de multiplicidades geométricas = {total_geom} (n = {n})")
```

```python
print(f"  Resultado: {'SÍ' if es_diag else 'NO'}\n")

# 4) Si es diagonalizable, construimos P y D y verificamos P^{-1} A P = D
if es_diag:
    eig_vects_sorted = sorted(eig_vects, key=lambda t: t[0])  # ordenar por λ (opcional)
    P_cols, D_diag = [], []
    for lam, _, basis in eig_vects_sorted:
        for v in basis:
            P_cols.append(sp.Matrix(v))
            D_diag.append(lam)
    P = sp.Matrix.hstack(*P_cols)
    D = sp.diag(*D_diag)

    print("P (columnas = eigenvectores):")
    sp.pprint(P); print()
    print("D (diagonal de eigenvalores):")
    sp.pprint(D); print()

    print("Verificación P^{-1} A P =")
    sp.pprint(sp.simplify(P.inv()*A*P)); print()
else:
    print("Explicación:")
    print("  No hay suficientes eigenvectores linealmente independientes para formar P.")
```

```
Matriz A =
⎡2  1  0⎤
⎢       ⎥
⎢0  2  0⎥
⎢       ⎥
⎣0  0  2⎦

Valores propios (m_a):
  λ = 2   (m_a = 3)

Espacio propio para λ = 2:  (A - λI) v = 0
A - λI =
⎡0  1  0⎤
⎢       ⎥
⎢0  0  0⎥
⎢       ⎥
⎣0  0  0⎦

Base del espacio propio:
  v_1 =
⎡1⎤
⎢ ⎥
⎢0⎥
⎢ ⎥
⎣0⎦
```

```
  v_2 =
⌈0⌉
| |
|0|
| |
⌊1⌋

Multiplicidad geométrica m_g(λ=2) = 2

¿A es diagonalizable?
  Suma de multiplicidades geométricas = 2 (n = 3)
  Resultado: NO

Explicación:
  No hay suficientes eigenvectores linealmente independientes para formar P.
```

---

## title: "Ecuaciones en diferencia y métodos de aproximación"

## Método de Herón para aproximar raíces cuadradas

El **método de Herón**, también llamado método babilónico, es un procedimiento iterativo para calcular raíces cuadradas.
Dado un número ( S ) y una estimación inicial ( y_0 ), la fórmula general es:

$$y(k+1) = \frac{1}{2}\left(y_k + \frac{S}{y_k}\right)$$

Cada iteración mejora la aproximación al valor real $\sqrt{S}$ El proceso se basa en una idea equivalente al método de Newton aplicado a

$$f(y) = y^2 - S$$

La iteración se detiene cuando la diferencia ( |y_{k+1} - y_k| ) es suficientemente pequeña.

### Ejemplo aproximación de raiz cuadrada de 2 con $y0 = 1$

Aplicando el método:

$$\begin{aligned} y_1 &= \tfrac{1}{2}(1 + 2/1) = 1.5, \\ y_2 &= \tfrac{1}{2}(1.5 + 2/1.5) = 1.4166, \\ y_3 &= \tfrac{1}{2}(1.4166 + 2/1.4166) = 1.4142. \end{aligned}$$

Con solo tres iteraciones, el resultado se aproxima al valor real:

raiz de 2 = 1.41421356

Al elevar ( 1.4142^2 ) se obtiene aproximadamente ( 2 ), confirmando la exactitud de la aproximación.

### Ecuación en diferencia tipo Fibonacci

Consideremos la ecuación en diferencia:

$$y_{n+2} - y_{n+1} - y_n = 0, \qquad y_0 = 1,\ y_1 = 1.$$

Su polinomio característico es:

$$r^2 - r - 1 = 0.$$

Sus raíces son:

$$r_1 = \frac{1+\sqrt{5}}{2}, \qquad r_2 = \frac{1-\sqrt{5}}{2}.$$

Por lo tanto, la solución general es:

$$y_n = A\, r_1^n + B\, r_2^n.$$

Aplicando las condiciones iniciales, se obtiene la **forma cerrada de Fibonacci**:

$$y_n = \frac{r_1^{n+1} - r_2^{n+1}}{\sqrt{5}}.$$

### Método general para ecuaciones en diferencia lineales con coeficientes constantes

Para una ecuación de orden (k):

$$a_k y_{n+k} + a_{k-1} y_{n+k-1} + \cdots + a_0 y_n = g(n),$$

los pasos generales son:

1. Plantear la **ecuación característica** (caso homogéneo ( g(n) = 0 )).
2. Encontrar las **raíces** del polinomio y su multiplicidad.
3. Formar la **solución homogénea** según el tipo de raíces (reales, repetidas o complejas).
4. Proponer una **solución particular** dependiendo de ( g(n) ).
5. Sumar ambas y aplicar **condiciones iniciales** para determinar las constantes.

### Ejemplo no homogéneo

$$y_{n+2} - 3y_{n+1} + 2y_n = 4^n, \qquad y_0 = 0,\ y_1 = 4.$$

Ecuación característica:

$$r^2 - 3r + 2 = 0 \quad \Rightarrow \quad r = 1,\ 2.$$

Solución homogénea:

$$y_n^{(h)} = A + B\, 2^n.$$

Propuesta particular:

$$y_n^{(p)} = C\,4^n \quad \Rightarrow \quad C = \frac{1}{6}.$$

Solución general:

$$y_n = A + B\,2^n + \tfrac{1}{6}4^n.$$

Determinando las constantes con $y_0$ y $y_1$, se obtiene la secuencia final.

## Interpretación computacional

Las ecuaciones en diferencia pueden implementarse fácilmente en código para simular sus valores iterativos.

**Ejemplo 1 — Método de Herón:**

```
def heron_sqrt(S, y0=1.0, tol=1e-10, max_iter=10):
    """
    Aproxima la raíz cuadrada de S mediante el método de Herón.
    """
    y = y0
    for _ in range(max_iter):
        y_next = 0.5 * (y + S / y)
        if abs(y_next - y) < tol:
            break
        y = y_next
    return y
print("Aproximación de sqrt(2):", heron_sqrt(2))
```

```
Aproximación de sqrt(2): 1.4142135623746899
```

**Ejemplo 2 — Ecuación en diferencia no homogénea:**

```
def solve_second_order(a1, a0, g, y0, y1, N):
    """
    Resuelve una ecuación en diferencia del tipo:
        y_{n+2} + a1*y_{n+1} + a0*y_n = g(n)
    """
    y = [y0, y1]
    for n in range(0, N-2):
        y_next = -a1 * y[-1] - a0 * y[-2] + g(n)
        y.append(y_next)
    return y

# Ejemplo con y_{n+2} - 3y_{n+1} + 2y_n = 4^n
g = lambda n: 4**n
seq = solve_second_order(-3, 2, g, y0=0, y1=4, N=8)
print("Secuencia y_n:", seq)
```

```
Secuencia y_n: [0, 4, 13, 35, 95, 279, 903, 3175]
```

## Conclusión

Las ecuaciones en diferencia constituyen una herramienta fundamental para modelar **procesos discretos** y **series temporales**, con aplicaciones en economía, ingeniería y algoritmos de aprendizaje.
El método de Herón, por su parte, ejemplifica el poder de los procedimientos iterativos para alcanzar soluciones precisas a partir de aproximaciones iniciales simples.

## Valor futuro y fórmulas financieras

El valor futuro (F) de una inversión (P) a una tasa de interés (i) durante (n) períodos se expresa como:

$$F = P(1+i)^n$$

Cada período multiplica el capital por $(1+i)$.
La relación entre períodos consecutivos puede verse como:

$$F_n = (1+i)F_{n-1},$$

una ecuación en diferencia que representa crecimiento exponencial discreto.

## Del interés compuesto a la ecuación en diferencia

El modelo discreto:

$$F_{n+1} = (1+i)F_n$$

describe el interés compuesto acumulativo.
Si $(i = 0.1)$ y $(F_0 = 1000)$:

$$(F_1 = 1100, ; F_2 = 1210,\ F_3 = 1331, \ldots$$

Esta forma recursiva permite analizar o simular el comportamiento del dinero a lo largo del tiempo.

## Ventajas del enfoque con ecuaciones en diferencia

- Permite observar la **evolución temporal** del capital paso a paso.
- Facilita la **simulación** mediante algoritmos iterativos.
- Acepta **términos variables**, como aportes o retiros periódicos.
- Se adapta a análisis predictivos y de **riesgo financiero**.

### Modelos financieros como ecuaciones en diferencia

**Interés compuesto:**

$$F_{n+1} = (1 + i)F_n$$

**Serie uniforme (aportes constantes (A)):**

$$F_{n+1} = (1 + i)F_n + A$$

**Serie con gradiente aritmético:**

$$F_{n+1} = (1 + i)F_n + (A + Gn),$$

donde $G$ es el cambio progresivo en los pagos.
Cada uno representa una variación del modelo general de crecimiento financiero discreto.

### Simulación de modelos financieros

Ejemplo de **interés compuesto**:

```
F = 1000
i = 0.1
for n in range(5):
    F = (1 + i) * F
    print(f"Periodo {n+1}: F = {F:.2f}")
```

```
Periodo 1: F = 1100.00
Periodo 2: F = 1210.00
Periodo 3: F = 1331.00
Periodo 4: F = 1464.10
Periodo 5: F = 1610.51
```

Ejemplo de **serie uniforme**:

```
F = 0
i = 0.08
A = 200
for n in range(6):
    F = (1 + i) * F + A
    print(f"Periodo {n+1}: F = {F:.2f}")
```

```
Periodo 1: F = 200.00
Periodo 2: F = 416.00
Periodo 3: F = 649.28
Periodo 4: F = 901.22
Periodo 5: F = 1173.32
Periodo 6: F = 1467.19
```

Estas simulaciones reproducen exactamente la ecuación en diferencia asociada a cada modelo.

### Interpretación dinámica del valor del dinero

La ecuación:

$$F_{n+1} - F_n = iF_n$$

muestra que el cambio de un período a otro es proporcional al capital actual.
Representa un **crecimiento exponencial discreto**, donde el dinero genera más valor conforme aumenta.
Este enfoque permite modelar fenómenos financieros como inversiones, préstamos o inflación en el tiempo.

---

### title: "Ecuaciones logísticas y dinámica poblacional"

### Introducción al modelo logístico discreto

El **modelo logístico discreto** describe la evolución de una población que crece con una tasa limitada por la competencia interna.
La ecuación básica es:

$$x_{n+1} = r\,x_n(1 - x_n)$$

donde: - $x_n$: población normalizada (entre 0 y 1), - $r$: parámetro de crecimiento o tasa de reproducción, - $x_{n+1}$: población en el siguiente período.

Este modelo capta cómo una población aumenta rápidamente al principio, pero se estabiliza cuando se acerca al límite de recursos.
Fue propuesto originalmente por **Pierre-François Verhulst (1845)** y es un ejemplo clásico de **modelo no lineal discreto**.

### Modelo logístico de la población de conejos

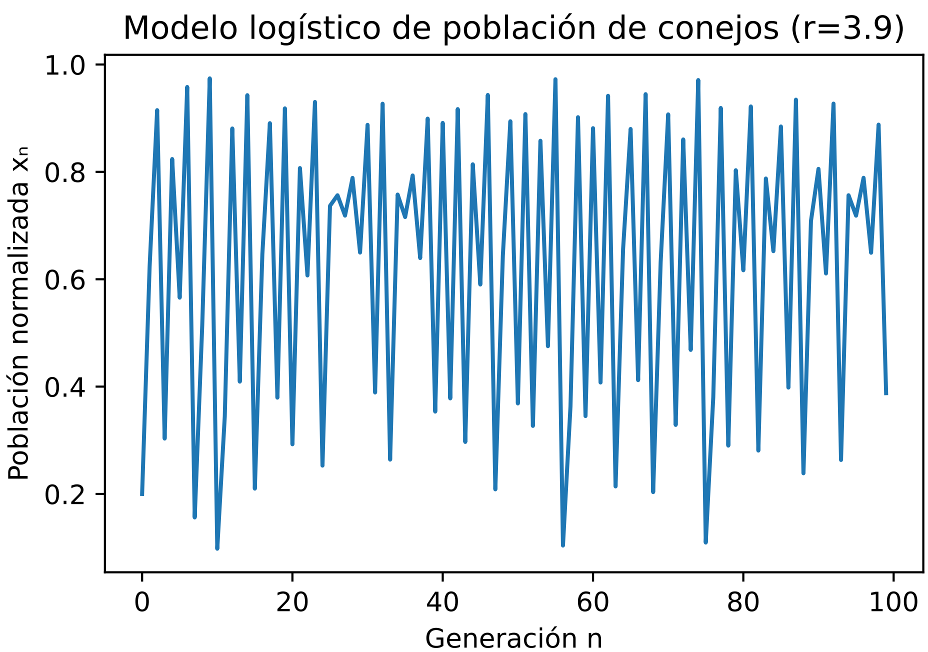
Supongamos una población de **conejos** en un ambiente con recursos limitados.
Si $x_n$ representa la proporción de conejos respecto a la capacidad máxima del entorno y $r$ es la tasa de crecimiento, la evolución está dada por:

$$x_{n+1} = r\,x_n(1 - x_n)$$

Este modelo genera distintos comportamientos dependiendo del valor de $r$: - Si $r < 1$: la población desaparece (tiende a 0). - Si $1 < r < 3$: la población se estabiliza en un valor fijo. - Si $3 < r < 3.57$: aparecen **ciclos** de período 2, 4, 8, etc. - Si $r > 3.57$: la población entra en **caos**.

### Implementación en Python del modelo de conejos

```
import numpy as np
import matplotlib.pyplot as plt

def logistic_population(r, x0, n):
    x = np.zeros(n)
    x[0] = x0
    for i in range(1, n):
        x[i] = r * x[i-1] * (1 - x[i-1])
    return x

# Parámetros
r = 3.9
x0 = 0.2
n = 100
x = logistic_population(r, x0, n)

plt.plot(range(n), x, lw=1.5)
plt.title(f"Modelo logístico de población de conejos (r={r})")
plt.xlabel("Generación n")
plt.ylabel("Población normalizada x$_n$")
plt.show()
```




**Interpretación:**
- Para (r=2.5), la población converge a un valor estable.
- Para (r=3.2), oscila entre dos niveles.
- Para (r=3.9), el comportamiento es caótico.

---

### Evolución temporal y sensibilidad a las condiciones iniciales

El modelo presenta **sensibilidad a las condiciones iniciales**:
si ($x_0 = 0.2000$) y ($x'_0 = 0.2001$), la diferencia entre ambas trayectorias puede crecer exponencialmente para valores altos de $r$.

Esta propiedad es una característica del **caos determinista**, donde el sistema sigue una regla exacta pero el resultado se vuelve impredecible.

### Diagrama de telaraña (Cobweb plot)

El **diagrama de telaraña** permite visualizar gráficamente la evolución de la población:

1. Se dibuja la curva $y = r\,x(1 - x)$ y la recta $y = x$.
2. Se parte desde un punto inicial $x_0$.
3. Se sube a la curva (para obtener $x_1$) y luego se proyecta a la recta identidad (para regresar al eje x).
4. Repitiendo el proceso se genera una figura que muestra cómo evoluciona la población.

**Interpretación:**
- Si el trazado converge al punto fijo, el sistema es estable.
- Si alterna entre varios valores, existe un ciclo periódico.
- Si no hay patrón repetido, el sistema es caótico.

### Diagrama de bifurcación

El **diagrama de bifurcación** muestra el comportamiento de largo plazo de la ecuación logística al variar el parámetro $r$.

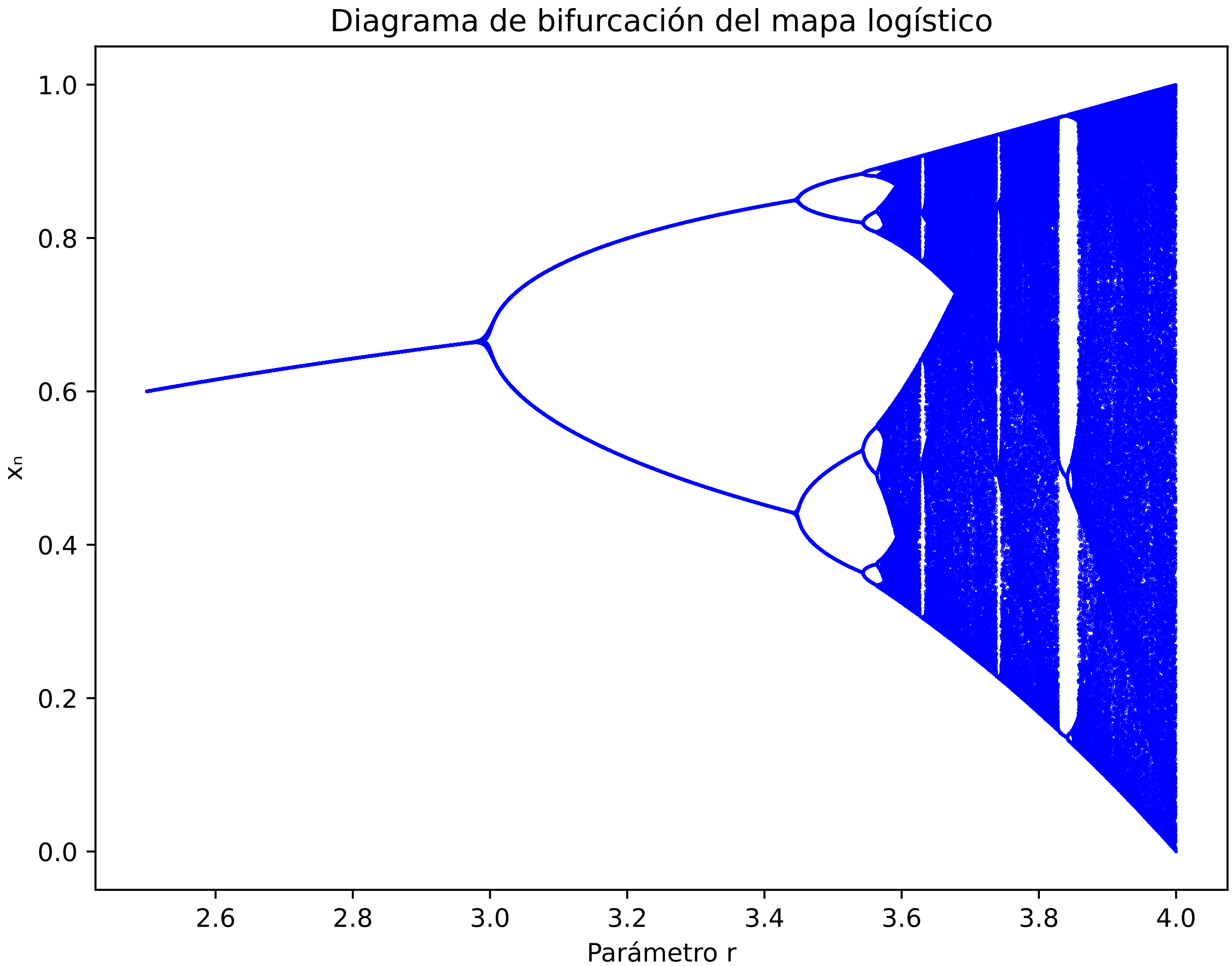
Para cada valor de $r$, se calculan muchas iteraciones y se grafican los valores finales de $x_n$.

- $r < 1$: la población muere.
- $1 < r < 3$: la población se estabiliza.
- $3 < r < 3.57$: se duplican los períodos (2, 4, 8…).
- $r > 3.57$: el sistema entra en caos.

La secuencia de duplicaciones de período obedece a la **constante de Feigenbaum ($\delta \approx 4.6692$)**, que describe la relación entre los intervalos sucesivos de bifurcación.

```python
def bifurcation(r_min=2.5, r_max=4.0, steps=3000, transient=200, plot_points=200):
    rs = np.linspace(r_min, r_max, steps)
    xs = []
    rs_all = []
    for r in rs:
        x = 0.5
        for _ in range(transient):
            x = r * x * (1 - x)
        for _ in range(plot_points):
            x = r * x * (1 - x)
            xs.append(x)
            rs_all.append(r)
```

```
    plt.figure(figsize=(8,6))
    plt.scatter(rs_all, xs, s=0.1, color="blue")
    plt.title("Diagrama de bifurcación del mapa logístico")
    plt.xlabel("Parámetro r")
    plt.ylabel("x$_n$")
    plt.show()

bifurcation()
```




### Ejemplos de comportamiento según el valor de $r$

| Valor de $r$ | Tipo de comportamiento | Descripción breve |
|---|---|---|
| 2.5 | Estable | Convergencia a punto fijo $x^* = 1 - 1/r = 0.6$. |
| 3.2 | Período 2 | Alternancia entre dos valores. |
| 3.5 | Período 4 o 8 | Ciclo de mayor complejidad. |
| 3.9 | Caótico | Trayectorias impredecibles y sensibles a $x_0$. |

### Relación con el algoritmo de Shor y periodicidad

El **algoritmo de Shor** (usado en computación cuántica para factorización) busca el **período** de una función modular:

$$f(a) = r^a \bmod N$$

donde $N$ es el número a factorizar y $r$ es una base coprima con $N$.
El algoritmo halla el **período** $p$ tal que:

$$r^p \equiv 1 \pmod N$$

y usa esa periodicidad para obtener los **factores primos** de $N$.

De forma conceptual, esto se asemeja al comportamiento periódico del **mapa logístico** antes del caos: ambos sistemas repiten estados o patrones cada cierto número de iteraciones.

### Resumen conceptual

- La **ecuación logística discreta** modela crecimiento poblacional con saturación.
- Al aumentar $r$, el sistema pasa de la estabilidad al caos.
- El **diagrama de bifurcación** muestra esta transición por duplicaciones de período.
- El **diagrama de telaraña** ilustra la convergencia, ciclos y caos.
- El **caos determinista** aparece para valores altos de $r$, donde pequeñas variaciones iniciales producen grandes diferencias.
- La búsqueda de periodicidad en sistemas como el logístico conecta con conceptos del **algoritmo de Shor**, que también se basa en la detección de períodos en funciones modulares.

### Series de Fourier — Idea general

Sea $f$ una función periódica de período $2L$ que cumple condiciones de Dirichlet. Entonces puede representarse como:

$$f(x) \sim \frac{a_0}{2} + \sum_{n=1}^{\infty} \Big[ a_n \cos\big(\tfrac{n\pi x}{L}\big) + b_n \sin\big(\tfrac{n\pi x}{L}\big) \Big].$$

En puntos de continuidad, la serie converge a $f(x)$; en discontinuidades converge al promedio lateral $\frac{f(x^-)+f(x^+)}{2}$.

### Coeficientes (período $2L$)

$$a_0 = \frac{1}{L}\int_{-L}^{L} f(x)\,dx, \qquad a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\big(\tfrac{n\pi x}{L}\big)\,dx, \qquad b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\big(\tfrac{n\pi x}{L}\big)\,dx.$$

**Caso $L = \pi$ (período $2\pi$)):**

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\,dx, \qquad a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)\,dx, \qquad b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)\,dx.$$

### Forma compleja

$$f(x) \sim \sum_{n=-\infty}^{\infty} c_n e^{in\pi x/L}, \quad c_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{-in\pi x/L}\,dx.$$

Relación con $a_n, b_n$ para $L = \pi$:

$$c_0 = \frac{a_0}{2}, \quad c_{\pm n} = \tfrac{1}{2}(a_n \mp ib_n).$$

### Simetrías útiles

- Si $f$ es par $f(-x) = f(x)$: $b_n = 0 \to$ serie solo de cosenos.
- Si $f$ es impar $f(-x) = -f(x)$: $a_0 = 0, a_n = 0 \to$ serie solo de senos.

### Series de medio rango

**Senos (extensión impar):**

$$f(x) \sim \sum_{n=1}^{\infty} B_n \sin\left(\tfrac{n\pi x}{L}\right), \quad B_n = \frac{2}{L}\int_0^L f(x)\sin\left(\tfrac{n\pi x}{L}\right)dx.$$

**Cosenos (extensión par):**

$$f(x) \sim \frac{A_0}{2} + \sum_{n=1}^{\infty} A_n \cos\left(\tfrac{n\pi x}{L}\right), \quad A_n = \frac{2}{L}\int_0^L f(x)\cos\left(\tfrac{n\pi x}{L}\right)dx.$$

### Convergencia y energía

**Criterio de Dirichlet:** si $f$ es acotada y suave a trozos, su serie de Fourier converge a $f(x)$ en puntos de continuidad y al promedio en saltos. **Parseval:**

$$\frac{1}{L}\int_{-L}^{L} |f(x)|^2\,dx = \frac{a_0^2}{2} + \sum_{n=1}^{\infty}(a_n^2 + b_n^2).$$

### Ejemplos típicos

**1.** $f(x) = x$ (impar, período $2\pi$):

$$x = 2\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}\sin(nx).$$

**2.** Onda cuadrada: $f(x) = \operatorname{sgn}(\sin x) \to$

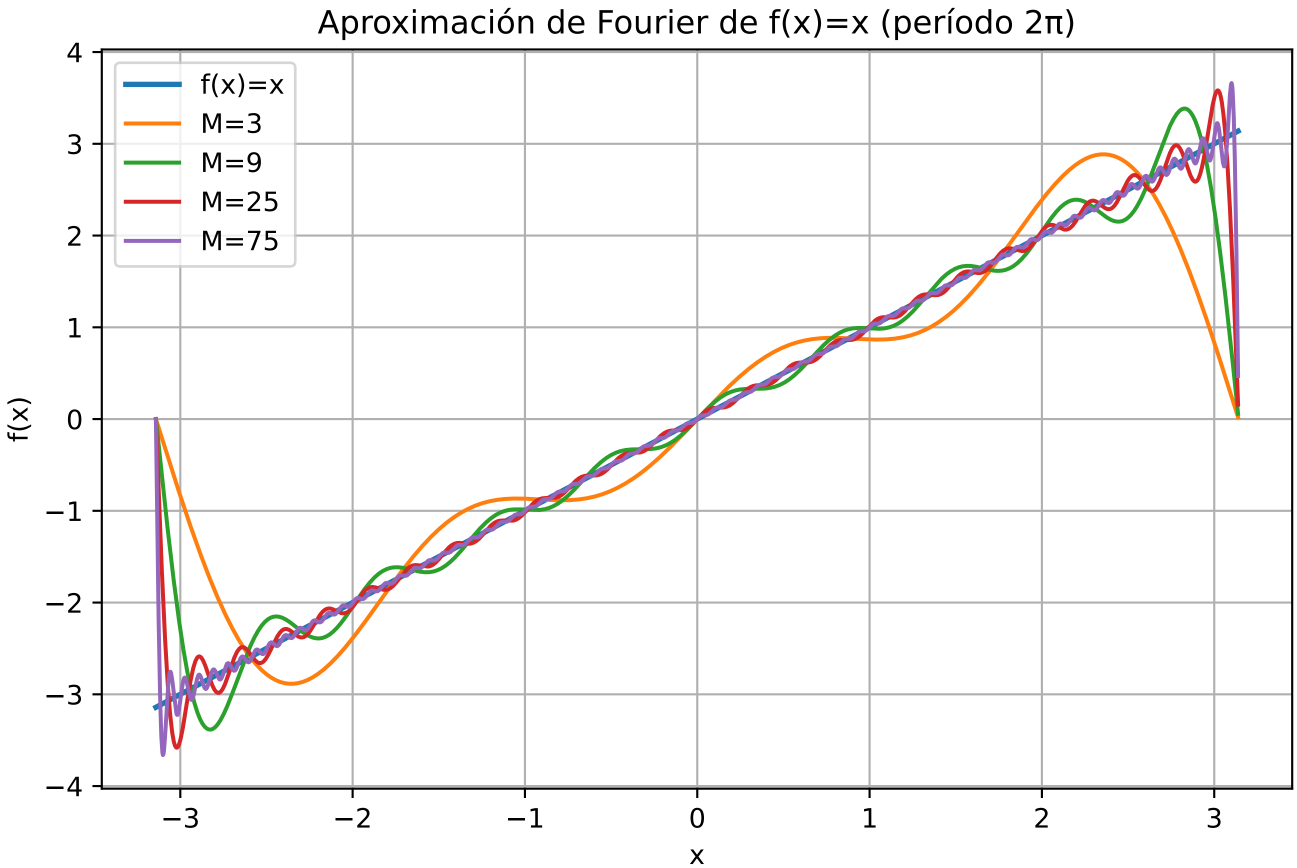
$$f(x) = \frac{4}{\pi}\sum_{k=0}^{\infty} \frac{1}{2k+1}\sin((2k+1)x).$$

**3.** Diente de sierra: $f(x) = x/\pi \to$

$$f(x) = \frac{2}{\pi}\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}\sin(nx).$$

### Demostración numérica: serie parcial de f(x)=x

```python
import numpy as np
import matplotlib.pyplot as plt

N = 2000
x = np.linspace(-np.pi, np.pi, N, endpoint=False)
f = x

def partial_fourier_x(M):
    s = np.zeros_like(x)
    for n in range(1, M+1):
        bn = 2 * ((-1)**(n+1)) / n
        s += bn * np.sin(n*x)
    return s

plt.figure(figsize=(8,5))
plt.plot(x, f, label="f(x)=x", linewidth=2)
for M in [3, 9, 25, 75]:
    sM = partial_fourier_x(M)
    plt.plot(x, sM, label=f"M={M}")
plt.legend()
plt.title("Aproximación de Fourier de f(x)=x (período 2π)")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.grid(True)
plt.show()
```



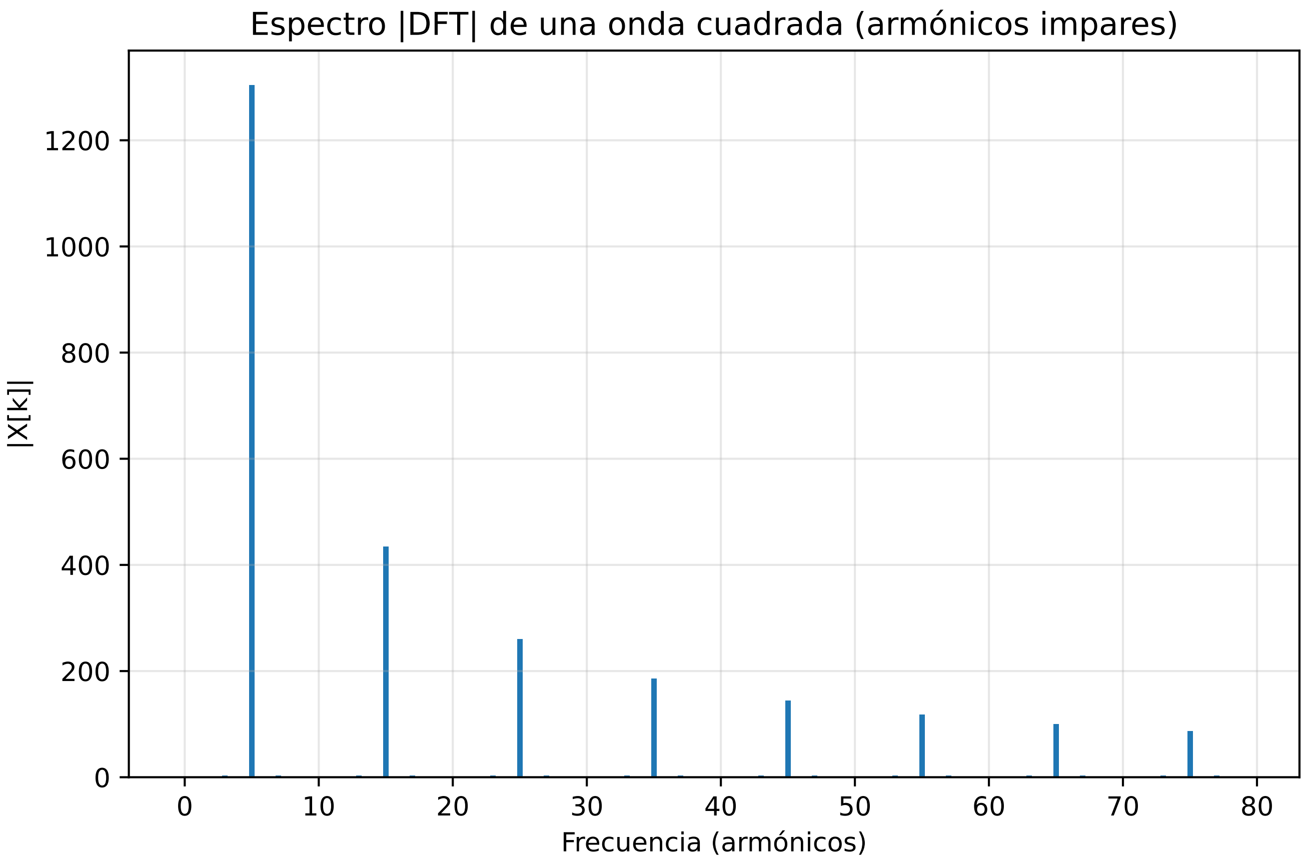

### Transformada Discreta de Fourier (DFT)

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi kn/N}, \quad k = 0, \dots, N-1.$$

**Ejemplo: onda cuadrada y su espectro**

```
import numpy as np
import matplotlib.pyplot as plt

N = 2048
t = np.arange(N)/N
f0 = 5
x_sig = np.sign(np.sin(2*np.pi*f0*t))

X = np.fft.rfft(x_sig)
freqs = np.fft.rfftfreq(N, d=1/N)

plt.figure(figsize=(8,5))
plt.bar(freqs[:80], np.abs(X)[:80], width=0.4)
plt.title("Espectro |DFT| de una onda cuadrada (armónicos impares)")
plt.xlabel("Frecuencia (armónicos)")
plt.ylabel("|X[k]|")
```

```
plt.grid(True, alpha=0.3)
plt.show()
```




### Otras transformadas

- **Transformada de Fourier continua:**

$$\mathcal{F}\{f\}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}\,dt.$$

- **Transformada de Laplace:**

$$\mathcal{L}\{f\}(s) = \int_{0}^{\infty} f(t)e^{-st}\,dt.$$

- **Transformada Z:**

$$X(z) = \sum_{n=0}^{\infty} x[n]z^{-n}.$$

  Cada una generaliza la idea de la serie de Fourier a dominios distintos (continuo, discreto o complejo). ## Conclusión

Las series de Fourier y sus transformadas asociadas son herramientas esenciales para el análisis de señales, resolución de ecuaciones diferenciales y modelado de fenómenos periódicos o transitorios.