

# 1 Aplicación de algoritmos genéticos al diseño de filtros IIR en punto fijo

## 1.1 Extracto

El desempeño de los filtros digitales en general, y de los filtros de respuesta impulsional infinita o IIR en particular<sup>1</sup>, se ve afectado por las representaciones numéricas finitas utilizadas para su implementación. Las herramientas de análisis y síntesis típicas<sup>2</sup>, lineales, no intentan modelar los efectos que impone una aritmética con restricciones<sup>3</sup>. Ésto se torna particularmente notorio en representaciones de punto fijo de relativo bajo número de bits<sup>4</sup> -valiosas por su sencillez, reducido costo y alta velocidad-, al punto de deformar completamente la respuesta del filtro proyectado.

Se vuelve necesario recurrir a otros métodos para arribar a una solución satisfactoria, maximizando una métrica cuantificable de ese grado de satisfacción, en ocasiones en detrimento de otros aspectos de la solución. Los algoritmos genéticos, una subclase dentro del conjunto de algoritmos evolutivos, llevan a cabo esta búsqueda mediante operaciones inspiradas en el proceso de selección natural. De carácter probabilístico, no garantizan la solución óptima pero permiten la exploración de universos de discurso extensos en un tiempo acotado.

<sup>1</sup> Por caso, la realimentación combinada con operatorias de truncado y desborde puede resultar en oscilaciones sostenidas.

<sup>2</sup> Por caso, la transformada Z y el mapeo bilineal en tándem con *frequency warping* para extender soluciones en tiempo continuo a tiempo discreto.

<sup>3</sup> Producto de acumuladores finitos para albergar los resultados parciales de sumas y multiplicaciones, el redondeo, truncado y/o saturación necesarios para mantener el resultado en valores representables, etc.

<sup>4</sup> Conforme el número de bits crece y se abandona el punto fijo por el punto flotante, la discrepancia entre representación continua y discreta se vuelve despreciable.

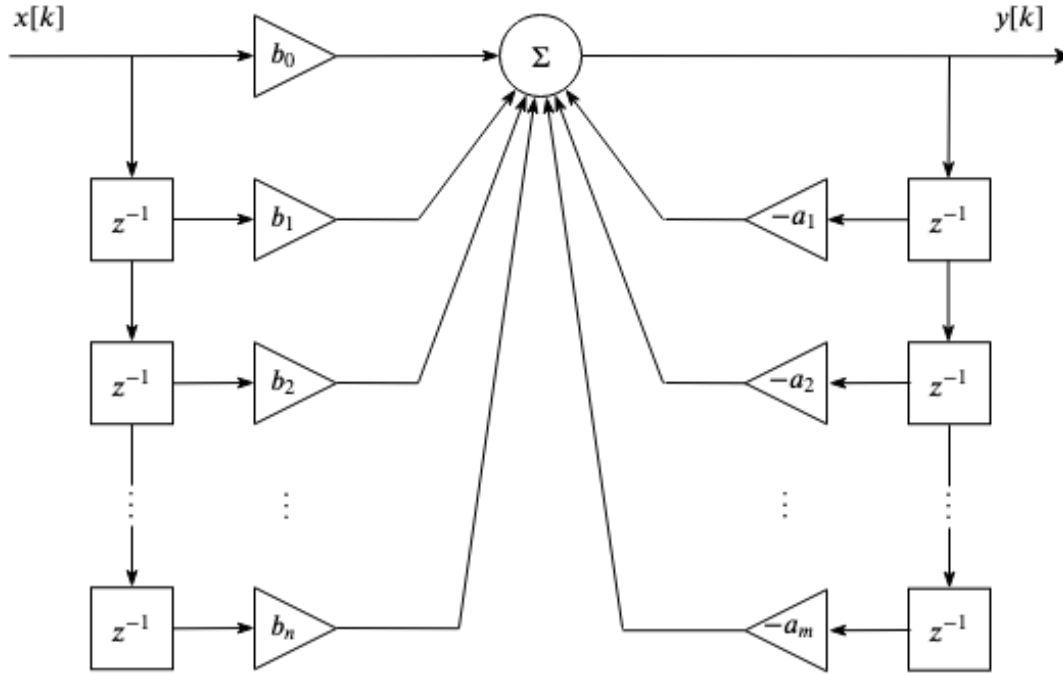
## 1.2 Estudio preliminar

### 1.2.1 Procesamiento digital de señales

**Filtrado en tiempo discreto** Todo filtro puede ser descripto por su ecuación en diferencias:

$$y[k] = \sum_{n=0}^N b_n \cdot x[k-n] - \sum_{m=1}^M a_m \cdot y[k-m] \quad (1)$$

Esta ecuación permite computar la respuesta muestra a muestra (respuesta temporal si se asume un período de muestreo  $t_s$  dado) e indica la estructura más simple (directa) del filtro.



Asumiendo coeficientes  $b_n$  y  $a_m$  invariantes en el tiempo, la transformada Z de la ecuación en diferencias resulta:

$$\frac{Y(z)}{X(z)} = \frac{\sum_{n=0}^N b_n \cdot z^{-n}}{1 + \sum_{m=1}^M a_m \cdot z^{-m}} z = re^{j\Omega} \quad (2)$$

Las raíces del polinomio numerador de coeficientes  $b_n$  constituyen los denominados ceros  $z_n$  de esta función transferencia y las raíces del polinomio denominador de coeficientes  $a_m$  constituyen los polos  $p_m$ . Para  $\|z\| = r = 1$ , obtenemos la respuesta en frecuencia del filtro (esto es, la transformada de Fourier en tiempo discreto).

**Efectos de la representación numérica** Los valores de entrada  $x[k]$ , de salida  $y[k]$ , y coeficientes  $b_n$  y  $a_m$  deben ser representados en forma binaria para su procesamiento. En lo que sigue, se asume una representación binaria signada en complemento a 2 de punto fijo de  $N_e \cdot N_f$  bits, que permite procesar magnitudes fraccionales utilizando unidades aritmético-lógicas para enteros signados:

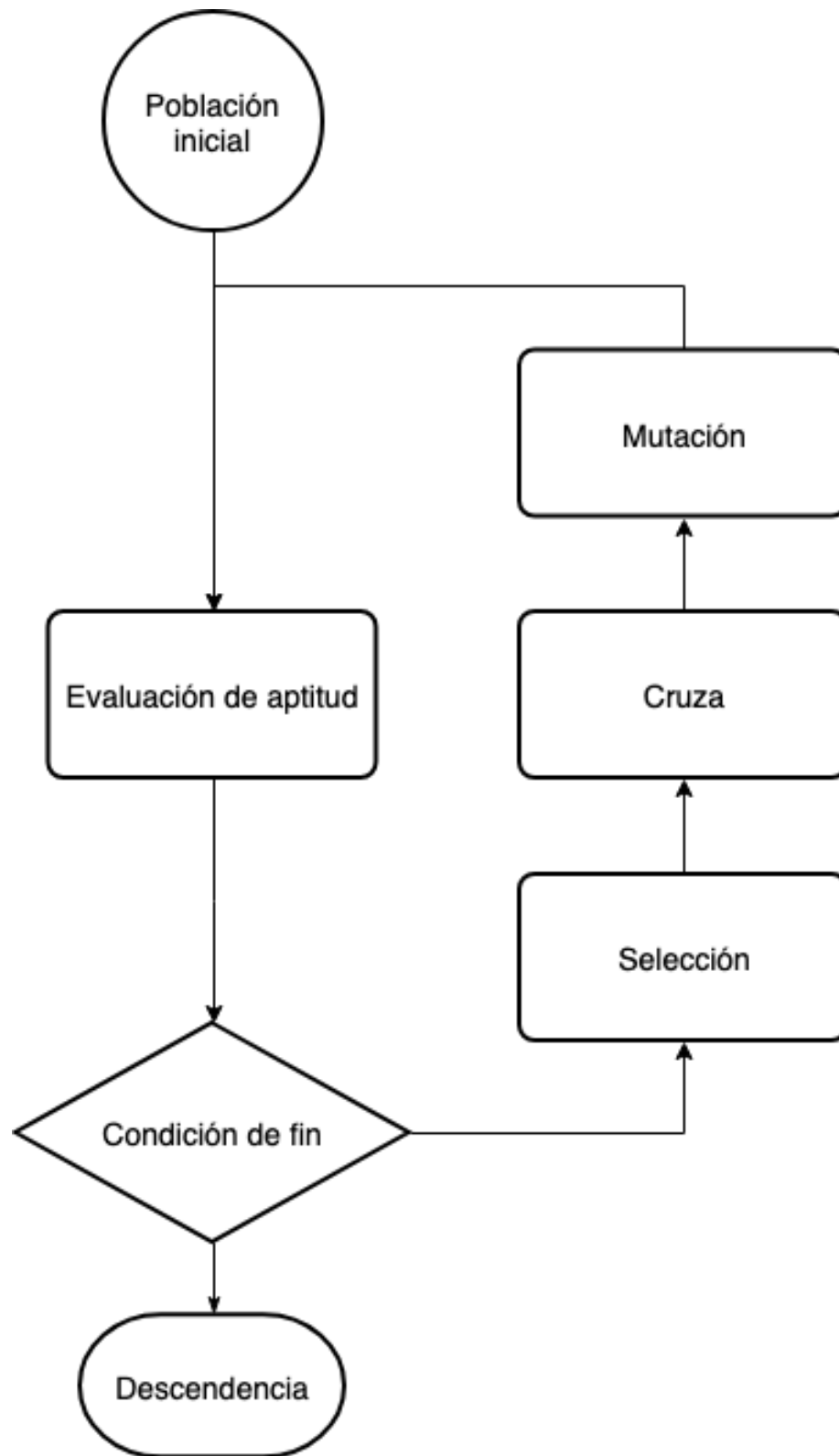
$$-2^{N_b-1} \leq x \leq 2^{N_b-1} - \frac{1}{2^{N_f}}$$

Coeficientes producto un diseño con mayor precisión numérica deben ser truncados o redondeados para ser representados de esta forma. El resultado parcial de cada multiplicación requiere  $2(N_b + N_f)$  bits para su representación exacta, lo cual también implica, tarde o temprano, su truncado o redondeo. Las sucesivas sumas deben alojarse en un acumulador de  $N_a$  bits, por lo que están sujetas a desborde o saturación.

Cada una de estas operatorias no lineales reduce el rango dinámico efectivo de la representación y distorsiona el filtro predeterminado por otros medios.

### **1.2.2 Algoritmo genético**

En su forma más simple, un algoritmo de esta clase puede ser descrito por el siguiente diagrama de flujo:



Una población inicial de individuos, en principio generada de forma aleatoria (pero bien puede

ser el resultado de conocimiento o procesos previos), es seleccionada en función de la aptitud de su fenotipo (o conjunto de características observables) y luego sometida a operadores de cruza y mutación de su genotipo (o conjunto de genes que codifica dichas características) para generar descendencia. Esta operatoria se repite a lo largo de generaciones hasta que una condición de fin o término dada se cumple. Los genes que redundan en individuos más aptos tenderán, en promedio, a incrementar su frecuencia de ocurrencia.

También en su forma más simple, la implementación de este algoritmo codifica los genes en secuencias binarias y opera sobre subconjuntos de éstas. Como se verá más adelante, el desempeño de esta heurística inspirada en la biología es fuertemente dependiente de la representación del genotipo de los individuos, de la diversidad genética inicial, de la clausura (o su ausencia) de los operadores genéticos respecto al universo de discurso y de la tendencia de los mismos a preservar los genes más aptos a través de las generaciones, etc., al punto de exhibir severos problemas de convergencia si el delicado balance entre los factores involucrados se ve afectado.

### 1.3 Aplicación

#### 1.3.1 Algoritmo

**Genotipo (o representación)** Sin hipótesis simplificadora alguna, una representación compacta de un filtro digital requiere de todos sus coeficientes.

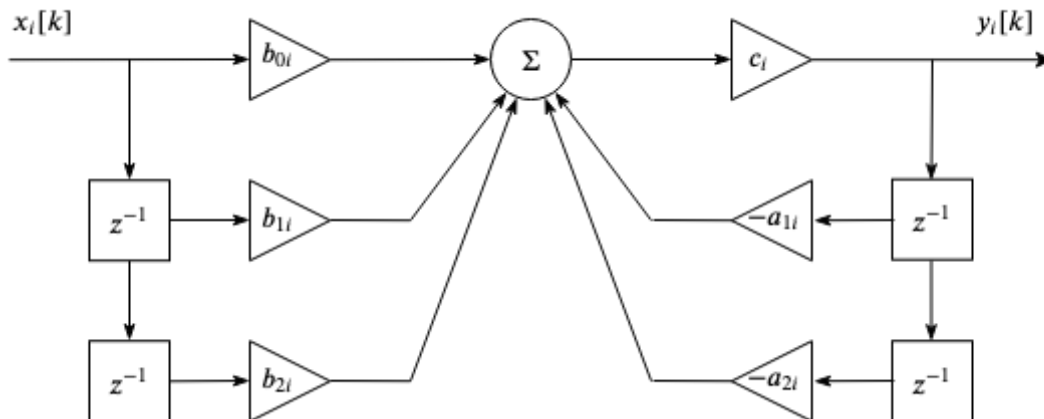
Parte de las dificultades en la síntesis de filtros digitales en punto fijo radica en el hecho de que el procedimiento normalmente se efectúa en  $\mathbb{R}$ , en lugar del subconjunto  $U \subset \mathbb{R}$  con aritmética modular en el que se lo implementa. Por ello, los coeficientes se representarán en  $1.N_b$  desde un comienzo.

La estructura del filtro digital también tiene consecuencias sobre su desempeño. La sensibilidad a los errores numéricos crece con el orden. Las sumas parciales pueden ocasionar el desborde o saturación de los acumuladores que las almacenan. La decomposición en secciones bicuadráticas, de segundo orden, y su ordenamiento tal que se minimice la posibilidad de desborde o saturación es una solución típica en implementaciones tradicionales a estos problemas.

Con estas consideraciones, el genotipo de cada filtro queda conformado por una cascada de secciones bicuadráticas  $[SOS_0, SOS_1, \dots, SOS_i]$ , donde la ecuación en diferencias de la sección  $SOS_i$  es:

$$y_i[k] = c_0 \cdot (b_0 \cdot x_i[k] + b_1 \cdot x_i[k-1] + b_2 \cdot x_i[k-2] - a_1 \cdot y_i[k-1] - a_2 \cdot y_i[k-2]) \quad (3)$$

o simplemente  $[b_0, b_1, b_2, a_1, a_2, c_0]$ .



Notar que el término  $a_0$  queda tácito, pues es siempre 1. Para lidiar con órdenes impares, una sección de primer orden puede introducirse haciendo  $b_2 = a_2 = 0$ . El agregado de una ganancia de etapa  $c_0$  permite cambios de escala de los coeficientes a modo de mantenerlos en el intervalo de valores representables.

## Operadores

**Evaluación** La aptitud de cada filtro se evalúa en relación al desempeño deseado. En la síntesis tradicional, el filtro exhibe una respuesta al impulso o una respuesta en frecuencia dadas o con ciertas características a partir de la utilización de teoría de aproximación, muestreo, convolución con ventanas, etc. En este caso, de forma empírica se ha determinado que la métrica de aptitud que mejor resultados arroja es:

$$f_k = \frac{1}{\mu_{er_k}^2 + \sigma_{er_k}^2}$$

donde

$$er_k[i] = \frac{h_k[i] - h_t[i]}{\max_i \|h_t[i]\|}$$

es el error relativo de la respuesta impulsional del filtro  $h_k$  respecto a la respuesta impulsional de la plantilla  $h_t$ ;

$$\mu_{er_k} = \frac{1}{N} \sum_{i=0}^N er_k[i]$$

es la media de dicho error relativo  $er_k$ ;

$$\sigma_{er_k}^2 = \frac{1}{N-1} \sum_{i=0}^N (er_k[i] - \mu_{er_k})^2$$

es la varianza de dicho error relativo  $er_k$ . El valor apropiado de N dependerá de cada respuesta a aproximar.

Observar que el denominador de la función  $f_k$  es numéricamente equivalente a la energía de  $er_k$  entendida como variable aleatoria.

Es importante mencionar, sin embargo, que el cómputo de la respuesta impulsional  $h_k$  durante y sólo durante la evaluación del filtro se realiza en su representación normalizada con la máxima precisión numérica disponible. El uso de aritmética en punto fijo durante el proceso evolutivo degrada severamente la convergencia.

**Selección** Los operadores de selección mediante muestreo de la población, tomando la aptitud, o bien una estratificación construida a partir de ella, como probabilidad de ocurrencia no ha mostrado buen desempeño. Frecuentemente, el algoritmo presentaba problemas de convergencia.

El operador de torneo, que consiste en la selección determinística del más apto en grupos de tamaño fijo (y ajustable) conformados de forma aleatoria, tantas veces como individuos hay en la población, ha mostrado mejores resultado y es la que esta instancia del algoritmo genético

emplea. Es, además, fácilmente ajustable: la presión de selección aumenta con el tamaño del torneo. Torneos grandes atentan contra la diversidad del pool genético y deben evitarse.

**Cruza** El operador de cruce recombina los genes de los progenitores, preservando<sup>5</sup> los genes deseables o aptos y eventualmente combinándolos en su descendencia.

Los operadores de cruce binaria, como ser el de corte en uno o dos puntos aplicados sobre la secuencia completa de coeficientes, no han mostrado buen desempeño, probablemente atribuible a la baja recombinación de singularidades (polos o ceros) que pueden inducir algunos cortes en la larga secuencia binaria de coeficientes y las significativas mutaciones que provocan en los puntos de corte.

La cruce uniforme de secciones bicuadráticas, es decir el intercambio probable de a pares ordenados de secciones de los progenitores para dar origen a su descendencia, ha mostrado mejores resultados. Más aún, la cruce uniforme de numeradores y denominadores de la función transferencia de cada sección, tomadas de a pares aleatorios ha mostrado aún mejores resultados y es la que esta instancia del algoritmo genético emplea. Se observa que este operador favorece el intercambio de polos y ceros y su redistribución.

<sup>5</sup> Si los genes no son preservados, el operador se torna una forma más de mutación.

**Mutación** El operador de mutación introducen (en principio) pequeñas perturbaciones en los genes que evitan la convergencia en extremos locales de la superficie de error y contribuyen a la exploración del universo de discurso. Este operador mantiene la diversidad genética.

Los operadores de mutación binaria, como ser la inversión de bits, no han mostrado buen desempeño, probablemente atribuible al peso desigual de cada bit en una representación numérica binaria y el consiguiente efecto en la magnitud de las perturbaciones (notar el impacto de la inversión del bit menos significativo comparado con el impacto de la inversión del bit más significativo o incluso el de signo). Se pierde control de la magnitud de la mutación.

La adición de una perturbación como variable aleatoria de distribución normal a los coeficientes, entendidos como valores numéricos, ha mostrado en cambio buenos resultados.

## Variaciones

**Población inicial** La generación de la población inicial, si bien aleatoria, se limita a conformar filtros con secciones de segundo orden estables y de fase mínima (es decir, polos y ceros dentro o sobre la circunferencia unidad en el plano  $Z$ ), asumiendo que la solución se hallará en la inmediaciones. Para ello, para cada sección de segundo orden:

$$H(z) = \frac{b(z)}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \quad (4)$$

y su recíproco, se aplica el criterio de estabilidad de Jury-Marden, que garantiza que dicha función transferencia es estable si:

$$\begin{aligned} a_0 &= 1 \\ \|a_2\| &< 1 \\ a_2 &> a_1 - 1, a_1 \geq 0 \\ a_2 &> -a_1 - 1, a_1 < 0 \end{aligned}$$

Adicionalmente, y pese a que el algoritmo no intenta minimizar orden, filtros con distinto número de secciones son gestados y sometidos al proceso evolutivo para luego optar por los más simples.

**Historial de más aptos** Los individuos más aptos a lo largo de todas las generaciones son conservados, evitando perder buenas soluciones producto del carácter aleatorio del proceso evolutivo.

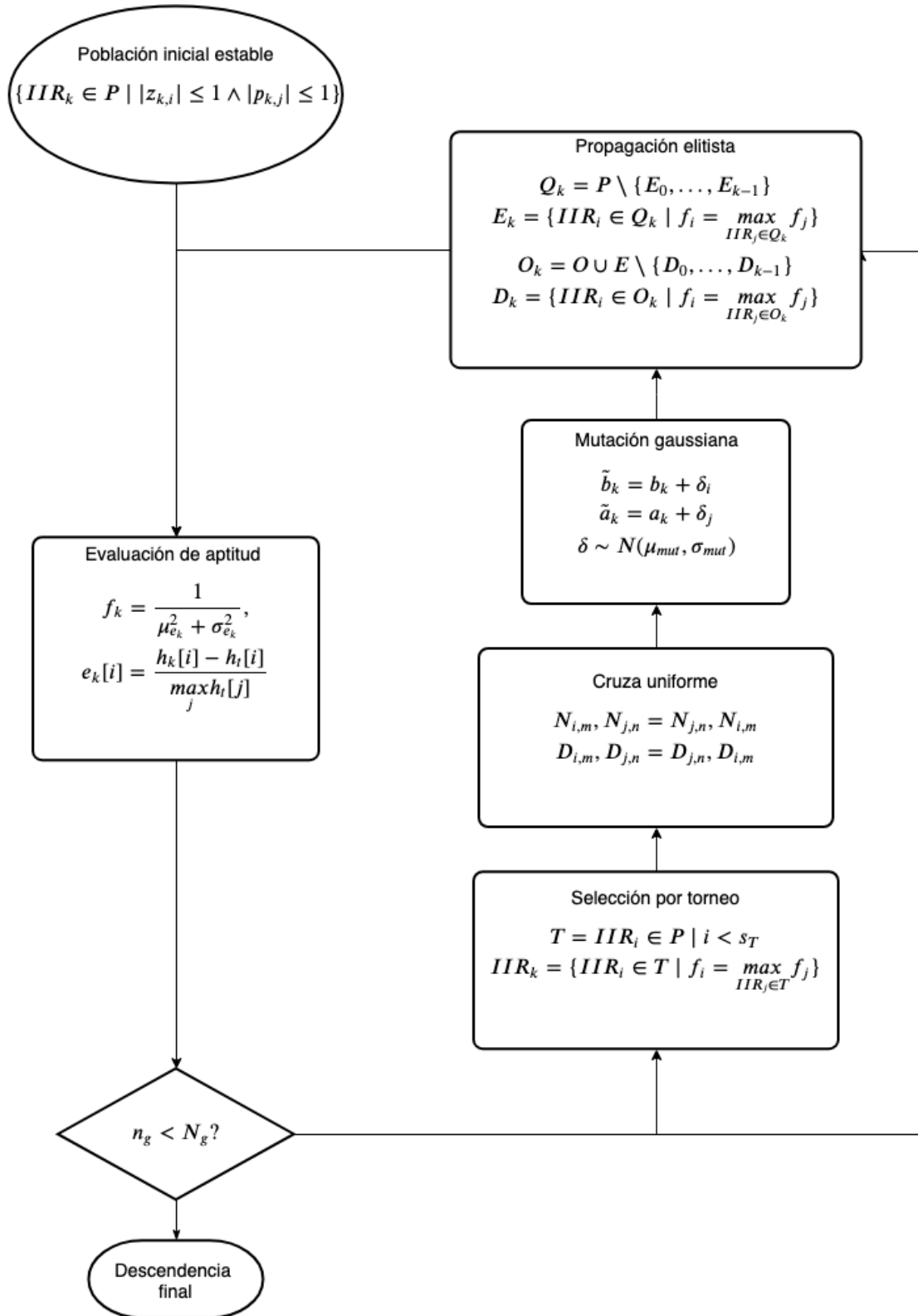
**Propagación elitista** Para mejorar la convergencia, pero limitando la exploración en el proceso, el recambio generacional de la población se ha modificado introduciendo una proporción de individuos de la población, los más aptos (o la *elite*), en su descendencia y seleccionando a los mejores del conjunto unión para la próxima generación.

Esta proporción y el tamaño de los grupos de selección por torneo están fuertemente ligados. Torneos grandes con una proporción de individuos de elite considerable provocan pérdida de diversidad y convergencia prematura.

**Condición de término** El algoritmo avanza a la población a lo largo de un número fijo de generaciones.



## Diagrama



### 1.3.2 Implementación

El algoritmo se implementó como módulo en Python 3.6, utilizando los paquetes [numpy](#), [scipy](#) y [deap](#).

Se reproduce debajo:

---

```
from deap import base
from deap import algorithms
from deap import tools
from deap import creator

import random
import functools
import numpy as np
import scipy.signal as signal

creator.create(
    'ResponseMismatch', base.Fitness, weights=(1.0,)
)
creator.create(
    'IIR', list, fitness=creator.ResponseMismatch, nbits=int
)
IIR = creator.IIR

def fitsos(sos, nbits):
    """
    Ajusta la ganancia de una etapa de segundo orden en punto fijo
    para que sus coeficientes puedan representarse en 1.(`nbits`-1)

    :param sos: etapa de segundo orden de filtro digital IIR
        en punto fijo.
    :param nbits: cantidad de bits de la representación en
        punto fijo de los coeficientes de la etapa.
    """
    # Computa el límite numérico de la representación entera signada.
    n = 2**(nbits - 1)
    # Busca el coeficiente de máximo valor absoluto.
    c = sos[np.argmax(np.abs(sos[-1] * sos[:-1]))]
    if abs(c) >= n:
        # Escala todos los coeficientes para que el coeficiente
        # de máximo valor absoluto sea -1 en punto fijo.
        sos[:-1] = -n * sos[-1] * sos[:-1] // abs(c)
        # Conserva el factor de escala aplicado.
        sos[-1] = min(-1, -abs(c) // n)
    return sos
```

```

def impulse(iir, ts, n):
    """
    Computation of the impulse response of a digital IIR filter
    at a fixed point.

    The filter is represented as a sequence of stages ordered by
    second order, each one represented by the coefficients of its
    difference equation plus a gain: [b0, b1, b2, a1, a2, k]. The coefficient a0
    is assumed to be unity. First order stages can be obtained by
    making b2 = a2 = 0.

    :param iir: digital IIR filter at fixed point.
    :param ts: sampling period, in seconds.
    :param n: number of samples.
    :return: time and impulse response, as a tuple.
    """
    # Initialize to zero the output vectors of
    # each one of the M stages of the filter plus the
    # input. Each vector can contain N samples
    # plus 2 additional samples to conform the
    # delay line.
    y = np.zeros([len(iir) + 1, n + 2], dtype=int)
    # Initialize the input vector with a discrete
    # delta.
    y[0, 2] = 2**(iir.nbits-1) - 1
    # Compute for each discrete instant the outputs
    # of each stage, from the first to the last
    # in that order.
    for j in range(2, n + 2):
        for i, sos in enumerate(iir, start=1):
            b0, b1, b2, a1, a2, k = sos
            # Compute the difference equation, truncating
            # and saturating the result to be represented
            # at fixed point 1.('nbits'-1)
            y[i, j] = np.clip((k * (
                b0 * y[i - 1, j] +
                b1 * y[i - 1, j - 1] +
                b2 * y[i - 1, j - 2] -
                a1 * y[i, j - 1] -
                a2 * y[i, j - 2]
            )) >> (iir.nbits - 1), -2**(iir.nbits - 1), 2**(iir.nbits - 1) - 1)
    # Return impulse response renormalizing the output
    # to the interval [-1, 1)
    # output of
    t = ts * np.arange(n)
    im = y[-1, 2:] / 2**(iir.nbits - 1)

```

```

return t, im

def iir2sos(iir):
    """
    Convierte un filtro digital IIR en punto fijo a su representación
    como secuencia de secciones de segundo orden en punto flotante (ver
    `scipy.signal.sos2tf` como referencia).

    :param iir: filtro digital IIR en punto fijo.
    :return: filtro digital en representación SOS.
    """
    # Computa el límite numérico de la representación entera signada.
    n = 2**(iir.nbits - 1)
    # Escala el filtro digital en punto fijo acorde a la ganancia y
    # normaliza al intervalo [-1, 1) en punto flotante.
    return np.array([
        (*(sos[-1] * sos[:3] / n), 1.,
        *(sos[-1] * sos[3:5] / n))
        for sos in iir
    ])

def genStablePrototype(nlimit, nbits=32):
    """
    Genera un filtro digital IIR en punto fijo estable en
    forma aleatoria.

    :param nlimit: orden máximo admitido para el filtro.
    :param nbits: cantidad de bits utilizados para la
        representación numérica de los coeficientes.
    :return: filtro digital IIR en punto fijo generado.
    """
    iir = IIR()
    # Computa el límite numérico de la representación entera signada.
    n = 2 ** (nbits - 1)
    # Selecciona el orden del filtro en forma aleatoria
    # del intervalo [1, nlimit].
    order = max(int(random.random() * (nlimit + 1)), 1)
    # Si el orden es impar se introduce una etapa de primer orden.
    if order % 2 != 0:
        # Cero y polo de la etapa se ubican dentro o sobre el
        # círculo unidad.
        b0 = n
        b1 = np.random.randint(-n, n-1)
        a1 = np.random.randint(-n, n-1)
        sos = np.array([b0, b1, 0, a1, 0, 1])
        # Ajusta la ganancia de la sección para su representación.

```

```

        fitsos(sos, nbits)
        # Incorpora la etapa al filtro.
        iir.append(sos)
# Introduce N etapas de segundo orden para alcanzar
# el orden seleccionado.
for _ in range(order // 2):
    # Ceros y polos de la etapa se ubican dentro del círculo unidad.
    b0 = n
    b2 = np.random.randint(-n+1, n-1)
    a2 = np.random.randint(-n+1, n-1)
    b1 = np.random.randint(-b2-n, b2+n)
    a1 = np.random.randint(-a2-n, a2+n)
    sos = np.array([b0, b1, b2, a1, a2, 1])
    # Ajusta la ganancia de la sección para su representación.
    fitsos(sos, nbits)
    # Incorpora la etapa al filtro.
    iir.append(sos)
if hasattr(iir, 'nbits'):
    # Preserva el número de bits en el filtro.
    iir.nbits = nbits
return iir

def cxUniformND(iir1, iir2, ndpb):
    """
    Cruza numeradores y denominadores de filtros digitales IIR en
    punto fijo, potencialmente de distinto orden, produciendo dos
    descendientes. El orden de las etapas a cruzar es modificado
    aleatoriamente. Variante de `deap.tools.cxUniform`.

    :param iir1: primer filtro progenitor.
    :param iir2: segundo filtro progenitor.
    :param ndpb: probabilidad de cruza de numerador y/o denominador.
    """
    # Tomando el filtro candidato de menor orden, itera las
    # secciones de a pares tomados en forma aleatoria.
    for i, j in zip(
        random.sample(list(range(len(iir1))), len(iir1)),
        random.sample(list(range(len(iir2))), len(iir2))
    ):
        # Obtiene las etapas de cada filtro a cruzar.
        sos1 = iir1[i]
        sos2 = iir2[j]
        if random.random() < ndpb:
            # Cruza los numeradores de las etapas
            sos1[:3], sos2[:3] = sos2[:3], sos1[:3]
        if random.random() < ndpb:
            # Cruza los denominadores de las etapas

```

```

        sos1[3:5], sos2[3:5] = sos2[3:5], sos1[3:5]
        # Ajusta la ganancia de la primera sección para que los
        # coeficientes del filtro puedan representarse en punto
        # fijo para el número de bits del filtro candidato.
        fitsos(sos1, iir1.nbits)
        # Ajusta la ganancia de la primera sección para que los
        # coeficientes del filtro puedan representarse en punto
        # fijo para el número de bits del filtro candidato.
        fitsos(sos2, iir2.nbits)
    return iir1, iir2

def evTimeResponse(iir, target, ts):
    """
    Evalúa la aptitud de la respuesta temporal de un filtro
    digital IIR en punto fijo según la similitud que su
    respuesta temporal presenta respecto a la respuesta
    objetivo.

    :param iir: filtro digital IIR en punto fijo.
    :param target: respuesta al impulso objetivo.
    :param ts: período de muestreo, en segundos.
    :return: aptitud del filtro provisto.
    """
    # Computa la respuesta al impulso del filtro candidato
    # en su representación SOS.
    _, (im,) = signal.dimpulse(
        (*signal.sos2tf(iir2sos(iir)), ts), n=len(target)
    )
    # Computa el error relativo entre respuesta al impulso
    # del filtro candidato y respuesta al impulso esperada.
    et = (im - target) / np.max(np.abs(target))
    # Evalua la aptitud del filtro candidato como el recíproco
    # de la potencia de error relativo.
    return (1. / (np.mean(et)**2 + np.var(et)),)

def mutCoeffGaussian(iir, mu, sigma, indpb):
    """
    Muta los coeficientes de un filtro digital IIR en punto
    fijo mediante perturbaciones numéricas. Variante de
    `deap.tools.mutGaussian`.

    :param mu: media de la distribución gaussiana de la que se toman
        las perturbaciones a aplicar.
    :param sigma: desvío estandar de la distribución gaussiana de la
        que se toman las perturbaciones a aplicar.
    :param indpb: probabilidad de perturbar un coeficiente.

```

```

"""
# Itera cada sección del filtro.
for sos in iir:
    # Conformar una máscara de los coeficientes de la
    # sección actual del filtro, según la probabilidad
    # dada.
    mask = (np.random.random(len(sos)-1) < indpb)
    # Perturba los coeficientes a partir de una distribución
    # normal con media y desvío estándar dados.
    sos[:-1][mask] += np.random.normal(
        mu, sigma, np.count_nonzero(mask)
    ).astype(int)
    # Ajusta la ganancia de la sección para que los coeficientes
    # del filtro puedan representarse en punto fijo para el
    # número de bits del filtro.
    fitsos(sos, iir.nbits)
return iir,

def eaSimplePlusElitism(population, toolbox, cxpb, mutpb, eprop, ngen,
                        stats=None, halloffame=None, verbose=__debug__):
    """
    Variante de `deap.algorithms.eaSimple` con una proporción de elitismo.
    """
    logbook = tools.Logbook()
    logbook.header = ['gen', 'nevals'] + (stats.fields if stats else [])

    # Evalúa los individuos con aptitud inválida.
    invalid_ind = [ind for ind in population if not ind.fitness.valid]
    fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    if halloffame is not None:
        halloffame.update(population)

    record = stats.compile(population) if stats else {}
    logbook.record(gen=0, nevals=len(invalid_ind), **record)
    if verbose:
        print(logbook.stream)

    # Comienza el proceso evolutivo.
    for gen in range(1, ngen + 1):
        # Selecciona la próxima generación de individuos.
        offspring = toolbox.select(population, len(population))

        # Varía el pool de individuos, aplicando cruza y mutación.
        offspring = algorithms.varAnd(offspring, toolbox, cxpb, mutpb)

```

```

    # Evalua los individuos con aptitud inválida.
    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    # Actualiza el grupo de mejores individuos.
    if halloffame is not None:
        halloffame.update(offspring)

    # Reemplaza la población actual con los mejores del conjunto
    # compuesta por su descendencia y la elite.
    elite_count = int(len(population) * eprop)
    elite = tools.selBest(population, elite_count)
    population[:] = tools.selBest(offspring + elite, len(population))

    # Toma nota de las estadísticas de la generación actual.
    record = stats.compile(population) if stats else {}
    logbook.record(gen=gen, nevals=len(invalid_ind), **record)
    if verbose:
        print(logbook.stream)

return population, logbook

def configure_genetic_approx(*, nbits=16, nlimit=8, nsln=10, cxpb=0.7, ndpb=0.5,
                             mutpb=0.2, mutmean=0.0, mutstd=0.3, coeffpb=0.1,
                             tournsize=3, nind=1000, eprop=0.005, ngen=400,
                             verbose=__debug__):
    """
    Configura una función de aproximación de filtros digitales IIR en punto
    fijo con una dada respuesta al impulso.

    :param nbits: cantidad de bits utilizados para la representación numérica
        de los coeficientes la solución.
    :param nlimit: orden máximo admitido para la solución.
    :param nsln: cantidad de soluciones a conservar de entre las más aptas.
    :param cxpb: probabilidad de cruza genética de soluciones.
    :param ndpb: probabilidad de intercambio de numerador y denominador
        de soluciones seleccionadas para la cruza.
    :param mutpb: probabilidad de mutar una solución.
    :param mutmean: media de perturbaciones utilizadas para la mutación,
        en el intervalo [-1, 1).
    :param mutstd: desvío estándar de las perturbaciones utilizadas para
        la mutación, en el intervalo [-1, 1).
    :param coeffpb: probabilidad de perturbación de un coeficiente de
        una solución seleccionada para la mutación.

```



```

:param tournsize: cantidad de soluciones a someter en cada instancia
de torneo de selección.
:param nind: cantidad de soluciones en el pool genético a explorar.
:param eprop: proporción de elitismo o la cantidad de soluciones de elite
respecto al total de soluciones en el pool.
:param ngen: cantidad de generaciones a evolucionar.
:return: función de aproximación de filtro digital IIR en punto fijo.
"""
def approx(target, ts, nlimit=nlimit, nsln=nsln):
    """
    Aproxima un filtro digital IIR en punto fijo para que presente la
    respuesta al impulso objetivo.

    :param target: respuesta al impulso objetivo.
    :param ts: período de muestreo, en segundos.
    :param nlimit: orden máximo admitido para la solución.
    :param nsln: cantidad de soluciones a conservar de entre las más aptas.
    :return: filtro digital IIR en punto fijo.
    """
    toolbox = base.Toolbox()
    toolbox.register(
        'individual', genStablePrototype, nlimit=nlimit, nbits=nbits
    )
    toolbox.register(
        'population', tools.initRepeat, list, toolbox.individual
    )
    toolbox.register('mate', cxUniformND, ndpb=ndpb)
    toolbox.register('select', tools.selTournament, tournsize=tournsize)
    toolbox.register(
        'mutate', mutCoeffGaussian, mu=mutmean*2**(nbits-1),
        sigma=mutstd*2**(nbits-1), indpb=coeffpb
    )
    toolbox.register('evaluate', evTimeResponse, target=target, ts=ts)

    stats = tools.Statistics(
        lambda individual: individual.fitness.values
    )
    stats.register('mean_fitness', np.mean)
    stats.register('fitness_stddev', np.std)
    stats.register('min_fitness', np.min)
    stats.register('max_fitness', np.max)

    hall = tools.HallOfFame(
        maxsize=nsln, similar=lambda x, y: (
            np.all(np.equal(np.shape(x), np.shape(y)))
            and np.all(np.equal(x, y))
        )
    )

```

```

population = toolbox.population(nind)
offspring, logbook = eaSimplePlusElitism(
    population, toolbox, cxpb=cxpb, mutpb=mutpb,
    eprop=eprop, ngen=ngen, stats=stats, halloffame=hall,
    verbose=verbose
)

return hall, offspring, logbook
return approx

```

---

### 1.3.3 Desempeño

A continuación, se aproximan las respuestas impulsionales de filtros aproximados analíticamente y se contrastan los resultados obtenidos por el algoritmo para un filtro IIR en punto fijo 1.15 (típico Q15 de 16 bits) con la solución cerrada. En cada una de las comparaciones que siguen, los siguientes parámetros permanecen constantes:

Parámetro	Valor
Máximo orden del filtro	10
Frecuencia de sampleo	44 kHz
Número de bits	16
Número de individuos	500
Tamaño de torneo de individuos	3
Probabilidad de cruza de individuos	0.7
Probabilidad de cruza de num. y den.	0.5
Probabilidad de mutación de individuos	0.2
Probabilidad de mutación de coeficiente	0.1
Media de perturbación	0.0
Desvío estándar de perturbación	0.3
Número de generaciones	500
Proporción de elitismo	0.5%
Tamaño de historial de individuos	10

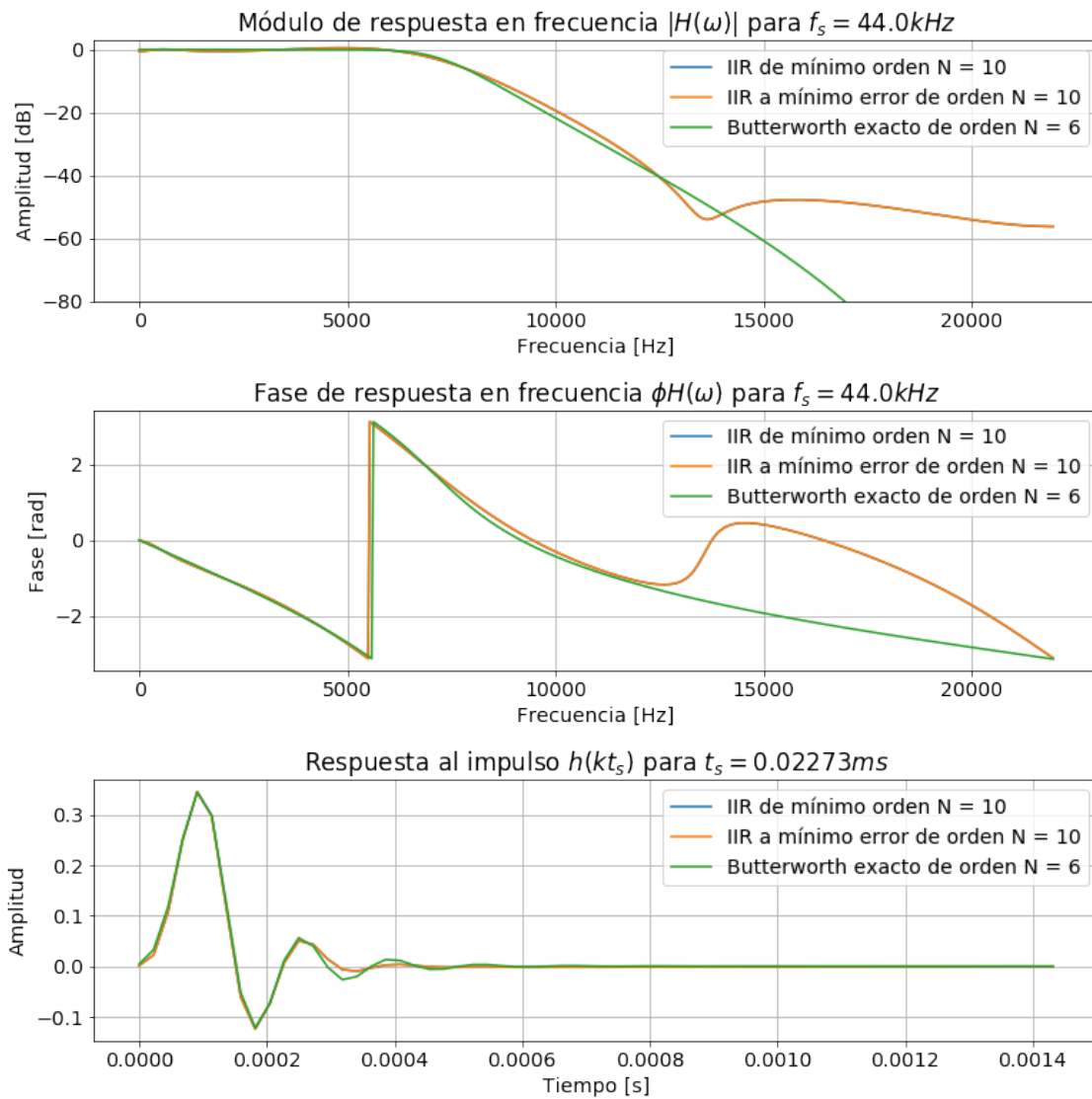
**Filtro pasa bajo tipo Butterworth** La aproximación de Butterworth logra máxima planicidad de módulo de respuesta en la banda de paso, a costa de menor pendiente en la banda de transición hacia la banda de paso. Para el caso del filtro pasa bajo en tiempo continuo, esta aproximación puede describirse mediante:

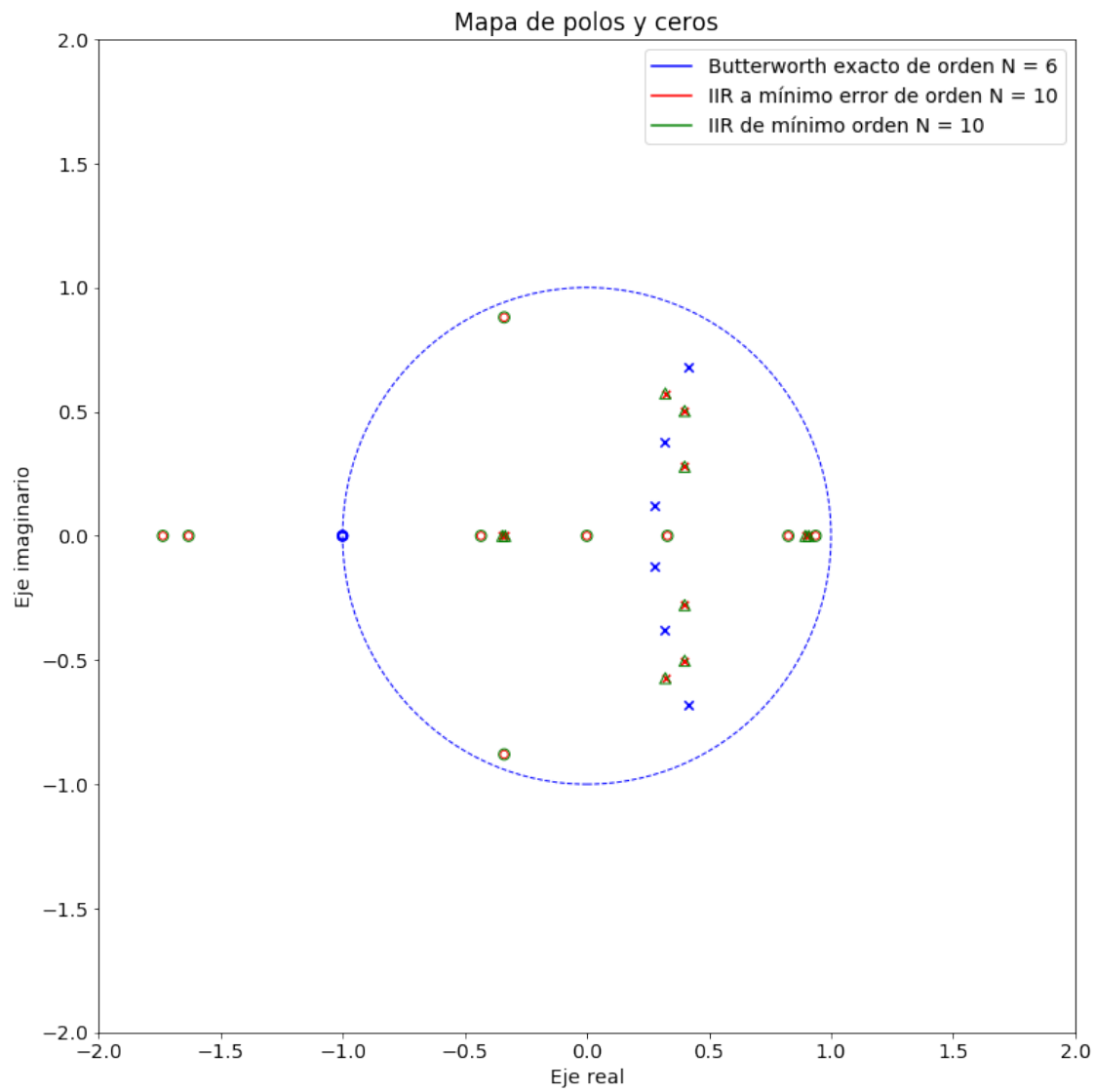
$$\|H(\omega)\|^2 = \frac{1}{\sqrt{1 + \epsilon^2 \frac{\omega^2}{\omega_c^2}}}$$

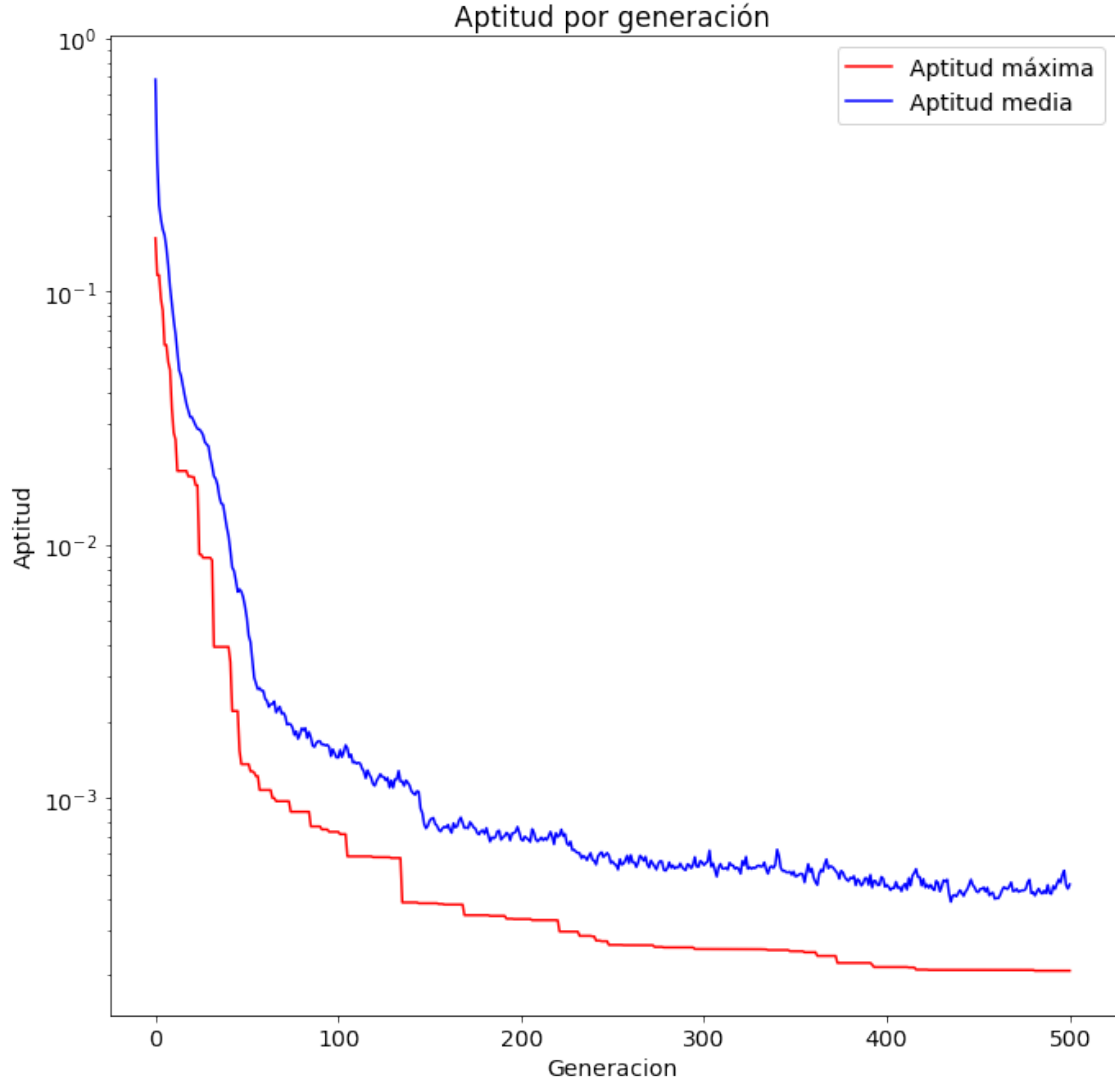
De  $\omega$  se mapea a  $s = \sigma + j\omega$ , para luego arribar a  $z$  vía transformada bilineal  $s = \frac{2}{t_s} \frac{z-1}{z+1}$  y posterior corrección por *frequency warping*.

Se diseña entonces un filtro pasa bajos con atenuación de  $\leq 1dB@6.6kHz$  y de  $\geq$

40dB@13.2kHz, para luego utilizar la respuesta impulsional resultante para aproximar la implementación con filtro IIR en punto fijo 1.15.







**Filtro pasa alto tipo Chebyshev** La aproximación de Chebyshev minimiza el error respecto a la plantilla de módulo de atenuación a costa de ripple en la banda de paso. Para el caso del filtro pasa bajo en tiempo continuo, esta aproximación puede describirse mediante:

$$\|H(\omega)\|^2 = \frac{1}{\sqrt{1 + \epsilon^2 T_n^2(\frac{\omega}{\omega_c})}}$$

donde

$$T_0(x) = 0 \tag{5}$$

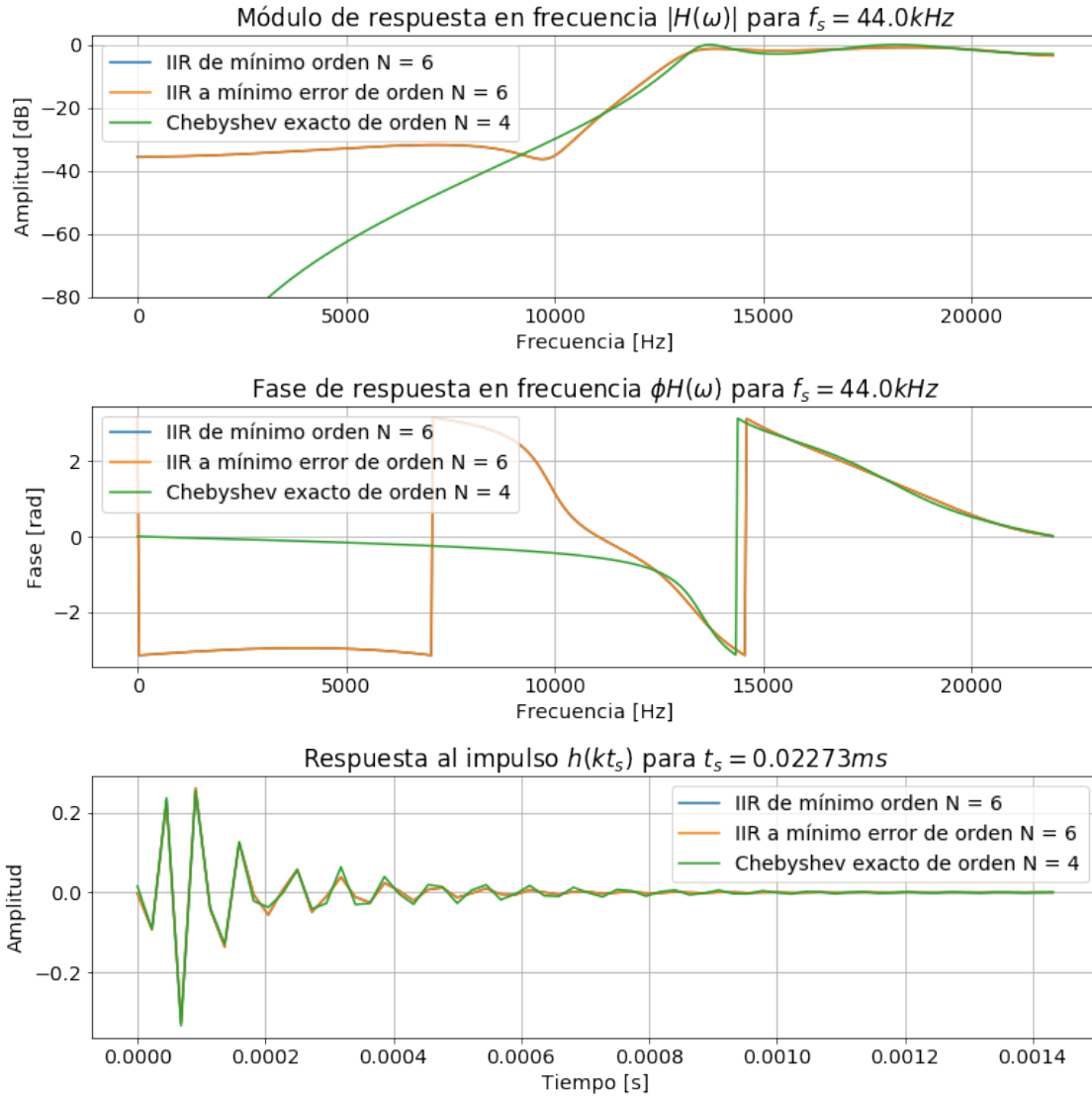
$$T_1(x) = x \tag{6}$$

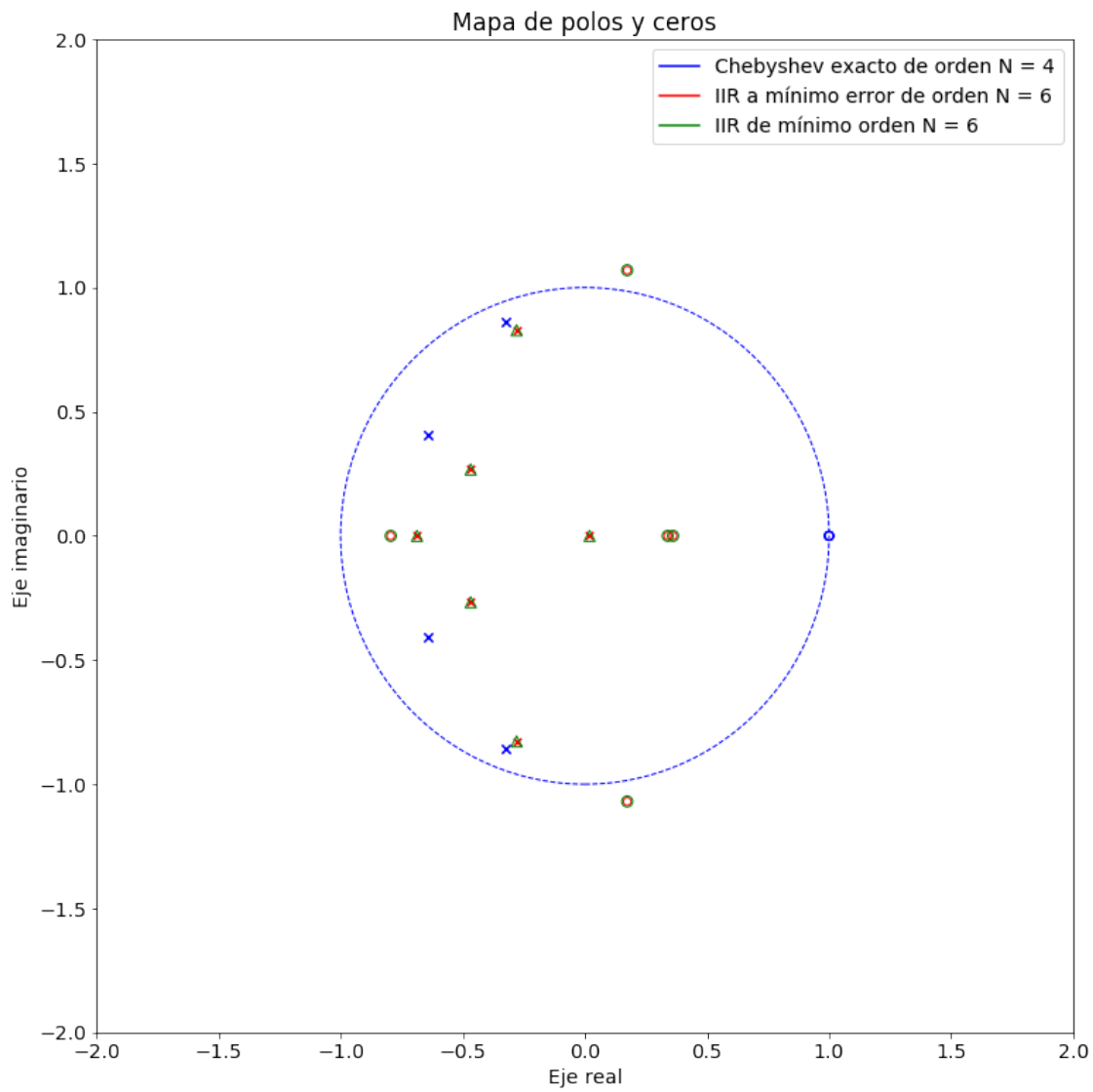
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \tag{7}$$

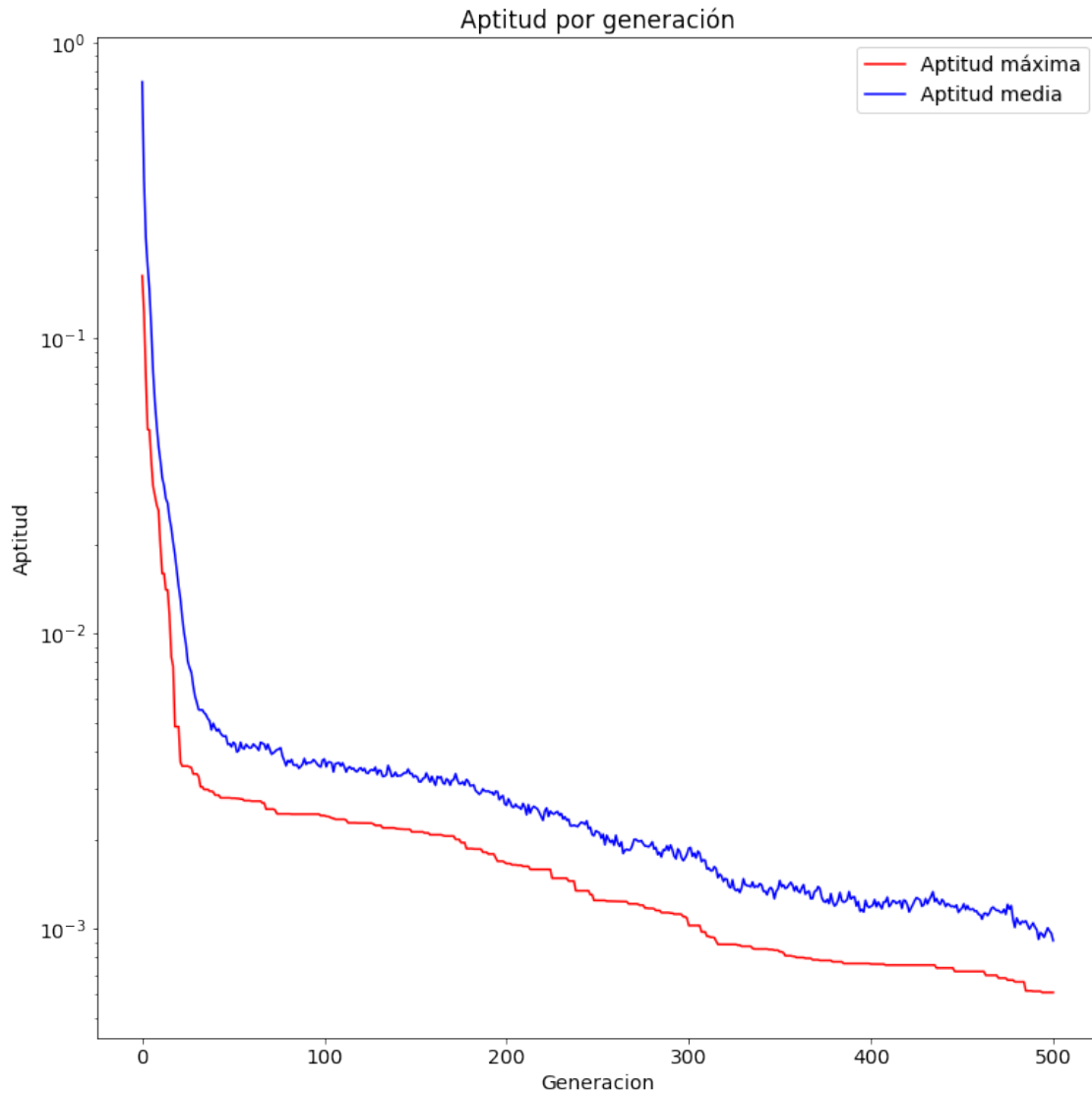
son los polinomios de Chebyshev del primer tipo.

De  $\omega$  se mapea a  $s = \sigma + j\omega$  y se aplica el kernel de transformación  $s = \frac{1}{s'}$ , para luego arribar a  $z$  vía transformada bilineal  $s' = \frac{2}{t_s} \frac{z-1}{z+1}$  y posterior corrección por *frequency warping*.

Se diseña entonces un filtro pasa altos con atenuación de  $\geq 40\text{dB}$  @  $f = 6.6\text{kHz}$  y de  $\leq 3\text{dB}$  @  $f = 13.2\text{kHz}$ , para luego utilizar la respuesta impulsional resultante para aproximar la implementación con filtro IIR en punto fijo 1.15.







**Filtro pasa banda tipo Chebyshev Inverso** La aproximación de Chebyshev inversa no presenta ripple en la banda de paso sino en la banda de atenuación, a costa de una pendiente en la banda de transición. Para el caso del filtro pasa bajo en tiempo continuo, esta aproximación puede describirse mediante:

$$\|H(\omega)\|^2 = \frac{1}{\sqrt{1 + \frac{1}{\epsilon^2 T_n^2(\frac{\omega}{\omega_c})}}}$$

donde



$$T_0(x) = 0 \quad (8)$$

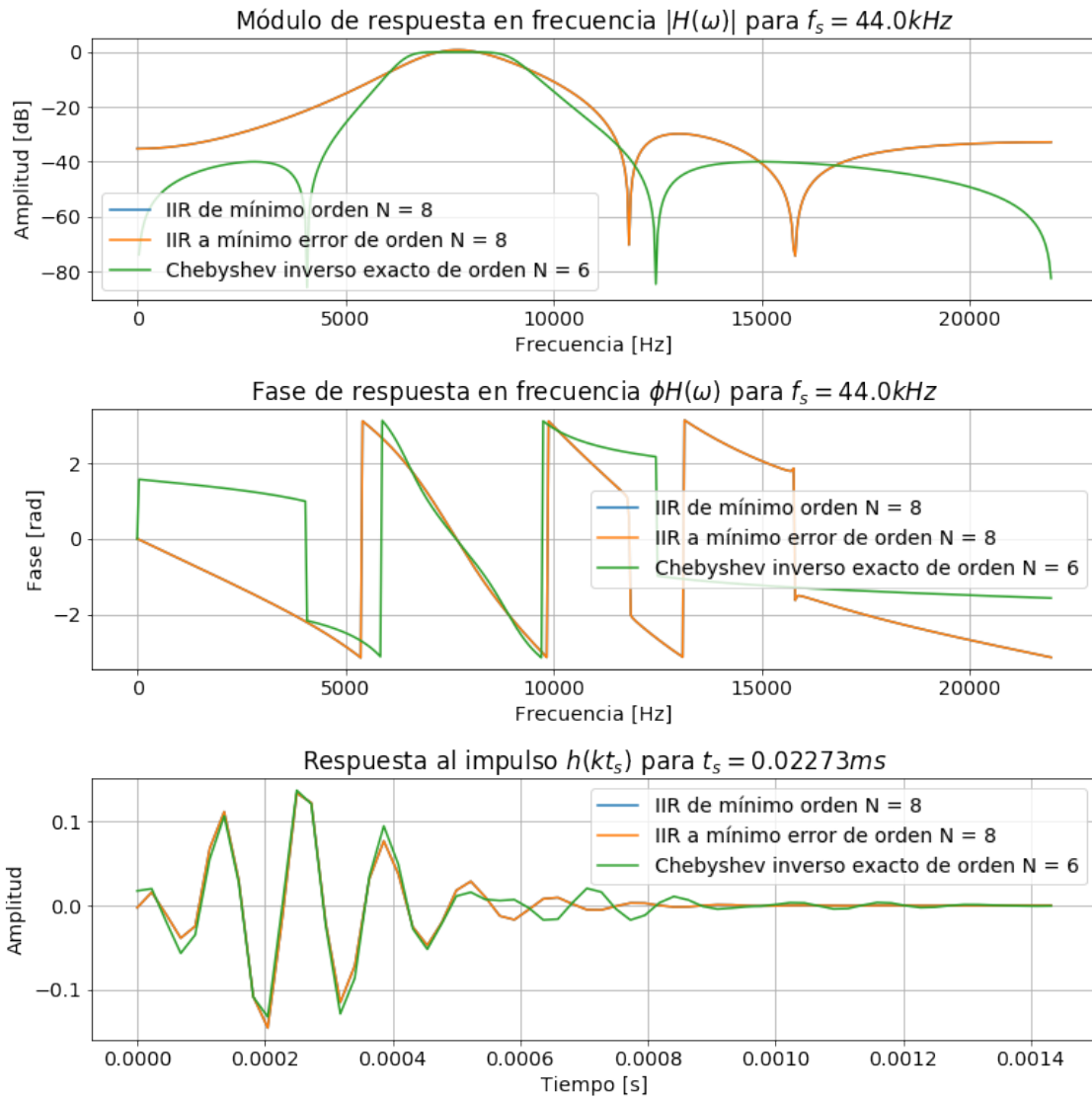
$$T_1(x) = 1 \quad (9)$$

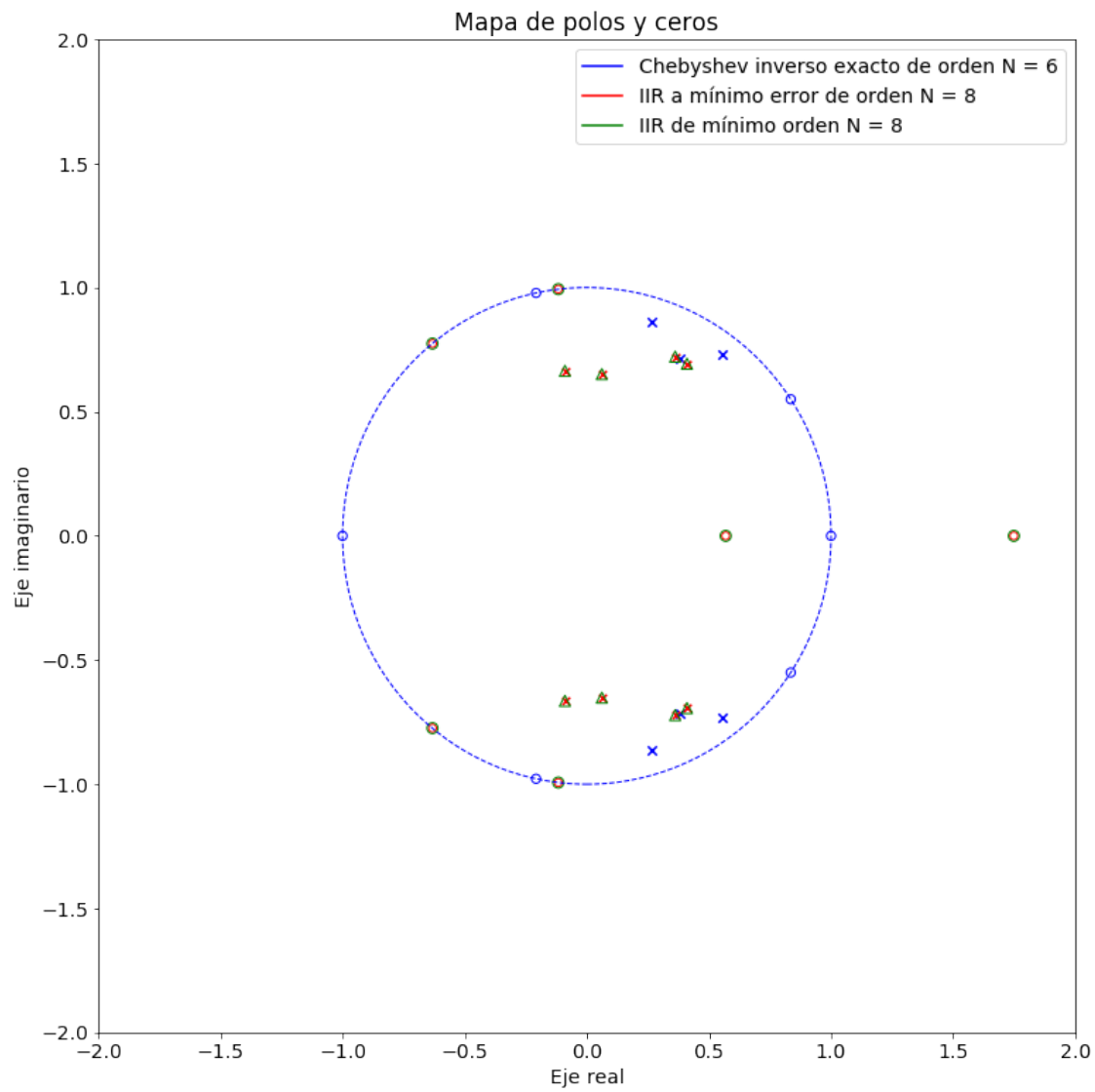
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad (10)$$

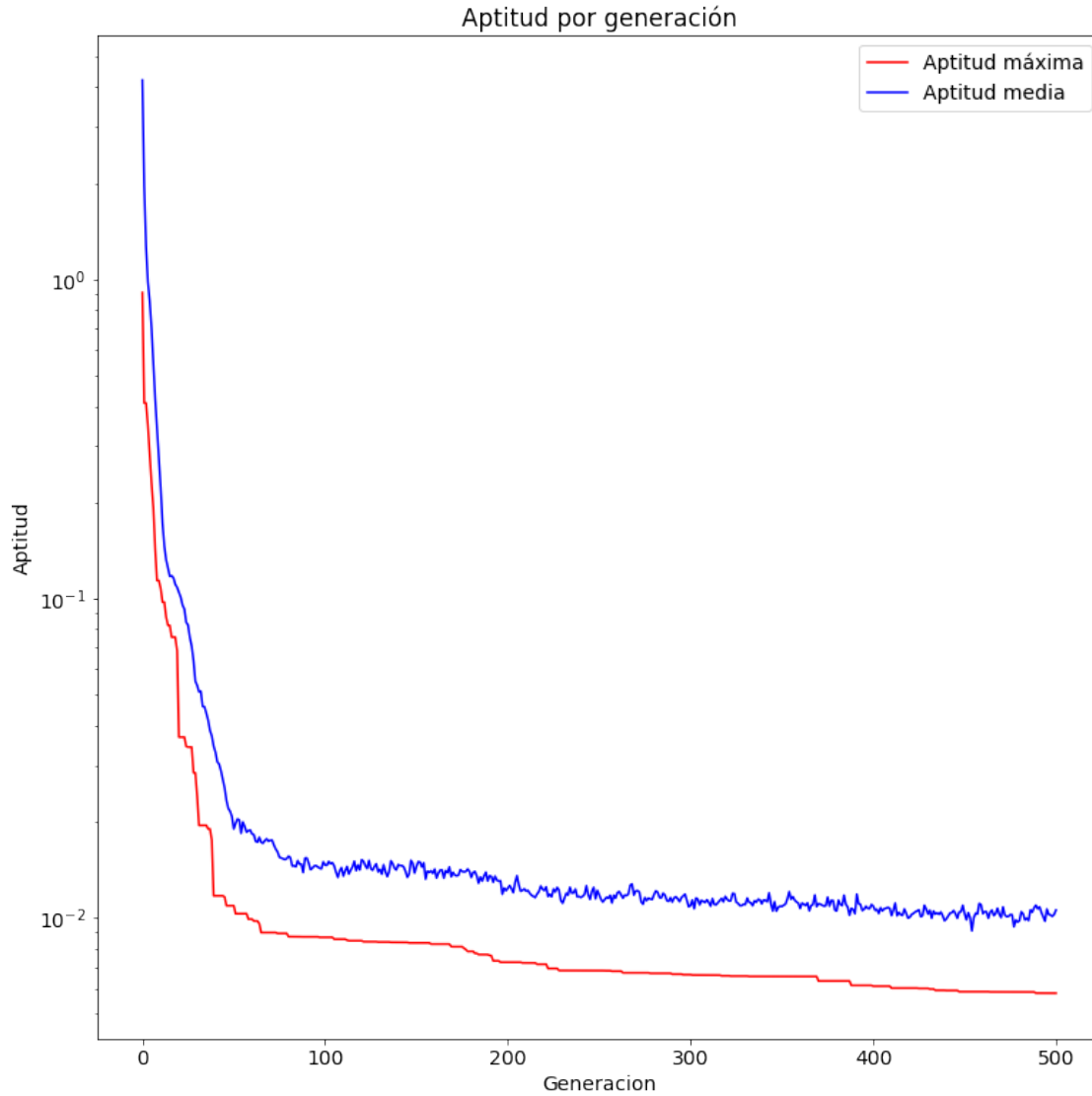
son los polinomios de Chebyshev del primer tipo.

De  $\omega$  se mapa a  $s = \sigma + j\omega$  y se aplica el kernel de transformación  $s = Q(\frac{s'}{\omega_o} + \frac{\omega_o}{s'})$ , para luego arribar a  $z$  vía transformada bilineal  $s' = \frac{2}{t_s} \frac{z-1}{z+1}$  y posterior corrección por *frequency warping*.

Se diseña entonces un filtro pasa banda con atenuación  $\leq 1dB @ 6.6kHz < f < 8.8kHz$  y  $\geq 40dB @ f > 13.2kHz \vee f < 2.2kHz$ , para luego utilizar la respuesta impulsional resultante para aproximar la implementación con filtro IIR en punto fijo 1.15.







**Filtro multi banda tipo FIR con ventana Hamming** Los filtros FIR o de respuesta impulsional finita pueden ser descriptos empleando la ecuación en diferencias de los filtros IIR con  $a_0 = 1 \wedge a_m = 0 \forall m \neq 0$ . Para su diseño, el uso del método de ventana parte de una plantilla de módulo de respuesta en frecuencia ideal o *brickwall*, simétrica respecto a las frecuencias negativas. La respuesta impulsional asociada es una sumatoria de senos cardinales, ergo no causal ni finita. Se procede entonces a la aplicación de una ventana de morfología apropiada que acote temporalmente dicha respuesta.

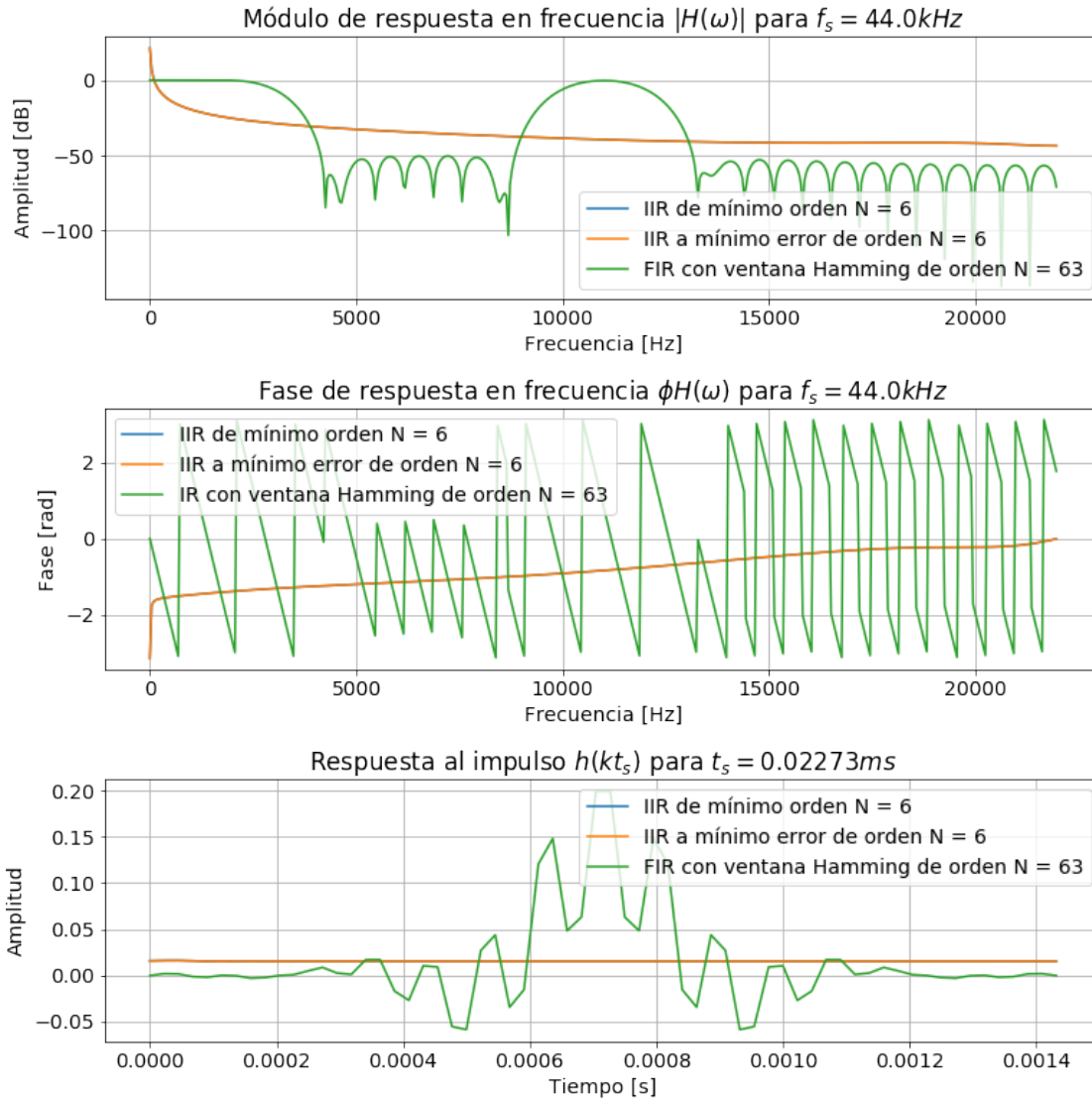
$$h_w[k] = h[k] \cdot w[k]$$

Una de muchas ventanas es la de Hamming:

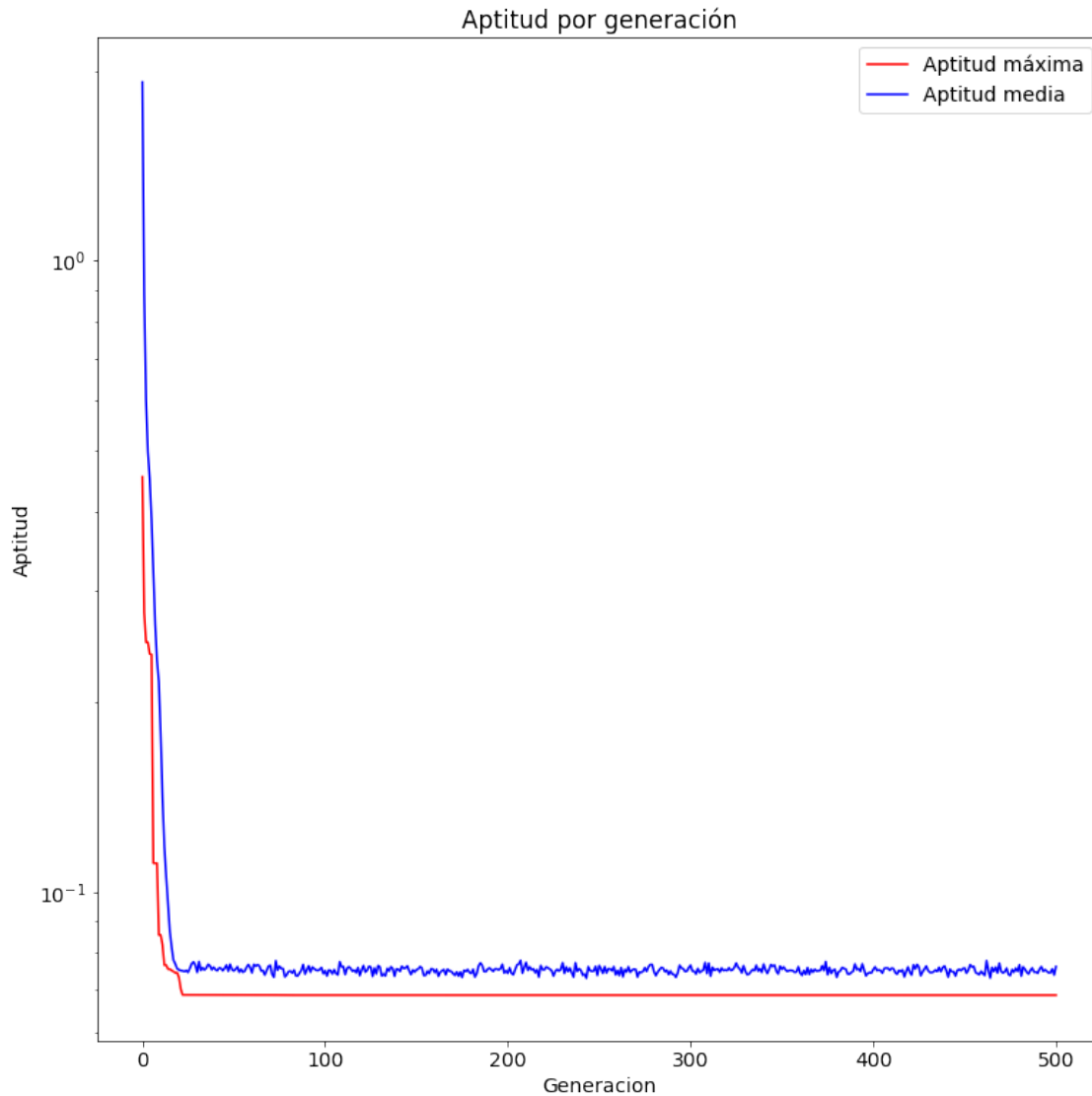
$$w[k] = 0.53836 - 0.46164 \cos\left(\frac{2\pi k}{N-1}\right)$$

donde  $0 \leq k < N$ . Esta ventana es similar a una ventana de coseno elevado pero, a diferencia de ésta, cancela el primer lóbulo de la respuesta.

Se diseña entonces un filtro multi banda con  $T = 64$  coeficientes o *taps* y bandas de paso  $0\text{Hz} \leq f < 3\text{KHz}$  y  $10\text{KHz} \leq f < 12\text{KHz}$ , para luego utilizar la respuesta impulsional resultante para aproximar la implementación con filtro IIR en punto 1.15.







## 1.4 Conclusiones

Se ha descrito un algoritmo capaz de aproximar respuestas impulsionales, y con menor exactitud la respuesta en frecuencia asociada, en filtros digitales IIR en punto fijo como cascada de secciones bicuadráticas.

En términos de desempeño, el conjunto se mantiene estable a pesar de las limitaciones de la representación numérica y la aritmética asociada. Como contrapartida, la fase de la respuesta en frecuencia resultante guarda poca o ninguna relación con la plantilla y el módulo exhibe notables errores (en escala logarítmica), en especial en las inmediaciones de los ceros. Se observa, además, la tendencia del algoritmo a converger a respuestas impulsionales constantes según la plantilla, probablemente atribuible a la función de aptitud y a la superficie de error que impone. Ésto restringe considerablemente la aplicación de este algoritmo, por lo que es preciso profundizar en las causas o en última instancia modificar el enfoque, ya sea acotando la población inicial a las

inmediaciones de soluciones analíticas o utilizando la población final como punto de partida para otra instancia de aproximación.

En términos de aplicación, al requerir una respuesta impulsional para la aproximación se supone más información que la que usualmente se encuentra disponible. La aproximación directa de módulo o fase de respuesta en frecuencia es típica y este algoritmo no ha mostrado buenos resultados con funciones de aptitud basadas en métricas afines. La derivación de una respuesta impulsional a partir de esta información es un posible camino a seguir, con sus propios desafíos.

Finalmente, existe una amplia variedad de algoritmos inspirados en la naturaleza e incluso híbridos con otros tipos de técnicas de optimización que aún restan por explorar. Cabe resaltar, no obstante, la cuidadosa definición de la representación y los operadores genéticos, adecuados al universo de discurso, y el mantenimiento de un delicado balance entre diversidad genética y presión de selección que este (tipo de) algoritmo requiere para converger a soluciones cuando mucho subóptimas sin tornarse una búsqueda por fuerza bruta aleatoria.

## **1.5 Referencias**

- [1] R. Lyons, Understanding digital signal processing. Upper Saddle River [etc.]: Pearson Education International, 2013.
- [2] A. EIBEN, Introduction To Evolutionary Computing: SPRINGER-VERLAG BERLIN AN, 2016.