Algorithm Mapping...

Adaptive Filters...
Filter coefficients/parameters not fixed but adjustable

Different Structures: Noise Cancellation, System Identification, Adaptive Predictor, Adaptive notch, Adaptive Channel Equalization

Algorithm Mapping...

Adaptive Filters...
LMS and RLS algorithms
The LMS is well suited for a number of applications, including adaptive echo and noise cancellation, equalization, and prediction.

LMS, Sign-Error LMS, Sign-Data LMS, Sign-sign LMS, Normalized LMS

\[ w_k(n+1) = w_k(n) + \beta \text{sgn}[e(n)]x(n-k) \]

\[ w_k(n+1) = w_k(n) + \beta \text{sgn}[e(n)] \text{sgn}[x(n-k)] \]

\[ w_k(n+1) = w_k(n) + \beta \text{sgn}[e(n)]w(n-k) \]

\[ w_k(n+1) = w_k(n) + \frac{\beta}{\sigma} w(n-k)e(n) \]

\[ \beta: \text{adaptation gain} \]

No multiplication needed in sign-sign
But for C6000 structure these structures are slower than LMS! ← Conditionals

RLS: Recursive Least square
The RLS is based on starting with the optimal solution and then using each input sample to update the impulse response in order to maintain that optimality.

Faster Convergence, Slower Execution

Algorithm Mapping...

Adaptive Filters...
Adaptive Linear Combiners → Single or Multiple inputs
one of the most useful adaptive filter structures

The configuration consists of K input signals, each of which is weighted by \( w(k) \) and combined to form the output:

\[ y(n) = \sum_{k=0}^{K} w(k,n)x(k,n) \]

vector formulation:

\[ X(n) = [x(0,n) \ x(1,n) \ \cdots \ x(K,n)]^T \]

\[ W(n) = [w(0,n) \ w(1,n) \ \cdots \ w(K,n)]^T \]

Same notation for both cases

Using vector notation, \( n \) is time index:

\[ y(n) = X(n)W(n) - W(n)X(n) \]

Example: Two weights, \( K = 1 \)

\[ y(n) = \sum_{k=0}^{K} w(k,n)x(n-k) = w(0,n)x(n) + w(1,n)x(n-1) \]

\[ y(n) = [x(n) \ x(n-1)] [w(0) \ w(1)] [x(n) \ x(n-1)]^T \]

\[ = x(n)w(0) + x(n-1)w(1) \]

In the case of a single input, the structure reduces to a \((K + 1)\)-tap FIR filter
Each delayed input is weighted and summed to produce the output
Just an FIR filter with adjustable coefficients
Algorithm Mapping...

**Adaptive Filters**

**Performance Function**

Adjust the weights to minimize error power, using basic structure and combiner...

\[ e(n) = d(n) - y(n) \]
\[ e^2(n) = d^2(n) - 2d(n)y(n) + y^2(n) \]
\[ e^2(n) = d^2(n) - 2d(n)X^T(n)W + W^T X(n)X^T(n)W \]

**Simple single weight case:**

\[ e^2(n) = d^2(n) - 2d(n)x(n)w(0) + x^2(n)w^2(0) \]
\[ \frac{de^2}{dw(0)} = -2d(n)x(n) + 2x^2(n)w(0) = 0 \]
\[ w(0) = \frac{d(n)}{x(n)} \]
\[ E[e^2(n)] = E[d^2(n)] - 2E[d(n)x(n)]w(0) - E[x^2(n)]w^2(0) \]
\[ w(0) = \frac{1}{x} \]

**3-dimensional quadratic surface (bowl) for two weight case**

\[ \beta < \alpha < \lambda_{\text{max}} \]

**Algorithm Mapping...**

**Adaptive Filters**

**Performance Function...**

In practice, the weights start at some initial value \( w \), and are adjusted in increments toward the minimum value of the performance function

\[ e^2(n) = d^2(n) - 2d(n)X^T(n)W + W^T X(n)X^T(n)W \]

**Taking the expected values:**

\[ E[e^2(n)] = E[d^2(n)] - 2E[d(n)X(n)]W + W^T E[X(n)X^T(n)]W \]

**Looking for the minimum**

Steepest descent or gradient method

Stepwise fashion until minimum is reached

\[ \text{grad}[E[e^2]] = \text{grad}[P] = \begin{bmatrix} \frac{\partial P}{\partial w(0)} & \frac{\partial P}{\partial w(1)} & \cdots & \frac{\partial P}{\partial w(K)} \end{bmatrix}^T \]

Direction of the extreme point is opposite to gradient direction, or \(-\text{grad}[P]\):

\[ W(n+1) = W(n) - \beta \text{grad}[P] \]

At the extreme point:

\[ \frac{\partial P}{\partial w(0)} = 0, \quad \frac{\partial P}{\partial w(1)} = 0, \quad \cdots, \quad \frac{\partial P}{\partial w(K)} = 0 \]

**Algorithm Mapping...**

**Adaptive Filters**

**Performance Function...**

Ignoring the Expectation for a while

\[ \text{grad}[P] \approx \text{grad}[e^2] \rightarrow \text{grad}[e^2] = 2e \text{ grad}[e] \]

\[ e(n) = [d(n) - X^T(n)W(n)] \]

**LMS**

\[ e^2(n) = 2e \text{ grad}[d(n) - X^T(n)W(n)] \]
\[ \frac{de}{dw(0)} = -2e \frac{d(n)}{x(n)} \]
\[ \frac{de}{dw(1)} = -2e \frac{d(n)}{x(1)} \]
\[ \frac{de}{dw(K)} = -2e \frac{d(n)}{x(K)} \]

\[ \text{grad}[e^2(n)] = -2e(n)X(n) \]

\[ \text{LMS Algorithm} \rightarrow \text{ C6000 Code example} \]

**Algorithm Mapping...**

**Adaptive Filters**

**Performance Function...**

Ignoring the Expectation for a while

\[ e(n) = [d(n) - X^T(n)W(n)] \]
\[ C(W_n) = \sum_{i=0}^{n} \lambda^{n-i} e(i)^2 = \sum_{i=0}^{n} \lambda^{n-i} e(i) e^*(i) \quad 0 < \lambda \leq 1 \]

\[ \frac{\partial C(W_n)}{\partial w^*(i)} = \sum_{k=0}^{n} \lambda^{n-i} e(i) \frac{\partial e^*(i)}{\partial w^*(k)} = \sum_{k=0}^{n} \lambda^{n-i} e(i) x^*(i - k) = 0 \]

Replacing \( e(n) \) and rearranging:

\[ \sum_{i=0}^{n} \lambda^{n-i} d(i) = \sum_{i=0}^{n} \lambda^{n-i} e(i) x^*(i - k) = 0 \]
\[ \sum_{i=0}^{n} \lambda^{n-i} e(i) x^*(i - k) = \sum_{i=0}^{n} \lambda^{n-i} d(i) x^*(i - k) \]
\[ R_x(n) w_n = r_d(n) \rightarrow w_n = R_x^{-1}(n) r_d(n) \]

Dependency on the past is determined by \( \lambda \)
Algorithm Mapping...

**Adaptive Filters...**

**RLS...**

Recursive formulation:
\[ w_n = w_{n-1} + \Delta w_{n-1} \]
\[ x(i) = [x(i), x(i-1), \ldots, x(i-p)]^T \]

**Woodbury Relation**
\[ \delta < 0.01/\sigma^2 \]

**Recursion:**
\[ \alpha(n) = P(n-1) - g(n)x^T(n)P(n-1)x(n) \]
\[ g(n) = P(n-1)x^T(n)H^T(n) \]
\[ w_n = P(n-1)x^T(n) + d(n)x^T(n) \]

**Kalman Filters**

- **Good resources at** [http://www.cs.unc.edu/~welch/kalman/](http://www.cs.unc.edu/~welch/kalman/)
- **Kalman filter** is an optimal recursive data processing algorithm
- Kalman filter is optimal with respect to virtually any criterion that makes sense!
  - Can use all the information that is provided to it
  - Processes all the measurements even erroneous ones
  - Minimum error energy criteria (Least Square Sense)
- **Kalman filter** does not need all the previous data to be available at once

**Kalman Filter**

- **Formulation**
- **Stochastic Difference Equation**

\[ x_k = A x_{k-1} + B u_{k-1} + w_{k-1} \]
\[ p(w) \sim N(0, Q) \]
\[ p(u) \sim N(0, R) \]
\[ p(v) \sim N(0, R) \]
\[ x_k \in \mathbb{R}^N \]
\[ z_k \in \mathbb{R}^M \]
\[ n \in \mathbb{R}^I \]

**Computation**

Defining two estimations and corresponding errors:
- a priori estimation:
  \[ \hat{x}_k \in \mathbb{R}^N \]
  \[ e_k = x_k - \hat{x}_k \]
- a posteriori estimation:
  \[ \hat{x}_k \in \mathbb{R}^N \]
  \[ e_k = x_k - \hat{x}_k \]
Algorithm Mapping…

Adaptive Filters... Kalman Filters...

Computation...

Covariance Matrices (n×n, prior and posterior) of these errors:

\[ P_k' = E[e_k e_k^T] \quad P_k = E[e_k e_k^T] \]

Goal of Kalman filtering: posterior estimate of state as a linear combination of

1) prior estimate and 2) the measurement

\[ \hat{x}_k = \hat{x}_{k^-} + K(z_k - H\hat{x}_{k^-}) \]

innovation or residual

discrepancy between (1) and (2)

K: n×m → called gain or blending matrix → chosen to minimize \( P_k \)

Stochastic Justification

Roots: Probability of the a priori estimate conditioned on all prior measurements

(Bayes’ rule: Normal prior and posterior)

For normally distributed noises:

\[ p(x_k|z_k) \sim N(E[x_k], E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T]) \]

\[ E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] = P_k \]

Stochastic Justification

Example: Simple 1D Forecaster, Smoother

A system that uses consecutive measurements for estimating the average:

\[ \mu_n = \frac{1}{n} \sum_{i=1}^{n} x_i \]

Average is more trustworthy than a single sample, so the weight of \( \mu \) is bigger...

Same thing can be done for variance estimation

\[ \sigma_n^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2 \]

These equations are used in calculators when numbers are entered sequentially

No need to keep all the numbers for calculating mean and variance

General flavor of Kalman filters → kind of recursive least square procedure

Algorithm Mapping...

Adaptive Filters... Kalman Filters...

Stochastic Justification...

Example: Multiple instruments of different accuracy

\( x_i \): the reading of the first instrument, variance of error \( \sigma_i^2 \), \( 1 \leq i \leq N \)

For N=2

→ if \( \sigma_1 = \sigma_2 \): average of measurements will be the best guess

→ if \( \sigma_1 < \sigma_2 \): \( x_1 \) will be the best guess

→ otherwise: weighted sum will be the best guess:

\[ x = \frac{\sigma_2}{\sigma_1 + \sigma_2} x_1 + \frac{\sigma_1}{\sigma_1 + \sigma_2} x_2 \]

\[ \hat{x} = x_1 + K(x_2 - x_1) \]

\[ K = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \]

\[ p_1(x) = \frac{1}{\sqrt{2\pi\sigma_1}} e^{-\frac{1}{2}(x-x_1)^2/\sigma_1^2} \]

\[ p_2(x) = \frac{1}{\sqrt{2\pi\sigma_2}} e^{-\frac{1}{2}(x-x_2)^2/\sigma_2^2} \]

\[ p(x) = p_1(x)p_2(x) = \int \left[ e^{-\frac{1}{2}(x-x_1)^2/\sigma_1^2} \right]^{\frac{1}{2}} \left[ e^{-\frac{1}{2}(x-x_2)^2/\sigma_2^2} \right]^{\frac{1}{2}} \]

Improved...

Algorithm Mapping...

Adaptive Filters... Kalman Filters...

Optimization

\( K \) is chosen to minimize a posteriori error covariance

Derivation

→ Substitute blending equation into definition of \( e_k \) → Calculate \( P_k \)

→ Taking the derivative of the trace of \( P_k \) with respect to \( K \)

→ Setting to zero and solve for \( K \)

One form of the solution:

\[ K_k = P_k H^T (HP_k H^T + R)^{-1} \]

Intuition:

\[ \lim_{R_k \to 0} K_k = H^{-1} \quad \lim_{P_k \to 0} K_k = 0 \]

The discrete Kalman algorithm resembles a feedback system:

1. The filter estimates the process state at some time

2. Then obtains feedback in the form of (noisy) measurements
Algorithm Mapping...

**Adaptive Filters... Kalman Filters...**

*Algorithm*

Two types of equations:
- For projecting forward (in time) the current state and error covariance estimates → obtain a priori estimates for the next time step
- For incorporating a new measurement into the a priori estimate (the feedback) → obtain an improved a posteriori estimate

\[
K_k = P_k H^T (H P_k H^T + R)^{-1}
\]

\[
\hat{x}_k = A \hat{x}_{k-1} + Bu_{k-1}
\]

\[
\hat{x}_k = \hat{x}_k + K_k(z_k - H \hat{x}_k)
\]

\[
P_k = (I - K_k H) P_{k-1}
\]

After each time and measurement update pair, the process is repeated with the previous a posteriori estimates used to project or predict the new a priori estimates.

Recursive form makes it feasible (e.g., compare to Wiener filters, needing all the data).

**Implementation**

1) The measurement noise covariance is measured prior to operation of the filter
   - Offline sample measurements
2) Process noise is often difficult to measure, simple assumption will be good enough
3) System dynamic and measurement parameters must be known or guessed
   - simple (poor) process model can produce acceptable results if enough uncertainty is injected into the process via the selection of Q

**Extended Kalman Filters**

If the process or measurement equations are non-linear → Extending the Kalman stuff

EKF:

Kalman filter that linearizes about the current mean and covariance using the partial derivatives (Jacobian Matrices) → noise is ignored for this part

\[
x_k = f(x_{k-1}, u_{k-1}, w_{k-1})
\]

\[
z_k = h(x_k, v_k)
\]

Kalman filter removes noise by assuming a pre-defined system model
→ Model must be meaningful

1) Understand the problem: break down the problem into the math basics
2) Model the state process: start with a basic model, may not work effectively at first, but this can be refined later
3) Model the measurement process: Analyze how you are going to measure the process
4) Model the noise: This needs to be done for both the state and measurement process. The base Kalman filter assumes Gaussian white noise, so make the variance/covariance (of error) meaningful
5) Test the filter: Often overlooked, use synthetic data if necessary (e.g., if the process is not safe to test on a live environment). See if the filter is behaving as it should.
6) Refine the filter: Try to change the noise parameters (filter), as this is the easiest to change. If necessary go back further, you may need to rethink the situation.
Algorithm Mapping...

Adaptive Filter... Kalman Filters ...

Practical Stuff...

Example: Water tank level meter

Level of the float is used for measurement, electronic or mechanical float

System understanding:

- Average level \( \rightarrow \) Filling/empting or static cases
- Float and tank level \( \rightarrow \) Sloshing or stagnant cases

Model the state process:

Starting with the simplest possible model, static level = constant \( \leftarrow \) scalar

\[ x_k = x_{k-1} \]

Model the measurement process:

Could be a scaled measurement but for simplicity \( \rightarrow \) \( H = 1 \)

Model the noise:

Scalar noise: \( R = r \) measurement, \( Q = q \) process

suitable process noise powers are tried to find a complete model

Test the filter

Predict:

\[ x_k = x_{k-1} + K_k (z_k - x_{k-1}) \]
\[ p_k^- = p_{k-1} + q \]
\[ z_k = x_k + r_k \]

Update:

\[ K_k = p_{k-1} (p_k^- + r_k)^{-1} \]
\[ p_k^+ = p_{k-1} (1 - K_k) \]

Initialize: \( x_0 = 0, p_0 = 1000, L_0 = 1, r = 0.1, q = 0.0001 \) (accurate model is assumed)

\[ x_1^* = 0 \]
\[ p_1^* = 1000 + 0.0001 \]

Single iteration has brought us to a close answer with small error

\[ x_1 = 0.8999 \]
\[ p_2 = 0.1 + 0.0001 \]

Error variance is decreasing

Small q shows the trust in the model!

And limits the system flexibility

Small q meant we trusted the model!

Very noisy measurements \( \%20 \) error Produces much better accuracy of \( \%15 \)

Let's try the same state/measurement model

\[ L_k = L_{k-1} + f \]

Filling case with constant rate: \( L_0 = 0, f = 0.1 \)

Let's try the same state/measurement model
Algorithm Mapping...

Adaptive Filter... Kalman Filters ...
Practical Stuff...
Example: Water tank level meter... Filling: \( L_k = L_{k-1} + f \) \( L_0 = 0, f = 0.1 \)
Relaxing the model \( q = 0.01 \) and \( q = 0.1 \)

Almost no difference to the measured value
There is a minor noise removal but not much

1) having a badly defined model,
   \( \rightarrow \) will not get a good estimate
2) can relax the model by increasing the model error
   \( \rightarrow \) will let the Kalman filter rely more on the measurement values
   \( \rightarrow \) but still allow some noise removal...

What will an increase in the measurement noise 'r' do?

Improving the model \( \rightarrow \) the filling model

Assuming \( \Delta t = 1 \) for simplicity
\( x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \)
\( p = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix} \)
\( q_f = 0.00001 \)
\( r = 0.3 \)

- The filter quickly adapts to the true value.
- Kalman filter was told nothing about the actual filling rate, and it figured it out all by itself, even with a bad unsure initialization.
- If you give a Kalman filter a bad initialization, it takes the first measurement as a "good" initialization.
Algorithm Mapping...

Adaptive Filter... Kalman Filters...
Practical Stuff...
Example: Water tank level meter...
Applying the new model to the static case (no filling)... The filter stabilizes in the exact same time frame as before the model stabilizes with a fill rate of 0...

Algorithm Mapping...

Adaptive Filter... Kalman Filters...
Practical Stuff...
Example: Water tank level meter... Sloshing case
The water is at a constant level but sloshing in the tank
\[ L = c \sin(\omega t) + l \]
\[ l = 1, \quad c = 0.5, \quad \omega = 0.05 \times 2\pi \]

1) Estimation is less noisy but there is a lag
2) The amplitude of estimation is decreasing
Again the model is not good enough
EKF:
\[ A = \begin{bmatrix} 1 & \omega \cos \omega \theta \\ 0 & 1 \end{bmatrix} \]

See: http://www.cs.unc.edu/~welch/kalman/kftool/index.html

Algorithm Mapping...

Special Topics on Filtering...  Multirate Signal Processing...
Decimation and Interpolation \( \rightarrow \) Downsampling and Upsampling
Changing the sampling rate \( \rightarrow \) because of system architecture or to reduce processing (Downsampling, Compressing)

Sampling rate reduction \( \rightarrow \) folding frequency
\[ y(n) = x(nM) \]
\[ f_s/M = \frac{f_s}{M} \]
\[ f_s = f_{Ny} / 2M \]
\[ f_{max} < f_s/M \]
\[ \Omega_{stop} = 2\pi f_{Ny} / 2M = \frac{\pi}{M} \text{ radians} \]

Z and frequency domain:
\[ Y_D(z) = \sum_{n=-\infty}^{\infty} y(n)z^{-n} = \sum_{n=-\infty}^{\infty} x(Mn)z^{-n} \]
Defining:
\[ x_1(n) = \begin{cases} x(n) & \text{if } n \text{ is an integer multiple of } M \\ 0 & \text{otherwise} \end{cases} \]
Then:
\[ Y_D(z) = \sum_{n=-\infty}^{\infty} x_1(Mn)z^{-n} = \sum_{n=-\infty}^{\infty} x_1(kz^{-k/M}) \]
\[ x_1(n) = c_M(n)x(n) \]
\[ c_M(n) = \begin{cases} 1 & \text{if } n \text{ is an integer multiple of } M \\ 0 & \text{otherwise} \end{cases} \]
\[ W_k = e^{-j2\pi k/M} \]
\[ Y_D(z) = X(z^{1/M}) = \sum_{k=0}^{M-1} X(z^{1/M}W_k) \]
And finally:
\[ Y_D(z^{2M}) = \sum_{k=0}^{M-1} X(z^{(2\pi/2M)}) \]

Signal must become bandlimited \( < \pi/M \) \( \rightarrow \) Needs a prefilter
Algorithm Mapping...

Special Topics on Filtering ... Multirate Signal Processing...

Decimation...

Aliasing $f > f_s / M$

If $f_{\text{max}} < f_s / M / 2$

no alias

Otherwise

aliasing filter needed

A gap is needed to have

a filter with reasonable transition width

The bigger is the gap

the smaller is the filter order

Algorithm Mapping...

Special Topics on Filtering ... Multirate Signal Processing...

Interpolation (Upsampling, Expanding)

Increasing the sampling rate

$y(n) = \begin{cases} 
  x(m) & m = nL \\
  0 & \text{otherwise}
\end{cases}$

$\text{zero-stuffing}$

$x(n): 8 \quad 8 \quad 4 \quad -5 \quad -6 \ldots$

$L=3$

$w(m): 800 \quad 800 \quad 400 \quad -500 \quad -600 \ldots$

Algorithm Mapping...

Special Topics on Filtering ... Multirate Signal Processing...

Interpolation... Expander

$Z$ and frequency domain:

$Y_E(z) = \sum_{n=-\infty}^{\infty} y_E(n)z^{-n}$

$- \sum_{k=-\infty}^{\infty} x(k)z^{-kL}$

$\Omega_{\text{sup}} = 2\pi \frac{L}{2} \times \frac{T}{L} = \frac{\pi}{L}$ radians

And

$Y_E(e^{j\omega}) = X(e^{j\omega L})$

Multiple images of compressed $x$ spectra are created in $0$ to $2\pi$ interval

Lowpass filter is needed after rate change to remove the extra images

Algorithm Mapping...

Special Topics on Filtering ... Multirate Signal Processing...

Decimation...

FIR anti aliasing filter design example:

$\text{Sampling rate} = 6,000 \text{ Hz}$

$\text{Input audio frequency range: 0–800 Hz}$

$\text{Passband ripple} : 0.02 \text{ dB} > 0.019$

$\text{Stopband attenuation} : 50 \text{ dB} < 53 \text{ dB}$

Downsample factor $M = 3$ $\rightarrow$ folding at $1000 \text{ Hz}$
**Special Topics on Filtering ... Multirate Signal Processing...**

**Interpolation...**

Aliasing $f > \frac{f_s}{2}$

Interpolation filter needed to remove extra images

A gap is needed to have a filter with reasonable transition width

The bigger is the gap, the smaller is the filter order

---

**FIR interpolation filter design example:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling rate</td>
<td>$6,000$ Hz</td>
</tr>
<tr>
<td>Input audio frequency range</td>
<td>$0$–$800$ Hz</td>
</tr>
<tr>
<td>Passband ripple</td>
<td>$0.02$ dB &gt; $0.019$</td>
</tr>
<tr>
<td>Stopband attenuation</td>
<td>$50$ dB &lt; $53$ dB</td>
</tr>
<tr>
<td>Upsample factor $L$</td>
<td>$3$</td>
</tr>
</tbody>
</table>

---

**Cascade Equivalence**

A cascade of a factor-of-$M$ down-sampler and a factor-of-$L$ up-sampler is interchangeable with no change in the input-output relation: $y_1[n] = y_2[n]$ if and only if $M$ and $L$ are relatively prime

$\text{x}[n] \downarrow M \rightarrow H(z) \rightarrow y_1[n] \\
\equiv x[n] \rightarrow H(z^M) \downarrow M \rightarrow y_1[n]$

**Cascade equivalence #1**

$\text{x}[n] \uparrow L \rightarrow H(z^L) \rightarrow y_2[n] \\
\equiv x[n] \rightarrow H(z) \uparrow L \rightarrow y_2[n]$

**Exercise - 9:** Validate all

---

**Non-Integer Sampling Rate Conversion**

**Example 1:** CD Audio $44.1$ to mp3 $48$kHz in 3 stages: $L/M = 48/44.1 = 160/147 = (4/3)(8/7)(5/7)$

**Example 2:**
Audio input $x(n)$ (2.5 KHz BW) is sampled at the rate of $6,000$ Hz, Audio output $y(m)$ is operated at the rate of $9,000$ Hz
Algorithm Mapping...

Algorithm Mapping...

Algorithm Mapping...

Algorithm Mapping...
Algorithm Mapping...

Special Topics on Filtering ... Multirate Signal Processing...
Polyphase Decomposition... (*)

Matrix form

\[
X(z) = \begin{bmatrix}
1 & z^{-1} & \ldots & z^{-(M-1)}
\end{bmatrix}
\begin{bmatrix}
X_0(z^M) \\
X_1(z^M) \\
\vdots \\
X_{M-1}(z^M)
\end{bmatrix}
\]

Same formulation can be established for filters, especially FIR filters

For IIRs the decomposition is not as straightforward as for FIRs

So most of the applications are based on FIRs

Example of FIR polyphase decomposition:

\[
H(z) = \sum_{n=0}^{8} h[n] z^{-n}
\]

\[
H(z) = E_0(z^4) + z^{-1} E_1(z^4) + z^{-2} E_2(z^4) + z^{-3} E_3(z^4)
\]

For IIRs the decomposition is not as straightforward as for FIRs

So most of the applications are based on FIRs

Example of IIR polyphase decomposition:

\[
H(z) = \sum_{n=0}^{15} h[n] z^{-n}
\]

\[
H(z) = E_0(z^4) + z^{-1} E_1(z^4) + z^{-2} E_2(z^4) + z^{-3} E_3(z^4)
\]

For IIRs the decomposition is not as straightforward as for FIRs

So most of the applications are based on FIRs

Example of IIR polyphase decomposition:

\[
H(z) = \sum_{n=0}^{15} h[n] z^{-n}
\]

\[
H(z) = E_0(z^4) + z^{-1} E_1(z^4) + z^{-2} E_2(z^4) + z^{-3} E_3(z^4)
\]

For IIRs the decomposition is not as straightforward as for FIRs

So most of the applications are based on FIRs

Type I Realization and Transposed Type I:

based on direct M-branch polyphase decomposition of \( H(z) \)

\[
H(z) = \sum_{k=0}^{M-1} z^{-k} E_k(z^M)
\]

And substituting in the direct form decomposition:

Which leads to another realization called Type II ...

\[
R_\ell(z^M) = E_{M-1-\ell}(z^M), \quad 0 \leq \ell \leq M-1
\]

And substituting in the direct form decomposition:

Which leads to another realization called Type II ...

Algorithm Mapping...

Special Topics on Filtering ... Multirate Signal Processing...
Polyphase Decomposition... (*)

FIR Polyphase structures

Type I Realization and Transposed Type I:

based on direct M-branch polyphase decomposition of \( H(z) \)

\[
H(z) = \sum_{k=0}^{M-1} z^{-k} E_k(z^M)
\]

And substituting in the direct form decomposition:

Which leads to another realization called Type II ...

\[
R_\ell(z^M) = E_{M-1-\ell}(z^M), \quad 0 \leq \ell \leq M-1
\]

And substituting in the direct form decomposition:

Which leads to another realization called Type II ...

Algorithm Mapping...

Special Topics on Filtering ... Multirate Signal Processing...
Polyphase Decomposition... (*)

FIR Polyphase structures

Type I Realization and Transposed Type I:

based on direct M-branch polyphase decomposition of \( H(z) \)

\[
H(z) = \sum_{k=0}^{M-1} z^{-k} E_k(z^M)
\]

And substituting in the direct form decomposition:

Which leads to another realization called Type II ...

\[
R_\ell(z^M) = E_{M-1-\ell}(z^M), \quad 0 \leq \ell \leq M-1
\]
**Algorithm Mapping...**

**Special Topics on Filtering ... Multirate Signal Processing...**

**Polyphase Decomposition...** FIR Polyphase structures...

Applications in Computationally efficient Decimation

Using Polyphase Type I and Cascade equivalence #I

\[ x[n] \xrightarrow{H(z)} v[n] \xrightarrow{M} y[n] \]

Decimator structure based on Type I polyphase decomposition

---

**Algorithm Mapping...**

**Special Topics on Filtering ... Multirate Signal Processing...**

**Polyphase Decomposition...** FIR Polyphase structures... (*)

Applications in Computationally efficient Interpolation

Interpolator based on Type I polyphase decomposition

---

**Algorithm Mapping...**

**Special Topics on Filtering ... Multirate Signal Processing...**

**Polyphase Decomposition...** FIR Polyphase structures...

Applications in Computationally efficient Decimation...

For direct implementation:
- All computations need to be completed in one sampling period, and for the following \( M-1 \) sampling periods the arithmetic units remain idle.
- The modified decimator structure also requires \( N \) multiplications and \( N-1 \) additions per output sample being computed.

For Polyphase:
- However, here the arithmetic units are operative at all instants of the output sampling period which is \( M \) times that of the input sampling period.
- Similar savings are also obtained in the case of the interpolator structure developed using the polyphase decomposition.
- More efficient interpolator and decimator structures can be realized by exploiting the symmetry of filter coefficients in the case of linear-phase filters \( H(z) \).

---

**Algorithm Mapping...**

**Special Topics on Filtering ... Multirate Signal Processing...**

**Polyphase Decomposition...** FIR Polyphase structures...

Sampling Rate Conversion Applications → Multi-stage Decimation

The multistage approach for downsampling rate conversion can be used to dramatically reduce the anti-aliasing filter length.

Example: 2-stage example → total decimation rate \( M = M_1 \times M_2 \)

\[ x(n) \xrightarrow{\text{Anti-aliasing filter } H_1(z)} \xrightarrow{M_1} y(n) \]

\[ y(n) \xrightarrow{\text{Anti-aliasing filter } H_2(z)} \xrightarrow{M_2} y_M(n) \]

Rule:
- First anti aliasing filter \( \rightarrow \) from \( f_s / 2M \) to \( f_s / M_1 - f_s / 2M \)
- Second anti aliasing filter \( \rightarrow \) from \( f_s / 2M \) to \( f_s / 2M_1 \)

final Nyquist limit \( f_s / 2M \) (where useful information bandwidth should stop!)
Algorithm Mapping...

**Special Topics on Filtering**  
*Multirate Signal Processing*

**Sampling Rate Conversion Applications**  
*Multi-stage Decimation*

---

**CIC Filters**

One of Multiplier-less Filter Structures (Hogenauer 1981)

Application in fast HW: data convertors, narrow band filters, ...

**Building blocks:**

1) Integrators: \(y[n] = y[n-1] + x[n]\)  \(H_1(z) = \frac{1}{1-z^{-1}}\)

2) Comb filters: \(y[n] = x[n] - x[n - RM]\)  \(H_C(z) = 1 - z^{-RM}\)

---

**Cascaded Integrator Comb Filters**

- **Parameters:** \(M, R\)
  - \(RM = 1\)  
  - Integrator inverse  
  - Highpass 20db/decade

---

**Interpolating CIC** (cascade equivalence II)

- **Frequency response:**
  - For a CIC of \(N\) integrator and \(N\) comb running at \(f_s\)
  - \(H(z) = H^N_1(z)H^N_C(z) = \left(\frac{1-z^{-RM}}{1-z^{-1}}\right)^N = \sum_{k=0}^{RM-1} z^{-k}\)

1) In spite of containing integrators (IIR), the filter is an FIR  
2) Stable and linear phase (Symmetric)  
3) Cascade of \(N\) FIRs, each with rectangular impulse response  
4) All the coefficients = 1  
   - no multiplication
Algorithm Mapping...

**Special Topics on Filtering... CIC Filters ➔ Cascaded Integrator Comb Filters...**

**Frequency response:** ...

\[ H(f) = \left| \frac{\sin \pi Mf}{\sin \frac{\pi f}{R}} \right|^N \]

Large 'R', \( \sin x \approx x \)

\[ |H(f)| \approx \left| RM\frac{\sin \pi Mf}{\pi Mf} \right|^N \text{ for } 0 < f < \frac{1}{M} \]

Only 1 dB MS difference for \( R M \geq 10 \), \( 1 \leq N \leq 7 \)

- sinc function: Nulls at multiples of \( f = 1/M \) (1/\( MR \) in \( fs \) scale) stopband: \( f - f_c \) to \( f + f_c \)

\( R, M \), and \( N \) can be chosen from the tables and design rules in previous page (see Alteraan455.pdf examples)

**Exercise:** How?

\[
abs(sin(x)/x)^1 \\
abs(sin(x)/x)^2 \\
abs(sin(x)/x)^3
\]

(see Altera an455.pdf examples on droop filter design)

Algorithm Mapping...

**Special Topics on Filtering... CIC Filters ➔ Cascaded Integrator Comb Filters...**

Passband attenuation is a function \( N \), number of stages

- Increasing \( N \) improves the aliasing removal but increases the passband 'droop'

**DC gain** is a function of rate change

- In MATLAB:
  ```matlab
  syms x
  ezplot(abs(sin(x)/x)^1))
  hold
  ezplot(abs(sin(x)/x)^2))
  ezplot(abs(sin(x)/x)^3))
  ```

**Algorithm Mapping...**

**Special Topics on Filtering... CIC Filters ➔ Cascaded Integrator Comb Filters...**

**Frequency response:** ...

**Passband attenuation** is a function \( N \), number of stages

- Increasing \( N \) improves the aliasing removal but increases the passband 'droop'

**DC gain** is a function of rate change

- In MATLAB:
  ```matlab
  syms x
  ezplot(abs(sin(x)/x)^1))
  hold
  ezplot(abs(sin(x)/x)^2))
  ezplot(abs(sin(x)/x)^3))
  ```

(see Altera an455.pdf examples on droop filter design)

Algorithm Mapping...

**Special Topics on Filtering... CIC Filters ➔ Cascaded Integrator Comb Filters...**

**Frequency Response...**

**Aliasing/Imaging Attenuation for Large Rate Change Factors**

<table>
<thead>
<tr>
<th>Differential Gain (dB)</th>
<th>Aliasing/Imaging Attenuation at ( f_c ) (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>95</td>
</tr>
<tr>
<td>4</td>
<td>120</td>
</tr>
<tr>
<td>5</td>
<td>145</td>
</tr>
</tbody>
</table>

The same number of bits are needed for all the registers of combs and integrators.

**Algorithm Mapping...**

**Special Topics on Filtering... CIC Filters ➔ Cascaded Integrator Comb Filters...**

**Bit Allocation**


Assuming sign-extended 2's complement arithmetic, with rounding or truncation

**For CIC decimator**

Filter Gain from input to output is: \( G = (RM)^N \) Bin = #bits at input

High register width for large \( R \)

\[ B_{out} = \left[ N \log_2 RM + B_{in} \right] \]

**For CIC interpolator**

Filter Gain at the \( i \)-th stage is

\[ G_i = \begin{cases} \frac{2^N}{2^{N-i}(RM)^{N-i}} & i = 1, 2, \ldots, N \\ \frac{2^{N-i}}{2^{N-i-1}(RM)^{N-i-1}} & i = N + 1, \ldots, 2N \end{cases} \]

Bit width of the \( i \)-th register will be

\[ W_i = \left[ B_{in} + \log_2 G_i \right] \]

\[ W_N = B_{in} + N - 1 \]
Algorithm Mapping...

**Special Topics on Filtering... CIC Filters → Cascaded Integrator Comb Filters...**

**Implementation Issues**

- Because of the passband droop (safe passband is narrow) usually other FIRs at low sampling rate are used (droop filter):
  1. to equalize the passband
  2. to complete the task when less severe droop is chosen

- For $M=1$ interpolator, rounding/truncation cannot be used. Errors grow and system is unstable.

- In R-programmable CICs bit width must be chosen to handle largest and smallest rate changes:
  1. The largest rate change dictates the bit width of the stages
  2. The smallest rate change dictates the # bits to be kept at the final result

A shifter can be included at the final stage to select the proper bits: $B_{out}$ for the decimator and $W_{m}$ for the interpolator can be used to set the shift

---

**Algorithm Mapping...**

**Special Topics on Filtering... Half band Filters**

Transfer function of a half-band filter: $H(z) = \alpha + z^{-1}E_{r}(z^{2})$

FIR impulse response satisfies in: $h[2n] = \begin{cases} \alpha, & n = 0 \\ 0, & \text{otherwise} \end{cases}$ Advantage: Half of the coefficients are zero...

For $\alpha = 0.5 \rightarrow H(z) + H(-z) = 1$

If $H(z)$ coefficients are real $\rightarrow H(-e^{j\omega}) = H(e^{-j(\pi - \omega)})$

$\Rightarrow H(e^{j\omega}) + H(e^{j(\pi - \omega)}) = 1 \Leftrightarrow$ Symmetric with respect to the $\pi/2 (f_{s}/4)$

Easiest way to design $\rightarrow$ Parks- McClellan (PM)

(If equiripple property is not the issue windowing method can be used)

$\rightarrow$ With PM $h(n)$ relation is approximately satisfied

Zero coeffs are treated as unknown

The method is not optimum

Vaidyanathan Paper $\rightarrow$ "A trick for the design of FIR halfband filters"
Algorithm Mapping...

**Special Topics on Filtering... Multiplier-less Filters**

Canonical Signed Digits (CSD) number representation for coefficients

For dedicated applications the flexibility of multiplier is not necessary...

\[ x = 2^m \sum_{i=0}^{M} s_i 2^{-i} \quad s_i \in \{-1,0,1\} \quad p_i \in \{0,1,\ldots,M\} \quad i = 0/1 \quad \text{max M+1 ternary digits} \]

There are several sign digit representation...

\[ 15_{10} = 16_{10} - 1_{10} = 10001_{SD} \]

\[ 15_{10} = 16_{10} - 2_{10} + 1_{10} = 10011_{SD} \]

\[ 15_{10} = 16_{10} - 4_{10} + 3_{10} = 10111_{SD} \ldots \]

Minimal representation \( \Rightarrow \) minimum number of non-zero digits (min \( L \))

**CSD:** No two non-zero digits are adjacent \( \Rightarrow \) CSD of a given number is unique

**Classical Binary to CSD Conversion:** Starting from the LSB, replace ‘all 1’s sequences with 10...01

\[ 011_2 \to 101_1 \; , \; 0111_2 \to 1001_1 \; , \ldots \]

Example:

\[ 27_{10} = 11011_{SD} = 100101_{CSD} \]

The first substitution does not reduce the complexity, it produces a length-three strike, and the complexity reduces from three additions to two subtractions

Algorithm Mapping...

**Special Topics on Filtering... CSD**

Distribution of CSD numbers

2’s complement numbers are uniformly distributed in any interval
CSD numbers are very non-uniform

Example: 6 and 8-digits CSD with 2 nonzero digits
- Gaps of with 1/8 in both cases
- Not improved if number of nonzero digits are fixed
- Increasing number of nonzero digits is not a solution

In filter design with CSD coefficients

\( \text{Scaling is of great importance} \)

Filter Coefficient Optimization Methods

1) **MILP:** Mixed Integer Linear Programming
   - Slow method...


Algorithm Mapping...

**Special Topics on Filtering... CSD**

the classical CSD coding does not always produce the optimal CSD coding in terms of hardware complexity

Additions are substituted by subtractions, when there should be no such substitution...

\( \Rightarrow 101 \rightarrow 011 \)

**HW Optimal Binary to CSD Conversion:**

1) Starting from the LSB, replace ‘all 1’s sequences to 10...01
2) Also convert 1011 to 1101
3) Starting from the MSB convert 101 to 111

Sometimes it is cheaper to factor the multiplier into smaller numbers...

CSD Numbers can be used for fractional numbers \( \Rightarrow \) the same way used for 2’s complement binary

Algorithm Mapping...

**Special Topics on Filtering... CSD**

Coefficient Optimization Methods...

2) **Local Search**
   - Faster but with lower performance

Two stage local search:

1) Scale factor search
2) Bivariate local search in the neighborhood of scaled rounded CSD coefficients

Rules:

A) Rule of thumb: one nonzero digit in CSD coding for each 20 dB stopband attenuation

B) Dealing with non-uniform distribution: One additional nonzero digit in CSD coding is allocated to the impulse response samples more than 1/2

Since impulse responses have \( \sin(x)/x \) envelope, not so many coefficients exceed 1/2

So no much increase in processing...
**Algorithm Mapping...**

**Special Topics on Filtering... CSD...**

**Coefficient Optimization Methods... Local Search Method...**

1) **Scaling Strategy**
   a) Impulse response is scaled first so that the largest coefficients has a value of unity
   b) L and M are preselected, for instance based on the bit width and the rules A & B
   c) A table of all possible (M,L) CSD coefficients in [0,0.5) is created
   d) A table of all possible (M,L+1) CSDs in [0.5, 1] is created

The process of coefficients quantization is highly nonlinear

   \[ \Rightarrow \text{No way but the brute force search} \]

   e) For each scale factor in \([1/2, 1]\) the filter coefficients are rounded to the closest CSD in the tables and the peak weighted ripple is computed:

   \[ \min \delta_{\text{chosen}} \]

   \[ W: \text{ripple weighting factor}, b \text{ is average passband gain} \]

   \[ \delta = \max{[\delta_p/W, \delta_s]}/b \]

   \[ \delta_p, \delta_s: \text{are stopband and passband ripples} \]

   \[ \Rightarrow \text{Only one octave of scale factors is searched } \Rightarrow x2 \text{ does not change the filter shape} \]

   \[ \Rightarrow \text{For } \delta, \text{ Zero-padding the coefficient to closest power of } 2 \text{ to } 8x \text{ filter length and FFT} \]

   \[ \Rightarrow \text{Choosing the steps of scale factor search, trade of between speed and accuracy} \]

2) **Local Search Strategy**
   a) All possible pairs of coefficients are varied by +/- one quantization step size
   b) \( \delta \) is computed for each case and the set with min \( \delta \) is chosen
   c) Repeat for the process until no more improvements...

   \[ \Rightarrow \text{For an FIR filter of } K \text{ coefficient, } 2^K \text{ sets of coefficients must be tried} \]

**Example:**

25 -tap Linear phase FIR
40 dB stopband attenuation \( \Rightarrow L=2(+1) \)

\[ W = 1, \delta_p, \delta_s = 0.005 (-46dB) \]
9-digit CSD \( \Rightarrow M=8 \)

\( L=2 \) except for the largest coefficient
13 CSD coefficients, 11 adders needed

\[ \Rightarrow \text{Results in 38 dB attenuation} \]

(MILP gives 41 db)