Lecture 12

Generalized Decision Feedback

May 14, 2024

JOHN M. CIOFFI

Hitachi Professor Emeritus of Engineering

Instructor EE379B – Spring 2024

Announcements & Agenda

- Announcements
  - PS6 due May 22, HWH @ web site - due Wed 5/22.
  - PS5 due tomorrow
  - Return to single-user for 2 lectures (L12, L13)

- Problem Set 6 = PS6 (due **May 22**)
  1. 5.5 Singularity Elimination
  2. 5.7 Matrix Bias review
  3. 5.8 Input Rxx variation single user
  4. 5.9 GDFE canonical behavior
  5. 5.11 Cyclic antennas

- Agenda
  - Finish IC capacity region from L11
  - Channel Singularity
  - Input Singularity
  - MMSE GDFE
  - Specific Forms (VC and Triangular)

GDFE Foundation				
12	5/14	GDFE Basics	5.1-3	6/5
13	5/16	GDFE Input Optimization and Forms	5.3	-/-

Finish IC regions

Section 5.1.1.1

Symmetric 2x2 IC

$$\begin{bmatrix} y_2 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & \alpha \\ \alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} + \begin{bmatrix} n_2 \\ n_1 \end{bmatrix}$$

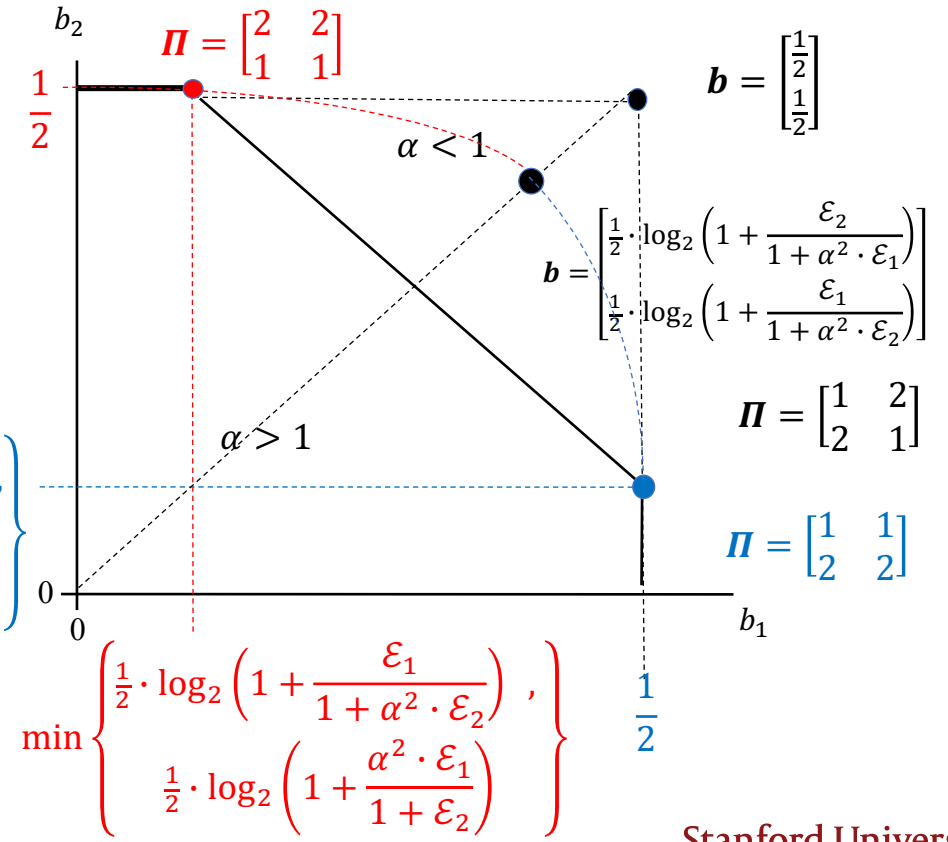
$R_{nn} = I$ and $\boldsymbol{\mathcal{E}} \preccurlyeq \mathbf{1}$

- When $\alpha \to 0$, there is no crosstalk and so $\mathcal{C}_{IC}(\boldsymbol{b})$ is a square.
- When $\alpha > 1$, $\mathcal{C}_{IC}(\boldsymbol{b})$ is a pentagon.
- When $0 < \alpha < 1$, $\mathcal{C}_{IC}(\boldsymbol{b})$ is intermediate

$$\min\left\{ \frac{1}{2}\cdot\log_2\left(1+\frac{\mathcal{E}_2}{1+\alpha^2\cdot\mathcal{E}_1}\right) , \; \frac{1}{2}\cdot\log_2\left(1+\frac{\alpha^2\cdot\mathcal{E}_2}{1+\mathcal{E}_1}\right) \right\}$$

These two are same for $\alpha = 1$ and equal energy, and determine Imin vector possibilities

Achievable Region when $\mathcal{E}_1 = \mathcal{E}_2 = 1$



Vector Gaussian IC Example, $L_{x,u} \equiv 1$; $L_{y,u} \equiv 2$

- 2 users and H is **4 x 2:** $\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_2 \\ \boldsymbol{y}_1 \end{bmatrix} = \begin{bmatrix} H_2 \\ H_1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} + \begin{bmatrix} \boldsymbol{n}_2 \\ \boldsymbol{n}_1 \end{bmatrix}.$ $\quad H_2 = [\boldsymbol{h}_{22} \quad \boldsymbol{h}_{21}] = \begin{bmatrix} .9 & .3 \\ .3 & .8 \end{bmatrix}$

$R_{\boldsymbol{nn}} = .01 \cdot I$ $\quad H_1 = [\boldsymbol{h}_{12} \quad \boldsymbol{h}_{11}] = \begin{bmatrix} .8 & .7 \\ .6 & .5 \end{bmatrix}$

```
>> H2 = [9     3
        3     8 ];
>> Rb2inv=H2'*H2+diag([1 1]);
>> Gbar2=chol(Rb2inv);
>> G2=inv(diag(diag(Gbar2)))*Gbar2;
>> S02=diag(diag(Gbar2))*diag(diag(Gbar2));
>>  0.5*log2(diag(S02)) =
b2 =  3.2539
b1 =  2.7526
>>  H1 = [8     7
         6     5 ];
>> Rb1inv=H1'*H1+diag([1 1]);
>> Gbar1=chol(Rb1inv);
>> S01=diag(diag(Gbar1))*diag(diag(Gbar1));
>> 0.5*log2(diag(S01)) =
b2 =  3.3291
b1 =  0.4128
```

Two MAC sets

Coincides with $\mathbb{I}_{min} = \begin{bmatrix} 3.25 \\ .413 \end{bmatrix}$

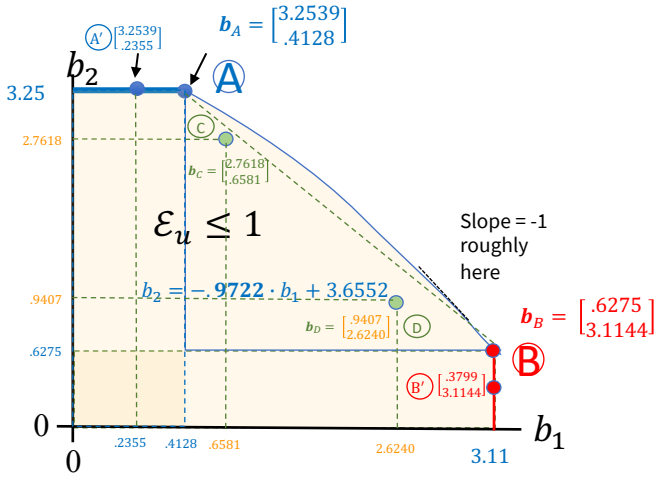
```
>> J2=hankel([0 1]);
>> Rb2inv=J2*H2'*H2*J2+diag([1 1]);
   74    51
   51    91
>> Gbar2=chol(Rb2inv);
>> S02=diag(diag(Gbar2))*diag(diag(Gbar2));
>> 0.5*log2(diag(S02)) =
b1 =  3.1047
b2 =  2.9018

>> Rb1inv=J2*H1'*H1*J2+diag([1 1]);
>> Gbar1=chol(Rb1inv);
>> S01=diag(diag(Gbar1))*diag(diag(Gbar1));
>> 0.5*log2(diag(S01)) =
b1 =  3.1144
b2 =  0.6275
```

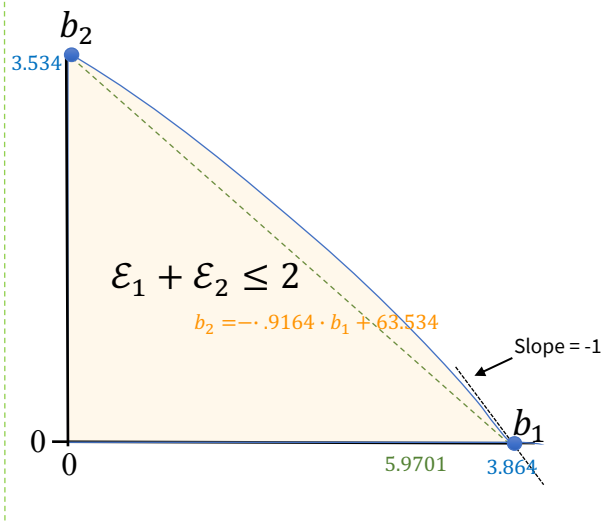
Two MAC Sets with Reversed order

Coincides with $\mathbb{I}_{min} = \begin{bmatrix} 3.11 \\ .628 \end{bmatrix}$

Try various energy points
With the same orders.

4x2 IC Example continued





- Vary energies near max points to see if local points above or below dimension-sharing, so curved boundary or flat
- Since the line has slope magnitude less than 1, then the curvature is above this line with max rate sum at magnitude 1.

PS 5.4 (2.31) – IC channel has mix, one 2x2 user and one scalar user

Chris Adelico-Ferrarin's MU_IC.m

```
function [b, GU, WU, S0, MSWMFU] = mu_ic(H, A, Lxu, Lyu, cb)
-------------------------------------------------------------------------
 Per-tonal (temporal dimension) multiuser interference channel
receiver and per-user bits - Chris Adelico Ferrarin - 2023

 Inputs: H, A, Lxy, Lyu, cb
 Outputs: b, GU, WU, S0, MSWMFU

 Definitions:

 H: noise-whitened channel matrix [HUU ⋯ HU1] sum-Lyu x sum-Lxu
                                  |⋮  ⋱  ⋮|
                                  |H2U ⋯ H21|
                                  [H1U ⋯ H11]
 A: Block Diag sq-root sum-Lxu x sum-Lxu discrete modulators,
   blkdiag([AU ... A1]);  The Au entries derive from each IC user's
   Lxu x Lxu input autocorrelation matrix, where the trace of each such
   autocorrelation matrix is user u's energy/symbol. This is per-tone.

 Lxu: # of input dimensions for each user U ... 1 in 1 x U row vector
 Lyu: # of output dimensions for each user U ... 1 in 1 x U row vector
 cb: = 1 if complex baseband or 2 if real baseband channel

 GU: unbiased feedback matrix sum-Lxu x sum-Lxu x U with matrices indexed
    from user U (e.g. GU(:,:,2) gives GU for user U-1). with
    matrices indexed from user U (e.g. WU(:,:,2) gives WU for user U-1).
 S0: sub-channel channel gains sum-Lxu x sum-Lxu x U with matrices indexed
    from user U (e.g. GU(:,:,2) gives S0 for user U-1).
 MSWMFU: unbiased mean-squared whitened matched filter, sum-Lxu x Ly x U
       with matrices indexed from user U (e.g. MSWMFU(:,:,2) gives

 WU: unbiased feedforward linear equalizer sum-Lxu x sum-Lxu x U MSWMFU for
user U-1).
 b - user u's bits/symbol 1 x U
   the user should recompute b if there is a cyclic prefix
```

Does not Find $\mathcal{I}_{min}$

- IC needs Lxu and Lyu
- A is like mu_mac
- Per tone
- Arrange input Hu matrices according to order $\boldsymbol{\Pi}$

```
H2 = [0.9 0.3; 0.3 0.8];    % From L12:6
H1 = [0.8 0.7; 0.6 0.5];
sigma2 = 0.01;
Ht = [H2; H1] / sqrt(sigma2);  % this is 4x2 matrix (2 outputs/input)
A = [1 0; 0 1];
Lxu = [1 1];
Lyu = [2 2];
cb = 2;
>> [b_A, GU_A, WU_A, S0_A, MSWMFU_A] = mu_ic(Ht, A, Lxu, Lyu, cb);
USER 2                          USER 1
>> b_A % =
      3.2539                    0.4128
GU_A(:,:,1) =                   GU_A(:,:,2) =
   1.0000    0.5667              1.0000    0.8600
        0    1.0000                   0    1.0000
WU_A(:,:,1) =                   WU_A(:,:,2) =
    0.0111         0              0.0100         0
   -0.0126    0.0225             -1.1026    1.2949
S0_A(:,:,1) =                   S0_A(:,:,2) =
   91.0000         0             101.0000         0
         0   45.4176                   0    1.7723
MSWMFU_A(:,:,1) =               MSWMFU_A(:,:,2) =
    0.1000    0.0333              0.0800    0.0600
   -0.0460    0.1423              0.2436   -0.1410
```

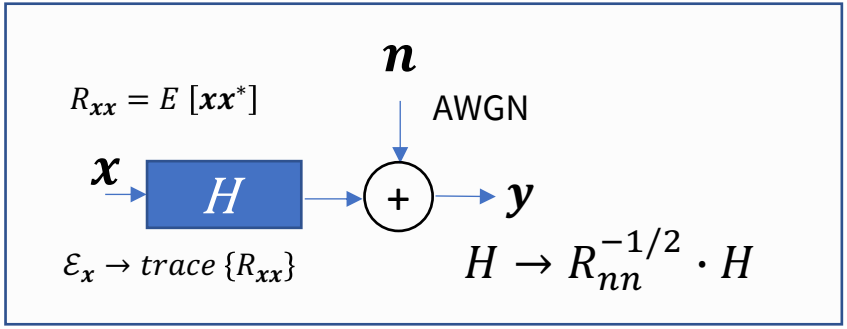
Order implied by H's column index

Channel Singularity

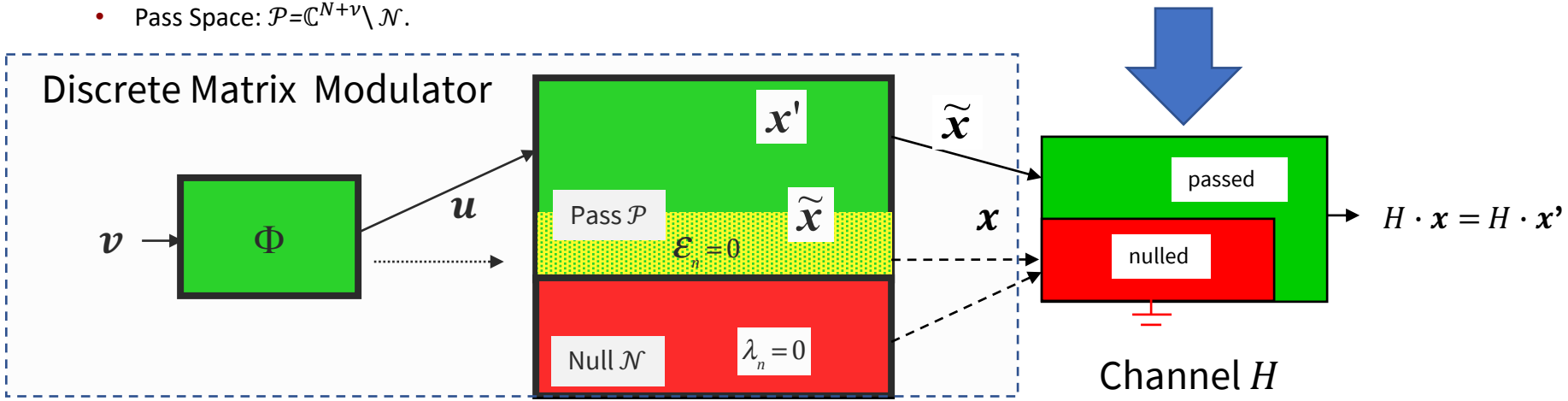
Section 5.1.1.1

Single-User Matrix AWGN Channel Returns!

- Treat as single-user here – dimension index n.
  - Whiten the noise with equivalent channel R_{nn}.
  - Finite-length block (symbol) of input samples has
    - dimensions that are subsymbols.



- All energy should go in channel pass space.
  - Xmit energy in channel null space $\mathcal{N}=\{x \mid H \cdot x = 0\}$ is "wasted."
  - Pass Space: $\mathcal{P}=\mathbb{C}^{N+\nu} \setminus \mathcal{N}$.



- Any AEP is over a sequence of such symbols (i.e, a codeword), so really single-user good codes are applied over codewords of symbols,
  - *Each containing N subsymbols.*

Eliminate Channel null space (Sec 5.1.1.1)

- A discrete modulator (temp matrix C) :
  - may insert energy into channel's null space &
  - sums over dimensions.
  - C's columns don't have to be "orthonormal."
- Find equivalent discrete modulator $\tilde{C}$
  - that corresponds only to the pass space,
  - & also the components of $\boldsymbol{x}$ on these new vectors, $\tilde{\boldsymbol{u}}$.

$$\boldsymbol{x} = \sum_{n=1}^{\bar{N}+\nu} \boldsymbol{c}_n \cdot u_n = C \cdot \boldsymbol{u}$$

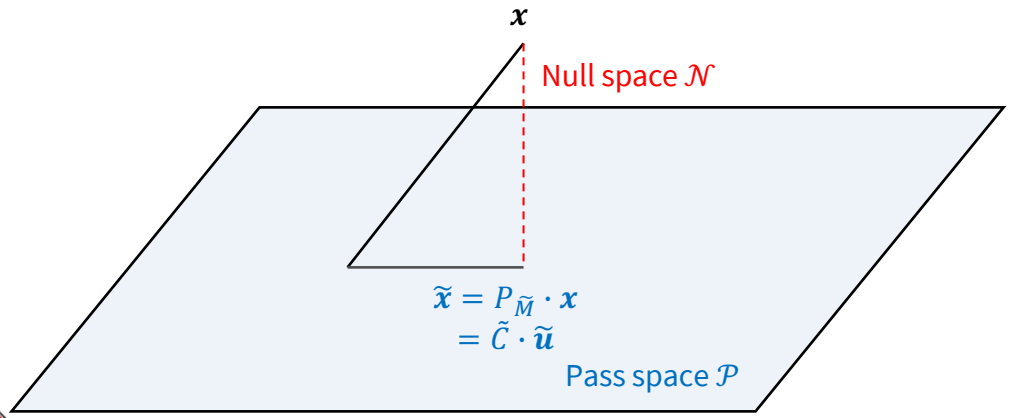
$$\boldsymbol{x} = \underbrace{\begin{bmatrix} c_{N+\nu-1} & \cdots & c_1 & c_0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} u_{N+\nu-1} \\ \vdots \\ u_1 \\ u_0 \end{bmatrix}}_{\boldsymbol{u}}$$

$$C \to \tilde{C}$$
$$\boldsymbol{u} \to \tilde{\boldsymbol{u}}$$

$$\tilde{\mathbf{x}} = \underbrace{\begin{bmatrix} \tilde{\mathbf{c}}_{\tilde{N}-1} & \cdots & \tilde{\mathbf{c}}_1 & \tilde{\mathbf{c}}_0 \end{bmatrix}}_{\tilde{C}} \underbrace{\begin{bmatrix} \tilde{u}_{\tilde{N}-1} \\ \vdots \\ \tilde{u}_1 \\ \tilde{u}_0 \end{bmatrix}}_{\tilde{\mathbf{u}}}$$

$$R_{\boldsymbol{x}\boldsymbol{x}} \neq R_{\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}}$$

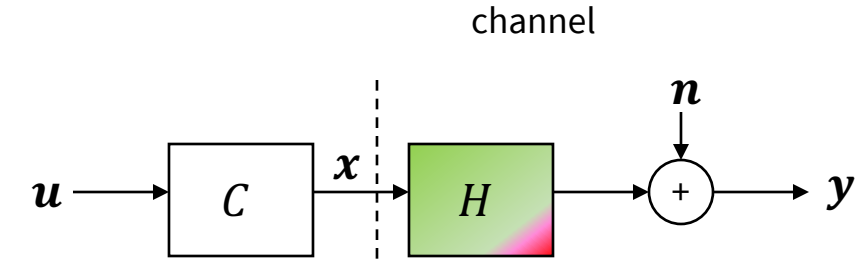
Part of $\boldsymbol{x}$ in pass space



See also Figure 5.4 Flowchart.

Equivalence

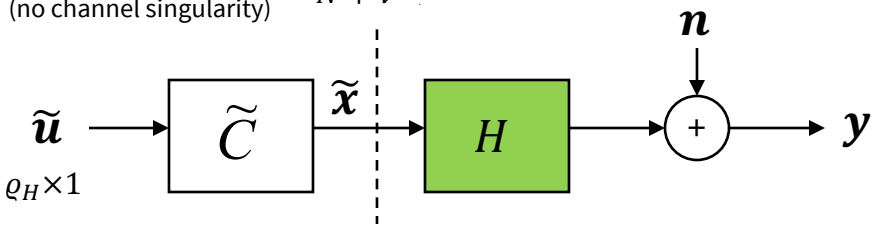
- The original discrete modulator:



equivalent channel
(no channel singularity)

$\bar{N} + \nu$

- The equivalent modulator:



- A decoder for the original $\boldsymbol{x}$ (or $\boldsymbol{u}$) cannot achieve reliable data rate for information outside $\tilde{\boldsymbol{x}} \in \mathcal{P}$ or outside $\tilde{\boldsymbol{u}}$, so avoid $\mathcal{N}$.
  - So the design might as well choose codewords $\in \mathcal{P}$.
  - Why waste energy on codeword separation in channel null space, $\boldsymbol{x}_B - \boldsymbol{x}_A \in \mathcal{N}$?
  - Because, if $\boldsymbol{x}_B \neq \boldsymbol{x}_A$, then $d_{AB} = \|H \cdot \boldsymbol{x}_B - H \cdot \boldsymbol{x}_A\| = 0$, which creates a MAP-decoder``coin flip'' decision for this symbol.
  - Side comment: Active MAC users may necessarily (no coordination) have nonzero energy in $\mathcal{P}$.

Fixmod.m & 1+D example

- Eliminate modulator's null-space components

```
function [Ct, Ot, Ruutt] = fixmod(H_NW, Ruu, C, tol)
-------------------------------------------------------------------------
fixmod removes the part of x that lies in H_NW's nullspace
    x->xt, u->ut
 Inputs: H_NW, Ruu, C, tol
   H_NW: Noise-whitened channel
   Ruu: Autocorrelation matrix of u
   C: discrete modulator matrix
   tol: used in licols to determine rank of matrix Ctemp

 Outputs: Ct, Ot, Ruutt
   Ct: new discrete modulator with components without H_NW's nullspace
     components
   Ot: Projection matrix of C2 onto C
   Ruutt: new autocorrelation matrix for u

 This program calls licols
-------------------------------------------------------------------------
```

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_2 \\ \boldsymbol{u}_1 \end{bmatrix}$$

$$\tilde{\boldsymbol{x}} = \tilde{C} \cdot \tilde{\boldsymbol{u}}$$

$$\tilde{\boldsymbol{u}} = \begin{bmatrix} \tilde{\boldsymbol{u}}_2 \\ \tilde{\boldsymbol{u}}_1 \end{bmatrix} = \boldsymbol{u}_2 + \tilde{O} \cdot \boldsymbol{u}_1$$

$$R_{\tilde{u}\tilde{u}} = R_{22} + \tilde{O} \cdot R_{12} + R_{21} \cdot \tilde{O}^* + \tilde{O} \cdot R_{11} \cdot \tilde{O}^*$$

```
>> H_NW =
   1   1   0
   0   1   1
>> Ruu = eye(3);
>> C = eye(3);
>> [Ct, Ot, Ruutt] = fixmod(H_NW, Ruu, C)
Ct =
   0.6667   0.3333
   0.3333   0.6667
  -0.3333   0.3333
Ot =
  -1.0000
   1.0000
Ruutt =
   2.0000  -1.0000
  -1.0000   2.0000
>> H_NW*Ct =
   1.0000   1.0000
  -0.0000   1.0000
```

$(x_k = u_k)$

$$\tilde{\boldsymbol{u}} = \begin{bmatrix} \tilde{u}_2 \\ \tilde{u}_1 \end{bmatrix} = \boldsymbol{u}_2 + \tilde{O} \cdot \boldsymbol{u}_1 = \begin{bmatrix} u_3 \\ u_2 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} u_1 = \begin{bmatrix} u_3 - u_1 \\ u_2 + u_1 \end{bmatrix}$$

$$= \begin{bmatrix} x_3 - x_1 \\ x_2 + x_1 \end{bmatrix}$$

$$\boldsymbol{y} = H \cdot \tilde{C} \cdot \tilde{\boldsymbol{u}} + \boldsymbol{n}$$

$$= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \tilde{u}_2 \\ \tilde{u}_1 \end{bmatrix} + \boldsymbol{n}$$

Output with new/old input

$$= \begin{bmatrix} x_3 + x_2 \\ x_2 + x_1 \end{bmatrix} + \boldsymbol{n}$$

Input Singularity

Section 5.1.1.2

Eliminate input singularity (Section 5.1.1.2)

- Singular **input** modes carry no information, so design also can silence corresponding channel modes.
  - See also Figure 5.6 Flowchart.

```
function [At, OA, Ruupp] = fixin(Ruutt, Ct, tol)

--------------------------------------------------------------------------
xt->xp, ut->up (Reducing ranking of ut->up, possible b/c Rxxtt singlar)
Inputs: Ruutt, Ct, tol
Outputs: At, OA, Ruupp

Ruutt: autocorrelation matrix of ut
Ct: "nullspace of H"-adjusted discrete modulator matrix
tol: used in licols to determine rank of matrix Ctemp
At: "nullspace of H & Rxx singularity"-adjusted discrete modulator matrix
OA: projection matrix of A2 onto A
Ruupp: autocorrelation matrix of up
--------------------------------------------------------------------------
```

$$\boldsymbol{x}' = \tilde{A} \cdot \boldsymbol{u}'$$

- Example has input of rank 2 after first reduction of channel null-space components, so no further reduction is possible.

```
>> Rxxtt=Ct*Ruutt*Ct'        (input in pass space)
   0.6667   0.3333  -0.3333
   0.3333   0.6667   0.3333
  -0.3333   0.3333   0.6667
>> rank(Rxxtt)  % =    2
>> rank(Ruutt)  % =    2
>> rank(Ct)  % =    2

>> [At, OA, Ruupp] = fixin(Ruutt, Ct)

At =     (same as Ct)
   0.6667   0.3333
   0.3333   0.6667
  -0.3333   0.3333
OA =  2 × 0 empty double matrix (happens: no more singularity)
Ruupp =  (looks same as Ruutt)
   2.0000  -1.0000
  -1.0000   2.0000
```

$$\boldsymbol{u}' = \tilde{\boldsymbol{u}}_2 + O_A \cdot \tilde{\boldsymbol{u}}_1$$

$$R_{\boldsymbol{x}'\boldsymbol{x}'} = \tilde{A} \cdot R_{\boldsymbol{u}'\boldsymbol{u}'} \cdot \tilde{A}^* = \tilde{A} \cdot \left(\tilde{R}_{22} + O_A \cdot \tilde{R}_{12} + \tilde{R}_{21} \cdot O_A^* + O_A \cdot \tilde{R}_{11} \cdot O_A^* \right) \cdot A^*$$

Singular input with 1 +D

- Another input that has no energy on one of the channel pass modes

```
>> [F,L,M]=svd(H_NW);
M =
  -0.4082   0.7071   0.5774
  -0.8165  -0.0000  -0.5774
  -0.4082  -0.7071   0.5774
L =
    1.7321         0         0
         0    1.0000         0
 >>M = -M;  F=-F; %for appearance
% let C=M, and put 1 unit of energy on one pass-space mode
% and another 2 units in null space
>> Ruu=[1 0 0 ; 0 0 0 ; 0 0 2];
>> Rxx=M*Ruu*M' =[ 5/6 -1/3 5/6
-1/3 4/3 -1/3
5/6 -1/3 5/6];
% note by inspection of M=C
>> C=[1/sqrt(6) -1/sqrt(2) -1/sqrt(3)
sqrt(2/3) 0 1/sqrt(3)
1/sqrt(6) 1/sqrt(2) -1/sqrt(3) ];
>> [Ct, Ot, Ruutt] = fixmod(H_NW, Ruu, C)
Ct =
   0.4082  -0.7071
   0.8165   0.0000
   0.4082   0.7071
Ot =  1.0e-15 *  (Ot is zero)
   0.1473
   0.0196
Ruutt =
   1.0000   0.0000
   0.0000   0.0000
```

$$\boldsymbol{x} = M \cdot \boldsymbol{u}$$

```
>> [At, OA, Ruupp] = fixin(Ruutt, Ct)
At =
   0.4082
   0.8165
   0.4082
OA =  9.9148e-16  (zero)
Ruupp =  1.0000
>> H=[1 1 0
0 1 1];
>> H_NW*At =
   1.2247
   1.2247
>> At'*H_NW'*H_NW*At =  3.0000
-------------------------------------------
>> Rxxtt=At*At' =
   0.1667   0.3333   0.1667
   0.3333   0.6667   0.3333
   0.1667   0.3333   0.1667
>> trace(Rxxtt) % =   1.0000
>> trace(Rxx) % =    3
```

- This becomes a scalar channel,
- after matched matrix filter.

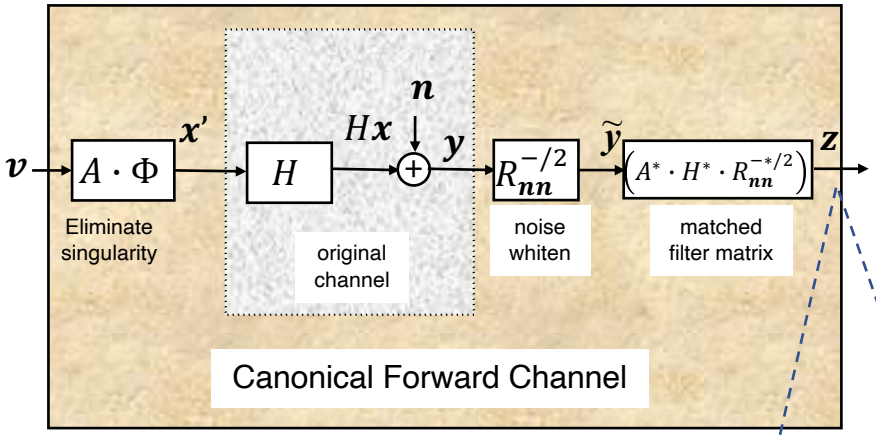
$$z = A^* \cdot H^* \cdot \boldsymbol{y} = 3 \cdot u + n'$$

So, the equivalent real channel after removing singularity and designing an input is a scalar.

MMSE-GDFE

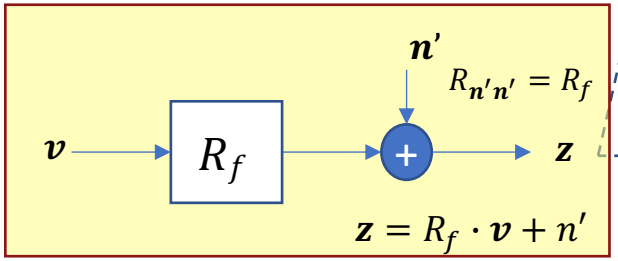
Section 5.1

The Canonical Channels -Refresher



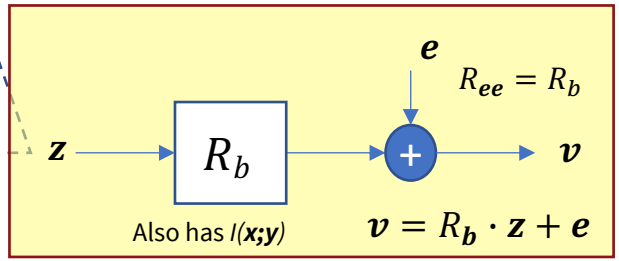
- v and z have same dimensionality, why?
- Non-singular (all dimensions now pass).
- $\mathcal{I}(v; z) = \mathcal{I}(x; y)$
  - All adjustments were 1-to-1 mappings
  - or eliminated only zero-information dimensions.
- R_f is the MMSE Filter to estimate z using v.
- R_b is the MMSE Filter to estimate u using z.

Forward Canonical Channel



Ok, but needs (complex) ML detector.

Backward Canonical Channel



(symmetry of mutual information)

dimensional detection on G

- Inverse of R_b

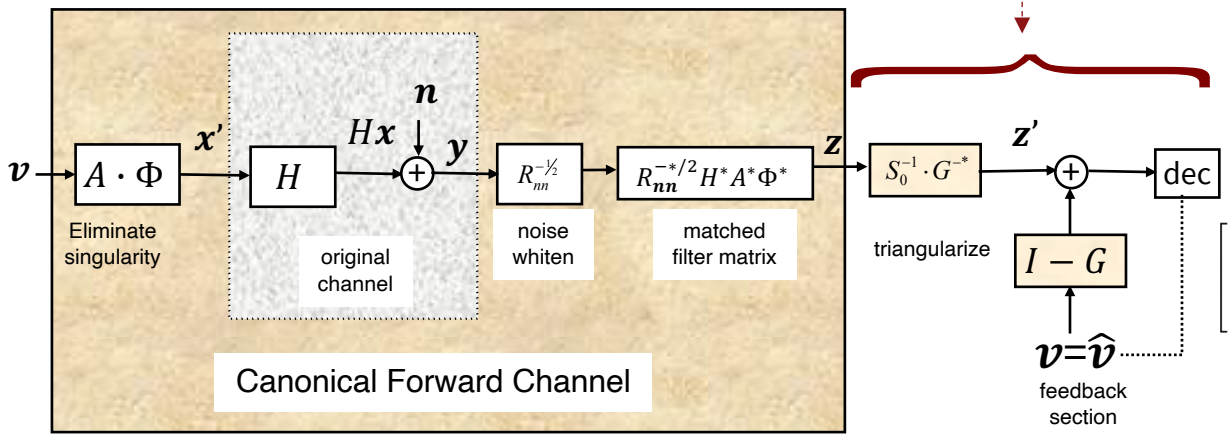
$$R_b^{-1} = R_f + I = G^* \cdot S_0 \cdot G$$

lower diag upper

$$G \cdot \boldsymbol{v} = G \cdot R_{\boldsymbol{b}} \cdot \boldsymbol{z} + G \cdot \boldsymbol{e} = S_0^{-1} \cdot G^{-*} \cdot \boldsymbol{z} + \boldsymbol{e}'$$

Cholesky Factorization

- Generalized Decision Feedback Equalizer



$$\boldsymbol{z}' = G \cdot \boldsymbol{v} - \boldsymbol{e}'$$

GDFE triangularizes the channel and diagonalizes error. We saw this as chain-rule ($\mathcal{I}$) and MMSE with MAC

$$E[|e_n'|^2] = S_{0,n}^{-1} = \frac{1}{S_{0,n}}$$

$$\begin{bmatrix} z'_{\tilde{N}^*-1} \\ z'_{\tilde{N}^*-2} \\ \vdots \\ z'_0 \end{bmatrix} = \begin{bmatrix} 1 & g_{\tilde{N}^*-1,\tilde{N}^*-2} & \cdots & g_{\tilde{N}^*-1,0} \\ 0 & 1 & \cdots & g_{\tilde{N}^*-2,0} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{\tilde{N}^*-1} \\ v_{\tilde{N}^*-2} \\ \vdots \\ v_0 \end{bmatrix} - \begin{bmatrix} e'_{\tilde{N}^*-1} \\ e'_{\tilde{N}^*-2} \\ \vdots \\ e'_0 \end{bmatrix}$$

solve by back substitution

So what? Let's look at those SNR_n's

- They are: $$\mathrm{SNR}_{bias,n} = 1 + \mathrm{SNR}_{v,n} = \frac{\mathbb{E}[|v_n|^2]}{\mathbb{E}[|e'_n|^2]} = 1 \cdot S_{0,n}$$

Bias applies as always to each dimension.

- Overall (geometric) SNR is:

$$\mathrm{SNR}_{\mathrm{GDFE}} = \left(\prod_{n=1}^{\bar{N}^*} \mathrm{SNR}_{bias,n} \right)^{\frac{1}{\bar{N}+\nu}} = \frac{|R_{\boldsymbol{vv}}|^{1/(\bar{N}+\nu)}}{|R_{\boldsymbol{e'e'}}|^{1/(\bar{N}+\nu)}} = |R_{\boldsymbol{ee}}|^{-\frac{1}{\bar{N}+\nu}} = 2^{2\bar{I}(\boldsymbol{v},\boldsymbol{z'})} = 2^{2\bar{I}(\boldsymbol{x},\boldsymbol{y})}$$

- Because $R_{\boldsymbol{vv}}=I$ – that is, the input $\boldsymbol{v}$ is "white."
- The designer can make $\boldsymbol{v}$ white (independent) ……
- Or, it might have been white input initially,
  - Even if some energy lost into null space.

Incidentally, Cholesky on forward canonical does not have this overall SNR (it's lower).

GDFE white-input design

- Factor R_{uu} so that

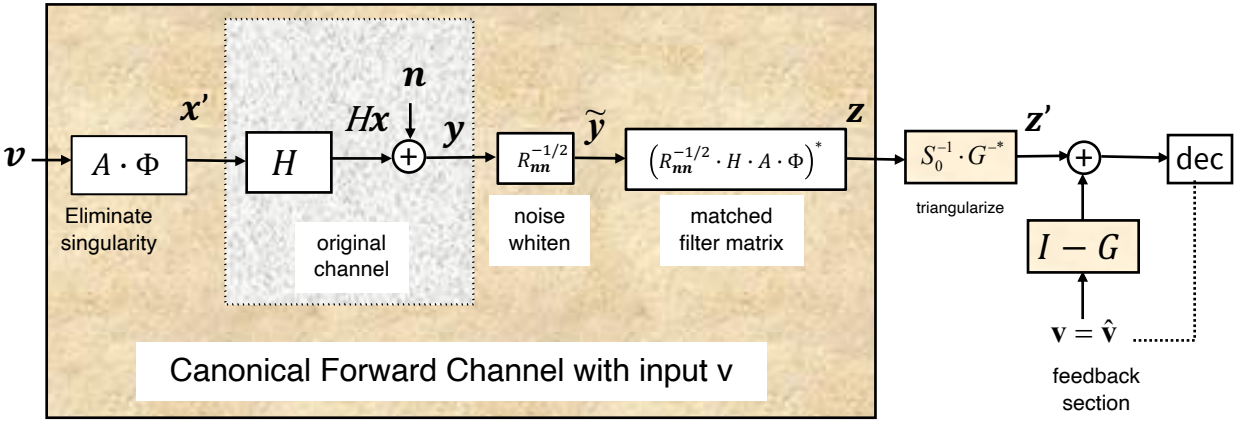
$$\boldsymbol{u} = \Phi \cdot \boldsymbol{v}$$

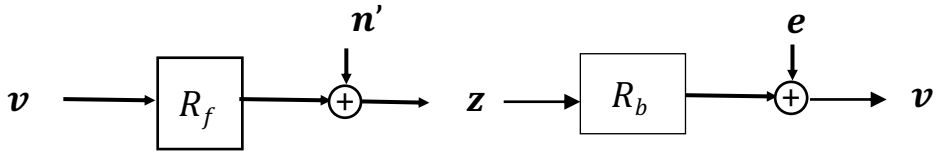
- And $R_{vv}=I$
  - Eigen-decomposition works
  - So does Cholesky
  - So do many other factorizations (infinite number).

$$R_{uu} = \Phi \cdot \Phi^*$$

- The singularity removal now adjusts (1-to-1) so that $A \rightarrow A \cdot \Phi$.
- $\mathcal{I}(\boldsymbol{z}; \boldsymbol{v}) = \mathcal{I}(\boldsymbol{x}; \boldsymbol{y})$.

Canonical-performance GDFE





- Backward channel has canonical performance; it does not need (full) ML detector.
- Equivalently, if earlier decisions within packet correct, then MMSE→ML.

Example 1+.9D^{-1} (real baseband)

ELIMINATE SINGULARITY

```
H=(1/sqrt(.181))*[.9 1 0
0 .9 1];
>> [Ct, Ot, Ruutt] = fixmod(H, eye(3), eye(3))
Ct =
   0.5945   0.3649
   0.3649   0.6715
  -0.3285   0.2956
Ot =
  -1.2346
   1.1111
Ruutt =
   2.5242  -1.3717
  -1.3717   2.2346
```

$$\tilde{\boldsymbol{u}} = \boldsymbol{u}_1 + \tilde{O} \cdot \boldsymbol{u}_2$$

```
>> [A, OA, Ruupp] = fixin(Ruutt, Ct)
A =
   0.5945   0.3649
   0.3649   0.6715
  -0.3285   0.2956
OA =
  2 x 0 empty double matrix
Ruupp =
   2.5242  -1.3717
  -1.3717   2.2346
>> Gubar=lohc(Ruupp);
>> Gu=Gubar*inv(diag(diag(Gubar))) ;
>> Xmit=A*Gubar;
>> Sx=diag(diag(Gubar))*diag(diag(Gubar))  =
   1.6821        0
        0   2.2346
```

$$\boldsymbol{u}' = \tilde{u}_1 + \tilde{O}_A \cdot \tilde{u}_2$$

DESIGN RECEIVER

```
>> Ht=H*A*Gu*sqrtm(Sx) =
   2.7436   1.5724
   0.0000   3.1623
>> Rf=Ht'*Ht;
>> Rbinv=Rf+eye(2);
>> Gbar=chol(Rbinv);
>> G=inv(diag(diag(Gbar)))*Gbar

G =
   1.0000   0.5059
        0   1.0000
>> S0=diag(diag(Gbar))^2 =
   8.5275        0
        0  11.2899
>> SNR=(det(S0))^(1/3)-1 %= 3.5832
>> 10*log10(SNR)  %=    5.5427 dB
>> W=inv(S0)*inv(G')
   0.1173        0
  -0.0448   0.0886
>> W*Ht'*Ht  %=
   0.8827   0.5059
   0.0448   0.9114
MMSE TRIANGULAR ERROR

UNBIASED GDFE FILTERS
>> WHunb=S0*inv(S0-eye(2))*W*Ht'*Ht =
   1.0000   0.5731
   0.0492   1.0000
>> Gunb=eye(2)+S0*inv(S0-eye(2))*(G-eye(2)) =
   1.0000   0.5731
        0   1.0000
```

BIT DISTRIBUTION

```
>> bbar=0.5*log2(diag(S0^(1/3)))  =

   0.5154
   0.5828

>> sum(bbar) =   1.0982
```

Best for $N=2$, $\nu=1$
With flat energy on R_{xx}

Same SNR as Vector Coding

computeGDFE.m

```
function [snrGDFEu, GU, WU, S0, MSWMFU, b, bbar, snrGLEu] =
computeGDFE(H, A, cb, Lx)
 ------------------------------------------------
  Inputs
  H: noise-whitened channel, Ly x Lx (one tone)
  A: any Lx x Lx square root of input autocorrelation matrix
        This can be generalized non-square square-root
  cb: =1 if H is complex baseband; =2 if H is real baseband
  Lx: optional input of Lx when not equal to size of A
      this is used to compute bits/dimension properly

  Outputs
  GU: unbiased feedback matrix
  WU: unbiased feedforward linear equalizer
  S0: sub-channel channel gains
  MSWMFU: unbiased mean-squared whitened matched filter
  b: bit distribution vector
  bbar: number of bits/dimension (real if cb=2, complex if cb=1)
  snrGDFEu - unbiased SNR in dB; assumes size of R_sqrt input
     the user should recompute SNR if there is a cyclic prefix
  b = bit distribution over symbol dimensions (real cb=2; complex
cb=1)
  bbar is sum(b)/Lx so total bits/(real cb=2; cplx cb=1) dimension
  snrGLEu is the linear GLE SNR

  Note: The R_sqrt need not be square non-singular, as long as
  R_sqrt*R_sqrt' = input autocorrelation matrix Rxx, but SNRLEu is off
 Thanks to Ethan Liang, corrected/updated J. Cioffi
```

If A is nonsquare, the Lx sets dimensionality

Recalling

```
H =
   2.1155   2.3505        0
        0   2.1155   2.3505
>> [Ct, Ot, Ruutt] = fixmod(H, eye(3), eye(3));
>> [A, OA, Ruupp] = fixin(Ruutt, Ct);
% previous square-root forms as
>> Gxbar=lohc(Ruupp);
>> Gx=Gxbar*inv(diag(diag(Gxbar))) ;
>> Xmit=A*Gxbar;
```

many square root choices

```
>> cb=2;
>> Lx=3;
[snrGDFEu, GU, WU, S0, MSWMFU, b, bbar,~] = computeGDFE(H, Xmit,cb,Lx)

snrGDFEu =   5.5427 dB
GU =
   1.0000   0.5731
        0   1.0000

MSWMFU =
   0.3645   0.0000
   0.0179   0.3073
b =                  bits/symbol
   1.5461
   1.7485
bbar =   1.0982   bits/dimension (real for this example because cb = 2)
```

Need for Receiver

```
>> Xmit =

   0.7710   0.0000
   0.4733   0.6690
  -0.4260   0.7433
```

Note that there are 3 dimensions per symbol (and 2 channel outputs/symbol – guard period)

Other Square roots?

MATLAB's matrix square root (symmetric)

```
>> [Ct, Ot, Ruutt] = fixmod(H, eye(3), eye(3));
>> [A, OA, Ruupp] = fixin(Ruutt, Ct);
>> Gxbar=sqrtm(Ruutt)
>> Gxbar =
   1.5186   -0.4668
  -0.4668    1.4201
>> Xmit=A*Gxbar;

>> [snrGDFEu, GU, WU, S0, MSWMFU, b,
bbar,~] = computeGDFE(H, Xmit,cb,Lx);
>> snrGDFEu  =   5.5427 dB

>> GU =
   1.0000    0.3680
        0    1.0000
>> MSWMFU  =
   0.3881   -0.1812
   0.1215    0.2378
>> b' =   1.3447    1.9499
>> bbar =   1.0982
```

Eigenvectors

```
>> [Ct, Ot, Ruutt] = fixmod(H, eye(3), eye(3));
>> [A, OA, Ruupp] = fixin(Ruutt, Ct);
>> [V,D]=eig(Ruutt);
>> Gxbar=V*sqrt(D) =
  -0.6690   -1.4411
  -0.7433    1.2970
>> Xmit=A*Gxbar;

>> [snrGDFEu, GU, WU, S0, MSWMFU, b,
bbar,~] = computeGDFE(H, Xmit,cb,Lx);
>> snrGDFEu =   5.5427 dB

>> GU =
   1.0000   -0.3459
        0    1.0000

>> MSWMFU =
  -0.2535   -0.1261
  -0.1648    0.3645
>> b' =   1.8760    1.4186
>> bbar =   1.0982
```

- Pick a random unitary matrix and postmultiply any Xmit here by it: → another GDFE
- Same performance, Different bit distribution

Start with 3x3 (singular) inputs?

Singular Sq Root

```
>> [Ct, Ot, Ruutt] = fixmod(H, eye(3), eye(3));
>> [A, OA, Ruupp] = fixin(Ruutt, Ct);
>> Rxxtt=A*Ruupp*A';
>> Gxtilde=sqrtm(Rxxtt) =
   0.5945   0.3649  -0.3285
   0.3649   0.6715   0.2956
  -0.3285   0.2956   0.7340
>> [snrGDFEu, GU, WU, S0, MSWMFU, b,
bbar] = computeGDFE(H, Gxtilde,2,3);

snrGDFEu =   5.5427 dB
GU =
   1.0000   1.1111   0.0000
        0   1.0000   0.9067
        0        0   1.0000
MSWMFU =
   0.4727   0.0000
   0.0783   0.3857
  -0.1923   0.4254

>> b' =   1.2264   1.3485   0.7196
>> bbar =   1.0982
```

With wasted energy on input

```
>> [snrGDFEu, GU, WU, S0, MSWMFU, b,
bbar] = computeGDFE(H, eye(3),2,3)

>> snrGDFEu =  5.5427 dB
>> GU =
   1.0000   1.1111        0
        0   1.0000   0.9067
        0        0   1.0000
>> MSWMFU =
   0.4727        0
   0.0783   0.3857
  -0.1923   0.4254

>> b' =   1.2264   1.3485   0.7196
>> bbar =   1.0982

>> trace(Rxxtt) =   2.0000 < 3
```

$R_{xx} = I\ (3\times3)$

Use that extra energy on input?

```
>> [snrGDFEu, GU, WU, S0, MSWMFU, b, bbar,~]
= computeGDFE(H, sqrt(1.5)*Rxxtt,2,3)

snrGDFEu =   6.8589 dB
GU =
   1.0000   1.1111   0.0000
        0   1.0000   0.9578
        0        0   1.0000
MSWMFU =
0.3860        0
   0.0479   0.3327
  -0.1619   0.3474

>> b' = 1.4736   1.5677   0.7819
>> bbar =   1.2744
```

- Placing energy in null space is a waste
- Last example has better energy placement

Zero-Forcing GDFE

- ZF designs ignore noise (essentially assuming it is zero for design, but not for SNR calculation).
- For the finite-length symbol case:
  - Do QR factorization of the channel (again with rq.m program at web site).

$$\widetilde{H} = \begin{bmatrix} 0 & R_{ZF} \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} q^* \\ Q^* \end{bmatrix}$$

- R_{ZF}=$S_{ZF} \cdot G_{ZF}$ so G_{ZF} defines the precoder or feedback section.
- Not necessarily canonical except in special cases (no crosstalk/ISI and worst-case noise).
  - But ZF-GDFE is pretty close to MMSE-GDFE in many cases.
  - ZF-GDFE is simpler to design (essentially just 1 QR factorization).
    - Thus, ZF is often found in various descriptions.

Generalized Linear Equalizer

- This is just MMSE linear estimate for any user.
- $\mathbb{E}[R_{ee}] = R_b^{-1}$ and so the MMSE for linear equalizer are the diagonal elements of R_b^{-1}
- The computeGDFE program has an optional last oufput that is the corresponding unbiased LE SNR.
- This can be compared with the GDFE SNR to estimate the performance gain of the nonlinear feedback section relative to linear best solution.
- Since many field systems today use linear, designers have a tool to see how much is lost with the linear approach.
- The LE bit distribution is $\frac{1}{2}\cdot\log_2(diag\{R_b\})^{-1}$ in bits for each real dimension at output.

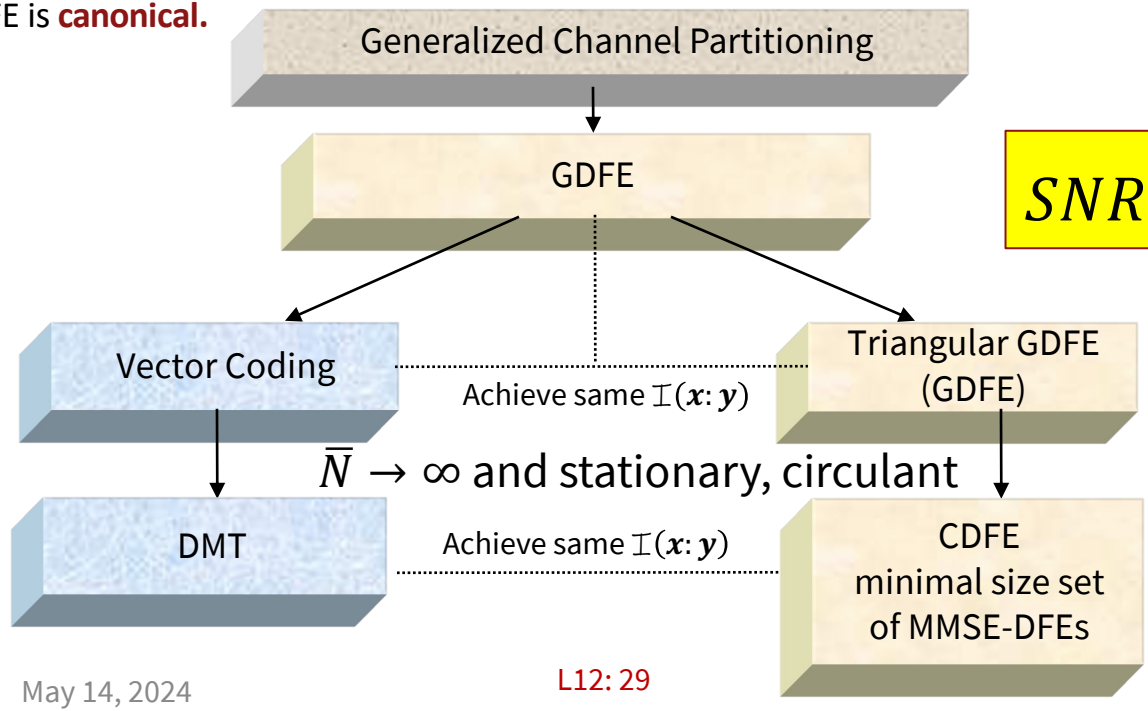
```
>> [snrGDFEu, GU, WU, S0, MSWMFU, b, bbar , snrLE] =
computeGDFE(H, sqrt(1.5)*Rxxtt,2,3)
snrLE =

2.0742 dB < 6.9 dB
```

Special GDFE Forms

Section 5.2

Canonical Performance for any Square Root

- GDFE has **canonical** performance:
  - Same good codes for AWGN ($\Gamma \rightarrow 0$ dB) work on GDFE-generated dimensions to drive $P_e \rightarrow 0$ if $\tilde{b} \leq \tilde{c}$.
  - Even though, it is not (sub-asymptotically) an ML detector (error propagation is not an issue if $P_e \rightarrow 0$).
- This extends Chap 3 CDEF Result extension for infinite-length (set of) MMSE-DFE(s).
  - MMSE-DFE is **canonical.**

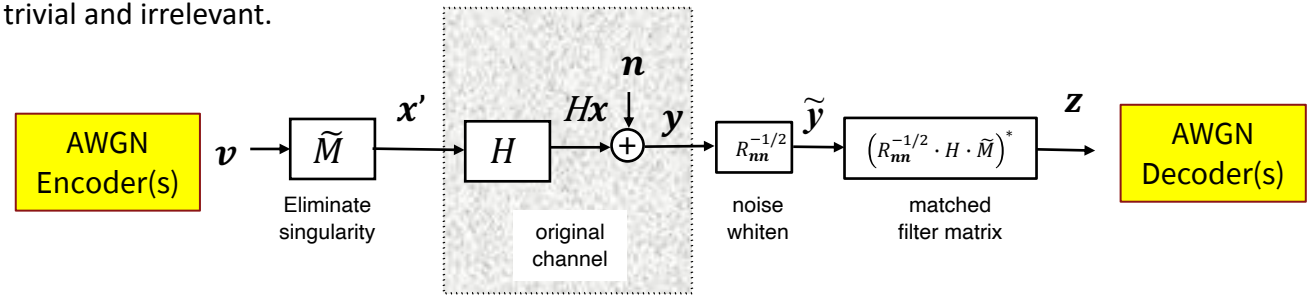


Vector Coding Special Case

- SVD of noise-equivalent channel is $R_{nn}^{-1/2} \cdot H = F \cdot \Lambda \cdot M^*$.

Assume (here) freq-time dimensions only ($\bar{N} + \nu$).

- VC uses any input of form $\boldsymbol{x} = \sum_{n=0}^{N+\nu-1} v_n \cdot \boldsymbol{m}_n \rightarrow \sum_{n=0}^{\wp_x - 1} v_n \cdot \boldsymbol{m}_n$ with input singularity removal (reorder).
  - Discrete modulator is $A = [\boldsymbol{m}_{\wp_x} \quad \cdots \quad \boldsymbol{m}_0]$; $\tilde{H} = R_{nn}^{-1/2} \cdot H \cdot \tilde{M}$.
- Forward channel simplifies to $R_f = \tilde{H}^* \cdot \tilde{H} = Diag\{\lambda^2_{\wp_x - 1}, \ldots, \lambda_0^2\}$,
  - so $R_b^{-1} = Diag\{\lambda^2_{\wp_x - 1} + 1 \quad, \cdots, \quad \lambda_0^2 + 1\} = S_0$.
- Trivially , $G = I$, so no feedback is needed (no error prop), and the GDFE detector is ML,
  - which involves ML decoder on each dimension independently acting on codewords of possibly many symbols.
  - Bias is trivial and irrelevant.



Vector Coding

- VC works for any dimensionality.
  - Vector DMT/OFDM – space-time-H SVD for each tone.
- Water-fill allows $\overline{\mathbb{I}}(\boldsymbol{x}; \boldsymbol{y}) \rightarrow \bar{\mathcal{C}}$.
- VC is both canonical and optimal; just can't do better (few flaws).
- DMT is frequency-time version of VC,
  - with asymptotically negligible cyclic prefix loss.
  - Vector DMT just does water-fill over many more dimensions in space-freq.

Triangular GDFEs

- Any GDFE where $G \neq I$ is triangular.

- Why not just Vector Code?
  - Latency of block – sometimes, the triangular/recursive structure allows "causal" implementation (lower delay).
    - This will involve "ignoring" guard periods and consequent non-optimal transients, but negligible with long symbols.
  - The transmitter dimensions may be separated physically (e.g., the MAC).
  - The receiver dimensions may be separated physically (e.g., the BC).

- So (canonical) GDFE's may fit better the situation than VC.

$$R_{\boldsymbol{uu}} = \Phi \cdot \Phi^* = G_{\boldsymbol{\phi}} \cdot S_x \cdot G_{\boldsymbol{\phi}}^* \quad G_{\boldsymbol{\phi}} \text{ is monic upper triangular (Cholesky}^{16}\text{)}$$

$$|R_{\boldsymbol{uu}}| = |S_x|$$

- Discrete modulator is $A = \left(P_{A,C}\right) \cdot G_{\boldsymbol{\phi}} \cdot S_{\boldsymbol{\phi}}^{1/2}$ where $\left(P_{A,C}\right)$ removes singularity to get to $\boldsymbol{u} = G_{\boldsymbol{\phi}} \cdot S_{\boldsymbol{\phi}}^{1/2} \cdot \boldsymbol{v}$.

[16] use lohc.m command

Circulant DFE

- The CDFE uses cyclic prefix, but attempts to model each energized band in time domain.
- The consequent square-circulant channel matrix H has full rank, so $\mathcal{N} = \emptyset$,
  - as long as H is not a matrix of all constant equal values.
- With circulant $\bar{N}\times\bar{N}$ input $R_{\boldsymbol{uu}} = \Phi \cdot \Phi^* = R_{\boldsymbol{xx}}$, the CDFE receiver ignores the cyclic prefix.
  - The time-domain convolution appears periodic for any symbol.
  - The factorization $R_{\boldsymbol{uu}}$=$\Phi \cdot \Phi^*$corresponds to "causal" filter from input $\boldsymbol{v}$.
    - Special case $R_{\boldsymbol{xx}} = \mathrm{I}$, then simply direct input to channel, $\boldsymbol{x} = \boldsymbol{u} = \boldsymbol{v}$ with cyclic prefix.
  - The CDFE Imitates EE379's MMSE-DFE as $\bar{N} \to \infty$; and indeed, it's better for $\bar{N} < \infty$.
- However, good input design (almost always) introduces singularity (think water-filling).
  - This requires some care to create single-carrier OFDM bands.
  - And, it's not as simple as just transmit data $\boldsymbol{x} = \boldsymbol{v}$ into the channel – interpolation of some type needed.
- Also known as "Single-carrier OFDM" in standards (CDFE name and publication predates SC-OFDM by more than 10 years).

CDFE Example

```
>> H=toeplitz([.9 zeros(1,7)]',[.9 1 zeros(1,6)]);
H(8,1)=1;
>> H=1/sqrt(.181)*H;
>> [snrGDFEu, GU, WU, S0, MSWMFU, b, bbar,LE] = computeGDFE(H, eye(8), 2, 9)
>> snrGDFEu =   7.1666 dB
>> GU =
   1.0000  0.4972       0       0       0       0       0  0.4972
        0  1.0000  0.6414       0       0       0       0 -0.2899
        0       0  1.0000  0.6930       0       0       0  0.1780
        0       0       0  1.0000  0.7128       0       0 -0.1113
        0       0       0       0  1.0000  0.7206       0  0.0702
        0       0       0       0       0  1.0000  0.7237 -0.0444
        0       0       0       0       0       0  1.0000  0.7531
        0       0       0       0       0       0       0  1.0000
>> MSWMFU =
   0.2115       0       0       0       0       0       0  0.2351
   0.1798  0.2729       0       0       0       0       0 -0.1371
  -0.1104  0.1601  0.2948       0       0       0       0  0.0841
   0.0691 -0.1002  0.1525  0.3033       0       0       0 -0.0526
  -0.0435  0.0631 -0.0961  0.1495  0.3066       0       0  0.0332
   0.0275 -0.0399  0.0608 -0.0945  0.1483  0.3079       0 -0.0210
  -0.0174  0.0253 -0.0385  0.0598 -0.0939  0.1478  0.3084  0.0133
  -0.0933  0.0418  0.0010 -0.0439  0.0961 -0.1687  0.2772  0.1647
>> b' =
   1.7297   1.5648   1.5156   1.4978   1.4909   1.4882   1.4871   1.0792
>> bbar =   1.3170
LE =   4.4409 % dB (linear is 2.73 dB worse).
```

- For very long symbol, → 8.4 dB
- G_u → .725 single feedback coefficient = (7.85/6.85)x.633
- W → constant-row feedforward filter
- CDFE's limit is Chapter 3's MMSE-DFE for infinite symbol length
- Same as DMT (which it should be):

```
>> [V,D]=eig(H);
>> [snrGDFEu, GU, WU, S0, MSWMFU, b, bbar,LE] = computeGDFE(H, V, cb, 9)
snrGDFEu =   7.1666
>> GU-eye(8) =   1.0e-13 *  (It's an identity for DMT!)
>> MSWMFU(:,1:3) = (note that it is complex – why? Channel is real? V is complex)
-1.5042 + 0.0000i   1.5042 + 0.0000i  -1.5042 + 0.0000i
  0.1018 + 0.1782i   0.0540 - 0.1980i  -0.1782 + 0.1018i
  0.1018 - 0.1782i   0.0540 + 0.1980i  -0.1782 - 0.1018i
 -0.0748 + 0.0831i   0.0831 + 0.0748i   0.0748 - 0.0831i
 -0.0748 - 0.0831i   0.0831 - 0.0748i   0.0748 + 0.0831i
  0.0792 + 0.0000i   0.0792 - 0.0000i   0.0792 + 0.0000i
  0.0798 + 0.0311i   0.0784 - 0.0345i   0.0311 - 0.0798i
  0.0798 - 0.0311i   0.0784 + 0.0345i   0.0311 + 0.0798i
(the receiver FFT is included when we use computeGDFE)

>> b' =
   0.0388   0.9942   0.9942   1.7297   1.7297   2.1943   2.0862   2.0862
Note it's different, but the sum is the same as CDFE
>> bbar =   1.3170
>> LE % =   7.1666 (linear is the same).
```

Triangular (with guard period) and no energy in $\mathcal{N}$

```
>> H=(1/sqrt(.181))*toeplitz([.9 zeros(1,7)]',[.9 1 zeros(1,7)])
H=(1/sqrt(.181))*toeplitz([.9 zeros(1,7)]',[.9 1 zeros(1,7)]);
>> [Ct, Ot, Ruutt] = fixmod(H, eye(9), eye(9));
>> [A, OA, Ruupp] = fixin(Ruutt, Ct);

A =
   0.7764   0.2012  -0.1811   0.1630  -0.1467   0.1320  -0.1188   0.1069
   0.2012   0.8189   0.1630  -0.1467   0.1320  -0.1188   0.1069  -0.0962
  -0.1811   0.1630   0.8533   0.1320  -0.1188   0.1069  -0.0962   0.0866
   0.1630  -0.1467   0.1320   0.8812   0.1069  -0.0962   0.0866  -0.0779
  -0.1467   0.1320  -0.1188   0.1069   0.9038   0.0866  -0.0779   0.0702
   0.1320  -0.1188   0.1069  -0.0962   0.0866   0.9221   0.0702  -0.0631
  -0.1188   0.1069  -0.0962   0.0866  -0.0779   0.0702   0.9369   0.0568
   0.1069  -0.0962   0.0866  -0.0779   0.0702  -0.0631   0.0568   0.9489
  -0.0962   0.0866  -0.0779   0.0702  -0.0631   0.0568  -0.0511   0.0460

>> Gxbar=lohc(Ruupp);
>> Gx=Gxbar*inv(diag(diag(Gxbar))) ;
>> Xmit=A*Gxbar;

   0.8812  -0.0000   0.0000   0.0000  -0.0000   0.0000  -0.0000   0.0000
   0.2283   0.8757   0.0000   0.0000  -0.0000  -0.0000  -0.0000   0.0000
  -0.2055   0.2397   0.8681  -0.0000   0.0000   0.0000   0.0000   0.0000
   0.1850  -0.2157   0.2554   0.8574  -0.0000   0.0000   0.0000   0.0000
  -0.1665   0.1942  -0.2299   0.2779   0.8416   0.0000   0.0000  -0.0000
   0.1498  -0.1747   0.2069  -0.2501   0.3120   0.8163   0.0000   0.0000
  -0.1348   0.1573  -0.1862   0.2251  -0.2808   0.3678   0.7710  -0.0000
   0.1213  -0.1415   0.1676  -0.2026   0.2527  -0.3310   0.4733   0.6690
  -0.1092   0.1274  -0.1508   0.1823  -0.2274   0.2979  -0.4260   0.7433
```

```
>> [snrGDFEu, GU, WU, S0, MSWMFU, b, bbar, LE] = computeGDFE(H,
Xmit,2,9);

>> snrGDFEu =    7.4896 dB

>> GU =
   1.0000   0.8573   0.0000   0.0000  -0.0000  -0.0000  -0.0000  -0.0000
        0   1.0000   0.7628   0.0000  -0.0000   0.0000   0.0000   0.0000
        0        0   1.0000   0.7174  -0.0000   0.0000   0.0000   0.0000
        0        0        0   1.0000   0.6899   0.0000   0.0000  -0.0000
        0        0        0        0   1.0000   0.6624   0.0000   0.0000
        0        0        0        0        0   1.0000   0.6189   0.0000
        0        0        0        0        0        0   1.0000   0.5233
        0        0        0        0        0        0        0   1.0000

>> MSWMFU =
   0.4165   0.0000   0.0000  -0.0000  -0.0000  -0.0000  -0.0000  -0.0000
   0.0471   0.3738   0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000
  -0.0294   0.0650   0.3560  -0.0000  -0.0000  -0.0000   0.0000   0.0000
   0.0178  -0.0394   0.0693   0.3487   0.0000   0.0000  -0.0000  -0.0000
  -0.0104   0.0231  -0.0407   0.0669   0.3452  -0.0000   0.0000   0.0000
   0.0058  -0.0129   0.0227  -0.0374   0.0599   0.3415   0.0000   0.0000
  -0.0029   0.0065  -0.0114   0.0188  -0.0302   0.0479   0.3328   0.0000
   0.0011  -0.0024   0.0042  -0.0069   0.0111  -0.0176   0.0279   0.3023

> b' =
   1.3789   1.4498   1.4859   1.5067   1.5249   1.5512   1.6042   1.7592
>> bbar =   1.3623
>> LE =    5.8689 (linear 1.6 dB worse)
```

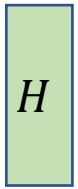
Outperforms CDFE
Same as VC w.r.t. DMT

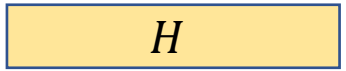
- Better!
- No clear convergence though

Try running computeGDFE to implement Vector Coding – what Xmit? (hint this is easy with this white input)

Massive MIMO = diagonal dominance

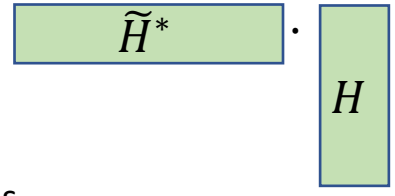
- The channel matrix H is very :
  - tall (many more receiver dimensions than ϱ_H) – uplink.
  - fat (many more transmitter dimensions than ϱ_H) – downlink.





- IF **tall** channel matrix $\tilde{H}$ has entries $\tilde{h}_{il}$ that are largely uncorrelated, except for same index, then

$$\mathbb{E}\left[\tilde{h}_{il} \cdot \tilde{h}_{kl}\right] = |h|^2 \cdot L_y \cdot \delta_{ik}.$$



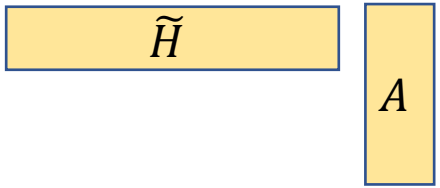
- THEN also $R_f = \tilde{H}^* \cdot \tilde{H}$ off-diag's contain uncorrelated values.
  - Law of large numbers – off diagonals ($\cdot\ 1/L_y$) go to zero, while diagonal grows relatively.

- Then R_b is also almost diagonal, like R_f.
- $G = I \rightarrow$ canonical with no feedback!
- Linear is sufficient and returns to ML.

Massive MIMO Downlink continued

- Fat case (downlink) means small number of receivers.
  - Best transmit A matrix (special square root in BC case, but even with single-user) attempts to match its long columns to the long rows of **fat** $\tilde{H}$, think water-filling (formal GDFE xmit optimization comes soon).
- IF **fat** channel matrix $\tilde{H}$ has entries $\tilde{h}_{il}$ that are largely uncorrelated, except for same index, then

$$\mathbb{E}[\tilde{h}_{il} \cdot a_{kl}] = \mathcal{E} \cdot L_y \cdot \delta_{ik}.$$



- THEN also $R_b = A^* \cdot \tilde{H}^* \cdot \tilde{H} \cdot A + I$ is diagonally dominant,
  - law of large numbers again.
- Here $R_{vv} = I$ by design, then and again there is no feedback.

Wireless: use many antennas at base

Wireline: vectored DSLs (the Gbps DSLs) often happens even when large square (not fat nor tall)

- $G = I \rightarrow$ canonical with no feedback!

Again: no precoder (linear is sufficient).

End Lecture 12

Triangular A in the singular case?

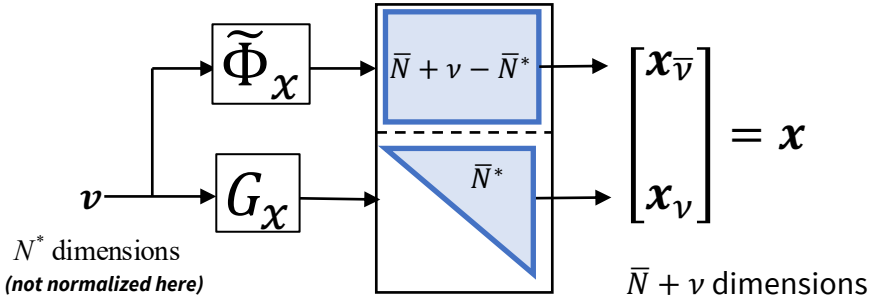
- Generalized Cholesky directly on R_{xx}

$$R_{xx} = \begin{bmatrix} R_{x_\nu x_\nu} & R_{x_\nu x_\nu} \\ R_{x_\nu x_\nu} & R_{xx}(\bar{N}^* - 1) \end{bmatrix}$$

- Steps to implement

Step 1: $R_{xx}(\bar{N}^* - 1) = G_\phi \cdot S_x \cdot G_\phi^*$

Step 2: $A = \begin{bmatrix} R_{x_{\bar{\nu}} x_\nu} \cdot G_{-\phi}^* \cdot S_x^{-1} \\ G_\phi \end{bmatrix} = \begin{bmatrix} \tilde{\Phi}_x \\ G_\phi \end{bmatrix}$



- Example

```
>> H=(1/sqrt(.181))*[.9 1 0
0 .9 1] =
   2.1155   2.3505        0
        0   2.1155   2.3505
>> [F,L,M]=svd(H)
F =
  -0.7071  -0.7071
  -0.7071   0.7071
```

```
L =
   3.8694        0        0
        0   2.2422        0
M =
  -0.3866  -0.6671   0.6368
  -0.8161  -0.0741  -0.5731
  -0.4295   0.7412   0.5158
>> F=-F;
>> M=-M;
```

```
>> Mt=M(1:3,1:2) =
   0.3866   0.6671
   0.8161   0.0741
   0.4295  -0.7412
>> rxx=Mt*Mt' =

   0.5945   0.3649  -0.3285
   0.3649   0.6715   0.2956
  -0.3285   0.2956   0.7340
```

Generalized Cholesky Example continued

Generalized Cholesky Steps 1 and 2:

```
>> J=hankel([0 1]);

>> Gxbar=lohc(Rxx(2:3,2:3)) =
   0.7433   0.3451
        0   0.8567

>> A=[ Rxx(1,2:3)*inv(Gxbar') ; Gxbar]   =
   0.6690  -0.3834
   0.7433   0.3451
        0   0.8567

>> A*A' = (check)
   0.5945   0.3649  -0.3285
   0.3649   0.6715   0.2956
  -0.3285   0.2956   0.7340
```

Transmitter

Now, finish with triangular GDFE

```
>> [snrGDFEu, GU, WU, S0, MSWMFU, b, bbar,~] = computeGDFE(H, A, cb, Lx);

snrGDFEu =  5.5427 dB

MSWMFU =
   0.2535   0.1261
  -0.1648   0.3645

GU =
   1.0000   0.3459
        0   1.0000

b' =   1.8760   1.4186
bbar =   1.0982
```

Receiver Forward

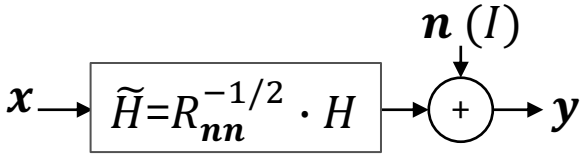
Receiver Feedback

- Generalized triangular transmitter
  - The Cholesky part is easier if the R_{uu} already exists
- Elimination of singularity beforehand usually is with transmit optimization (coming next)
- ComputeGDFE's non-square A input causes linear SNR to be incorrect
  - Essentially it is increasing the input energy, so may give too high value

```
>> Rf=(H*A*(2/3)*A'*H') =
   6.6667   3.3149
   3.3149   6.6667
>> Rbinv=Rf+eye(2) =
   7.6667   3.3149
   3.3149   7.6667
>> sGLE0=diag(diag(Rbinv)) =
   7.6667        0
        0   7.6667
>> snrGLEu=10*log10(det(sGLE0)^(1/(Lx))-1) =   4.6061
```

Alternate GLE Calculation

Fixmod Flowchart (Fig 5.4)



Step 1 $\widetilde{H} = F \cdot \Lambda \cdot M^*$ — $M = \left[\underbrace{\boldsymbol{m}_{\bar{N}_x} \cdots \boldsymbol{m}_{\varrho_H+1}}_{\mathcal{N}_H} \quad \underbrace{\boldsymbol{m}_{\varrho_H} \cdots \boldsymbol{m}_1}_{\widetilde{M}}\right]$

Step 2 $P_{\widetilde{M}} = \widetilde{M} \cdot \widetilde{M}^*$

Step 3 $\tilde{C}_{temp} = P_{\widetilde{M}} \cdot C = \left[\underbrace{\tilde{C}}_{\varrho_H \text{ ind}} \quad \underbrace{\tilde{C}_1}_{\text{lin dep}}\right]$ (may need re-indexing)

Step 4 $\tilde{O} = \left[\tilde{C}^* \tilde{C}\right]^{-1}\left(\tilde{C}^* \tilde{C}_1\right)$

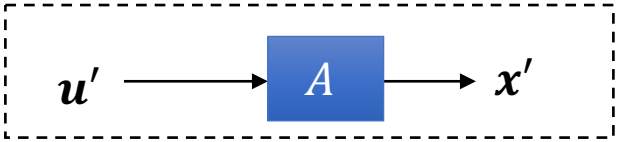
Step 5 $\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_1 & \varrho_H \text{ dimensions} \\ \boldsymbol{u}_2 & \bar{\boldsymbol{N}}_x - \varrho_H \text{ dimensions} \end{bmatrix}$ — $R_{\boldsymbol{uu}} = \begin{bmatrix} R_{22} & R_{21} \\ R_{12} & R_{11} \end{bmatrix}$

Step 6 $\widetilde{\boldsymbol{u}} = \boldsymbol{u}_2 + \tilde{O} \cdot \boldsymbol{u}_1$

Step 7 $\widetilde{\boldsymbol{x}} = P_{\widetilde{M}} \cdot \boldsymbol{x} = \tilde{C} \cdot \widetilde{\boldsymbol{u}}$

Step 8 $R_{\widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}}} = R_{22} + \tilde{O} \cdot R_{12} + \tilde{O}^* \cdot R_{21} + \tilde{O} \cdot R_{11} \cdot \tilde{O}^*$

Fixin Flowchart Fig 5.6



Given

$$\tilde{\boldsymbol{x}} = \tilde{C} \cdot \tilde{\boldsymbol{u}}$$

$$R_{\tilde{x}\tilde{x}} = \tilde{C} \cdot R_{\tilde{u}\tilde{u}} \cdot \tilde{C}^*$$

Step 1 $R_{\tilde{x}\tilde{x}} = Q \cdot \Psi \cdot Q^*$ — $Q = \left[\underbrace{\boldsymbol{q}_{\bar{N}_x} \cdots \boldsymbol{q}_{\bar{N}^*+1}}_{no\ input\ energy} \quad \underbrace{\boldsymbol{q}_{\bar{N}^*} \cdots \boldsymbol{q}_1}_{\tilde{Q}}\right]$

Step 2 $P_{\tilde{Q}} = \tilde{Q} \cdot \tilde{Q}^*$

Step 3 $\tilde{A}_{temp} = P_{\tilde{Q}} \cdot \tilde{C} = \left[\underbrace{\tilde{A}}_{\bar{N}^*\ lin\ ind} \quad \underbrace{\tilde{A}_1}_{lin\ dep}\right]$ Dimensions may need reordering

Step 4 $O_A = \left[\tilde{A}^* \cdot \tilde{A}\right]^{-1}\left(\tilde{A}^* \cdot \tilde{A}_1\right)$

Step 5 $\tilde{\boldsymbol{u}} = \begin{bmatrix} \tilde{\boldsymbol{u}}_2 & \bar{N}^* \text{dimensions} \\ \tilde{\boldsymbol{u}}_1 & \bar{\boldsymbol{N}}_x - \bar{N}^* \text{ dimensions} \end{bmatrix}$ — $R_{\tilde{u}\tilde{u}} = \begin{bmatrix} \tilde{R}_{22} & \tilde{R}_{21} \\ \tilde{R}_{12} & \tilde{R}_{11} \end{bmatrix}$

Step 6 $\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{u}}_1 + O_A \cdot \tilde{\boldsymbol{u}}_2$

Step 7 $\boldsymbol{x}' = P_{\tilde{Q}} \cdot \tilde{\boldsymbol{x}} = P_{\tilde{Q}} \cdot P_{\tilde{M}} \cdot \boldsymbol{x} = A \cdot \boldsymbol{u}'$

Step 8 $R_{\boldsymbol{u}'\boldsymbol{u}'} = \tilde{R}_{22} + O_A \cdot \tilde{R}_{12} + \tilde{R}_{21} \cdot O_A^* + O_A \cdot \tilde{R}_{11} \cdot O_A^*$