STANFORD

Lecture 12

Generalized Decision Feedback

May 17, 2023

JOHN M. CIOFFI

Hitachi Professor Emeritus of Engineering

Instructor EE392AA – Spring 2023

Announcements & Agenda

- Announcements
  - PS6 due May 24
    - Last regular homework (PS7 is double and due at course end)
  - PS5 due today
  - Return to single-user for 2 lectures (L12, L13)
  - Build some simple insights
  - Projects?

- Problem Set 6 = PS6 (due **May 24**)
  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

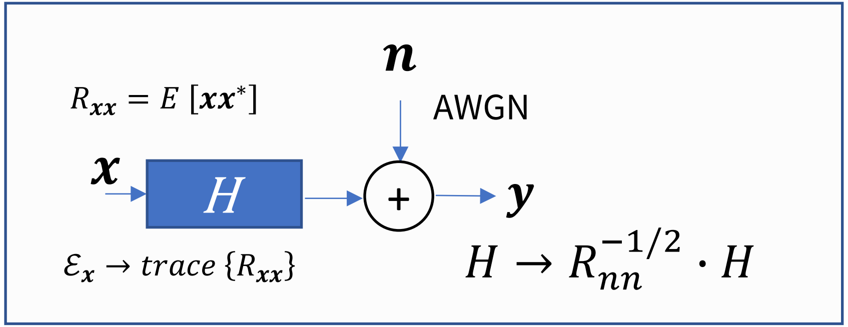
- Agenda
  - Channel Singularity
  - Input Singularity
  - MMSE GDFE
  - Specific Forms (VC and Triangular)

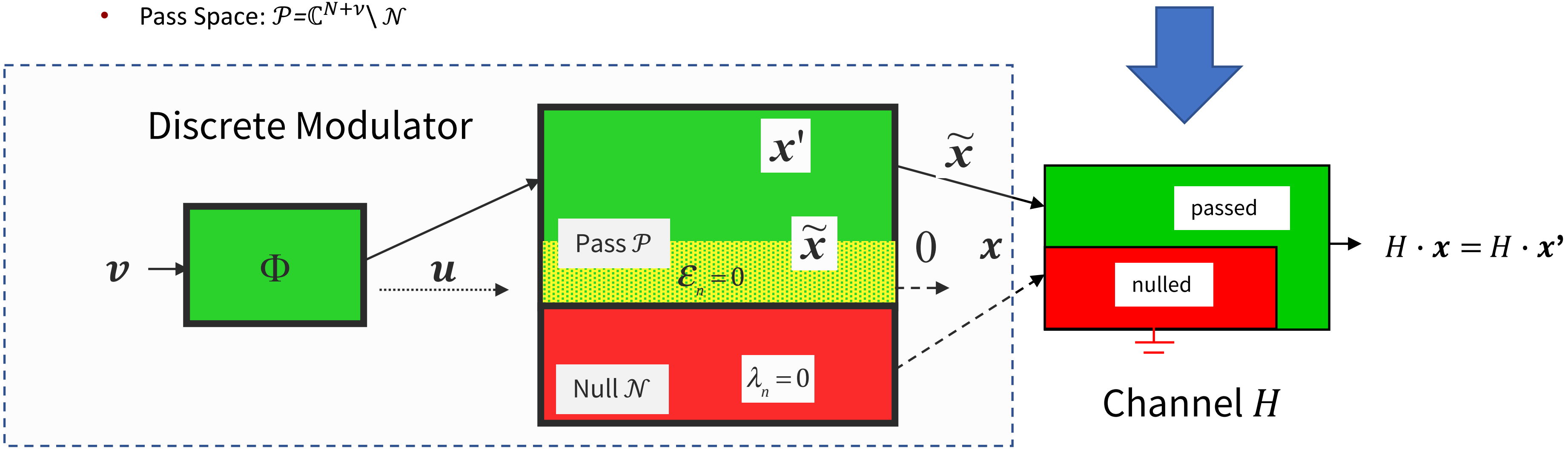
Channel Singularity

Section 5.1.1.1

Matrix AWGN Channel – *finite-length* symbol

- Treat as single-user here
  - Whiten the noise with equivalent channel $R_{\boldsymbol{nn}}$
  - Finite-length block (symbol) of input samples
    - Dimensions are subsymbols
- All energy should go in channel pass space
  - Xmit energy in channel null space $\mathcal{N}=\{\boldsymbol{x} \mid H \cdot \boldsymbol{x} = 0\}$ 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)

Eliminate Channel null space (Sec 5.1.1.1)

- discrete modulator
  - Which may insert energy into channel's null space
  - C's columns don't have to be "orthonormal"
  - Sum over dimensions (here shown as temporal)

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

$$\boldsymbol{x} = \underbrace{\begin{bmatrix} \mathbf{c}_{N+\nu-1} & \dots & \mathbf{c}_1 & \mathbf{c}_0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} u_{N+\nu-1} \\ \vdots \\ u_1 \\ u_0 \end{bmatrix}}_{\mathbf{u}}$$

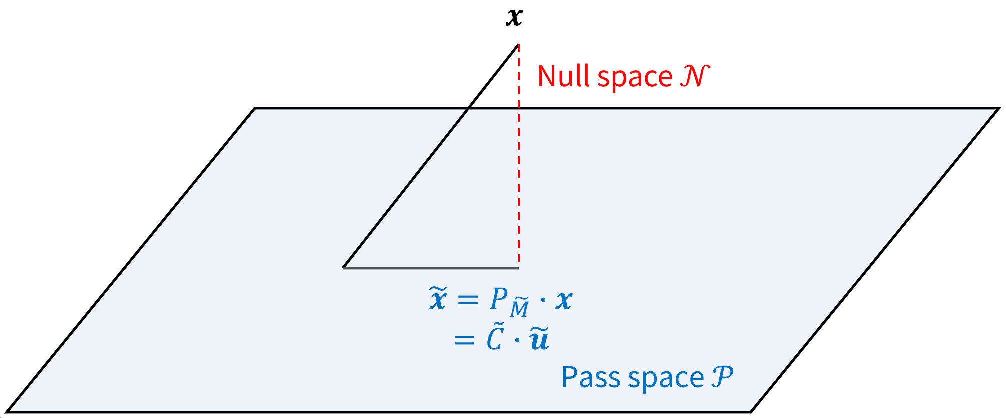
- Find equivalent discrete modulator $\tilde{C}$
  - That corresponds only to the pass space,
  - & also the components of $\boldsymbol{x}$ on these new vectors, $\tilde{\boldsymbol{u}}$.

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

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

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

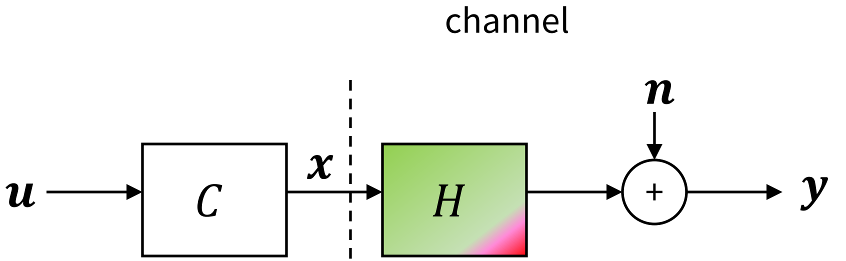
$$R_{\boldsymbol{xx}} \neq R_{\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}}$$



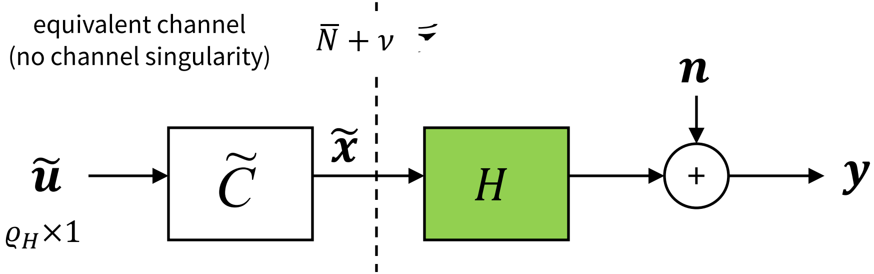
See also Figure 5.4 Flowchart

Equivalence

- The original discrete modulator:



- 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 avoid energy waste on codeword separation in channel null space $\mathcal{N}$?
  - Because, if $\boldsymbol{x}_A \in \mathcal{N}$ and $\boldsymbol{x}_B (\neq \boldsymbol{x}_A) \in \mathcal{N}$, then $d_{AB} = \|H \cdot \boldsymbol{x}_B - H \cdot \boldsymbol{x}_A\|$ =0 and creates ``coin flip'' decision.
  - Side comment: all active MAC users need 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
---------------------------------------------------------------------------
```

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

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

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

```
>> 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 remove corresponding channel modes
  - See also Figure 5.6 Flowchart

See 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
--------------------------------------------------------------------------
```

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

- The input after first reduction of channel null-space components is rank 2, 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
```

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

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

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
>>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=[ 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, 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
```

- Really a scalar channel
- After matched matrix filter

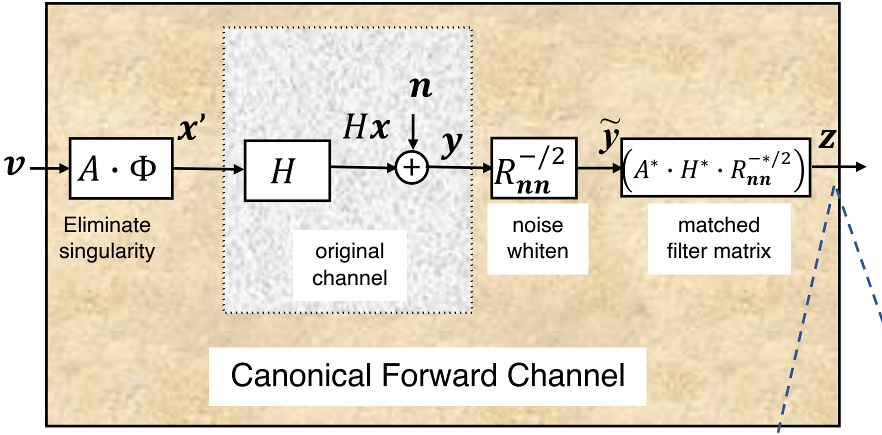
$$z = A^* \cdot H^* \cdot \boldsymbol{y} = 3 \cdot u + n'$$

```
>> [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*At  =
   1.2247
   1.2247
>> At'*H'*H*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
```

MMSE-GDFE

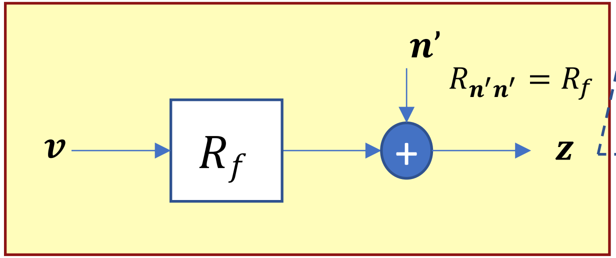
Section 5.1

The Canonical Channels -Refresher



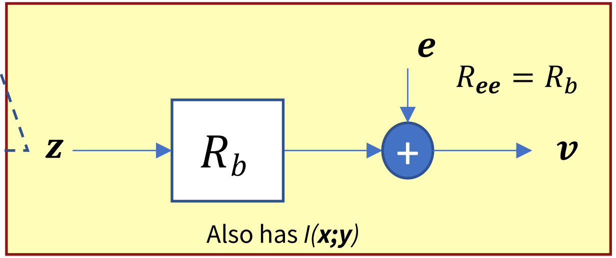
- $\boldsymbol{v}$ and $\boldsymbol{z}$ have same dimensionality, why?
- Non-singular (all dimensions now pass)
- $\mathcal{I}(\boldsymbol{v};\boldsymbol{z}) = \mathcal{I}(\boldsymbol{x};\boldsymbol{y})$
  - All 1-to-1 mappings
  - or only zero-information dimensions eliminated
- R_f is the MMSE Filter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

Forward Canonical Channel



Ok, but needs (complex) ML detector

Backward Canonical Channel



(duality of mutual information)

dimensional detection on G

- Inverse of R_b

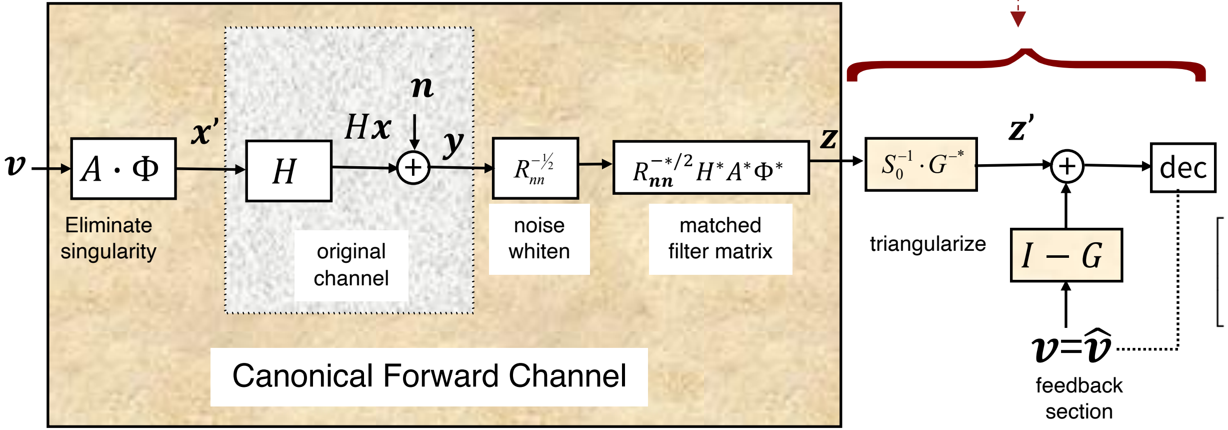
$$R_b^{-1} = R_f + I = G^* \cdot S_0 \cdot G$$

lower *diag* *upper*

Cholesky Factorization

- Generalized Decision Feedback Equalizer

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



$$z' = G \cdot v - e'$$

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

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

solve by back substitution

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

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

Bias applies as always to each dimension

- Overall (geometric) SNR

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

- Because $R_{vv}=I$ – that is, the input $\boldsymbol{v}$ is "white."
- The designer can make $\boldsymbol{v}$ white (independent) ……
- Or it might have been white input to begin with
  - 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 $R_{\boldsymbol{uu}}$ so that

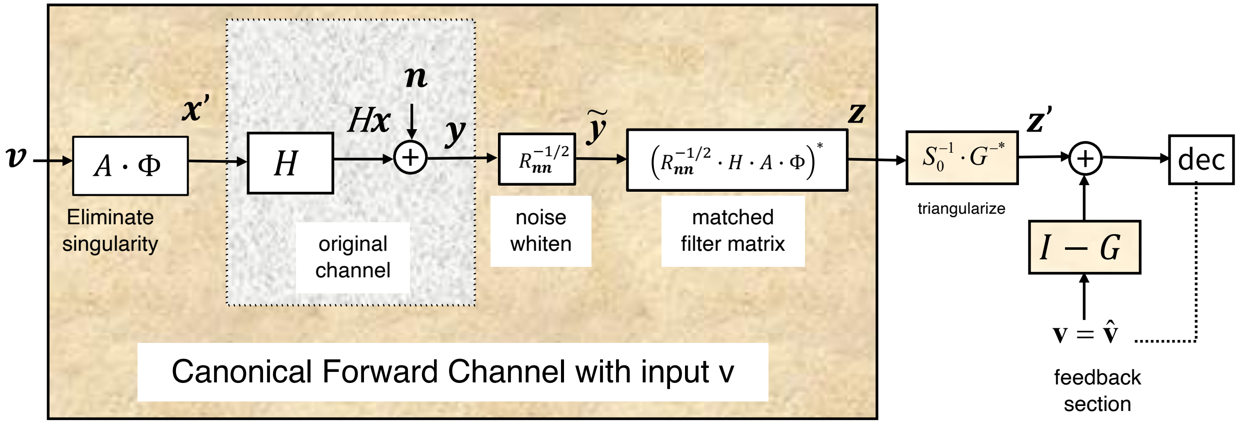
$$\boldsymbol{u} = \Phi \cdot \boldsymbol{v}$$

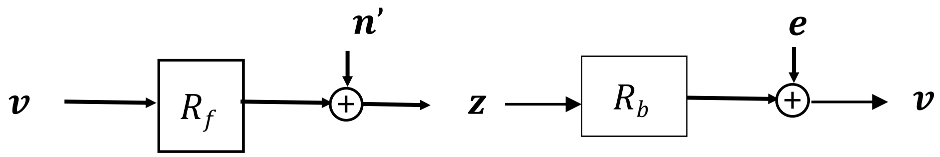
- And $R_{\boldsymbol{vv}}=I$
  - Eigen-decomposition works
  - So does Cholesky
  - So do many other factorizations (infinite number)

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

- 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 – does not need (full) ML detector

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
```

$$\tilde{\boldsymbol{u}} = \boldsymbol{u}_1 + \tilde{O} \cdot u_2$$

```
>> [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*Gx*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, Gxbar,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)
```

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

```
>> 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
```

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
[snrGDFEu, GU, WU, S0, MSWMFU, b, bbar] =
computeGDFE(H, (1.5)*Rxxtt,cb,Lx);
produces same
```

- 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 (see Yun's rq program at web site)

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

- $R_{ZF} = S_{ZF} \cdot G_{ZF}$ so G_{ZF} defines the precoder or feedback section

- Not necessarily canonical except in special cases (no crosstalk/ISI and worst-case noise)
  - Pretty close to MMSE-GDFE in many cases
  - Simpler to design (essentially just 1 QR factorization)

Generalized Linear Equalizer

- This is just MMSE linear estimate for any user.
- $\mathbb{E}[R_{\boldsymbol{ee}}] = R_b^{-1}$ and so the MMSE for linear equalizer are the diagonal elements of R_b^{-1}
- The computeGDFE program has an optional last input that will provide the corresponding unbiased 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 bit distribution is $\frac{1}{2} \cdot \log_2\left(diag\left\{R_b^{-1}\right\}\right)$ in bits for each real dimension at output

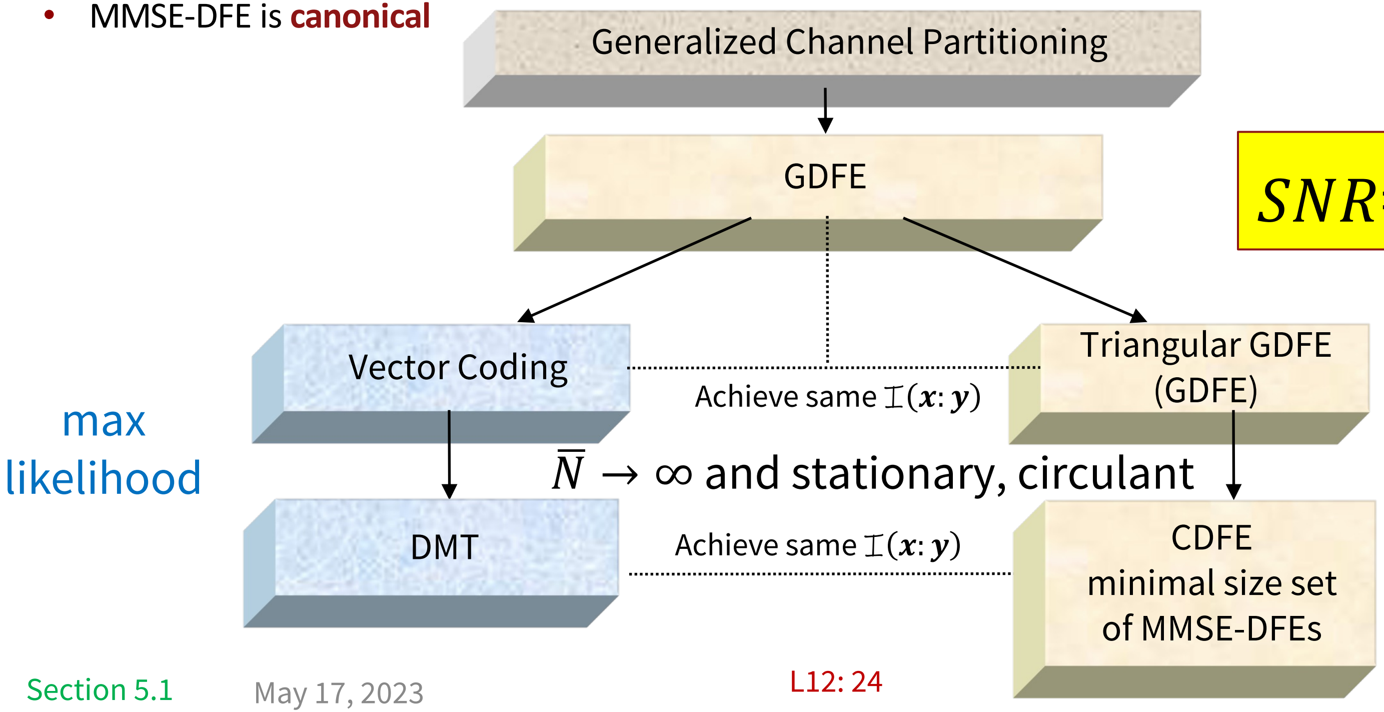
```
>> [snrGDFEu, GU, WU, S0, MSWMFU, b, bbar , snrLE] =
computeGDFE(H, sqrt(1.5)*Rxxtt,2,3)
snrLE =

 4.7453 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 $\tilde{b} \leq \tilde{\mathcal{C}}$.
  - Even though, it is not (usually) an ML detector (notice error propagation not an issue if $P_e \rightarrow 0$)
- CDEF Result extension, orginally for infinite-length (set of) MMSE-DFE(s) - 1994
  - MMSE-DFE is **canonical**



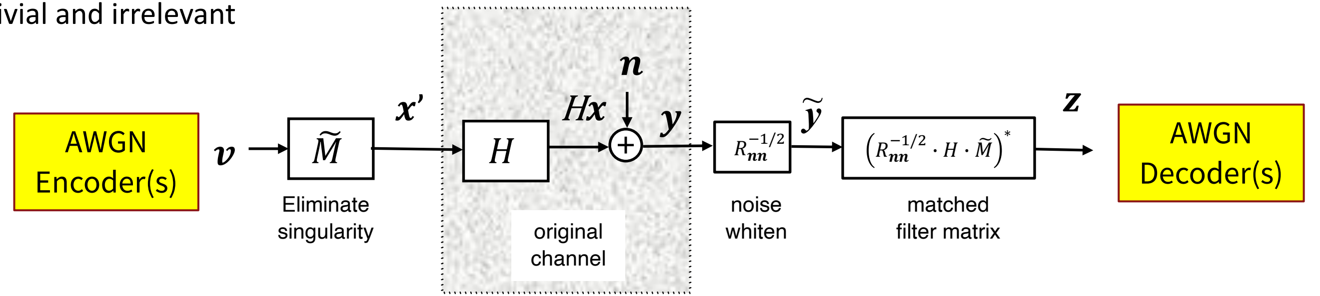
$$SNR = 2^{2\bar{\mathbb{I}}(\boldsymbol{x}:\boldsymbol{y})} - 1$$

Vector Coding Special Case

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

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 $A = [\boldsymbol{m}_{\wp_x} \quad \cdots \quad \boldsymbol{m}_0]$; $\tilde{H} = R_{\boldsymbol{nn}}^{-1/2} \cdot H \cdot \tilde{M}$
- Forward channel simplifies to $R_f = \tilde{H}^* \cdot \tilde{H} = Diag\{\lambda_{\wp_x - 1}^2, \ldots, \lambda_0^2\}$,
  - so $R_b^{-1} = Diag\{\lambda_{\wp_x - 1}^2 + 1 \quad , \cdots , \quad \lambda_0^2 + 1\} = S_0$
- Trivially , $G = I$, so no feedback needed (no error prop) and GDFE detector is ML
  - Which involves ML decoder on each dimension independently acting on codewords of possibly many symbols
  - Bias trivial and irrelevant



Vector Coding

- Works for any dimensionality
  - Vector DMT/OFDM – space-time-H SVD for each tone
- Water-fill allows $\bar{\mathcal{I}}(\boldsymbol{x}:\boldsymbol{y}) \rightarrow \bar{\mathcal{C}}$
- 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)
  - Dimensional separation may facilitate multiplexing of signals

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

$$R_{\boldsymbol{uu}} = \Phi \cdot \Phi^* = G_\phi \cdot S_x \cdot G_\phi^*$$

G_ϕ is monic upper triangular (Cholesky[16])

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

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

[16] use lohc.m command

Circulant DFE

- 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 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 $\bar{N} \rightarrow \infty$; 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,~] = 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
```

- 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,~] = 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
```

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] = 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
```

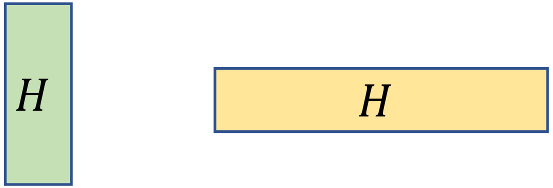
Outperforms CDFE
Same as VC w.r.t. DMT

- Better!
- No clear convergence though

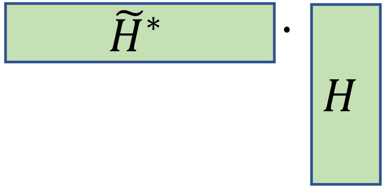
Try running computeGDFE to implement Vector Coding – what Xmit? (hint this is easy with this white input)

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



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

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

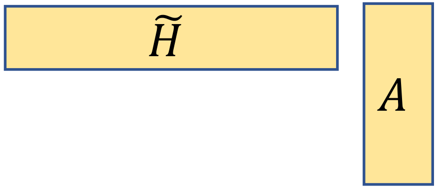
- Then R_b is also almost diagonal, like R_f

- $G = I \rightarrow$ canonical with no feedback!
  - Linear is sufficient and returns to ML

Massive MIMO Downlink continued

- Fat case (downlink) – 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

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



- THEN $R_b = A^* \cdot \tilde{H}^* \cdot \tilde{H} \cdot A + I$ is diagonal dominant
  - Law of large numbers again
- Here $R_{vv} = I$ by design, then and again there is no feedback
- **$G = I$ → canonical with no feedback!**

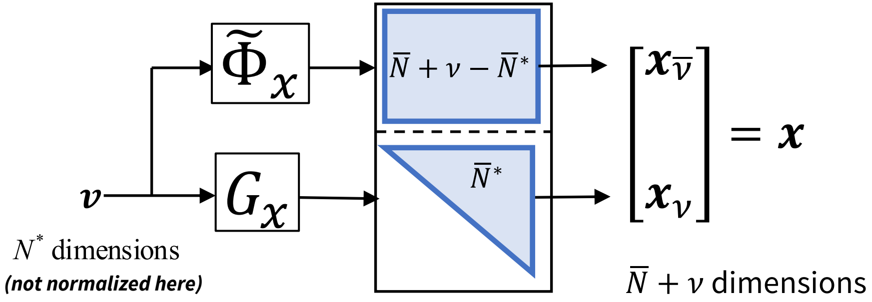
Again: no precoder (linear is sufficient)

Wireless: use many antennas at base

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

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

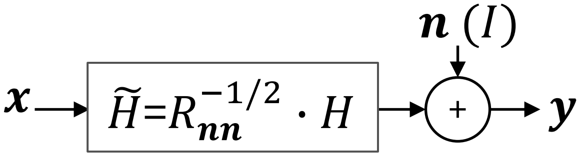
Receiver Feedback

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

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

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

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

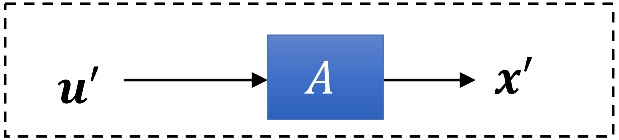
Step 5 $\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_1 & \varrho_H \text{ dimensions} \\ \boldsymbol{u}_2 & \bar{\boldsymbol{N}}_x - \varrho_H \text{ dimensions} \end{bmatrix}$ — $R_{\boldsymbol{uu}} = \begin{bmatrix} R_{22} & R_{21} \\ R_{12} & R_{11} \end{bmatrix}$

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

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

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

Fixin Flowchart Fig 5.6



Given

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

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

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

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

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

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

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

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