Efficient algorithms for the design of finite impulse response digital filters

Silviu Filip
under the supervision of N. Brisebarre and G. Hanrot
(AriC, LIP, ENS Lyon)

Journées Nationales de Calcul Formel (JNCF)
CIRM, Luminy, November 3-7, 2014

Digital Signal Processing

- became increasingly relevant over the past 4 decades:

$$\text{ANALOG} \rightarrow \text{DIGITAL}$$

Digital Signal Processing

- became increasingly relevant over the past 4 decades:

$$\text{ANALOG} \rightarrow \text{DIGITAL}$$

- think of:
  - data communications (ex: Internet, HD TV and digital radio)
  - audio and video systems (ex: CD, DVD, BD players)
  - many more

Digital Signal Processing

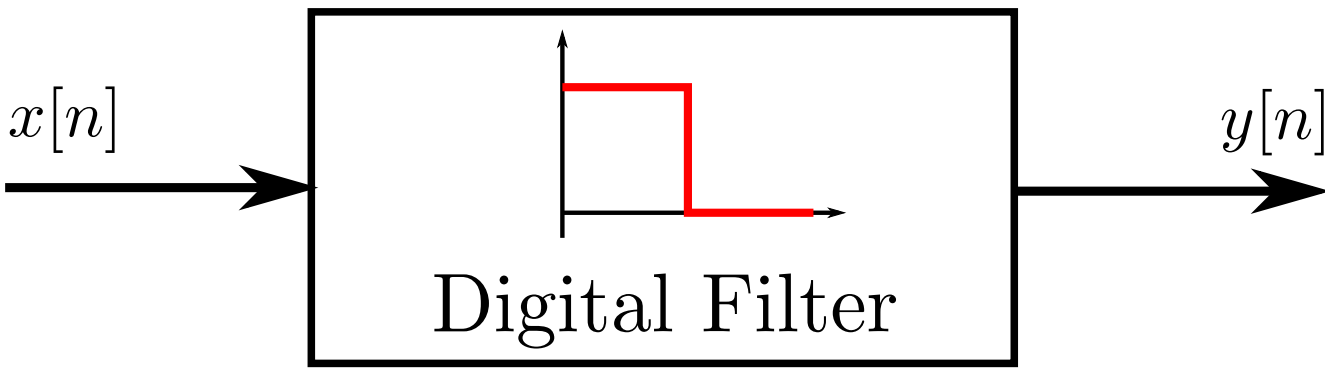
- became increasingly relevant over the past 4 decades:

$$\text{ANALOG} \rightarrow \text{DIGITAL}$$

- think of:
  - data communications (ex: Internet, HD TV and digital radio)
  - audio and video systems (ex: CD, DVD, BD players)
  - many more

What are the 'engines' powering all these?

Digital filters

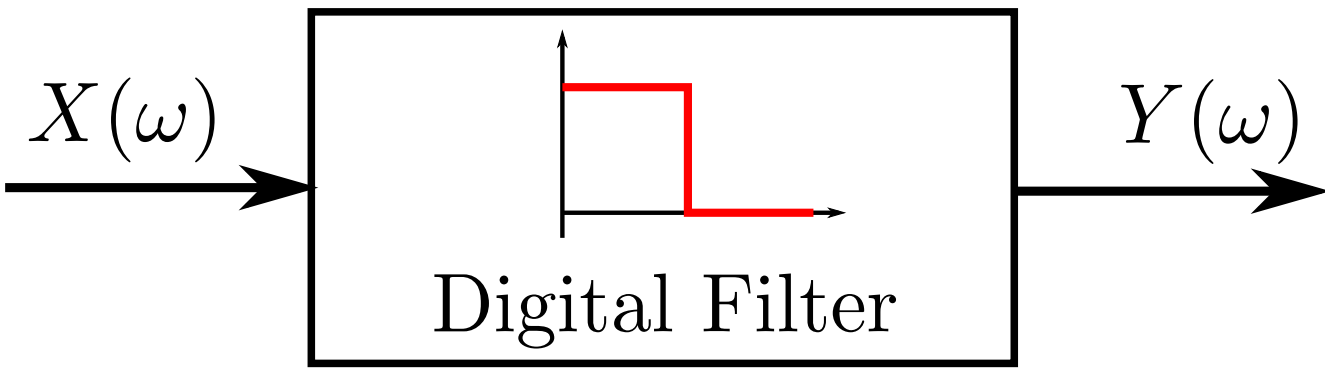


$$y[n] = x[n] \star h[n]$$

$\rightarrow$ we get two categories of filters

- finite impulse response (**FIR**) filters
- infinite impulse response (**IIR**) filters

Digital filters



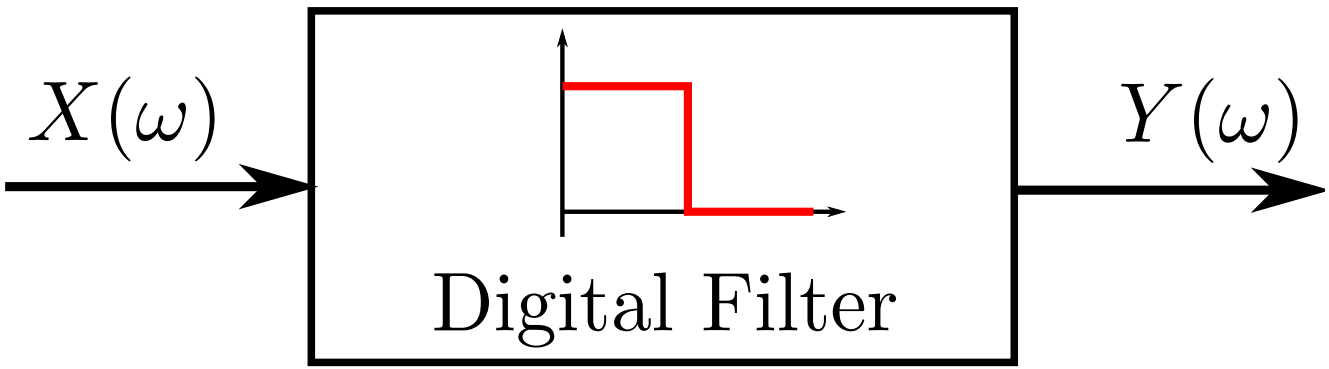
$$y[n] = x[n] \star h[n]$$

$\rightarrow$ we get two categories of filters

- finite impulse response (**FIR**) filters
- infinite impulse response (**IIR**) filters

$\rightarrow$ natural to work in the **frequency** domain

Digital filters



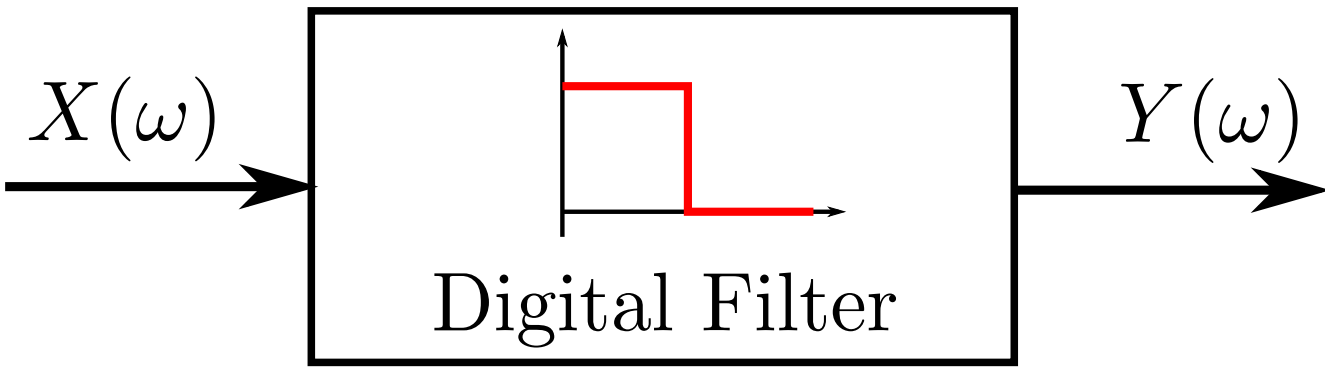
$$Y(\omega) = X(\omega)H(\omega), \omega \in [0, \pi]$$

$\rightarrow$ we get two categories of filters

- finite impulse response (**FIR**) filters
- infinite impulse response (**IIR**) filters

$\rightarrow$ natural to work in the **frequency** domain
H is the **transfer function** of the filter

Digital filters



$$Y(\omega) = X(\omega)H(\omega), \omega \in [0, \pi]$$

$\rightarrow$ we get two categories of filters

- finite impulse response (**FIR**) filters
  H is a polynomial
- infinite impulse response (**IIR**) filters
  H is a rational fraction

$\rightarrow$ natural to work in the **frequency** domain
H is the **transfer function** of the filter

The filtering toolchain

Steps:

1. **derive a concrete mathematical representation of the filter**
   $\rightarrow$ use theory of minimax approximation

The filtering toolchain

Steps:

1. **derive a concrete mathematical representation of the filter**
   $\rightarrow$ use theory of minimax approximation

2. **quantization of the filter coefficients using fixed-point or floating-point formats**
   $\rightarrow$ use tools from algorithmic number theory (euclidean lattices)

The filtering toolchain

Steps:

1. **derive a concrete mathematical representation of the filter**
   $\rightarrow$ use theory of minimax approximation

2. **quantization of the filter coefficients using fixed-point or floating-point formats**
   $\rightarrow$ use tools from algorithmic number theory (euclidean lattices)

3. **hardware synthesis of the filter**

The filtering toolchain

Steps:

1. **derive a concrete mathematical representation of the filter**
   $\rightarrow$ use theory of minimax approximation

2. **quantization of the filter coefficients using fixed-point or floating-point formats**
   $\rightarrow$ use tools from algorithmic number theory (euclidean lattices)

3. **hardware synthesis of the filter**

Today's focus: **first step** for FIR filters

Finite Impulse Response (FIR) filters

- large class of filters, with a lot of desirable properties

Usual representation: $H(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$

Finite Impulse Response (FIR) filters

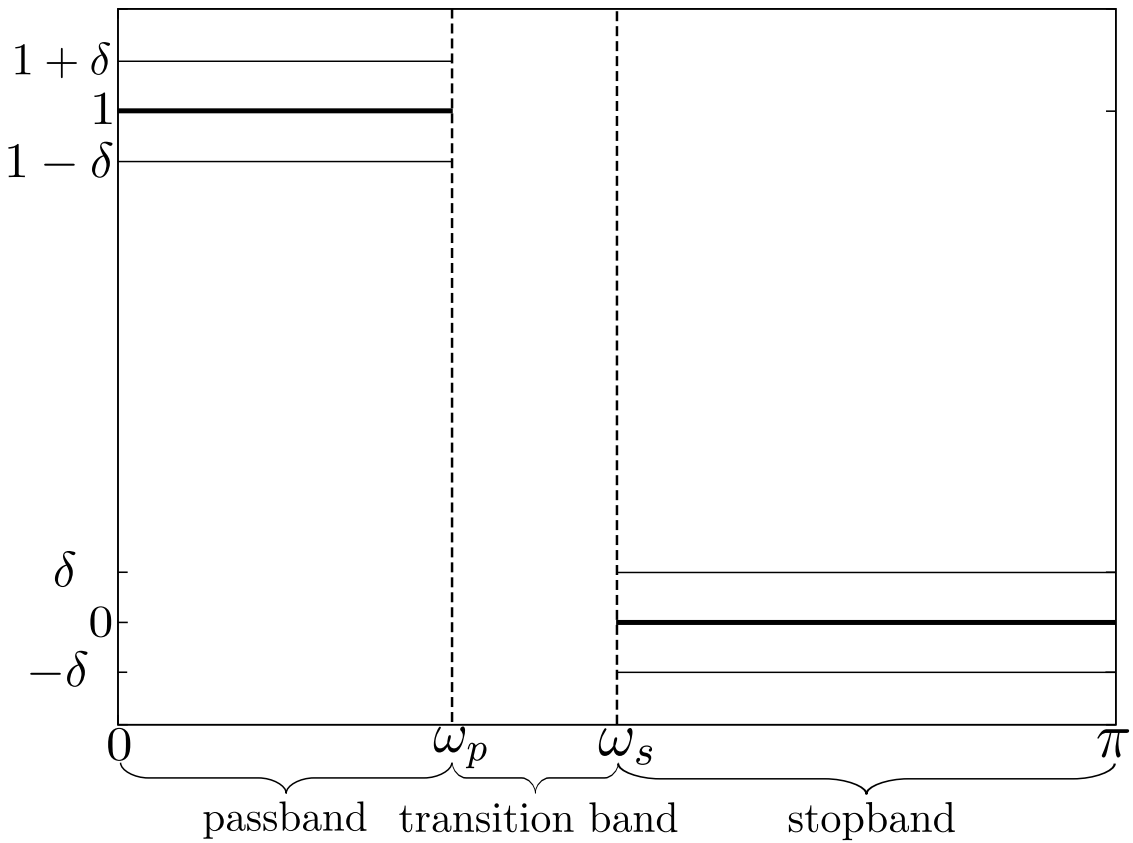
- large class of filters, with a lot of desirable properties

Usual representation: $H(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k) = \sum_{k=0}^{L} a_k T_k(\cos(\omega))$
$\rightarrow$ if $x = \cos(\omega)$, view H in the basis of Chebyshev polynomials

Finite Impulse Response (FIR) filters

- large class of filters, with a lot of desirable properties

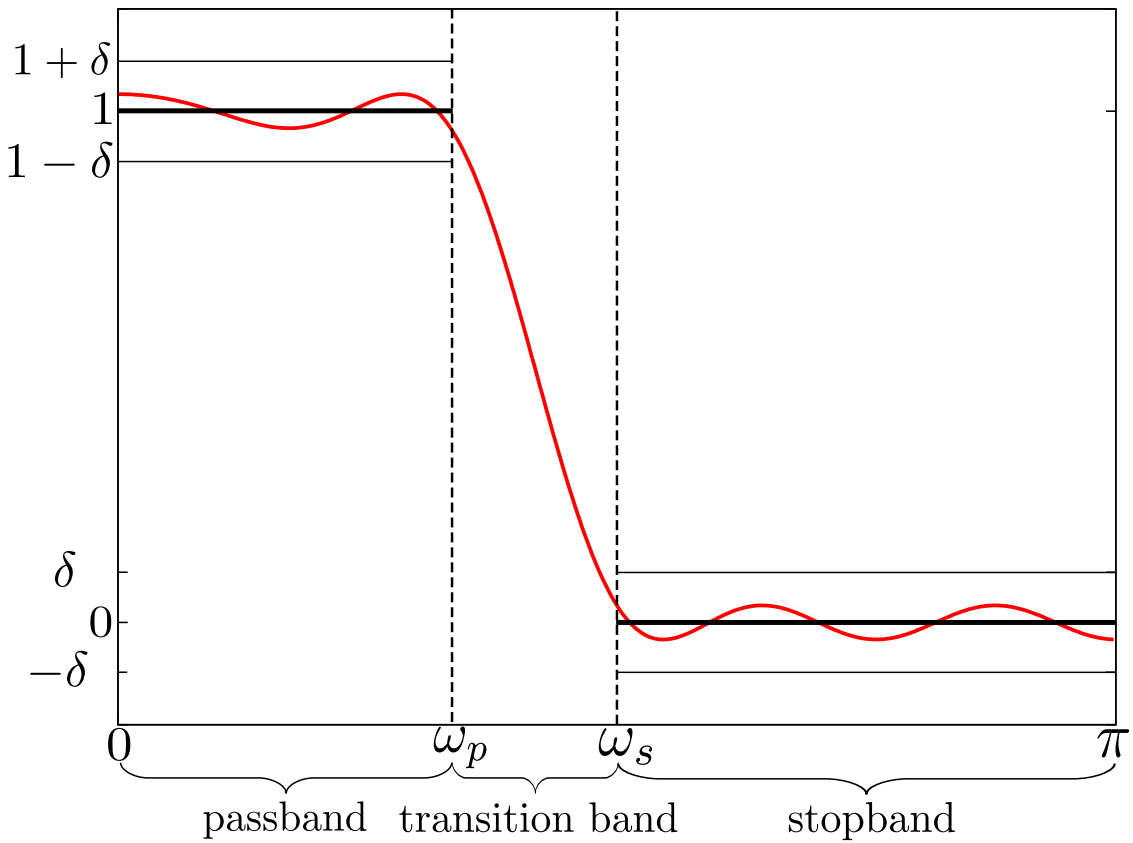
Usual representation: $H(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k) = \sum_{k=0}^{L} a_k T_k(\cos(\omega))$
$\rightarrow$ if $x = \cos(\omega)$, view H in the basis of Chebyshev polynomials
Specification:



Finite Impulse Response (FIR) filters

- large class of filters, with a lot of desirable properties

Usual representation: $H(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k) = \sum_{k=0}^{L} a_k T_k(\cos(\omega))$
$\rightarrow$ if $x = \cos(\omega)$, view H in the basis of Chebyshev polynomials
Specification:



$$H(\omega) = \sum_{k=0}^{8} a_k \cos(\omega k)$$

Optimal FIR design with real coefficients

The problem: Given a closed real set F, find an approximation $H(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$ of degree L for a continuous function $D(\omega), \omega \in F$ such that

$$\delta = \|E(\omega)\|_{\infty,F} = \max_{\omega \in F} |H(\omega) - D(\omega)|$$

is **minimal**.

Optimal FIR design with real coefficients

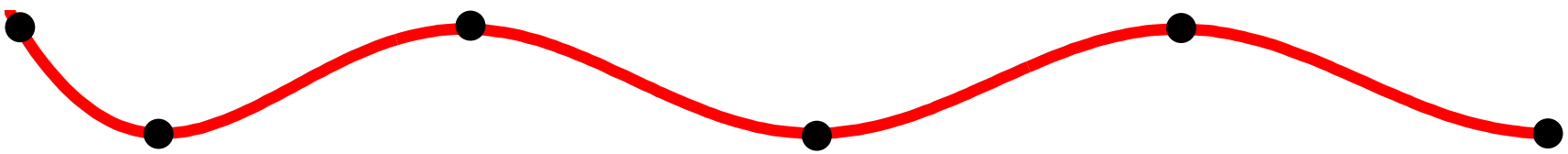
The solution: characterized by the **Alternation Theorem**

Theorem

*The **unique** solution* $H(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$ *has an **error function*** $E(\omega)$*, for which there exist* $L+2$ *values* $\omega_0 < \omega_1 < \cdots < \omega_{L+1}$*, belonging to* F*, such that*

$$E(\omega_i) = -E(\omega_{i+1}) = \pm\delta,$$

for $i = 0, \ldots, L$.



Optimal FIR design with real coefficients

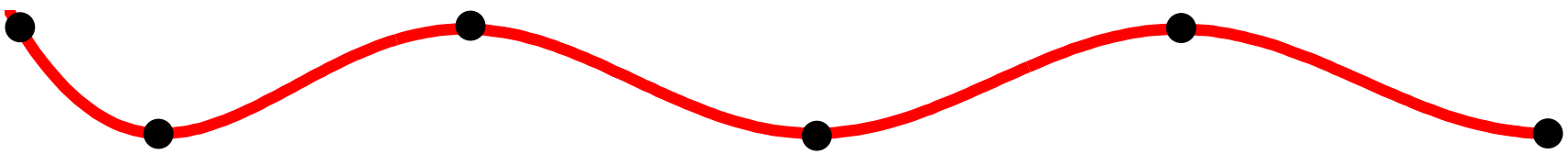
The solution: characterized by the **Alternation Theorem**

Theorem

*The **unique** solution* $H(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$ *has an **error function*** $E(\omega)$*, for which there exist* $L+2$ *values* $\omega_0 < \omega_1 < \cdots < \omega_{L+1}$*, belonging to* F*, such that*

$$E(\omega_i) = -E(\omega_{i+1}) = \pm\delta,$$

for $i = 0, \ldots, L$.



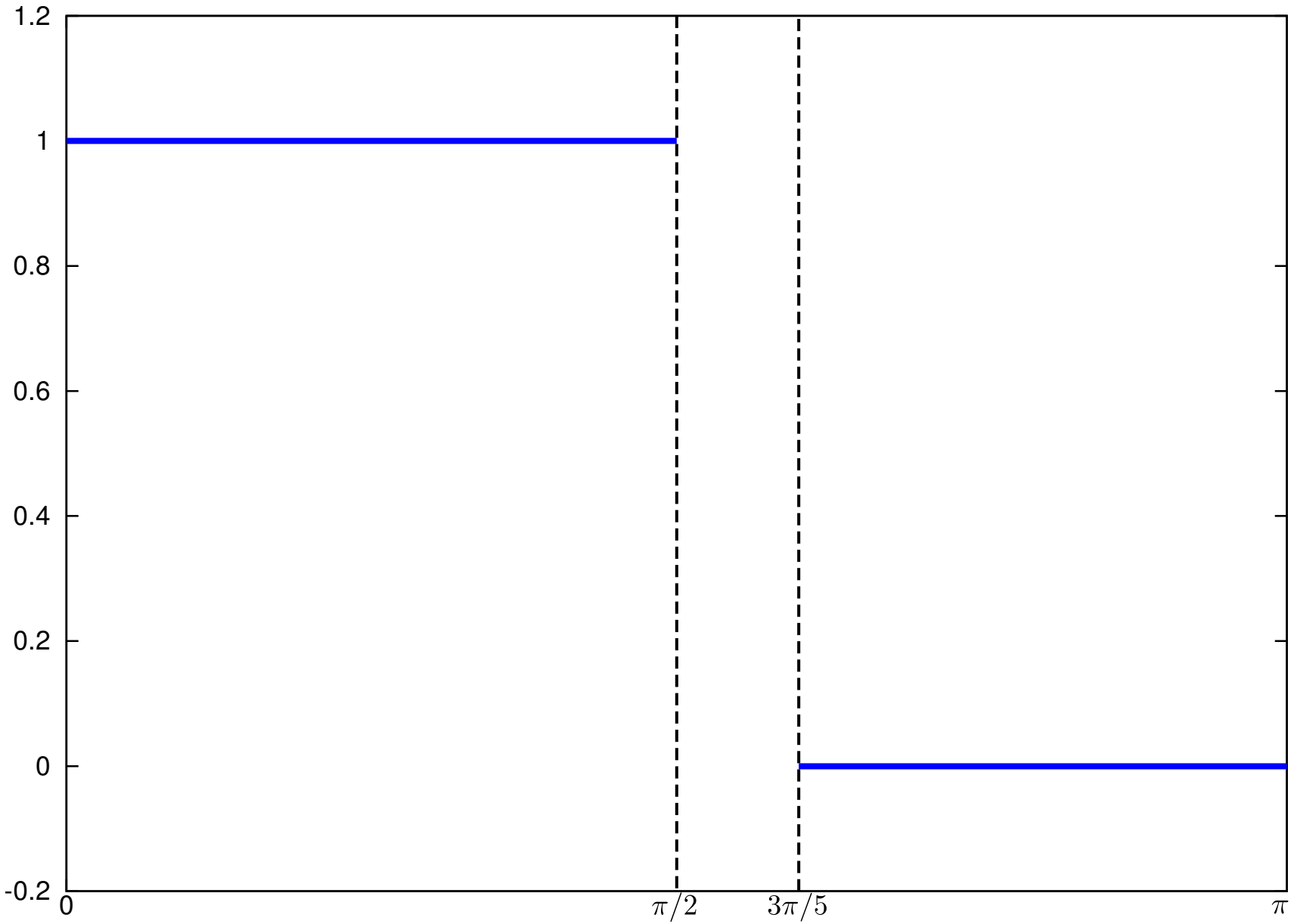
→ well studied in Digital Signal Processing literature

1972: Parks and McClellan

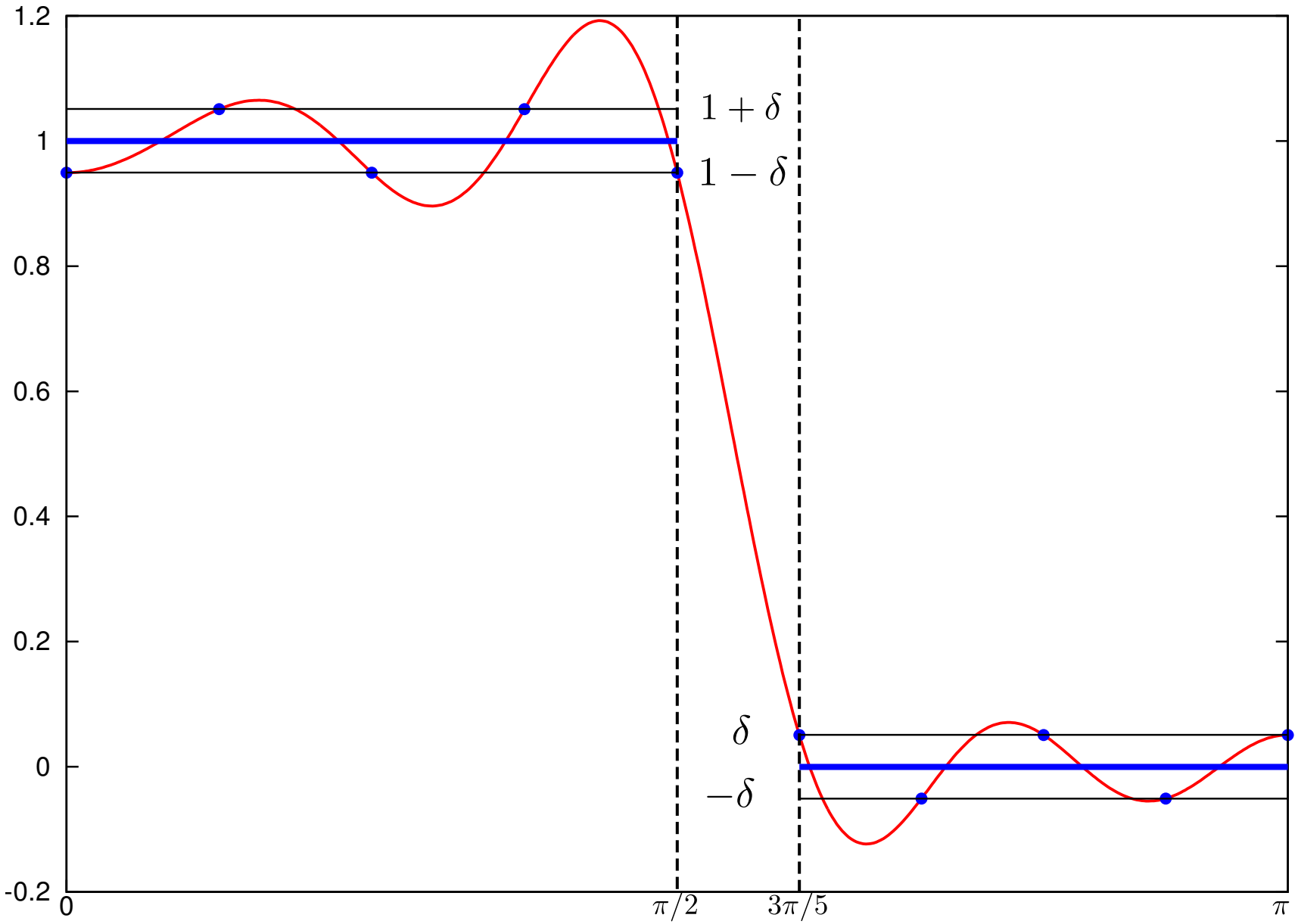
→ based on a powerful iterative approach from Approximation Theory:

1932: Remez

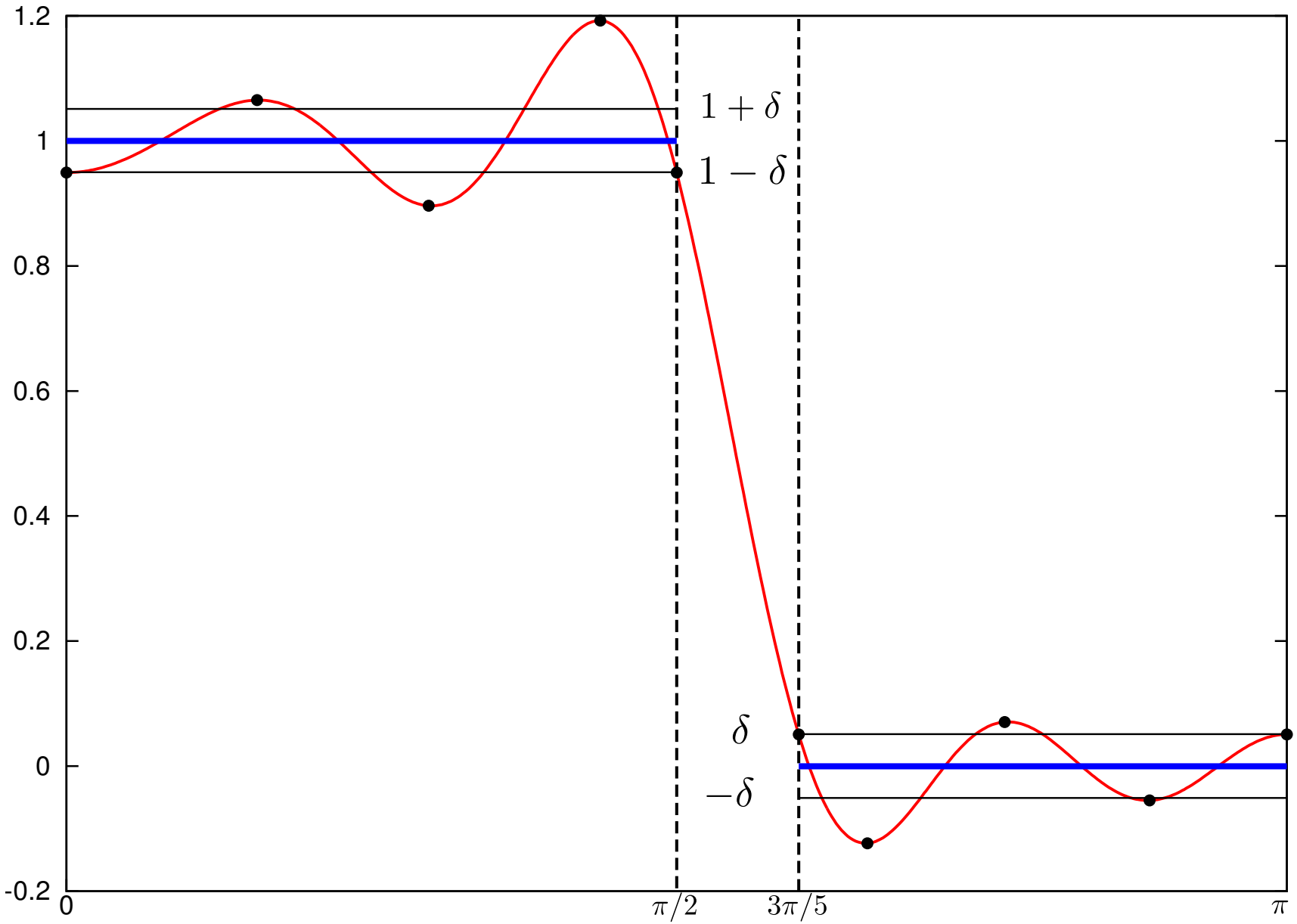
The Parks-McClellan design method: Example



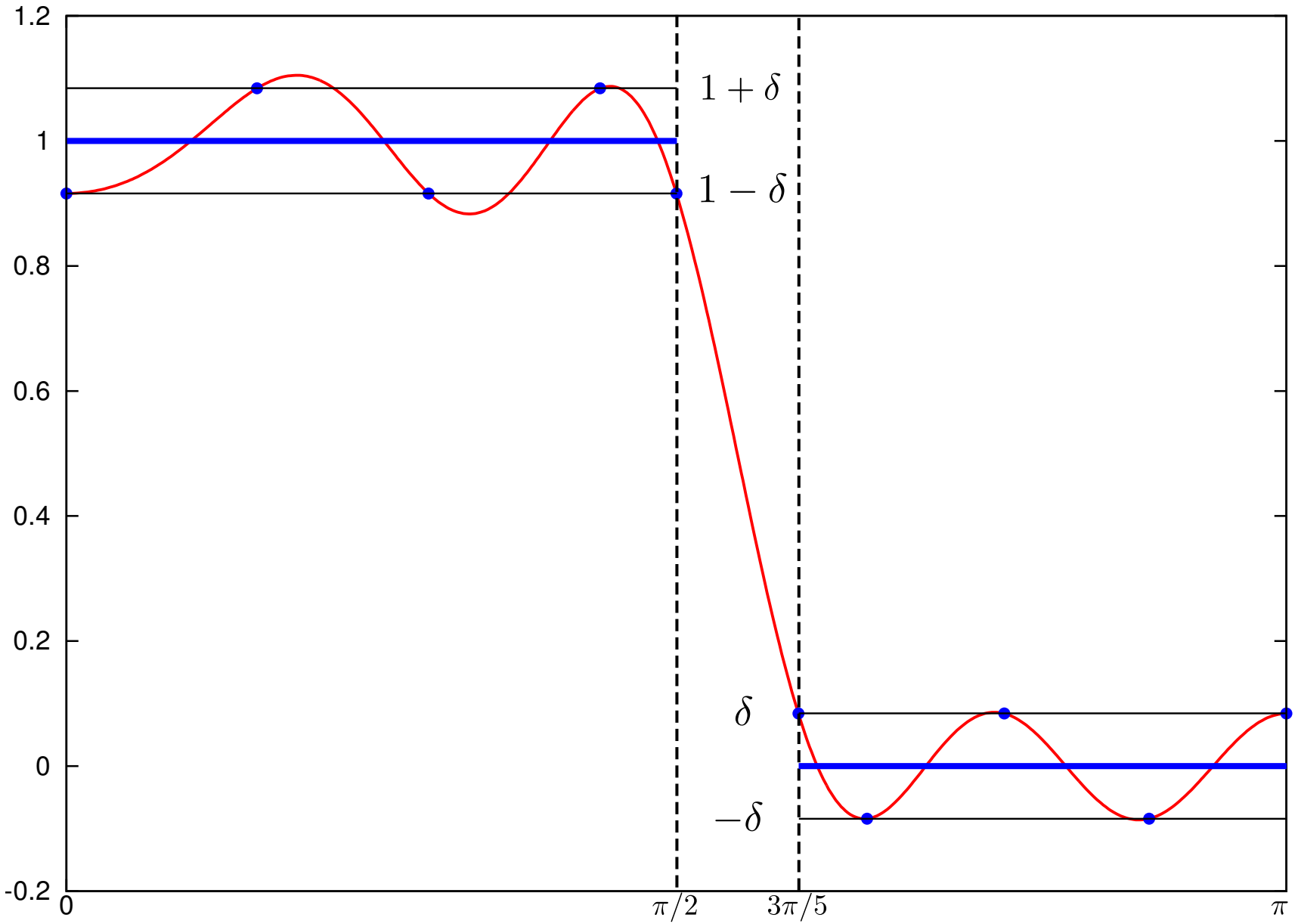
The Parks-McClellan design method: Example



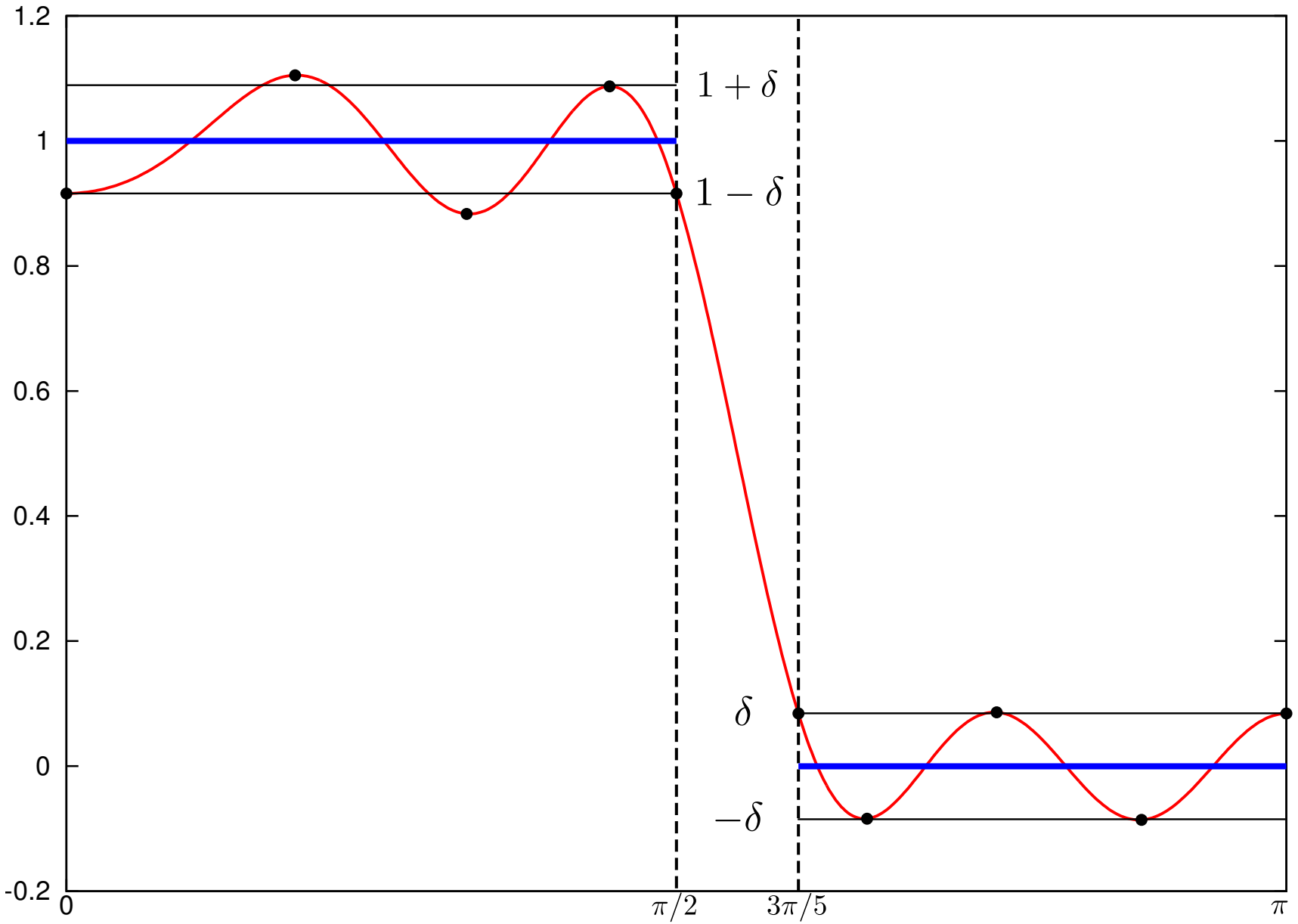
The Parks-McClellan design method: Example



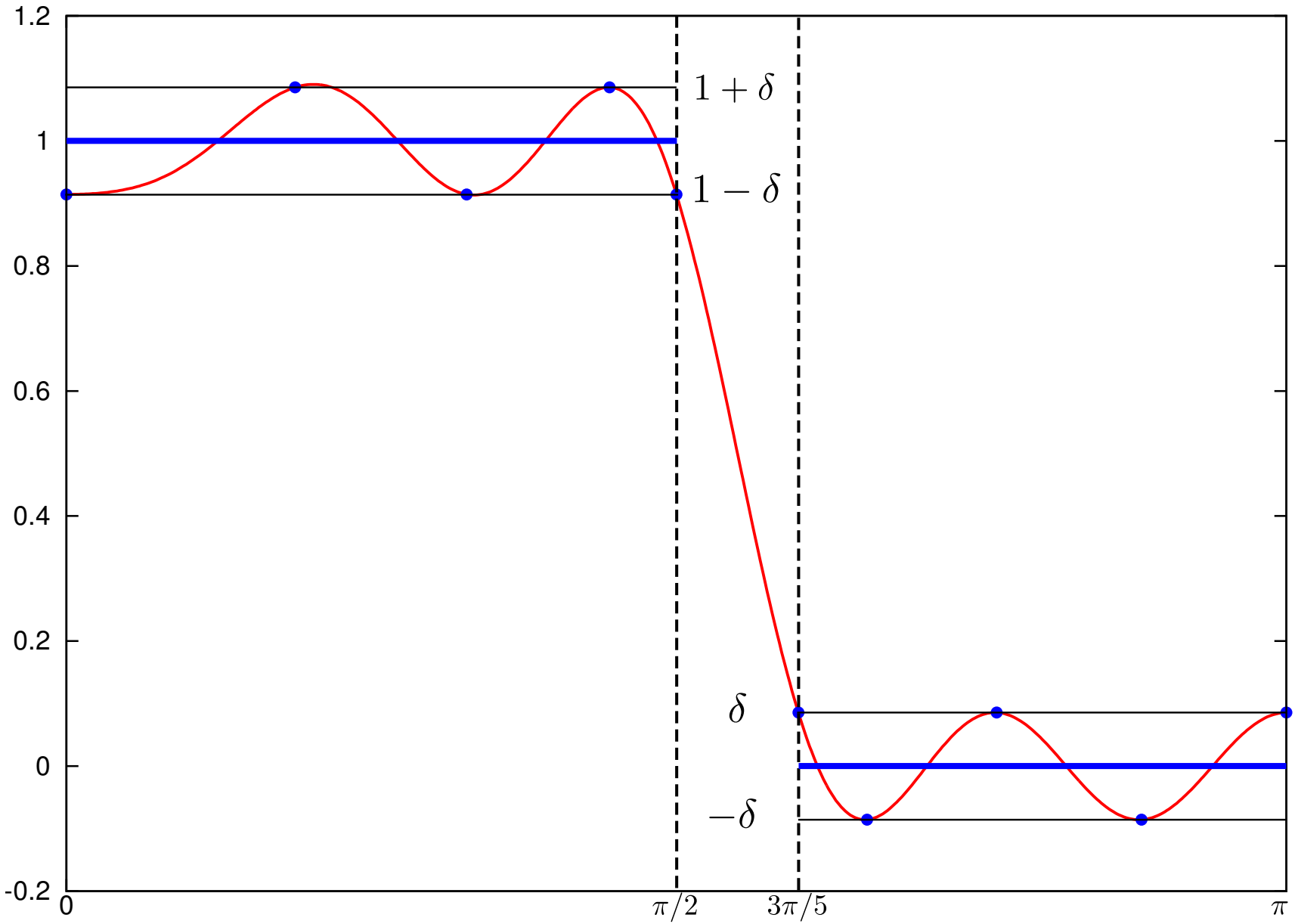
The Parks-McClellan design method: Example



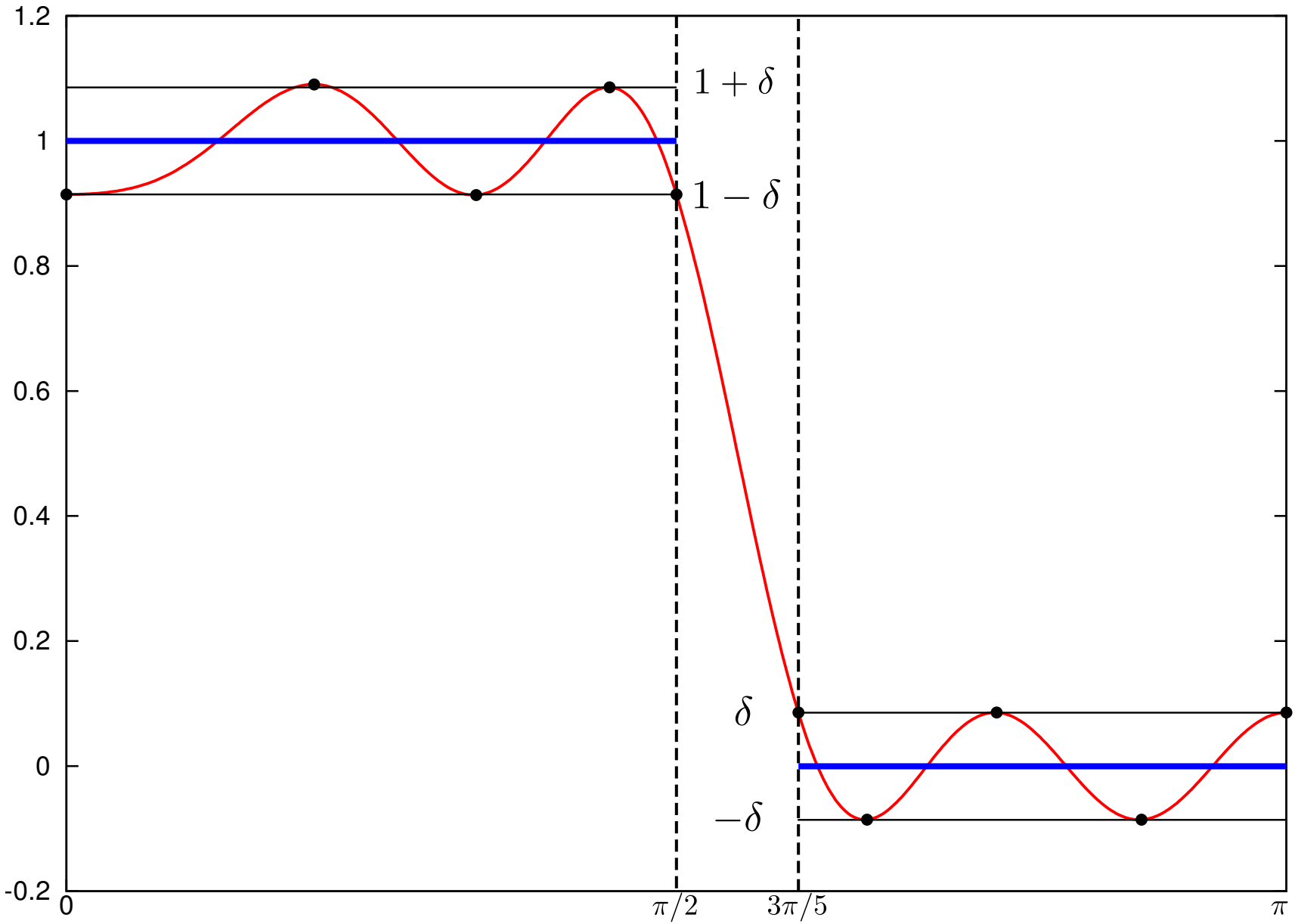
The Parks-McClellan design method: Example



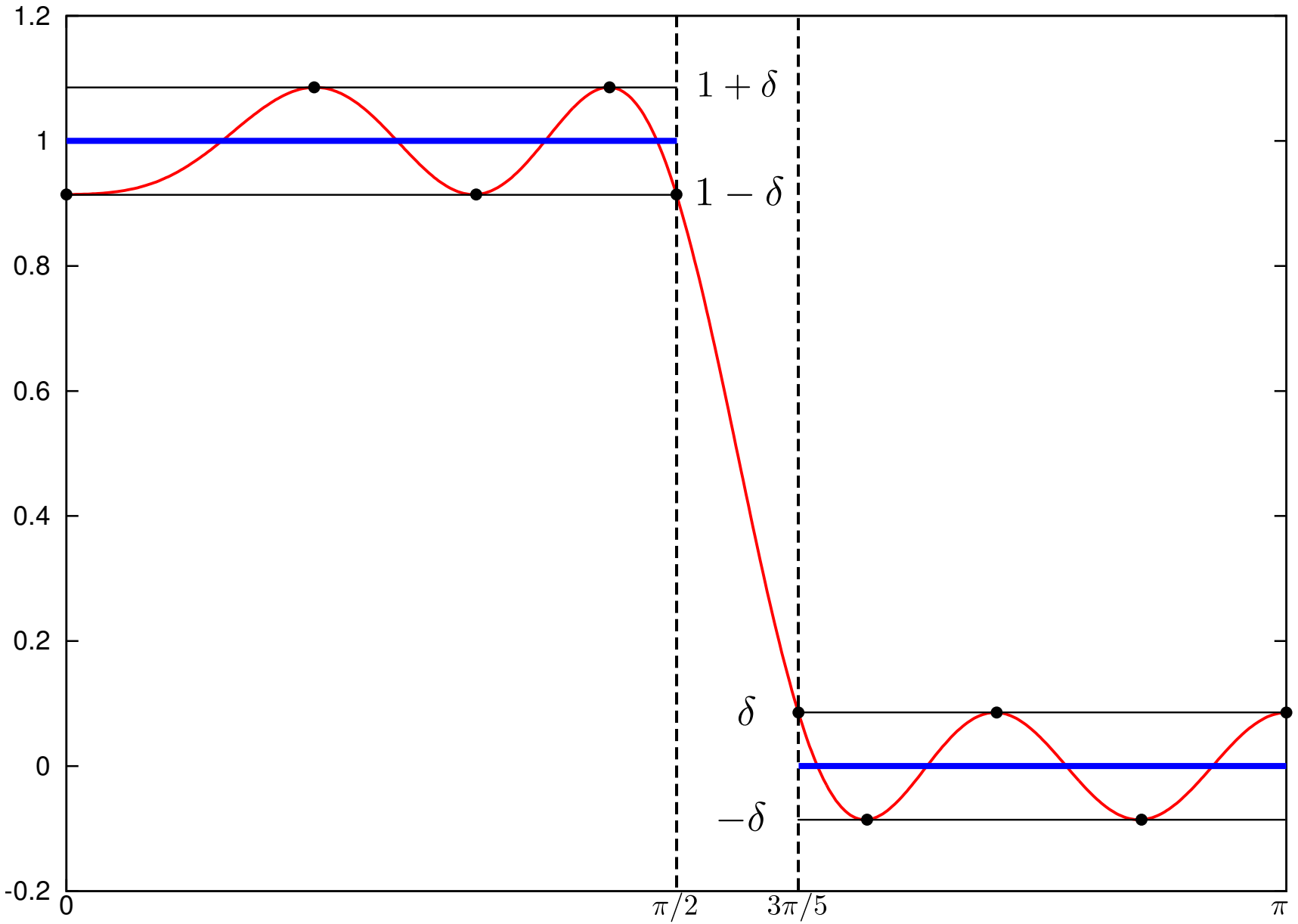
The Parks-McClellan design method: Example



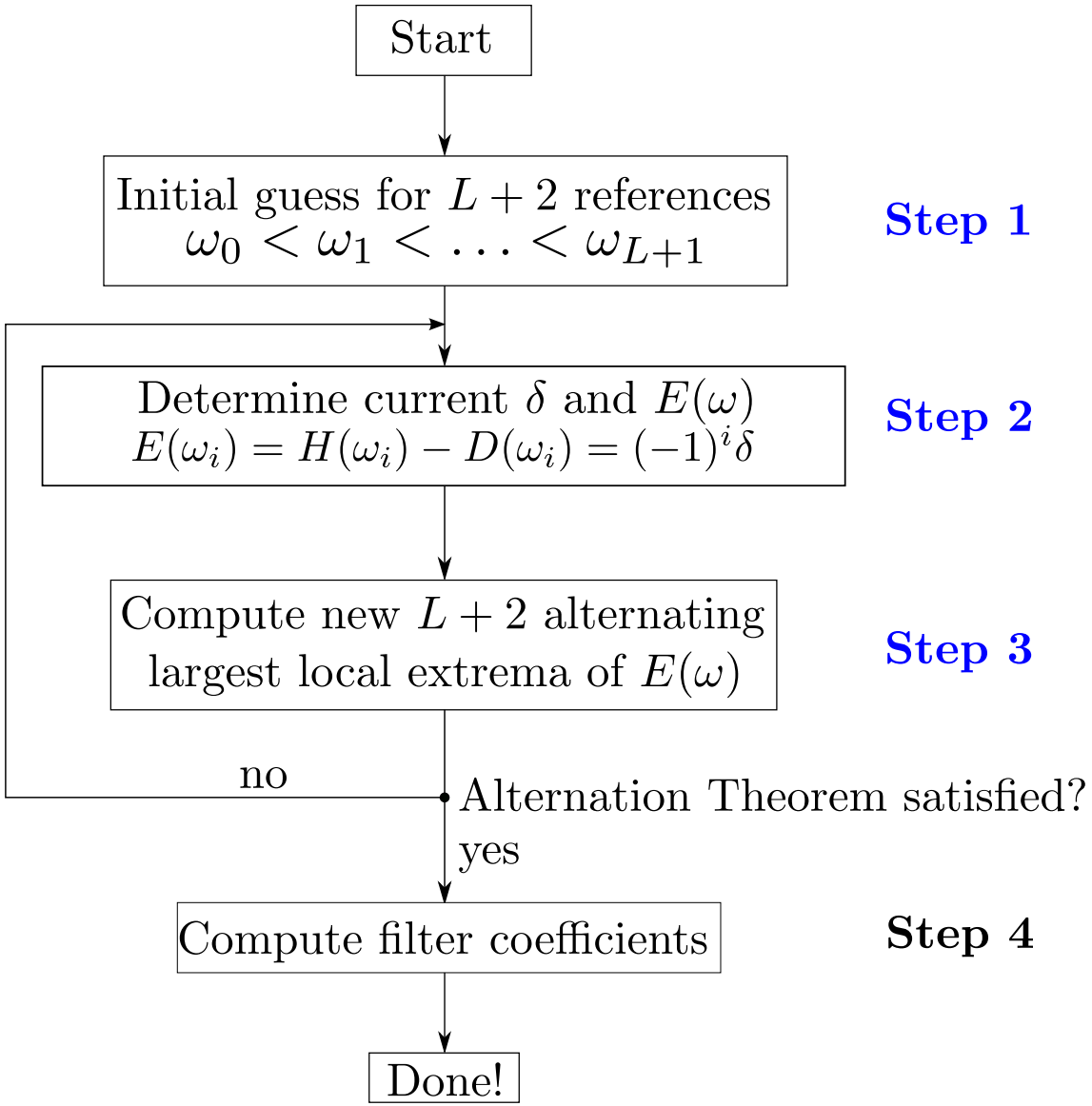
The Parks-McClellan design method: Example



The Parks-McClellan design method: Example



The Parks-McClellan design method: Steps



Step 1: Choosing the $L + 2$ initial references

Traditional approach: take the $L + 2$ references uniformly from F
$\rightarrow$ can lead to convergence problems

Step 1: Choosing the $L + 2$ initial references

Traditional approach: take the $L + 2$ references uniformly from F
$\rightarrow$ can lead to convergence problems

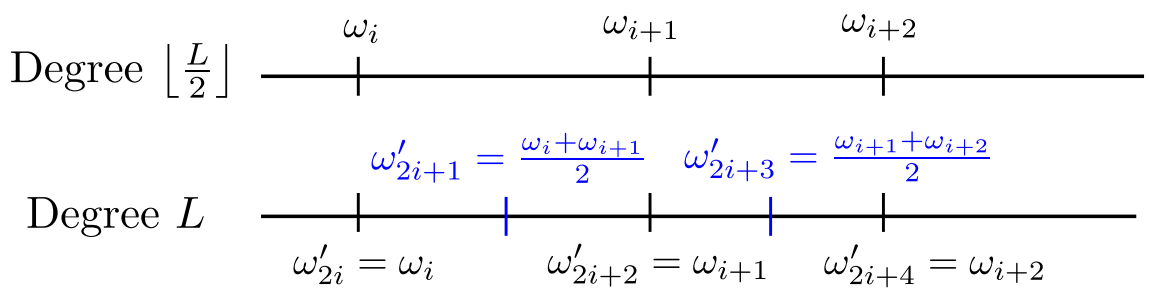
$\rightarrow$ **want to start** from better approximations
Existing approaches: not general enough and/or costly to execute

Step 1: Choosing the $L+2$ initial references

Traditional approach: take the $L+2$ references uniformly from F
$\rightarrow$ can lead to convergence problems

$\rightarrow$ **want to start** from better approximations
Existing approaches: not general enough and/or costly to execute

Our approach: extrema position extrapolation from smaller filters



$\rightarrow$ although empirical, it is rather robust in practice

Step 2: Computing the current error function $E(\omega)$ and δ

Amounts to solving a linear system in $a_0, \ldots, a_L$ and δ.

$$\begin{bmatrix} 1 & \cos(\omega_0) & \cdots & \cos(L\omega_0) & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \cos(\omega_L) & \cdots & \cos(L\omega_L) & (-1)^L \\ 1 & \cos(\omega_{L+1}) & \cdots & \cos(L\omega_{L+1}) & (-1)^{L+1} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_L \\ \delta \end{bmatrix} = \begin{bmatrix} D(\omega_0) \\ \vdots \\ D(\omega_L) \\ D(\omega_{L+1}) \end{bmatrix}$$

$\rightarrow$ solving system directly: can be numerically unstable

Step 2: Computing the current error function $E(\omega)$ and δ

Amounts to solving a linear system in $a_0, \ldots, a_L$ and δ.

$$\begin{bmatrix} 1 & \cos(\omega_0) & \cdots & \cos(L\omega_0) & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \cos(\omega_L) & \cdots & \cos(L\omega_L) & (-1)^L \\ 1 & \cos(\omega_{L+1}) & \cdots & \cos(L\omega_{L+1}) & (-1)^{L+1} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_L \\ \delta \end{bmatrix} = \begin{bmatrix} D(\omega_0) \\ \vdots \\ D(\omega_L) \\ D(\omega_{L+1}) \end{bmatrix}$$

$\rightarrow$ solving system directly: can be numerically unstable

$\rightarrow$ use **barycentric** form of Lagrange interpolation [Berrut&Trefethen2004]

Barycentric Lagrange interpolation

Problem: p polynomial with $\deg p \leqslant L$ interpolates f at points x_j, i.e.,

$$p(x_j) = f_j, j = 0, \ldots, L$$

Barycentric Lagrange interpolation

Problem: p polynomial with $\deg p \leqslant L$ interpolates f at points x_j, i.e.,

$$p(x_j) = f_j, j = 0, \ldots, L$$

$\rightarrow$ the barycentric form of p is:

$$p(x) = \frac{\displaystyle\sum_{j=0}^{L} \frac{w_j}{x - x_j} f_j}{\displaystyle\sum_{j=0}^{L} \frac{w_j}{x - x_j}},$$

where $w_j = \dfrac{1}{\prod_{k \neq j}(x_j - x_k)}$.

Cost: $O(L^2)$ for computing *all* w_j, $O(L)$ for evaluating $p(x)$.

Barycentric Lagrange interpolation

$\rightarrow$ numerically stable **if the family of interpolation nodes used has a small Lebesgue constant** [Mascarenhas&Camargo2014]

The Lebesgue constant: specific to each grid of points; quantifies the convergence/divergence properties of polynomial interpolants using those nodes

Barycentric Lagrange interpolation

$\rightarrow$ numerically stable **if the family of interpolation nodes used has a small Lebesgue constant** [Mascarenhas&Camargo2014]

The Lebesgue constant: specific to each grid of points; quantifies the convergence/divergence properties of polynomial interpolants using those nodes

$\rightarrow$ from empirical observation, the families of points used inside the Parks-McClellan algorithm (Step 1 + Step 3) usually converge to sets of points with small Lebesgue constant

Step 3: Finding the local extrema of $E(\omega)$

Traditional approach: evaluate $E(\omega)$ on a dense grid of uniformly distributed points (in practice it is usually $16L$)
$\rightarrow$ works well for degree $L < 100$, tends to fail in some cases for larger L

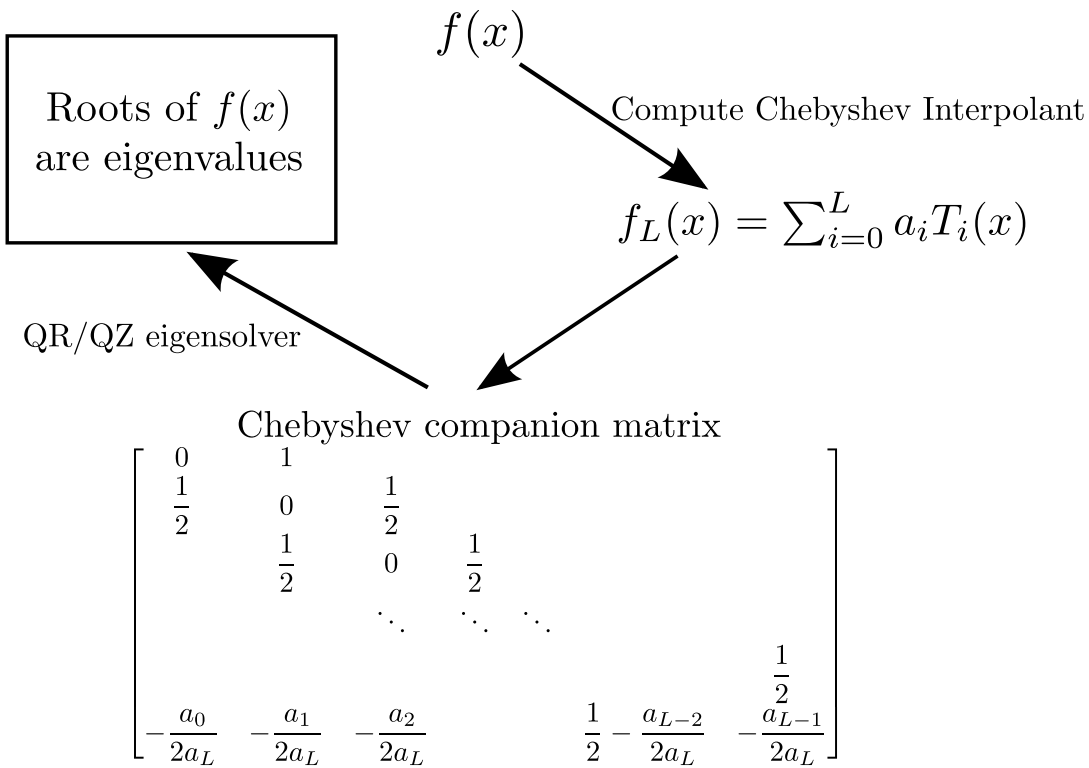
Step 3: Finding the local extrema of $E(\omega)$

Traditional approach: evaluate $E(\omega)$ on a dense grid of uniformly distributed points (in practice it is usually $16L$)

$\rightarrow$ works well for degree $L < 100$, tends to fail in some cases for larger L

Our approach:

$\rightarrow$ **Chebyshev-proxy rootfinder(CPR)** [Boyd2002,Boyd2013]



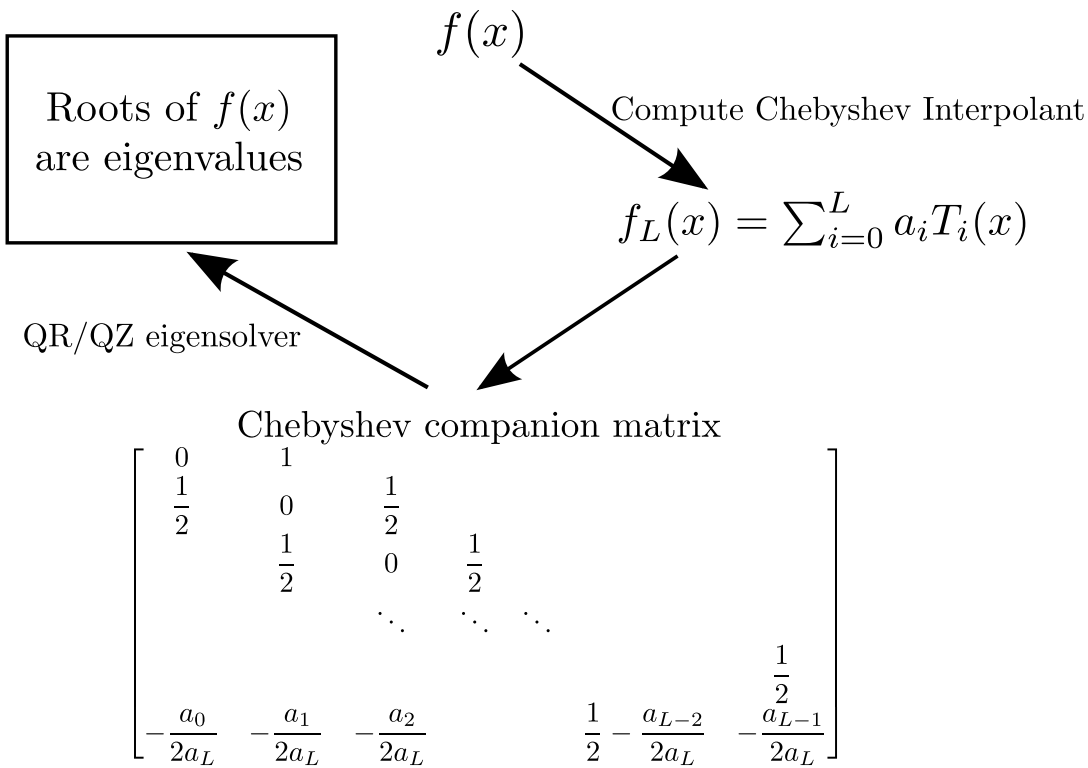
Step 3: Finding the local extrema of $E(\omega)$

Traditional approach: evaluate $E(\omega)$ on a dense grid of uniformly distributed points (in practice it is usually $16L$)

$\rightarrow$ works well for degree $L < 100$, tends to fail in some cases for larger L

Our approach:

$\rightarrow$ **Chebyshev-proxy rootfinder(CPR)** [Boyd2002,Boyd2013]



$\rightarrow$ [Pachon&Trefethen2009] use it to implement the Remez algorithm

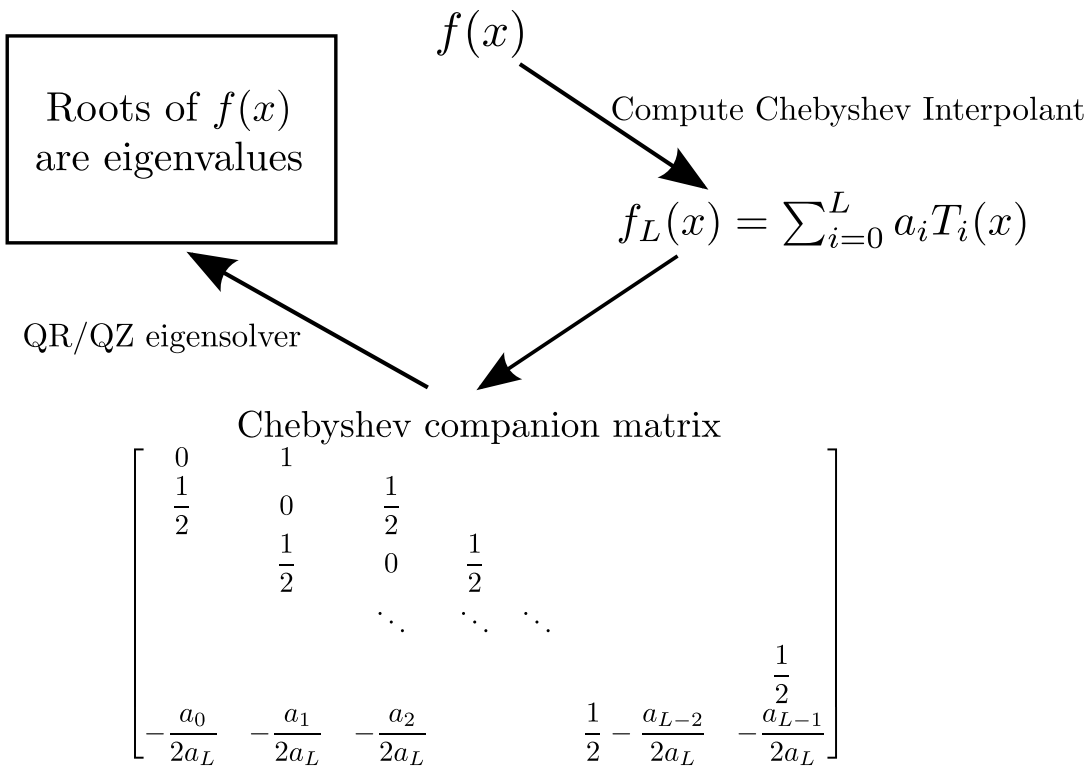
Step 3: Finding the local extrema of $E(\omega)$

Traditional approach: evaluate $E(\omega)$ on a dense grid of uniformly distributed points (in practice it is usually $16L$)

$\rightarrow$ works well for degree $L < 100$, tends to fail in some cases for larger L

Our approach:

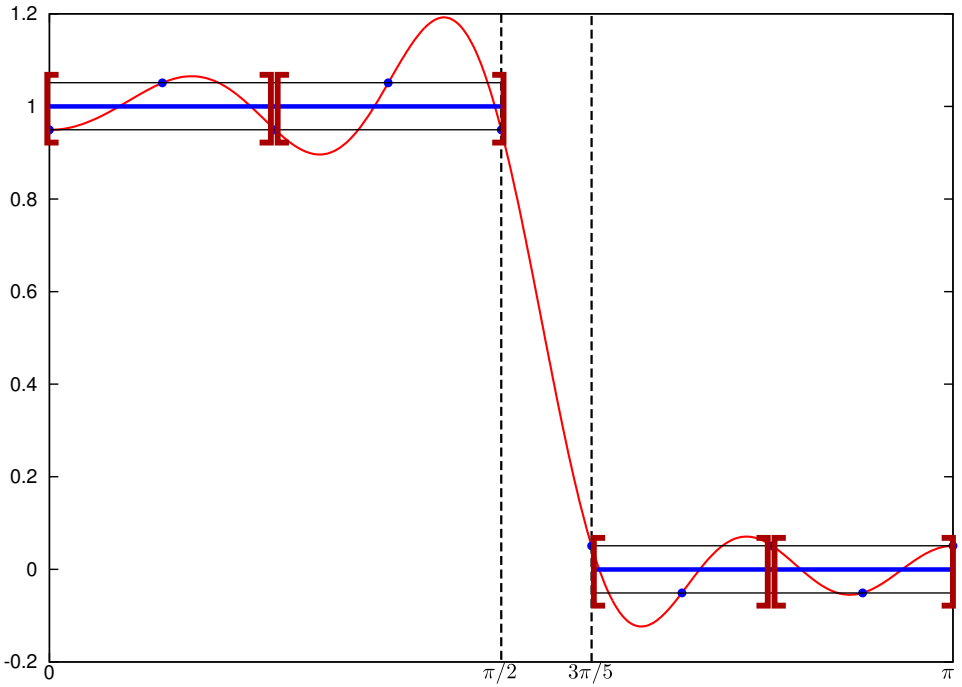
$\rightarrow$ **Chebyshev-proxy rootfinder(CPR)** [Boyd2002,Boyd2013]



$\rightarrow$ [Pachon&Trefethen2009] use it to implement the Remez algorithm

$\rightarrow$ apply a similar idea for FIR approximations

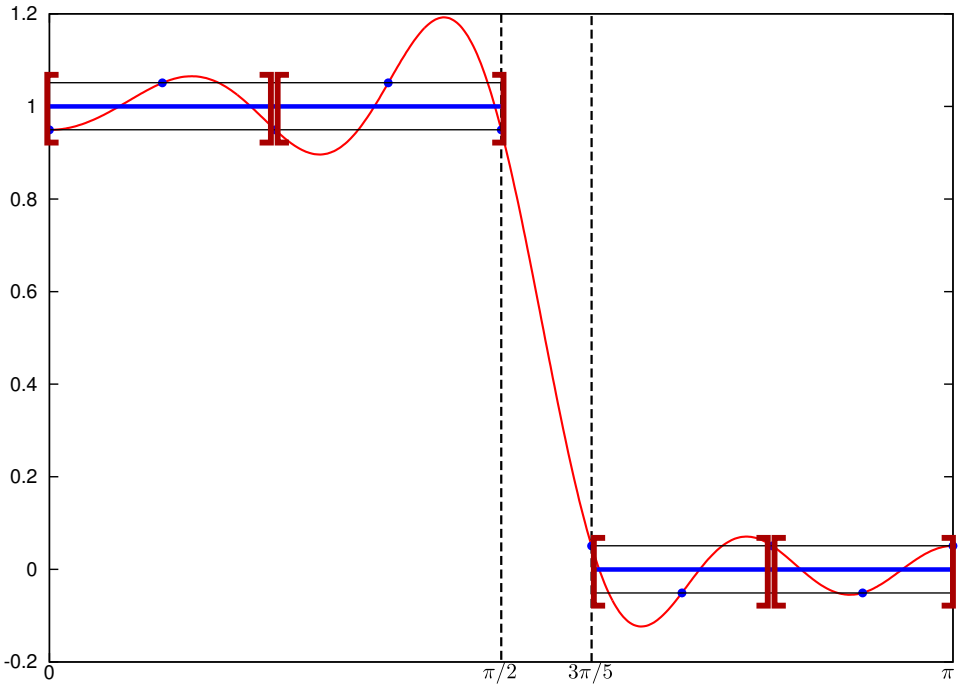
Step 3: Finding the local extrema of $E(\omega)$



Idea:

- split F into *appropriate* subintervals $F = \bigcup_{i=0}^{N} F_i$
- interpolate $E(\omega)$ on each F_i with small degree Chebyshev interpolants C_i
- compute roots of the derivative of C_i using a Chebyshev-proxy rootfinder

Step 3: Finding the local extrema of $E(\omega)$



Idea:

- split F into *appropriate* subintervals $F = \bigcup_{i=0}^{N} F_i$
- interpolate $E(\omega)$ on each F_i with small degree Chebyshev interpolants C_i
- compute roots of the derivative of C_i using a Chebyshev-proxy rootfinder

Advantages:

$\rightarrow$ robust, numerically stable root finding approach

$\rightarrow$ easy to parallelize

Cost: if $N = O(L)$, overall $O(L^2)$ arithmetic operations

Step 4: Recover coefficients of $H(\omega)$ upon convergence

$\rightarrow$ can use the *Inverse Discrete Fourier Transform*

Step 4: Recover coefficients of $H(\omega)$ upon convergence

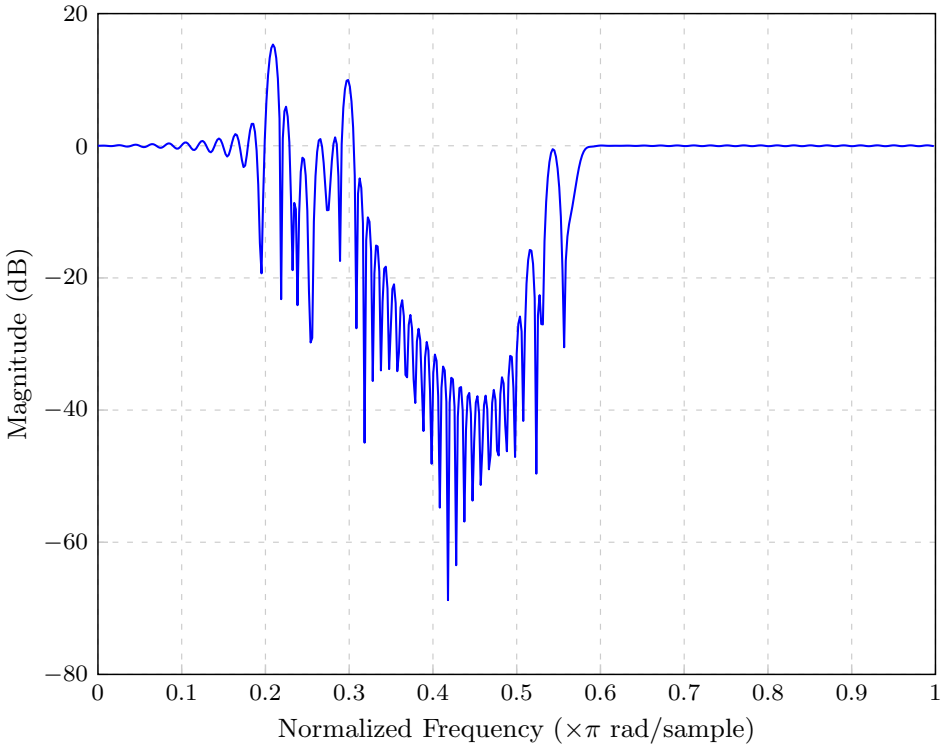
$\rightarrow$ can use the *Inverse Discrete Fourier Transform*

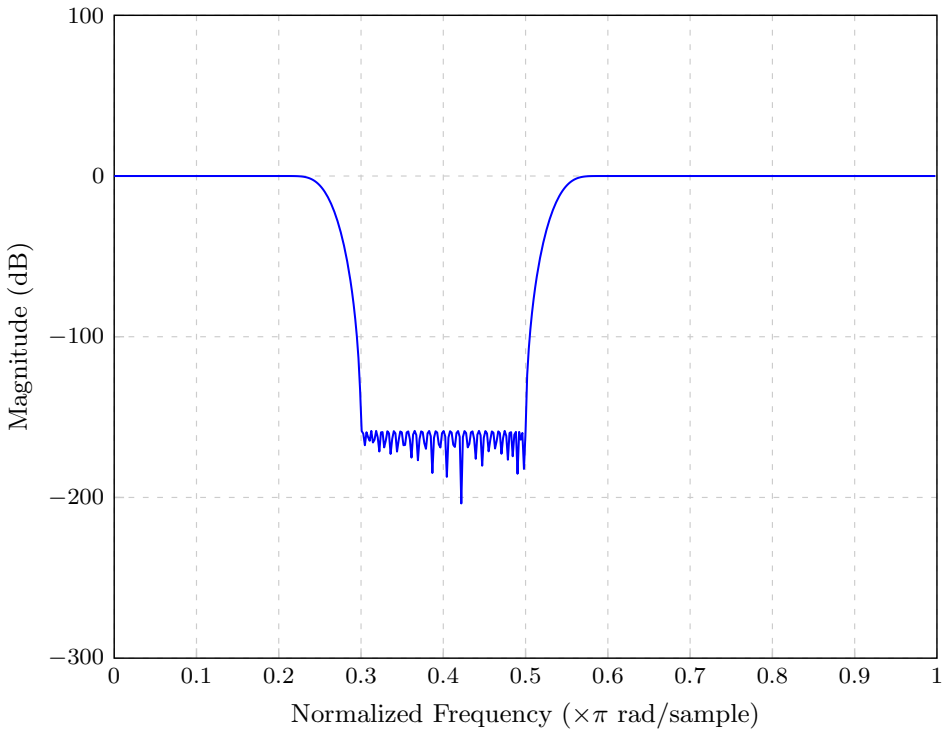
$\rightarrow$ implement it using Clenshaw's algorithm for computing linear combinations of Chebyshev polynomials (numerically robust approach)

Cost: $O(L^2)$ arithmetic operations

Our implementation: Examples & Results

Comparison with MATLAB & demo





Our implementation: Examples & Results

$\rightarrow$ design of channelizers for software radio systems

Specification: degree 53248 filter with stopband $\left[0, \frac{1}{8192}\pi\right]$ and passband $\left[\frac{3}{8192}\pi, \pi\right]$.

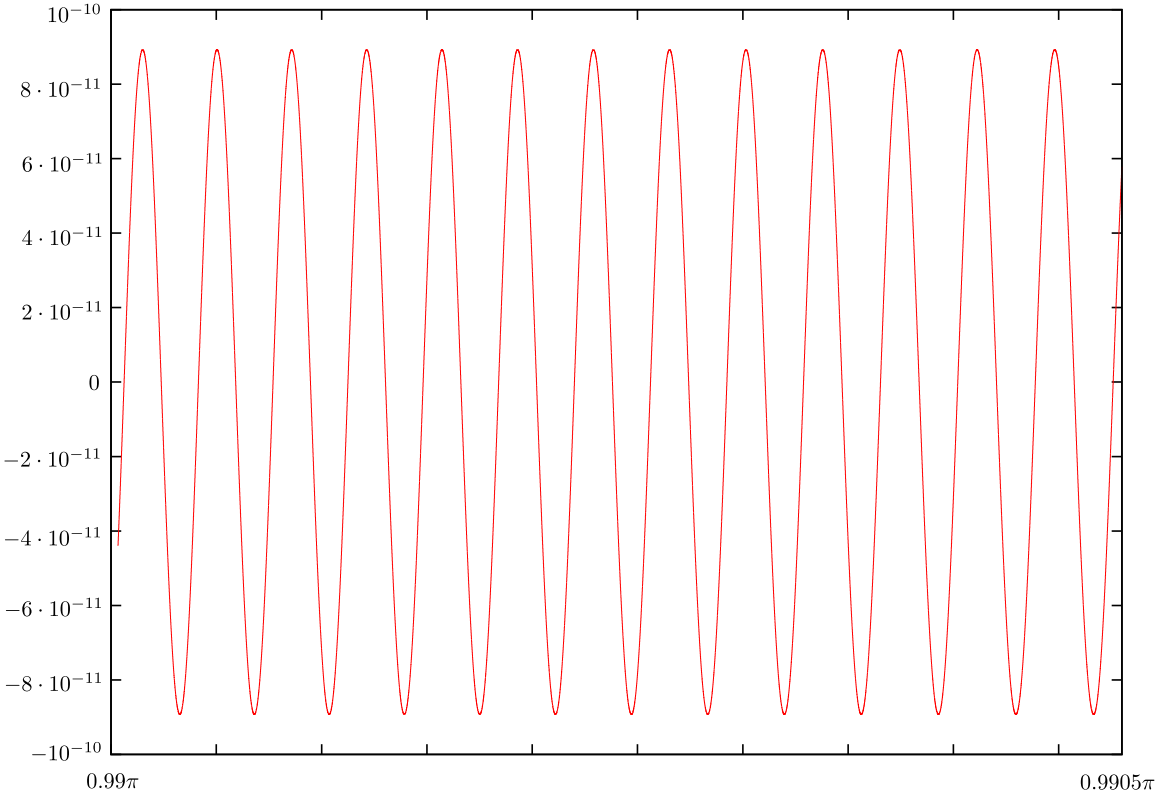


Figure: error function on $[0.99\pi, 0.9905\pi]$

Convergence: required 5 iterations

Perspectives

Conclusion:

- improved the practical behavior of a well known polynomial approximation algorithm for filter design
  $\rightarrow$ use numerically stable barycentric Lagrange interpolation + rootfinders without sacrifices in efficiency
- this new approach can take huge advantage of parallel architectures

Future work:

- release the code for our implementation as an open source library
- provide a complete toolchain for constructing FIR filters (approximation + quantification + hardware synthesis)
- tackle the IIR filter setting (rational fraction)
  - non-linear problem
  - constraints: poles located inside the unit circle