The Parks McClellan algorithm: a scalable approach for designing FIR filters

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

PEQUAN Seminar, February 26, 2015

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

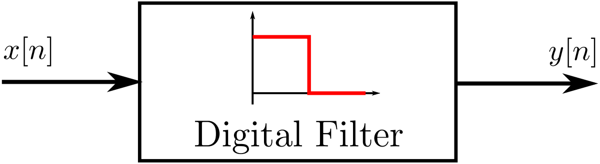
- became increasingly relevant over the past 4 decades:

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

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

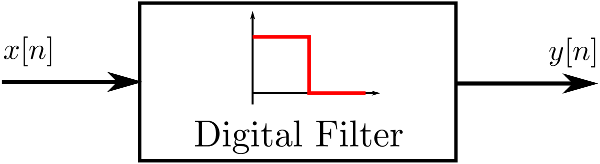
What are the 'engines' powering all these?

Digital filters



$$y[n] = \sum_{k=0}^{N} b_k x[n-k] - \sum_{k=1}^{M} a_k y[n-k], a_k, b_k \in \mathbb{R}$$

Digital filters

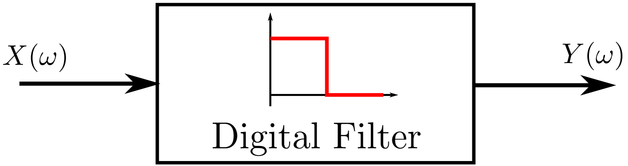


$$y[n] = \sum_{k=0}^{N} b_k x[n-k] - \sum_{k=1}^{M} a_k y[n-k], a_k, b_k \in \mathbb{R}$$

$\rightarrow$ we get two categories of filters

- finite impulse response (**FIR**) filters
  non-recursive structure (i.e. $a_k = 0, k = 1, \ldots, M$)
- infinite impulse response (**IIR**) filters
  recursive structure (i.e. $\exists k$ s.t. $a_k \neq 0$)

Digital filters



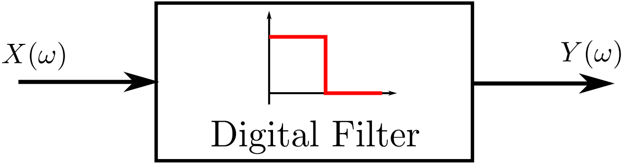
$$y[n] = \sum_{k=0}^{N} b_k x[n-k] - \sum_{k=1}^{M} a_k y[n-k], a_k, b_k \in \mathbb{R}$$

$\rightarrow$ we get two categories of filters

- finite impulse response (**FIR**) filters
  non-recursive structure (i.e. $a_k = 0, k = 1, \ldots, M$)
- infinite impulse response (**IIR**) filters
  recursive structure (i.e. $\exists k$ s.t. $a_k \neq 0$)

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

Digital filters



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

→ 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**

→ 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: $H(\omega) = \sum_{k=0}^{n} h_k \cos(\omega k)$

Finite Impulse Response (FIR) filters

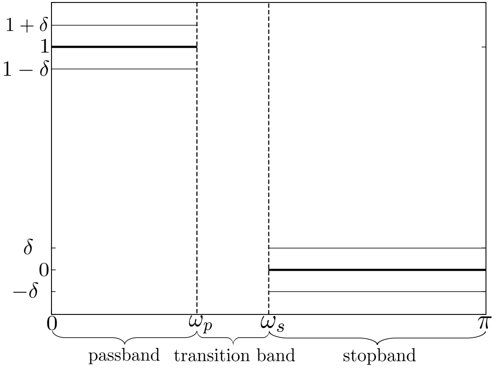
- large class of filters, with a lot of desirable properties

Usual representation: $H(\omega) = \sum_{k=0}^{n} h_k \cos(\omega k) = \sum_{k=0}^{n} h_k T_k(\cos(\omega))$
$\rightarrow$ 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

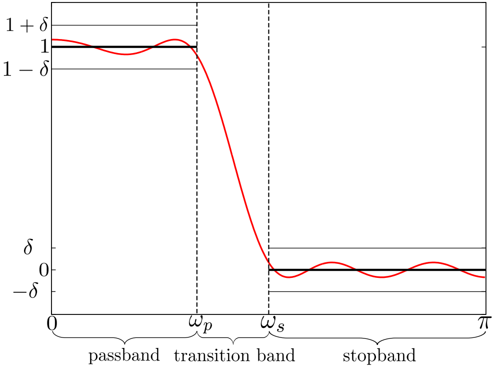
Usual representation: $H(\omega) = \sum_{k=0}^{n} h_k \cos(\omega k) = \sum_{k=0}^{n} h_k T_k(\cos(\omega))$
$\rightarrow$ if $x = \cos(\omega)$, view H in the basis of Chebyshev polynomials
Specification:



Finite Impulse Response (FIR) filters

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

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



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

Optimal FIR design with real coefficients

The problem: Given a closed real set $F \subseteq [0, \pi]$, find an approximation $H(\omega) = \sum_{k=0}^{n} h_k \cos(\omega k)$ of degree at most n for a continuous function $D(\omega), \omega \in F$ such that

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

is **minimal**.

W - real valued weight function, continuous and positive over F.

Optimal FIR design with real coefficients

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

Theorem

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

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

for $i = 0, \ldots, n$ *and* $\delta = \|E(\omega)\|_{\infty,F}$.

Optimal FIR design with real coefficients

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

Theorem

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

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

for $i = 0, \ldots, n$ *and* $\delta = \|E(\omega)\|_{\infty,F}$.

$\rightarrow$ well studied in Digital Signal Processing literature

1972: Parks and McClellan

$\rightarrow$ based on a powerful iterative approach from Approximation Theory:

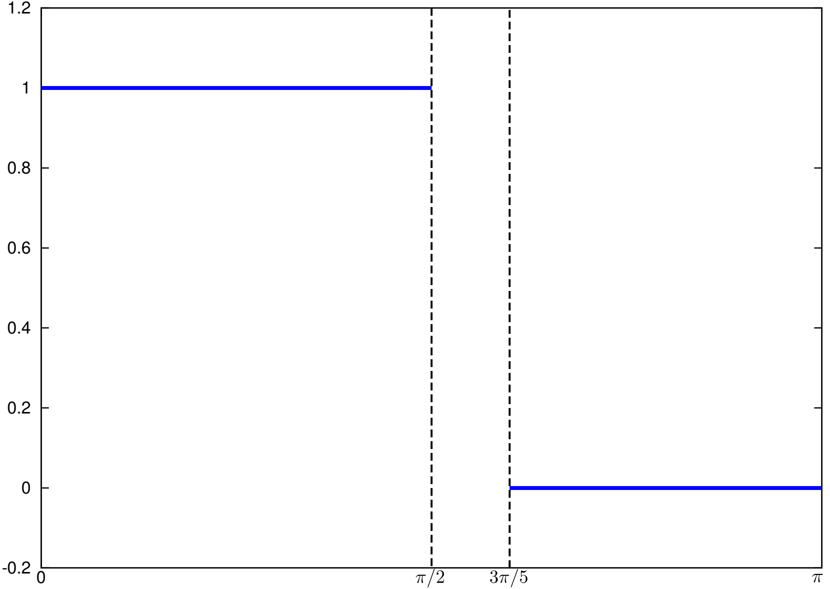
1934: Remez

The Parks-McClellan design method: Motivation

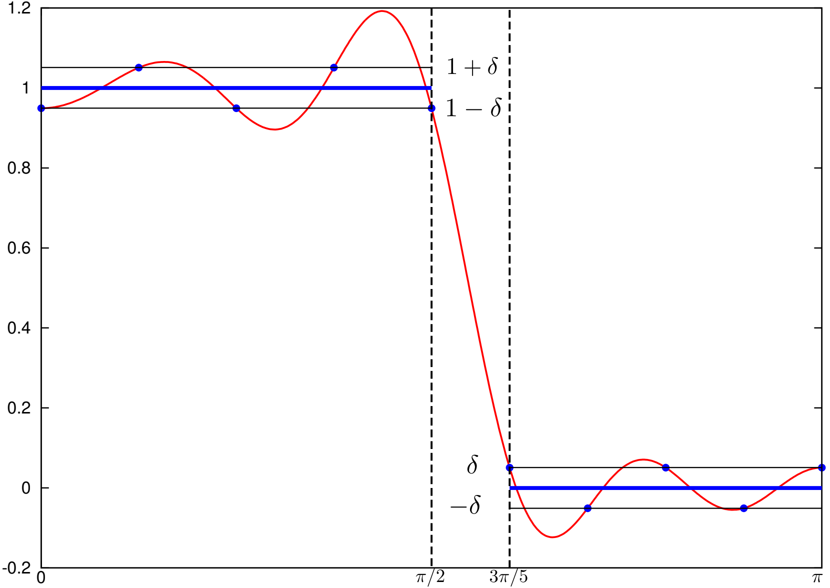
Why work on such a problem?

- one of the most well-known filter design methods
- no concrete study about its numerical behavior in practice
- need for high degree ($n > 500$) filters + existing implementations not able to provide them (e.g. MATLAB, SciPy, GNURadio)
- useful for attacking the coefficient quantization problem

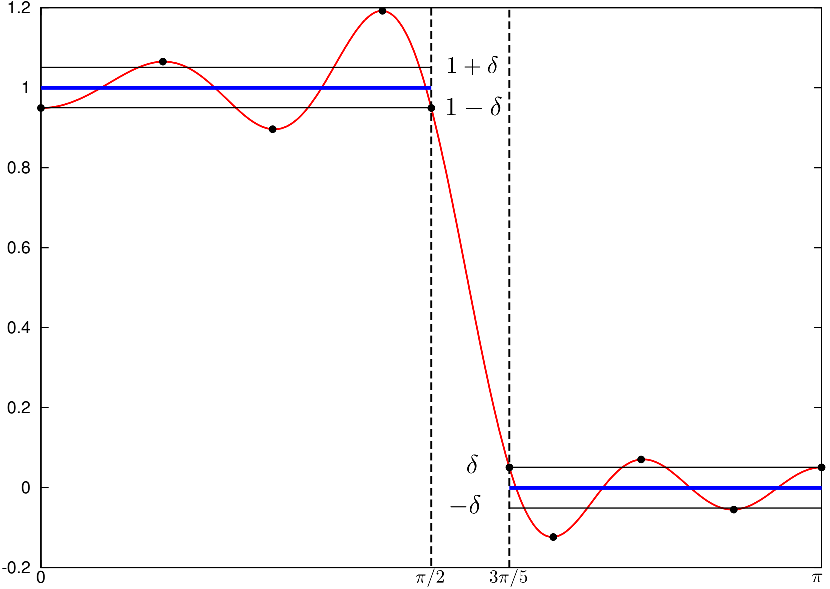
The Parks-McClellan design method: Example



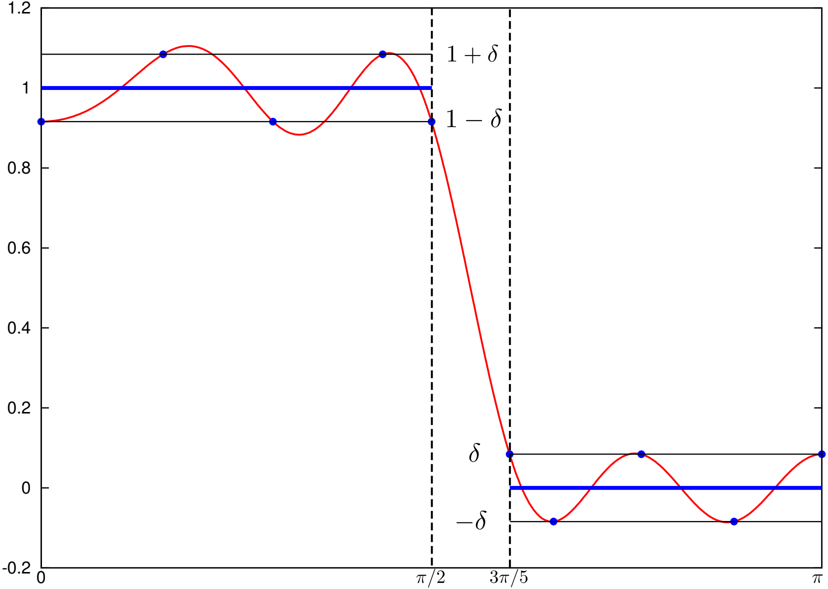
The Parks-McClellan design method: Example



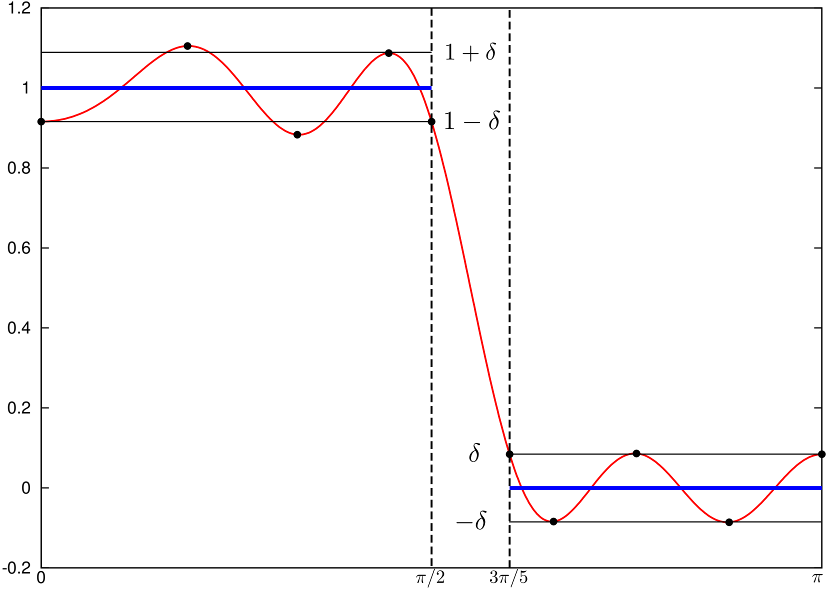
The Parks-McClellan design method: Example



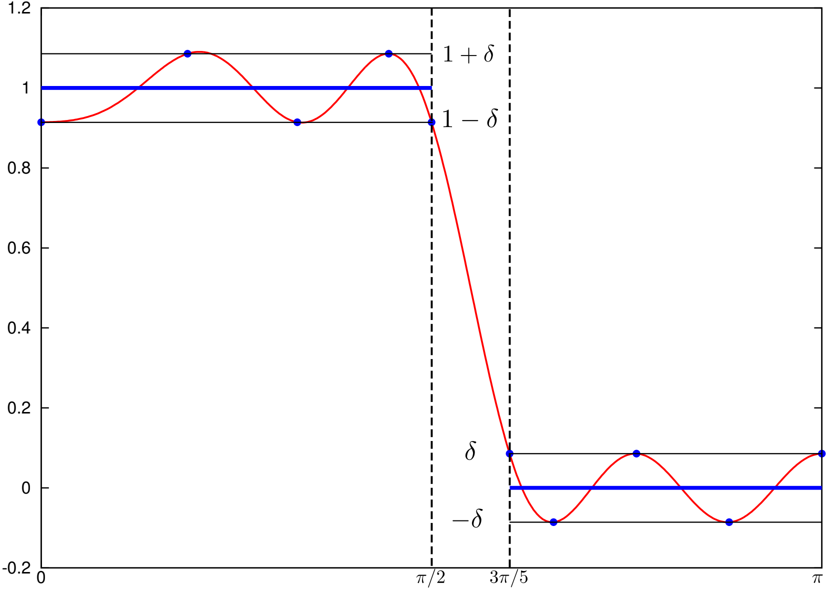
The Parks-McClellan design method: Example



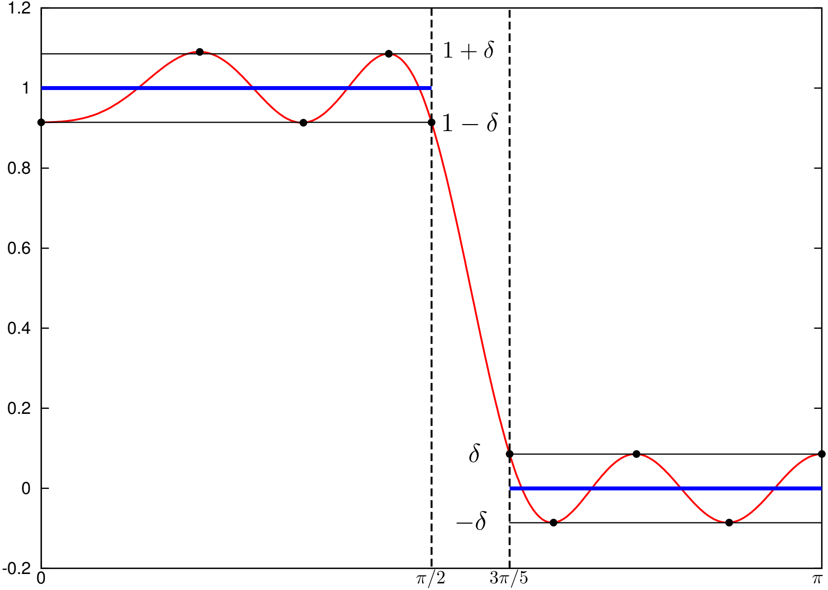
The Parks-McClellan design method: Example



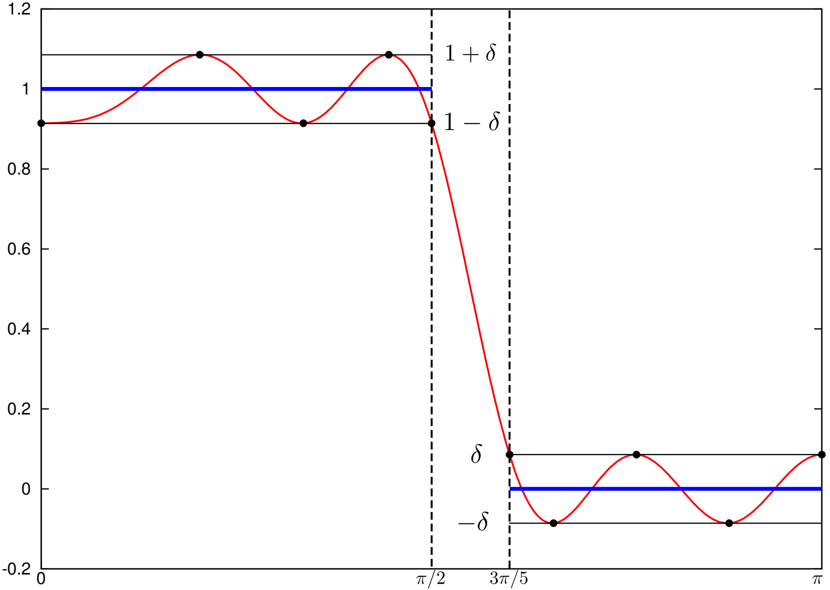
The Parks-McClellan design method: Example



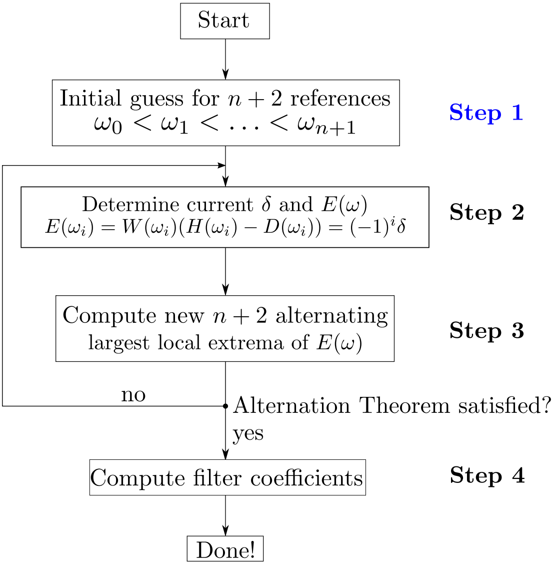
The Parks-McClellan design method: Example



The Parks-McClellan design method: Example



The Parks-McClellan design method: Steps



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

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

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

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

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

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

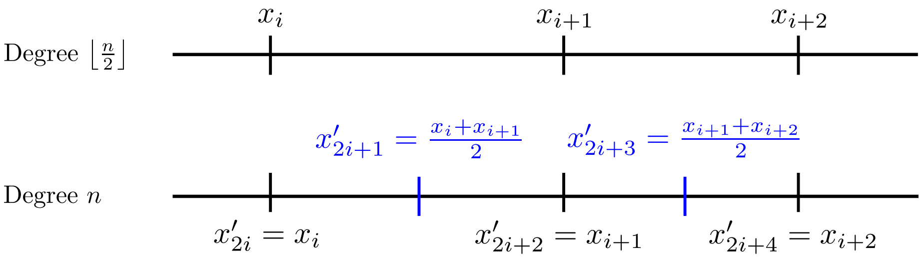
Traditional approach: take the $n+2$ references uniformly from F

$\rightarrow$ can lead to convergence problems

$\rightarrow$ **want to start** from better approximations

Existing approaches: most are not general enough and/or costly to execute

Our approach: extrema position extrapolation from smaller filters



$\rightarrow$ although empirical, this **reference scaling** idea is rather robust in practice

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

Examples

1. degree $n = 520$ unit weight filter with passband $[0, 0.99\pi]$ and stopband centered at π
   $\rightarrow$ removal of harmonic interference inside signals
2. degree $n = 53248$ lowpass filter with passband $\left[0, \frac{1}{8192}\pi\right]$ and stopband $\left[\frac{3}{8192}\pi, \pi\right]$
   $\rightarrow$ design of efficient wideband channelizers for software radio systems

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

Examples + comparison with uniform initialization

1. degree $n = 520$ unit weight filter with passband $[0, 0.99\pi]$ and stopband centered at π
   $\rightarrow$ removal of harmonic interference inside signals
2. degree $n = 53248$ lowpass filter with passband $\left[0, \frac{1}{8192}\pi\right]$ and stopband $\left[\frac{3}{8192}\pi, \pi\right]$
   $\rightarrow$ design of efficient wideband channelizers for software radio systems

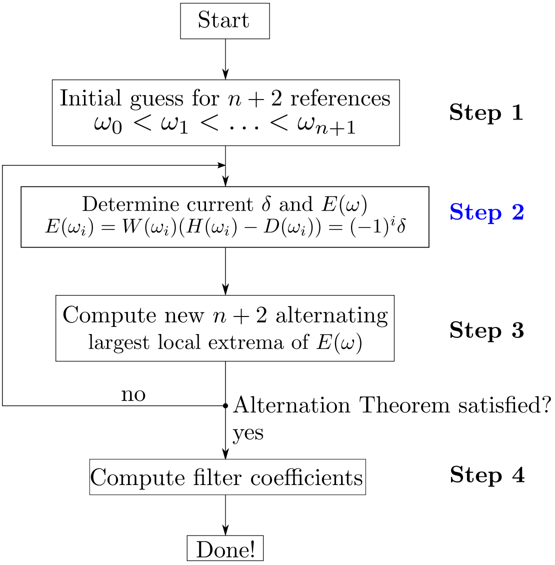
Example 1		Example 2	
Degree	Iterations	Degree	Iterations
520	12/**3**	53248	NC[1]/**3**

Advantages:

- reduced number of iterations
- improved numerical behavior

[1] our implementation did not converge when using uniform initialization

The Parks-McClellan design method: Steps



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

Amounts to solving a linear system in $h_0, \ldots, h_n$ and δ.

$$\begin{bmatrix} 1 & \cos(\omega_0) & \cdots & \cos(n\omega_0) & \frac{1}{W(\omega_0)} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \cos(\omega_n) & \cdots & \cos(n\omega_n) & \frac{(-1)^n}{W(\omega_n)} \\ 1 & \cos(\omega_{n+1}) & \cdots & \cos(n\omega_{n+1}) & \frac{(-1)^{n+1}}{W(\omega_{n+1})} \end{bmatrix} \begin{bmatrix} h_0 \\ \vdots \\ h_n \\ \delta \end{bmatrix} = \begin{bmatrix} D(\omega_0) \\ \vdots \\ D(\omega_n) \\ D(\omega_{n+1}) \end{bmatrix}$$

$\rightarrow$ 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 $h_0, \dots, h_n$ and δ.

$$\begin{bmatrix} 1 & \cos(\omega_0) & \cdots & \cos(n\omega_0) & \frac{1}{W(\omega_0)} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \cos(\omega_n) & \cdots & \cos(n\omega_n) & \frac{(-1)^n}{W(\omega_n)} \\ 1 & \cos(\omega_{n+1}) & \cdots & \cos(n\omega_{n+1}) & \frac{(-1)^{n+1}}{W(\omega_{n+1})} \end{bmatrix} \begin{bmatrix} h_0 \\ \vdots \\ h_n \\ \delta \end{bmatrix} = \begin{bmatrix} D(\omega_0) \\ \vdots \\ D(\omega_n) \\ D(\omega_{n+1}) \end{bmatrix}$$

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

$\rightarrow$ Parks & McClellan's idea: use **barycentric** form of Lagrange interpolation

Barycentric Lagrange interpolation

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

$$p(x_k) = f_k, k = 0, \ldots, n$$

Barycentric Lagrange interpolation

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

$$p(x_k) = f_k, k = 0, \ldots, n$$

$\rightarrow$ schoolbook solution:

$$p(x) = \sum_{k=0}^{n} f_k \ell_k(x), \qquad \ell_k(x) = \prod_{i=0, i \neq k}^{n} \frac{x - x_i}{x_k - x_i}$$

Cost: $O(n^2)$ operations for evaluating $p(x)$, *each* time

Barycentric Lagrange interpolation

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

$$p(x_k) = f_k, k = 0, \ldots, n$$

$\rightarrow$ schoolbook solution:

$$p(x) = \sum_{k=0}^{n} f_k \ell_k(x), \qquad \ell_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}$$

Cost: $O(n^2)$ operations for evaluating $p(x)$, *each* time
Can we do better?

Barycentric Lagrange interpolation

YES $\rightarrow$ the barycentric form of p:

$$p(x) = \frac{\displaystyle\sum_{k=0}^{n} \frac{w_k}{x - x_k} f_k}{\displaystyle\sum_{k=0}^{n} \frac{w_k}{x - x_k}}, \qquad w_k = \frac{1}{\prod_{i \neq k}(x_k - x_i)}$$

Cost: $O(n^2)$ operations for computing the weights w_k (done *once*) + $O(n)$ operations for evaluating $p(x)$

Barycentric Lagrange interpolation

→ we get:

$$\delta = \frac{\displaystyle\sum_{k=0}^{n+1} w_k D(\omega_k)}{\displaystyle\sum_{k=0}^{n+1} \frac{(-1)^k w_k}{W(\omega_k)}}, \qquad w_k = \frac{1}{\prod_{i \neq k}(x_k - x_i)}$$

and

$$H(\omega) = \frac{\displaystyle\sum_{k=0}^{n+1} \frac{w_k}{x - x_k} c_k}{\displaystyle\sum_{k=0}^{n+1} \frac{w_k}{x - x_k}},$$

where $x = \cos(\omega), x_k = \cos(\omega_k)$ and $c_k = D(\omega_k) - (-1)^k \dfrac{\delta}{W(\omega_k)}$.

Barycentric Lagrange interpolation

Why should we use it?
$\rightarrow$ numerically stable **if the family of interpolation nodes used has a small Lebesgue constant** [Higham2004;Mascarenhas&Camargo2014]

The Lebesgue constant: specific for each grid of points; measures the quality of a polynomial interpolant with respect to the function to be approximated

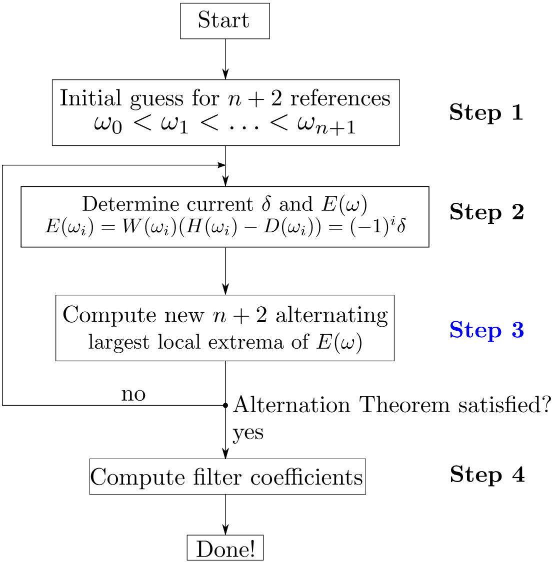
Barycentric Lagrange interpolation

Why should we use it?
$\rightarrow$ numerically stable **if the family of interpolation nodes used has a small Lebesgue constant** [Higham2004;Mascarenhas&Camargo2014]

The Lebesgue constant: specific for each grid of points; measures the quality of a polynomial interpolant with respect to the function to be approximated

$\rightarrow$ 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

The Parks-McClellan design method: Steps



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 $16n$)
$\rightarrow$ can sometimes fail to find all the extrema
$\rightarrow$ need for a more robust alternative

Chebyshev-proxy rootfinders [Boyd2006,Boyd2013]

$f(x)$

Interpolation

$f_m(x) = \sum_{k=0}^{m} a_k T_k(x)$

$$\begin{bmatrix} 0 & 1 & & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & & \\ & \frac{1}{2} & 0 & \frac{1}{2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & & & \frac{1}{2} \\ -\frac{a_0}{2a_m} & -\frac{a_1}{2a_m} & -\frac{a_2}{2a_m} & & \frac{1}{2}-\frac{a_{m-2}}{2a_m} & -\frac{a_{m-1}}{2a_m} \end{bmatrix}$$

Chebyshev companion matrix

QR/QZ eigensolver

$\boxed{\text{roots of } f(x) \text{ are eigenvalues}}$

Chebyshev-proxy rootfinders [Boyd2006,Boyd2013]

$\rightarrow$ **numerically stable** for finding the real roots of $f_m(x)$ located inside $[-1, 1]$
Cost: around $10m^3$ operations according to [Boyd2013]

Important questions:

- What value is suitable for m?
  Depends on f. Can be computed adaptively (like inside the `Chebfun`[2] Matlab™ package)
- Can the computational cost be reduced?
  YES. To $O(m^2)$. Through **interval subdivision** OR **direct quadratic solvers**.

[2] `http://www.chebfun.org/`

Interval subdivision

Key idea: use the *trigonometric form* of f_m (i.e. $x = \cos(\omega), \omega \in [0, \pi]$)

Interval subdivision

Key idea: use the *trigonometric form* of f_m (i.e. $x = \cos(\omega), \omega \in [0, \pi]$)

$\rightarrow$ several alternatives [Boyd2006] for finding the roots of f_m:

- k-m **subdivision** algorithms: split $[0, \pi]$ into m *uniform* subintervals + degree k Chebyshev interpolation on each subinterval
  Cost estimate: $22000m + 42m^2$ operations, for $k = 13$ ("tredecic" subdivision)

Interval subdivision

Key idea: use the *trigonometric form* of f_m (i.e. $x = \cos(\omega), \omega \in [0, \pi]$)

$\rightarrow$ several alternatives [Boyd2006] for finding the roots of f_m:

- k-m **subdivision** algorithms: split $[0, \pi]$ into m *uniform* subintervals + degree k Chebyshev interpolation on each subinterval
  Cost estimate: $22000m + 42m^2$ operations, for $k = 13$ ("tredecic" subdivision)

- **linear-with-cubic solve** algorithm: adaptive interval subdivision + cubic interpolation + zero-free interval testing
  Cost estimate: $400m^2$ operations, for problems with $O(m)$ simple roots

Interval subdivision

Key idea: use the *trigonometric form* of f_m (i.e. $x = \cos(\omega), \omega \in [0, \pi]$)

$\rightarrow$ several alternatives [Boyd2006] for finding the roots of f_m:

- k-m **subdivision** algorithms: split $[0, \pi]$ into m *uniform* subintervals + degree k Chebyshev interpolation on each subinterval
  Cost estimate: $22000m + 42m^2$ operations, for $k = 13$ ("tredecic" subdivision)

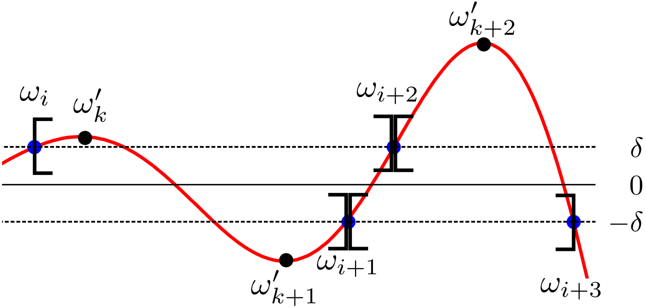
- **linear-with-cubic solve** algorithm: adaptive interval subdivision + cubic interpolation + zero-free interval testing
  Cost estimate: $400m^2$ operations, for problems with $O(m)$ simple roots

Which one to use? $\rightarrow$ problem-dependent

Interval subdivision: Our problem

local extrema of $E(\omega)$ $\rightarrow$ **roots** of $E'(\omega)$
What we know, at each iteration:

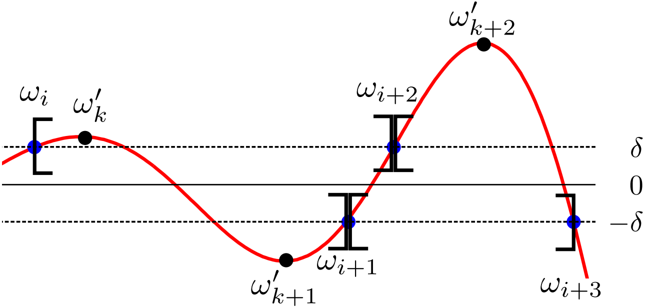
- $E(\omega)$ usually has *very close* to $n+2$ local extrema inside F
- placement information for the local extrema



Interval subdivision: Our problem

local extrema of $E(\omega)$ $\rightarrow$ **roots** of $E'(\omega)$
What we know, at each iteration:

- $E(\omega)$ usually has *very close* to $n+2$ local extrema inside F
- placement information for the local extrema



Our approach: k-n-type subdivision with **non-uniform** subintervals

Interval subdivision: Our solution

Why use it?

- works very well in practice
- $k = 4$ is usually sufficient $\rightarrow$ small computational cost
- no need for zero-free interval testing
- embarrassingly parallel approach

Direct quadratic solvers

$\rightarrow$ investigated in a number of recent articles.
$\rightarrow$ tend to be faster than classic QR/QZ schemes for $n > 80$.

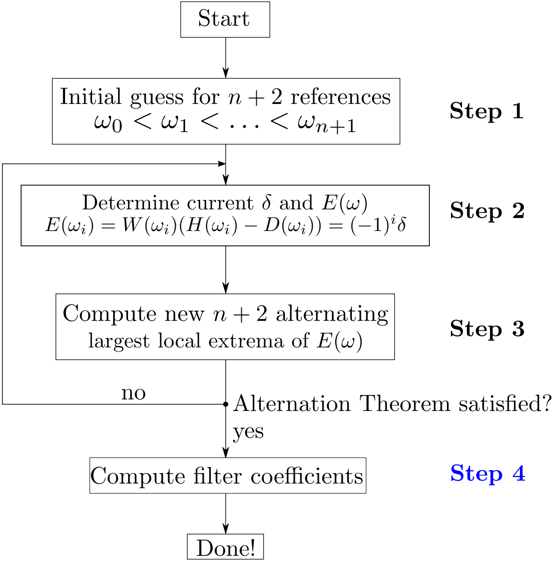
Direct quadratic solvers

$\rightarrow$ investigated in a number of recent articles.
$\rightarrow$ tend to be faster than classic QR/QZ schemes for $n > 80$.

Some questions:

- How do such methods compare to subdivision approaches?
- When is it worthwhile to use them with Chebyshev basis expansions?
- Can they be easily parallelized?

The Parks-McClellan design method: Steps



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: $O(n^2)$ arithmetic operations

Some remarks about convergence

Notation:
H^* - final minimax filter
H_k - the filter computed at the k-th iteration of the Parks-McClellan algorithm

Some remarks about convergence

Notation:
H^* - final minimax filter
H_k - the filter computed at the k-th iteration of the Parks-McClellan algorithm

$\rightarrow$ theoretically, **linear** convergence is always possible [Cheney1966], i.e.

$$\max_{\omega \in F} |W(\omega)(H^*(\omega) - H_k(\omega))| \leqslant A\theta^k,$$

for some $A > 0$ and $\theta \in (0, 1)$.

Some remarks about convergence

Notation:
H^* - final minimax filter
H_k - the filter computed at the k-th iteration of the Parks-McClellan algorithm

$\rightarrow$ theoretically, **linear** convergence is always possible [Cheney1966], i.e.

$$\max_{\omega \in F} |W(\omega)(H^*(\omega) - H_k(\omega))| \leqslant A\theta^k,$$

for some $A > 0$ and $\theta \in (0, 1)$.
$\rightarrow$ if $D(\omega)$ twice differentiable and $E^*(\omega) = W(\omega)(D(\omega) - H^*(\omega))$ equioscillates *exactly* $n + 2$ times, we have **quadratic** convergence [Veidinger1960]

Our implementation

$\rightarrow$ written in C++

Our implementation

$\rightarrow$ written in C++
$\rightarrow$ small number of external dependencies:

- `Eigen`;
- `Intel TBB`;
- `MPFR`.

Our implementation

→ written in C++
→ small number of external dependencies:

- `Eigen`;
- `Intel TBB`;
- `MPFR`.

→ optional parallelism with `OpenMP`

Our implementation

→ written in C++
→ small number of external dependencies:

- `Eigen`;
- `Intel TBB`;
- `MPFR`.

→ optional parallelism with `OpenMP`
→ comes in three flavors:

- `double`
- `long double`
- `MPFR`

Our implementation: Results

Examples:

1. degree $n = 100$ unit weight filter with passband $[0, 0.4\pi]$ and stopband $[0.5\pi, \pi]$
2. degree $n = 100$ unit weight filter with passbands $[0, 0.2\pi], [0.6\pi, \pi]$ and stopband $[0.3\pi, 0.5\pi]$
3. degree $n = 520$ unit weight filter with passband $[0, 0.99\pi]$ and stopband centered at π
4. degree $n = 53248$ lowpass filter with passband $\left[0, \frac{1}{8192}\pi\right]$ and stopband $\left[\frac{3}{8192}\pi, \pi\right]$

Our implementation: Results

→ running times (in seconds) on a 3.6 GHz 64-bit Intel Xeon(R) E5-1620

Problem	Uniform (sequential)	GNURadio	MATLAB	SciPy
Example 1 ($n = 100$)	0.0112	NC	0.1491	0.3511
Example 2 ($n = 100$)	0.0395	NC	NC	NC
Example 3 ($n = 520$)	0.3519	NC	NC	NC
Example 4 ($n = 53248$)	NC	NC	NC	NC

Example (degree)	Uniform (sequential)	Uniform (parallel)	Scaling (sequential)	Scaling (parallel)
Example 1 ($n = 100$)	0.0112	0.0073	0.0147	0.011
Example 2 ($n = 100$)	0.0395	0.0274	0.0339	0.0275
Example 3 ($n = 520$)	0.3519	0.2251	0.0982	0.0716
Example 4 ($n = 53248$)[3]	NC	NC	537.8	162.6

[3] used the `long double` version of our code

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:

- 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