Lecture 6

FIR Filter Design Using Transformations

2-D Difference Equations

2-D FIR Design using Transformations

- **Idea**: Turn a 1-D filter, e.g. a lowpass, into a 2-D filter (non-separable).
- Approach is simple, powerful, and elegant.
- Due to McClellan (1969).
- We use a transformation function $F(w_1,w_2)$ to turn a 1-D zero-phase prototype filter $H(w)$ into a 2-D filter $H(w_1,w_2)$.

Derivation of the Method

- First, we need to express the frequency response of the prototype zero-phase filter in the appropriate form.
- Let the length of $h[n]$ be $2N+1$. Since it is a zero-phase filter $h[n] = h[-n]$.
- $$\begin{aligned} H(e^{j\omega}) &= \sum_{n=-N}^{N} h[n]e^{-j\omega n} \\ &= h[0] + \sum_{n=1}^{N} h[n]e^{-j\omega n} + \sum_{n=-N}^{-1} h[n]e^{-j\omega n} \\ &= h[0] + \sum_{n=1}^{N} h[n](e^{-j\omega n} + e^{j\omega n}) \\ &= \sum_{n=1}^{N} a[n]\cos\omega n \end{aligned}$$

$$\sum_{n=1}^{N} h[-n]e^{j\omega n}$$

Chebyshev Polynomials

- $\cos wn$ can be expressed as a polynomial of degree n in the variable $\cos w$.
- Examples:
  - $\cos 2w = 2\cos^2 w - 1$
  - $\cos 3w = 4\cos^3 w - 3\cos w$
  - $\cos nw = 2\cos w \cos(n-1)w - \cos(n-2)w.$
- In general $\cos n\omega = T_n(\cos w)$
  - $T_n(x)$ – Chebyshev polynomial of degree n
    - $T_0(x) = 1$
    - $T_1(x) = x$
    - $T_n(x) = 2x\, T_{n-1}(x) - T_{n-2}(x)$

Transforming the Prototype

- Using Chebyshev polynomials, the prototype frequency response can be written as

$$H(\omega) = \sum_{n=0}^{N} a[n] T_n(\cos\omega)$$

- The designed filter is

$$H(\omega_1,\omega_2) = \sum_{n=0}^{N} a[n] T_n(F(\omega_1,\omega_2))$$

The Transformation Function

$$H(\omega_1,\omega_2) = \sum_{n=0}^{N} a[n] T_n(F(\omega_1,\omega_2))$$

- *F* should be chosen to be the frequency response of a zero-phase filter with $-1<F<1$.
  - If *F* is real, *H* will be real
  - If *F* is $(2Q+1)\text{x}(2Q+1)$, *H* will be $(2NQ+1)\text{x}(2NQ+1)$.
  - *H* will be constant on any contour in the (w_1,w_2)-plane on which $F(w_1,w_2)$ is constant.
  - The range of *H* will be the same as the range of the prototype.

How it works

$$H(\omega_1,\omega_2) = \sum_{n=0}^{N} a[n] T_n(F(\omega_1,\omega_2))$$

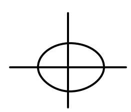
- Consider the transformation function

$$\cos w \rightarrow F(w_1, w_2) = A + B\cos w_1 + C\cos w_2.$$

- For any set of points

$\{(w_1,w_2)\text{: } F(w_1,w_2) = \text{const.}\}$,

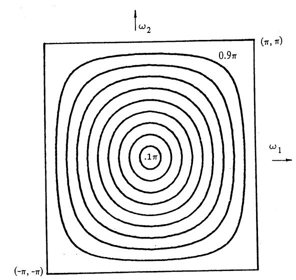
$F(w_1,w_2)$ = (different) constant



- Shape of contours depends on A,B,C
- Value of H on the contour depends on the prototype,

$$H(w_1,w_2) = H_p(\cos^{-1}(F(w_1,w_2)))$$

Example



$$-1 \le F \le 1 \longrightarrow 0 \le \omega \le \pi$$

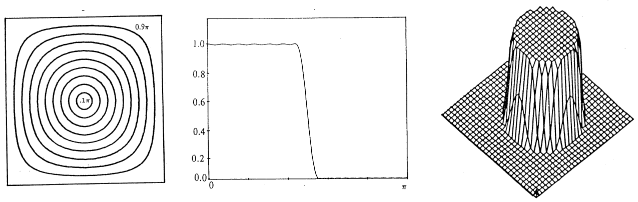
- First order transformation with "circular" contours

$$F(\omega_1,\omega_2) = -\frac{1}{2} + \frac{1}{2}\cos\omega_1 + \frac{1}{2}\cos\omega_2 + \frac{1}{2}\cos\omega_1\cos\omega_2$$

- Each contour is associated with a value of *w* through

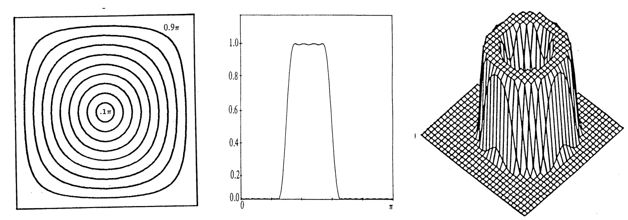
$$\cos\omega \longrightarrow F(\omega_1,\omega_2)$$

Example with LP Prototype



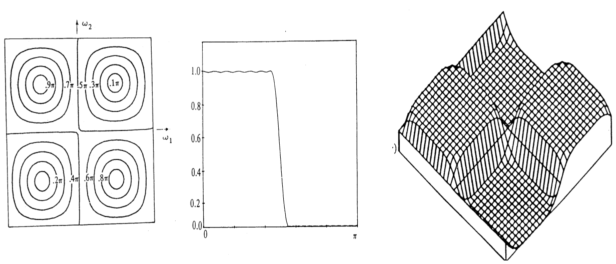
F (3x3) Hp (63 pts) H (63x63)

Example with BP Prototype



F (3x3) Hp (63 pts) H (63x63)

Another Example



$$F(\omega_1, \omega_2) = \sin\omega_1 \sin\omega_2$$

$$\cos\omega \longrightarrow F(\omega_1, \omega_2)$$

Issues

- Multidimensional IIR filters are usually avoided in image processing
  - Testing and guaranteeing stability difficult.
  - Zero-phase filtering requires a two-pass implementation.
  - Infinite support of the output problematical.
  - Boundary condition issues.
  - Image processing demands are modest.
- But…
  - They offer the promise of more filtering for a given level of complexity.
  - Separable IIR filters can be managed.
  - Some modeling applications require them.

2-D Difference Equations

- The concept of a difference equation can be extended to two (or more) dimensions.

$$\sum_{k_1}\sum_{k_2} b[k_1,k_2]y[n_1-k_1,n_2-k_2] = \sum_{r_1}\sum_{r_2} a[r_1,r_2]x[n_1-r_1,n_2-r_2]$$

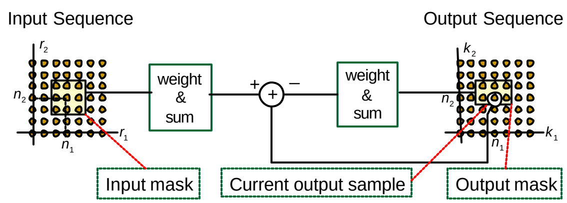
  - The sums have finite limits.
  - The size of b is related to the filter order.
  - The system is LTI if it is at initial rest.
  - The state space associated with a 2-D difference equation is of infinite dimensionality.

Recursive Computability

$$y[n_1,n_2] = \sum_{r_1,r_2} a[r_1,r_2]x[n_1-r_1,n_2-r_2] - \sum_{\substack{k_1,k_2\\ [k_1,k_2]\neq[0,0]}} b[k_1,k_2]y[n_1-k_1,n_2-k_2]$$

- To use this as an algorithm, the samples of y on the right need to be computed before the one on the left.
- If such an ordering exists, the system is **recursively computable**.
  - If there is one such ordering, there is an infinite number of them.

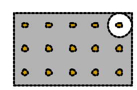
Input and Output Masks

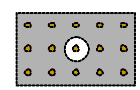


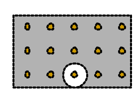
- The **input and output masks** isolate the samples of the input and output sequences that are used to compute the current output sample.
- Their shape is the shape of the support of a and b (rotated through 180°.

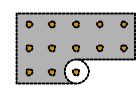
Output Masks

- Whether or not a difference equation is recursively computable depends upon the shape of the output mask.


Recursively computable

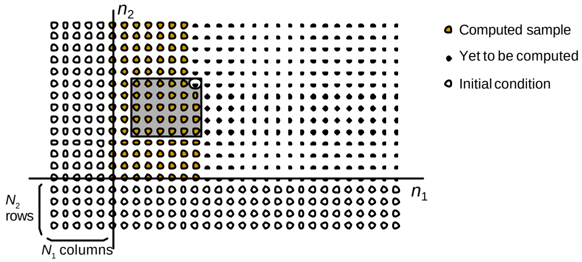

Not recursively computable


Not recursively computable


Recursively computable

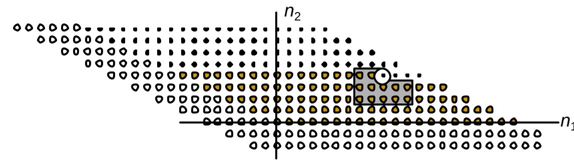
Quarter-Plane Masks

- Quarter-plane filters have masks with the “hole” in a corner (four types).



Nonsymmetric Half-Plane Masks

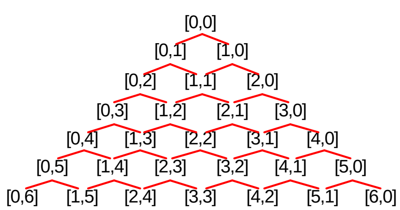
- Nonsymmetric half-plane masks have their “hole” at a jag in the side of the mask (eight types).



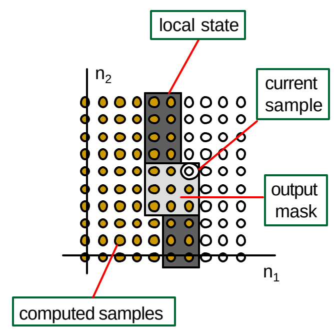
- Most general form of recursive system.
- To get a rectangular shaped output requires computing extraneous output samples.

Ordering the Computations

- There are an infinite number of possible orderings for recursively computable filters.
- Not all of them are equivalent in terms of storage, parallelism.

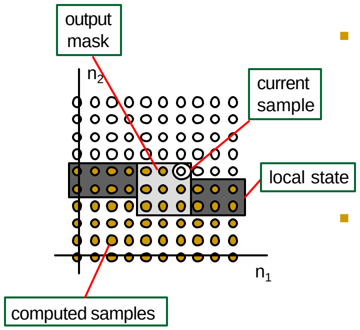


Column-wise Implementation



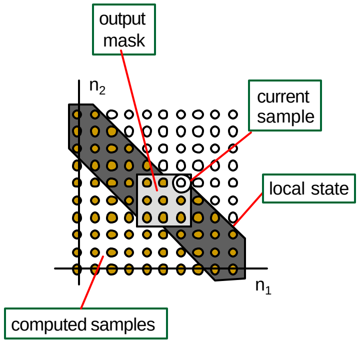
- For an $L_1 x L_2$ point output mask, the column-wise implementation requires approximately L_1-1 **columns** of storage.
- $L_1 L_2$ multiplications per output sample.

Row-wise Implementation



- For an L_1xL_2 point output mask, the row-wise implementation requires approximately L_2-1 **rows** of storage.
- L_1L_2 multiplications per output sample.

Diagonal Implementation



- For an L_1xL_2 point output mask, the row-wise implementation requires approximately L_1+L_2-1 **diagonals** of storage.
- L_1L_2 multiplications per output sample.
- Maximally parallel
  - All elements on the diagonal can be computed simultaneously.