STANFORD

Lecture 17

BC Design and the Central IC

May 30, 2024

JOHN M. CIOFFI

Hitachi Professor Emeritus of Engineering

Instructor EE379B – Spring 2024

Announcements & Agenda

- Announcements
  - PS7 – last normal homework is extended to June 4.
  - Section 5.6
  - PS7 solutions will be distributed early on June 5.
  - Final is 25 hours, starting Friday afternoon (email from Helen Niu)

- Agenda
  - BC discrete design process
  - Matlab design-process review & Capacity Region construction
  - IC Review
  - CIC Optimization

BC Discrete Design Process

Order Reversal in Duality

- Semantics – alternate dual definition $\tilde{H}_{dual} = \left(\mathcal{J}_y \cdot \tilde{H} \cdot \mathcal{J}_x\right)^* = \mathcal{J}_x \cdot \tilde{H}^* \cdot \mathcal{J}_y$.

$$\mathcal{J}_y \triangleq \begin{bmatrix} 0 & 0 & I_{L_{y,U}} \\ 0 & \iddots & 0 \\ I_{L_{y,1}} & 0 & 0 \end{bmatrix}$$

  - Duality maps a single MAC output corresponds to a single input on BC.
  - $\mathcal{J}_y$ re-indexes dimensions (not users); but **no-$\mathcal{J}_y$-use** maps easily to matlab's usual indexing.
  - The $\mathcal{J}_y$ use simply makes the math look correct and symmetric, but confuses matlab programming.

- All information, SVD, energy, etc are still preserved, as also true without $\mathcal{J}_y$ in $\tilde{H}_{dual}$.

- The mac2bc and bc2mac programs basically handle $\mathcal{J}_y$ tacitly in reordering outputs, or inputs respectively.
  - L16's Hbc=conj(permute**(** Hmac(:,:,**end:-1:1**) **, [2 1 3])**) for $N = 1$; *command presumes* Hbc*'s input has dimension 1 at top.*
  - This tacitly multiplies by $\mathcal{J}_y$.

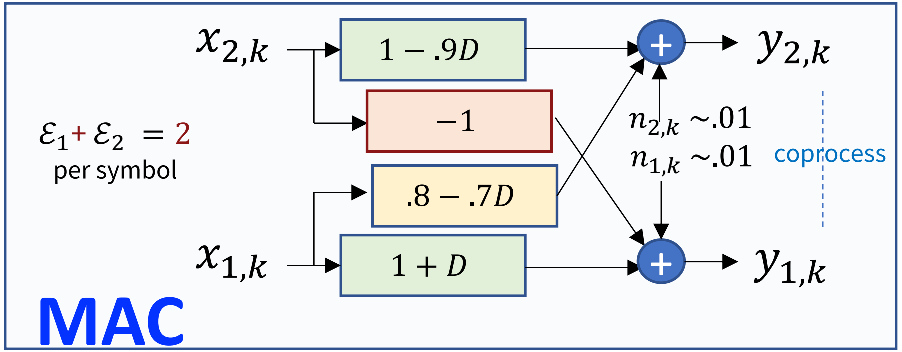
- Math Example:

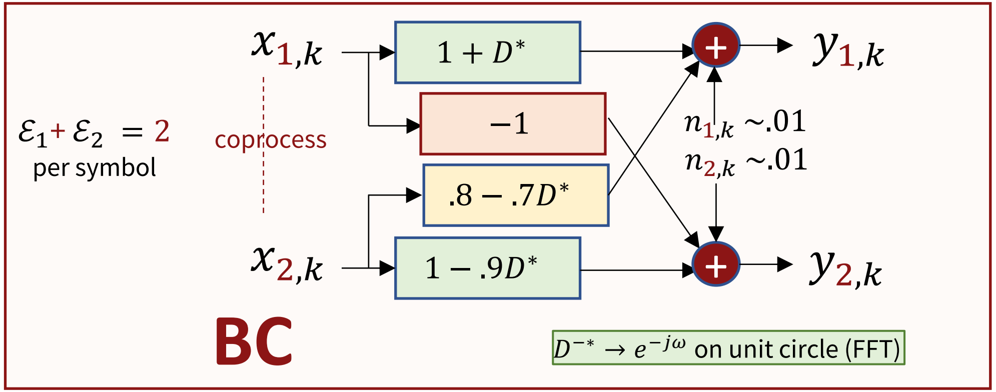
$$H_{dual}(D) = \left(\mathcal{J}_y \cdot \underbrace{\begin{bmatrix} 1-.9D & .8-.7D \\ -1 & 1+D \end{bmatrix}}_{H(D)} \cdot \mathcal{J}_x\right)^* = \begin{bmatrix} 1+D^* & .8-.7D^* \\ -1 & 1-.9D^* \end{bmatrix}$$

$D^* \rightarrow e^{j\omega}$ on unit circle (FFT)

- Essentially:
  - User direct user [magnitudes, negated phase] remain the same, but priority reverses.
  - Crosstalk flow "flips-filter" from transfer of $u \rightarrow u'$ on original to $u \leftarrow u'$ (with negated phase).

Dual Channels





- Coordination occurs on the other side.
  - **BC allows each user input to be 2D (2 sub-users/dim each).**

Almost same channel (D^*); but de/pre-coding order (priority) reverses

```
h = cat(3, [1  .8 ; -1   1], [-.9  -.7 ; 0  1] )*10;
H = fft(h, 8, 3);       % (the matlab FFT increases energy)

Hbc=zeros(2,2,8);
for n=1:8  % note N>1, so the permute command not applicable
Hbc(:,:,n)=H(:,end:-1:1,n)';  % note no Jy used here
end

>> for n=1:8
rho(n)=rank(H(:,:,n));
end
>> rho =   2   2   2   2   2   2   2   2
```

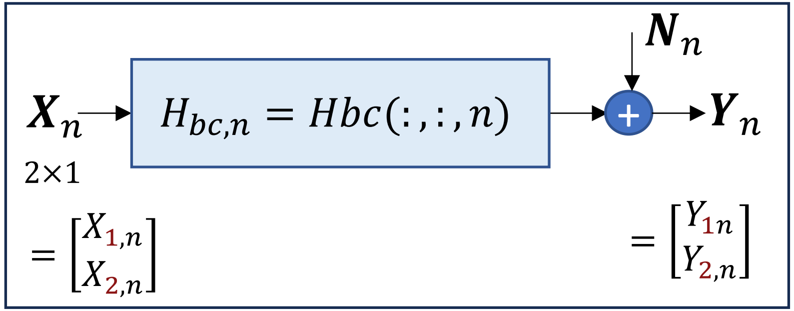
All tones have full rank.
(worst-case noise applies easily, but this design produces all Rxxb (u).)

With preliminaries now set ….

- Table

Hbc(:,:,1) =	Hbc(:,:,5) =
1.0000 + 0.0000i 20.0000 + 0.0000i	15.0000 + 0.0000i 0.0000 + 0.0000i
1.0000 + 0.0000i -10.0000 + 0.0000i	19.0000 + 0.0000i -10.0000 + 0.0000i
Hbc(:,:,2) =	Hbc(:,:,6) =
3.0503 - 4.9497i 17.0711 + 7.0711i	12.9497 + 4.9497i 2.9289 - 7.0711i
3.6360 - 6.3640i -10.0000 + 0.0000i	16.3640 + 6.3640i -10.0000 + 0.0000i
Hbc(:,:,3) =	Hbc(:,:,7) =
8.0000 - 7.0000i 10.0000 +10.0000i	8.0000 + 7.0000i 10.0000 -10.0000i
10.0000 - 9.0000i -10.0000 + 0.0000i	10.0000 + 9.0000i -10.0000 + 0.0000i
Hbc(:,:,4) =	Hbc(:,:,8) =
12.9497 - 4.9497i 2.9289 + 7.0711i	3.0503 + 4.9497i 17.0711 - 7.0711i
16.3640 - 6.3640i -10.0000 + 0.0000i	3.6360 + 6.3640i -10.0000 + 0.0000i

Now a BC, but still *there are* 8 tonal 2×2 channels.



- Design uses duality and the Rxxb = mac2bc(Rxxm , H) program.

- With appropriate tensors, this I/O set can be repeated for each tone $n = 1, \dots, 8$.

```
Rxxm=zeros(1,1,2);
Rxxm(1,1,:)=[8/9 8/9];
% same all n; thus, 3D tensor
   sufficient on this channel.

Rxxb=zeros(2,2,2,8); % 4D tensor
bbc=zeros(2,8);
```

The Rxxb's for the dual

```
for n=1:8
Rxxb(:,:,:,n)=mac2bc(Rxxm, reshape(H(:,:,n),2,1,2));
Hbc(:,:,n)=H(:,end:-1:1,n)';   % note N>1, so the permute command not applicable.
bbc(1,n)=real(log2(1+Hbc(1,:,n)*Rxxb(:,:,1,n)*Hbc(1,:,n)'));
bbc(2,n)=real(log2((1+Hbc(2,:,n)*(Rxxb(:,:,2,n)+Rxxb(:,:,1,n))*Hbc(2,:,n)')/(1+Hbc(2,:,n)*Rxxb(:,:,1,n)*Hbc(2,:,n)')));
end
bbc
bvec=sum(bbc')
bsum=sum(bvec)
Rxxb;
```

```
Bu =
  6.5043
  3.6736
sum(Bu) =
```

```
bbc =  3.6736   7.7329   7.9256   6.8322   5.4843   6.8322   7.9256   7.7329
       6.5043   7.1048   7.9703   8.5075   8.6822   8.5075   7.9703   7.1048
bvec =  54.1393  62.3515
```
bsum = **116.4908 so then 116.4908 /9 = 12.9434 bits/tone**

Note output order reversal.

(The BC bvec is interpreted with user 2 at the bottom/right, So 62.315, whereas the MAC places it at the top/left.)

- Output
  - 16?
- 8 tones
  - 2x2 each user

```
>> Rxxb
Rxxb(:,:,1,1) =
   0.5841 + 0.0000i   0.1018 + 0.0000i
   0.1018 + 0.0000i   0.0178 + 0.0000i
Rxxb(:,:,2,1) =
   0.0116 + 0.0000i  -0.1164 + 0.0000i
  -0.1164 + 0.0000i   1.1642 + 0.0000i
Rxxb(:,:,1,2) =
   0.5712 - 0.0000i   0.2092 + 0.3682i
   0.2092 - 0.3682i   0.3139 + 0.0000i
Rxxb(:,:,2,2) =
   0.3119 - 0.0000i  -0.2111 - 0.3695i
  -0.2111 + 0.3695i   0.5807 + 0.0000i

Rxxb(:,:,1,3) =
   0.3160 - 0.0000i   0.3152 + 0.2855i
   0.3152 - 0.2855i   0.5723 + 0.0000i
Rxxb(:,:,2,3) =
   0.5729 - 0.0000i  -0.3165 - 0.2849i
  -0.3165 + 0.2849i   0.3165 + 0.0000i
Rxxb(:,:,1,4) =
   0.2184 - 0.0000i   0.3553 + 0.1402i
   0.3553 - 0.1402i   0.6681 + 0.0000i
Rxxb(:,:,2,4) =
   0.6730 - 0.0000i  -0.3572 - 0.1389i
  -0.3572 + 0.1389i   0.2183 + 0.0000i

Rxxb(:,:,1,5) =
   0.1945 + 0.0000i   0.3655 + 0.0000i
   0.3655 + 0.0000i   0.6867 + 0.0000i
Rxxb(:,:,2,5) =
   0.7021 + 0.0000i  -0.3695 + 0.0000i
  -0.3695 + 0.0000i   0.1945 + 0.0000i
Rxxb(:,:,1,6) =
   0.2184 + 0.0000i   0.3553 - 0.1402i
   0.3553 + 0.1402i   0.6681 - 0.0000i
Rxxb(:,:,2,6) =
   0.6730 + 0.0000i  -0.3572 + 0.1389i
  -0.3572 - 0.1389i   0.2183 - 0.0000i

Rxxb(:,:,1,7) =
   0.3160 + 0.0000i   0.3152 - 0.2855i
   0.3152 + 0.2855i   0.5723 - 0.0000i
Rxxb(:,:,2,7) =
   0.5729 + 0.0000i  -0.3165 + 0.2849i
  -0.3165 - 0.2849i   0.3165 - 0.0000i
Rxxb(:,:,1,8) =
   0.5712 + 0.0000i   0.2092 - 0.3682i
   0.2092 + 0.3682i   0.3139 - 0.0000i
Rxxb(:,:,2,8) =
   0.3119 + 0.0000i  -0.2111 + 0.3695i
  -0.2111 - 0.3695i   0.5807 - 0.0000i
```

Duality works whether Rxx optimum or NOT

- Duality equates two **PER-USER** mutual-information quantities:
  - $\mathcal{I}_{MAC}(u\,/[u-1,\ldots,1]) = \mathcal{I}_{BC}(u)$.
  - $\mathcal{I}_{BC}(u) = \log_2\left(\frac{\left|I+\sum_i^{u} H_u^* \cdot R_{\overline{xx}}(i) \cdot H_u\right|}{\left|I+\sum_i^{u-1} H_u^* \cdot R_{\overline{xx}}(i) \cdot H_u\right|}\right)$ - this expression is only exact for H_u (not for H); it is not the chain rule.
  - It still corresponds to a MMSE estimator/decoder, implemented with lossless precoder, for user u.

- Chapter 2's worst-case noise finds an all-user BC (no receiver coordination) from MMSE-GDFE.

```
Rxxbsum=zeros(2,2,8);
 for n=1:8
   Rxxbsum(:,:,n)=Rxxb(:,:,1,n)+Rxxb(:,:,2,n);
  end

Rwcn=zeros(2,2,8);
sumRate=zeros(1,8);

for n=1:8
[Rwcn(:,:,n)
sumRate(n)]=wcnoise(Rxxbsum(:,:,n),Hbc(:,:,n),1);
end  % alternative to chain-rule for BC
```

sumRate=2*real(sumRate) =
10.2036   14.8377   15.8959   15.3396   14.1665   15.3396   15.8959   14.8377

>> sum(sumRate) = 116.5166 > 116.4908 % some secondary user "freeloading"

Loss with respect to single-user
10*log10((2^(**116.6695** /9)-1) / (2^(116.4908 /9)-1)) = **0.06 dB**

Best BC rate sum, so with optimized input, is

[Rxx, Rwcn, bmax]=bcmax(Rxxbsum, Hbc, 1); output is bits/real-dimension
>> 2*bmax = **116.5892** > 116.4908 , but of course less than **116.6695**

Matches macmax (L16:31, as it should)

size(Rwcn) = 2 2 8
size(Rxx) = 2 2 8 % bcmax provides the wcn and the best Rxx (sum over users) for BC

mu_bc.m – Precoder, Bloc-diag rcvr, given A

- The dual-BC design achieves the original MAC's target rates.
  - Dual-BC design does not use WCN.
  - Design U GDFEs (precoders) for each of the $\bar{N}$ tones (mu_bc program from Chapter 2, but now with $\bar{N}$ tones) & MSWMF, for $\{R_{xx}(u)\}$; any square root for each (AU)

```
function [Bu, GU, S0, MSWMFunb , B] = mu_bc(H, AU, Lyu , cb)
--------------------------------------------------------------------------
 Inputs: Hu, AU , Usize, cb
 Outputs: Bu, Gunb, Wunb, S0, MSWMFunb

 H: noise-whitened BC matrix [H1 ; ... ; HU] (with actual noise, not wcn)
   sum-Ly x Lx x N
 AU: Block-row square-root discrete modulators, [A1 ... AU]
    Lx  x (U * Lx) x N
 Lyu: # of (output, Lyu) 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 precoder matrices: (Lx U) x (Lx U) x N
    For each of U users, this is Lx x Lx matrix on each tone
 S0: sub-channel dimensional channel SNRs: (Lx U) x (Lx U) x N
 MSWMFunb: users' unbiased diagonal mean-squared whitened matched matrices
   For each of U cells and Ntones, this is an Lx x Lyu  matrix
 Bu - users bits/symbol 1 x U
   the user should recompute SNR if there is a cyclic prefix
 B - the user bit distributions (U x N) in cell array
```

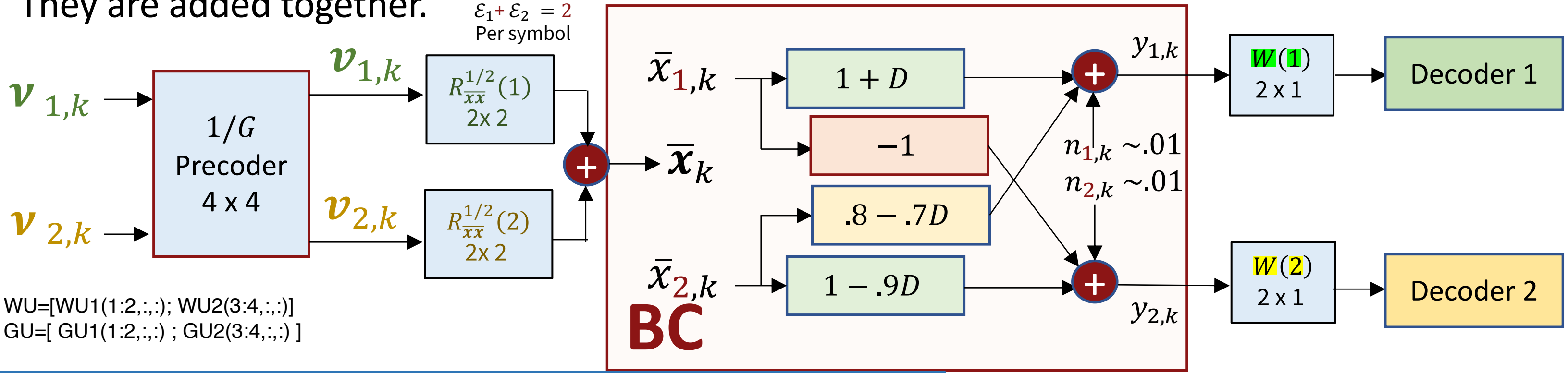
Bu =
6.5043
3.6736
sum(Bu) =

from Dual-MAC

```
AU=zeros(2,4,8);
for n=1:8
AU(:,:,n)=[ sqrtm(Rxxb(:,:,1,n))  sqrtm(Rxxb(:,:,2,n)) ];
end

[Bu, Gunb, S0, MSWMFunb, B] = mu_bc(Hbc, AU, [1 1] , 1);
Bu =   54.1393   62.3515

>> B =  2 × 8 cell array
   {[3.6736]}  {[7.7329]}  {[7.9256]}  {[6.8322]}  {[5.4843]}  {[6.8322]}  {[7.9256]}  {[7.7329]}
   {[6.5043]}  {[7.1048]}  {[7.9703]}  {[8.5075]}  {[8.6822]}  {[8.5075]}  {[7.9703]}  {[7.1048]}

sum(Bu) =  116.4908   (checks)

GU=zeros(4,4,8);
for n=1:8
GU(:,:,n)=[Gunb{1,n} ; Gunb{2,n}];  % convert to matrix form
end

MSWMFU=zeros(4,1,8);
for n=1:8
MSWMFU(:,:,n)=[MSWMFunb{1,n} ; MSWMFunb{2,n}]; % convert to matrix form
end
```

Precoders and Filters

- The transmit filters are the square-root matrices $R_{\boldsymbol{xx}}^{1/2}(u)$, which are 2-dimensional for EACH user.
- They are added together.



WU=[WU1(1:2,:,:); WU2(3:4,:,:)]
GU=[GU1(1:2,:,:) ; GU2(3:4,:,:)]

GU(:,:,1) =

1.0000 + 0.0000i	0.1743 + 0.0000i	-0.6324 + 0.0000i	6.3243 + 0.0000i
0.0000 + 0.0000i	1.0000 + 0.0000i	-3.6274 + 0.0000i	36.2743 + 0.0000i
0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i	-10.0000 + 0.0000i
0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i

GU(:,:,2) =

1.0000 + 0.0000i	0.3662 + 0.6445i	-0.5488 - 0.1177i	0.2320 + 0.7298i
0.0000 + 0.0000i	1.0000 + 0.0000i	-0.5038 + 0.5653i	1.0106 + 0.2142i
0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i	-0.6768 - 1.1846i
0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i

GU(:,:,3) =

1.0000 + 0.0000i	0.9974 + 0.9035i	-0.0513 - 0.5181i	-0.2293 + 0.3117i
0.0000 + 0.0000i	1.0000 + 0.0000i	-0.2867 - 0.2598i	0.0292 + 0.2861i
0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i	-0.5525 - 0.4972i
0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i

GU(:,:,4) =

1.0000 + 0.0000i	1.6268 + 0.6419i	1.0890 - 1.3469i	-0.8561 + 0.4902i
0.0000 + 0.0000i	1.0000 + 0.0000i	0.2965 - 0.9450i	-0.3525 + 0.4404i
0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i	-0.5308 - 0.2064i
0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i

GU(:,:,5) =

1.0000 + 0.0000i	1.8789 + 0.0000i	3.5784 + 0.0000i	-1.8834 + 0.0000i
0.0000 + 0.0000i	1.0000 + 0.0000i	1.9046 + 0.0000i	-1.0024 + 0.0000i
0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i	-0.5263 + 0.0000i
0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i

GU(:,:,6) =

1.0000 + 0.0000i	1.6268 - 0.6419i	1.0890 + 1.3469i	-0.8561 - 0.4902i
0.0000 + 0.0000i	1.0000 + 0.0000i	0.2965 + 0.9450i	-0.3525 - 0.4404i
0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i	-0.5308 + 0.2064i
0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i

GU(:,:,7) =

1.0000 + 0.0000i	0.9974 - 0.9035i	-0.0513 + 0.5181i	-0.2293 - 0.3117i
0.0000 + 0.0000i	1.0000 + 0.0000i	-0.2867 + 0.2598i	0.0292 - 0.2861i
0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i	-0.5525 + 0.4972i
0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i

GU(:,:,8) =

1.0000 + 0.0000i	0.3662 - 0.6445i	-0.5488 + 0.1177i	0.2320 - 0.7298i
0.0000 + 0.0000i	1.0000 + 0.0000i	-0.5038 - 0.5653i	1.0106 - 0.2142i
0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i	-0.6768 + 1.1846i
0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	1.0000 + 0.0000i

MSWMFU(:,:,1) =
MSWMFU(:,:,1) =

0.2960 + 0.0000i
1.6978 + 0.0000i
0.9222 + 0.0000i
-0.0922 + 0.0000i

MSWMFU(:,:,2) =

0.0616 + 0.0594i
0.1107 - 0.0327i
0.0716 + 0.1254i
-0.1058 + 0.0000i

MSWMFU(:,:,3) =

0.1051 + 0.0236i
0.0697 - 0.0395i
0.0586 + 0.0527i
-0.1060 + 0.0000i

MSWMFU(:,:,4) =

0.1855 - 0.0390i
0.0905 - 0.0597i
0.0562 + 0.0219i
-0.1059 + 0.0000i

MSWMFU(:,:,5) =

0.3217 + 0.0000i
0.1712 + 0.0000i
0.0556 + 0.0000i
-0.1056 + 0.0000i

MSWMFU(:,:,6) =

0.1855 + 0.0390i
0.0905 + 0.0597i
0.0562 - 0.0219i
-0.1059 - 0.0000i

MSWMFU(:,:,7) =

0.1051 - 0.0236i
0.0697 + 0.0395i
0.0586 - 0.0527i
-0.1060 - 0.0000i

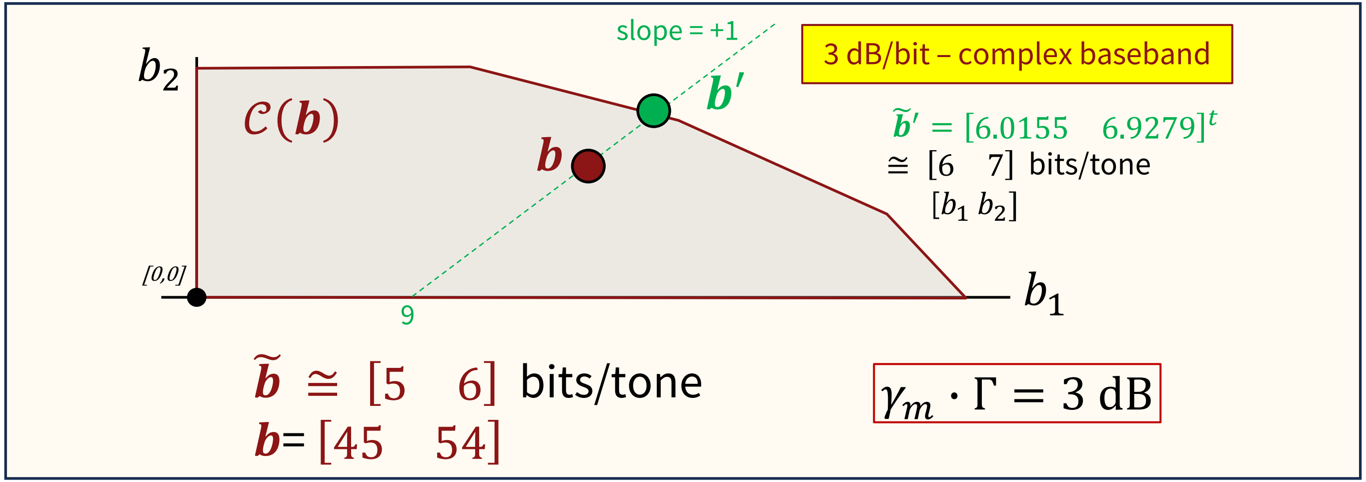
MSWMFU(:,:,8) =

0.0616 - 0.0594i
0.1107 + 0.0327i
0.0716 - 0.1254i
-0.1058 - 0.0000i

Design with Margin

- The rate vector $\boldsymbol{b}' = [54.1393 \quad 62.3515]^t \in \mathcal{C}(\boldsymbol{b})$'s boundary , and thus requires a $\Gamma = 0$ dB code.

- Use a code with $\Gamma = 1.5$ dB and leave 1.5 dB margin.



- The design (GDFE/precoder …) we just found with $\Gamma = 1.5$ dB code and 5 bits/Hz on user 2 and 6 bits/Hz on user 1 has 1.5 dB margin.
- This design works for any $\Gamma \cdot \gamma_m$= 3 dB if the energy for the point $\boldsymbol{b}'$ is used, but operated at rate $\boldsymbol{b}$.

Use of mac2bc - reminders

- It's use of svd "econ" mode is ok and implements the duality.
  - It's already in the mac2bc and bc2mac programs.
- The BC rate sum need not be the largest for the program's output Rxxb.
- If the input Rxxm is such that it corresponds to an MAC-Esum $\mathcal{C}(\boldsymbol{b})$ boundary point, then it is best.
  - But not necessarily at interior points, like those produced by admMAC and minPMAC for almost all admissible points (since most are not on the boundary).

Some Matlab Design-Process Help (PS7) & $\mathcal{C}(b)$ construction

PS 7: $N > 1$, constant $L_x = L_{x,u} \equiv \mathfrak{L}_x / U$

- **Generate Hbc** that can be used with mu_bc program.
  - Variable $L_{x,u}$: set $L_x = \max_u L_{x,u}$ and expand each $\widetilde{H}_u$ to have L_x - $L_{x,u}$ dummy zero columns.

```
Hmac=reshape(Hmac,Ly,Lxu,U,N);   % 4D tensor from 3D tensor FFT output, which was Ly x 𝔏x x N
Hmac=Hmac(:,:,order,:)
Hbc=zeros(Lxu,Ly,U,N);   % use MAC's Lxu and Ly !
for n=1:N   % note N>1
for i=1:U
Hbc(:,:,i,n)=Hmac(:,:,U+1-i,n)';
end
end
Hbc1=permute(reshape(Hbc, [Ly , 𝔏x, N]), [ 2 1 3]); % returns to 3D tensor – reorder because Ly , 𝔏x are from MAC
```

- **Generate Rxxm** that can be used with mac2bc program.

```
Info.Rxxs  % from slower minPMACMIMO is already (Lxu, Lxu, U, N) if Lxu is constant

E=celltomat(info.Eun) % so only need this if Lxu=1 because faster minPMAC was used. % Rxxm=cell2mat(info.Rxxs)
Rxxm=zeros(1,1,U,N); % for minPMAC
for n=1:N   Rxxm(1,1,:,n)=E(u, order,n);    end  % Since Lxu=1, conversion to mac2bc format (per tone) % any u=1,…,U works

Rxxb=zeros(Ly,Ly,U,N);
bbc=zeros(U,N); % for use on next slide
for n=1:N   Rxxb(:,:,:,n)=mac2bc(Rxxm(:,:,:,n), Hmac(:,:,:,n));  end % per tone use of mac2bc 3D tensor to 3D tensor
```

HWH 7: BC Design Completion

- Data calculation below is a check, shown here for $U = 3$.

```
bbc(1,n)=real(log2(1+Hbc(:,:,1,n)*Rxxb(:,:,1,n)*Hbc(:,:,1,n)'));
bbc(2,n)=real(log2((1+Hbc(:,:,2,n)*(Rxxb(:,:,2,n)+Rxxb(:,:,1,n))*Hbc(:,:,2,n)')/(1+Hbc(:,:,2,n)*Rxxb(:,:,1,n)*Hbc(:,:,2,n)')));
bbc(3,n)=real(log2((1+Hbc(:,:,3,n)*(Rxxb(:,:,3,n)+Rxxb(:,:,2,n)+Rxxb(:,:,1,n))*Hbc(:,:,3,n)')/(1+Hbc(:,:,3,n)*(Rxxb(:,:,2,n)+Rxxb(:,:,1,n))*Hbc(:,:,3,n)')));
end
bvec=sum(bbc')
bsum=sum(bvec)
```

- Prepare for BC Design – stack the A matrices horizontally.

```
A=zeros(Ly, Ly*U, N); % 3D tensor to match Hbc1 and mu_bc.m program
for n=1:N  A(:,:,n)=[ sqrtm(Rxxb(:,:,1,n))  …..  sqrtm(Rxxb(:,:,U,n)) ];  end

[Bu, Gunb, S0, MSWMFunb, B] = mu_bc(Hbc1, A, [Lyu] , cb);
```

- **mu_bc outputs cell arrays**, so allows variable (BC) $L_{y,u}$; when constant $L_{y,u} = L_y$ can translate for display:

GU=zeros($U \cdot \mathfrak{L}_y$, $U \cdot \mathfrak{L}_y$, N); % $\mathfrak{L}_y$ corresponds to BC here
for n=1:N GU(:,:,n)=[Gunb{1,n} ; …, Gunb{U,n}]; end
MSWMFU=zeros($U \cdot \mathfrak{L}_y$, L_y, N); for n=1:N MSWMFU(:,:,n)=[MSWMFunb{1,n}; … ; MSWMFunb{U,n}]; end

- More generally GU cells are **each** $L_{y,u} \times L_{y,u}$, while MSWMFU are each $\mathfrak{L}_y \times L_{y,u}$.

64-tone - one vertex

```
[Eun, theta, bun, FEAS_FLAG, bu_a, info]=minPMAC(H64, [445 412 132]', [1 1 1]',1);
FEAS_FLAG =    2
bu_a =  445.0000     412.0000     132.0000
        %  bu_v                 Eun         bun         theta    order   frac    cID
  445.81   411.19     132   {1x3x64 }  {1x3x64 }   {1x3}    {1x3}   0.99     1
  425.68   431.32     132   {1x3x64 }  {1x3x64}   {1x3}     {1x3}    0.01     1
>> info.order{:} =
   3    2    1
   3    1    2
>> info.frac'*info.bu_v = 445.0000  412.0000  131.9999
>> info.frac' =    0.9904    0.0096
>> sum(Eun') =   65.9553   60.2757   40.4453
>> sum(Eun,'all') =  166.6763 < 3 x 64
```

Small < 1% ;Might just use vertex 1
Large/small → numerical issues

```
H64mac =reshape(H64,2,1,3,64); % add extra tensor dimension for transpose
H64bc=zeros(1,2,3,64); % note reversal of first two dimensionalities
for n=1:64
     for i=1:3   H64bc(:,:,i,n) = H64mac(:,:,4-i,n)';  end
     H64bc1=permute(reshape(H64bc, [2 , 3, 64]), [ 2 1 3]);
end
```

```
Rxxm=zeros(1,1,3,64);
for n=1:64
          E=cell2mat(info.Eun);
          Rxxm(1,1,:,n)=E(1,:,n);     end
Rxxb=zeros(2,2,3,64);
bbc=zeros(3,64);
for n=1:64   Rxxb(:,:,:,n)=mac2bc(Rxxm(:,:,:,n), H64mac(:,:,:,n)); end
A=zeros(2, 6, 64);
for n=1:64  A(:,:,n)=[ sqrtm(Rxxb(:,:,1,n)) , sqrtm(Rxxb(:,:,2,n)),  sqrtm(Rxxb(:,:,3,n)) ];  end
```

```
>> [Bu, Gunb, S0, MSWMFunb, B] = mu_bc(H64bc1, A, [1 1 1], 1);
>> Bu =  131.9999  411.7251  445.2751
Buvertex1=Bu; % save for next slide
```

checks with reverse order for vertex 1

```
GU=zeros(6, 6, 64);
MSWMFU=zeros(6,1,64);
for n=1:64   GU(:,:,n)=[Gunb{1,n} ; Gunb{2,n} ; Gunb{3,n}];
MSWMFU(:,:,n)=[MSWMFunb{1,n}; MSWMFunb{2,n}  ; MSWMFunb{3,n}];  end
```

```
>> GU(:,:,23) =   % 6 x 6
  1.0000 + 0.0000i   0.0920 - 0.3464i   8.7367 + 1.1630i  10.8139 -15.1709i  -0.6748 - 4.2623i   1.9688 + 0.6639i
  0.0000 + 0.0000i   1.0000 + 0.0000i   3.1234 +24.3936i  48.6591 +18.2925i  11.0106 - 4.8735i  -0.3797 + 5.7850i
  0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i   0.9891 - 1.8681i  -1.3553 - 0.3648i   0.4582 - 0.4967i
  0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i  -0.1475 - 0.6474i   0.3091 + 0.0816i
  0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i  -0.2233 + 0.4266i
  0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
```

```
>> MSWMFU(:,:,23) =          % 6 x 1
  0.8840 - 0.3319i
  1.5285 + 2.1461i
  0.0553 - 0.0808i
  0.0460 + 0.0052i
  0.0535 - 0.0801i
 -0.1988 - 0.0213i
```

Note the mu_bc match is not quite perfect on the 99% point

```
>> A(:,:,23) =
  0.0805 - 0.0000i   0.0074 - 0.0279i   0.2114 + 0.0000i   0.2091 - 0.3950i   0.9368 - 0.0000i  -0.2092 + 0.3996i
  0.0074 + 0.0279i   0.0103 - 0.0000i   0.2091 + 0.3950i   0.9447 + 0.0000i  -0.2092 - 0.3996i   0.2172 - 0.0000i
```

- minPMAC for large N, U tends to have numerical issues
  - CVX version (minPMACMIMO) runs much longer, can tend to be more accurate

check other vertex

- Small error in 99% on vertex share rate can require large compensation on the 1% rate.

```
H64mac =reshape(H64,2,1,3,64);
H64mac=H64mac(:,:,[2 1 3],:);  % set order for other vertex
H64bc=zeros(1,2,3,64);
for n=1:64
for i=1:3   H64bc(:,:,i,n) = H64mac(:,:,4-i,n)'; end
end
H64bc1=permute(reshape(H64bc, [2 , 3, 64]), [ 2 1 3]);

Rxxm=zeros(1,1,3,64);
E=cell2mat(info.Eun);

for n=1:64
Rxxm(1,1,:,n)=E(2,[2 1 3],n);    end
Rxxb=zeros(2,2,3,64);
bbc=zeros(3,64);
for n=1:64   Rxxb(:,:,:,n)=mac2bc(Rxxm(:,:,:,n), H64mac(:,:,:,n)); end

A=zeros(2, 6, 64);
for n=1:64  A(:,:,n)=[ sqrtm(Rxxb(:,:,1,n)) , sqrtm(Rxxb(:,:,2,n)),  sqrtm(Rxxb(:,:,3,n)) ];
end

[Bu, Gunb, S0, MSWMFunb, B] = mu_bc(H64bc1, A, [1 1 1], 1);

>> Bu =  131.9999  416.7071  440.2931
>> rate = [Buvertex1 ; Bu]
  131.9999  411.7251  445.2751
  131.9999  416.7071  440.2931
>> info.frac'*rate  =
  131.9999  411.7730  445.2270   versus 412 and 445
```

```
Check with:
for n=1:64
bbc(1,n)=real(log2(1+H64bc(:,:,1,n)*Rxxb(:,:,1,n)*H64bc(:,:,1,n)'));
bbc(2,n)=real(log2((1+H64bc(:,:,2,n)*(Rxxb(:,:,2,n)+Rxxb(:,:,1,n))*H64bc(:,:,2,n)')/(1+H64bc(:,:,2,n)*Rxxb(:,:,1,n)*H64bc(:,:,2,n)')));
bbc(3,n)=real(log2((1+H64bc(:,:,3,n)*(Rxxb(:,:,3,n)+Rxxb(:,:,2,n)+Rxxb(:,:,1,n))*H64bc(:,:,3,n)')/(1+H64bc(:,:,3,n)*(Rxxb(:,:,2,n)+Rxxb(:,:,1,n))*H64bc(:,:,3,n)')));
end
bvec=sum(bbc') =  131.9999  411.7251  445.2751
bsum=sum(bvec) =  989.0000

>> newfrac=inv(rate(1:2,2:3)')*[412 ; 445]  =
   0.9448
   0.0552
```

Better design is:
19 symbols mac2bc vertex 1 &
1 symbol mac2bc vertex 2.

```
> E(:,:,21:25)
   1.1044   1.1722   0.4032
   1.1044   1.1722   0.4032

   1.0237   1.1371   0.5177
   1.0237   1.1371   0.5177

   0.9394   1.1115   0.6258
   0.9394   1.1115   0.6258

   0.8528   1.0964   0.7249
   0.8528   1.0964   0.7249

   0.7653   1.0920   0.8132
   0.7653   1.0920   0.8132
```

Energies are from minPMAC, but then used in mac2bc.

Note equal energies both vertices' of each tone

Capacity Region

5.5.5 Generation of the Vector BC Capacity Rate Region

The steps for tracing the vector BC Capacity Region are:

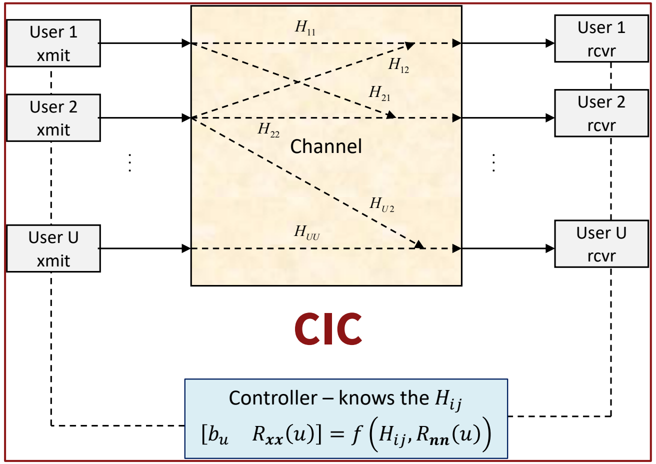
1. Create a dual vector MAC channel (with coefficients $\widetilde{H}$ and noise autocorrelation I).
2. for each $\boldsymbol{b}'$ with $b'_1 = 0, ..., b_{1,max}, \; ... \; b'_U = 0, ..., b_{U,max}$ with increments selected appropriately and maximums chosen sufficiently large to be outside the rate region (i.e., equal to the single user capacity for all other users zeroed)
   (a) Find the energy vector $\boldsymbol{\mathcal{E}}$ for a given $\boldsymbol{b}$ on the dual vector MAC using the minPMAC program of Section 5.4.
   (b) if $\sum_u \mathcal{E}_u \leq \mathcal{E}$, then the point is in the region, so $c_{new}(\boldsymbol{b}) = \{\boldsymbol{b}' \bigcup c_{old}(\boldsymbol{b})\}$.
3. Trace the boundary for of $\boldsymbol{b}$ in Step 2 for which $\sum_u \mathcal{E}_u = \mathcal{E}$.

- Check any point's admissibility on the dual MAC with admMAC

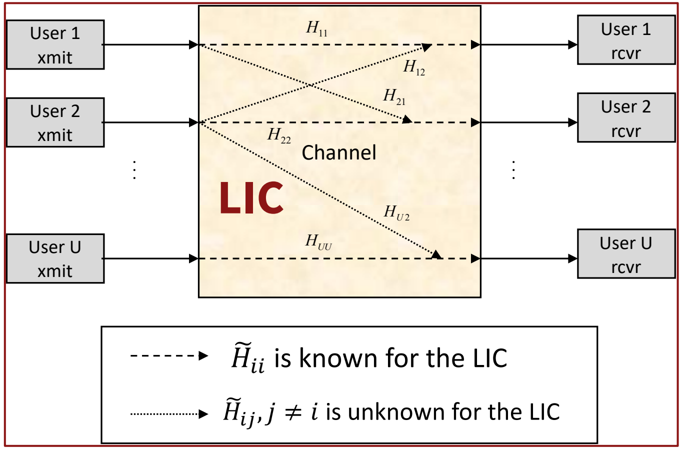
IC Review

Central or Local Control?

- **Centrally controlled IC (CIC):** controller directs users, &
  - centrally sets $b_{u,l,n}$, $\mathcal{E}_{u,l,n}$, $R_{\boldsymbol{xx}}(u,n)$, users' codes,
  - knows $H_{u\leftrightarrow u',l,n}$, $R_{\boldsymbol{nn}}(u,n)\ \forall\ u \in \boldsymbol{u}$.
  - Examples include:
    - Distributed Antenna Systems (DAS)
    - Cell-free with mobile-edge computation &
    - CIC is often associated with licensed spectra (e.g., cellular).
  - User receivers use successive decoding (GDFE).
  - Basically, this is the IC you know already.



- **Locally controlled IC (LIC):** each user transmitter can only control its own transmission.
  - Locally (transceiver u) sets $b_{u,l,n}$, $\mathcal{E}_{u,l,n}$, $R_{\boldsymbol{xx}}(u,n)$, & codes.
  - Local user knows only $H_{u\leftrightarrow u,l,n}$, $\mathcal{R}_{\boldsymbol{nn}}(u,n)$.
  - Examples include:
    - Collision detection, avoidance are popular.
    - Unlicensed spectra (Wi-Fi, Bluetooth).
  - Everyone else is noise (try to detect and remove them at your own risk).



Review MU Capacity step: Prior-User Set

- Order vector and inverse, or
  - Permutation (permutation matrix J), etc

$$\boldsymbol{\pi}_u = \begin{bmatrix} \pi(U') \\ \vdots \\ \pi(1) \end{bmatrix} \quad \boldsymbol{\pi}_u^{-1} = \begin{bmatrix} U' \\ \vdots \\ 1 \end{bmatrix} \qquad j = \pi(i) \rightarrow i = \pi^{-1}(j).$$

- Prior-User Set is $\mathbb{P}_u(\boldsymbol{\pi}) = \{j \mid \boldsymbol{\pi}^{-1}(j) < \boldsymbol{\pi}^{-1}(u)\}$.
  - That is "all the users before the desired user u in the given order $\boldsymbol{\pi}$.
  - Receiver u best decodes these "prior" users and removes them, while "post" users are noise.
  - $\boldsymbol{\pi}$ can be any order in $\mathbb{P}_u(\boldsymbol{\pi})$, but the most interesting is usually $\boldsymbol{\pi}_u$ (receiver u's order).

rcvr/ User i	$\boldsymbol{\pi}_4(i)$	$\boldsymbol{\pi}_3(i)$	$\boldsymbol{\pi}_2(i)$	$\boldsymbol{\pi}_1(i)$
$i = 4$	3	3	4	3
$i = 3$	4	2	3	2
$i = 2$	1	4	2	1
$i = 1$	2	1	1	4
$\mathbb{P}_u(\boldsymbol{\pi}_u)$	{1,2}	{2,4,1}	{1}	{4}

$$\boldsymbol{\Pi} = \begin{bmatrix} 3 & 3 & 4 & 3 \\ 4 & 2 & 3 & 2 \\ 1 & 4 & 2 & 1 \\ 2 & 1 & 1 & 4 \end{bmatrix}$$

Assumes $U' = 4$, so no subusers

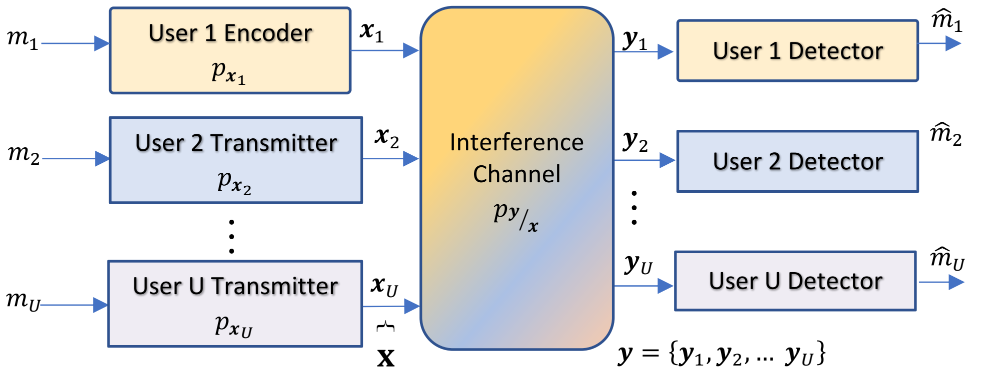
Generally could have $(16)^4$ Orders.

- **Data rates** (mutual information bounds) depend on those user rates that are decoded/cancelled, not xtalk noise – these latter are energies independent of code-rate choice.

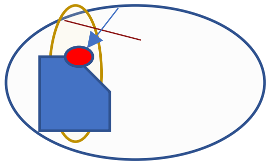
$\mathfrak{I}$	$\mathfrak{I}_4$	$\mathfrak{I}_3$	$\mathfrak{I}_2$	$\mathfrak{I}_1$
top	∞	$\mathcal{I}_3(3/1,2,4)$ 20	∞	∞
	$\mathcal{I}_4(4/1,2)$ 10	$\mathcal{I}_3(2/1,4)$ 9	∞	∞
	$\mathcal{I}_4(1/2)$ 5	$\mathcal{I}_3(4/1)$ 4	$\mathcal{I}_2(2/1)$ 4	$\mathcal{I}_1(1/4)$ 2
bottom	$\mathcal{I}_4(2)$ 1	$\mathcal{I}(1)$ 2	$\mathcal{I}_2(1)$ 2	$\mathcal{I}_1(4)$ 5

$$\mathcal{I}_{min}(\boldsymbol{\Pi}, p_{\boldsymbol{xy}}) = \begin{bmatrix} 4 \\ 20 \\ 1 \\ 2 \end{bmatrix}$$

The CIC



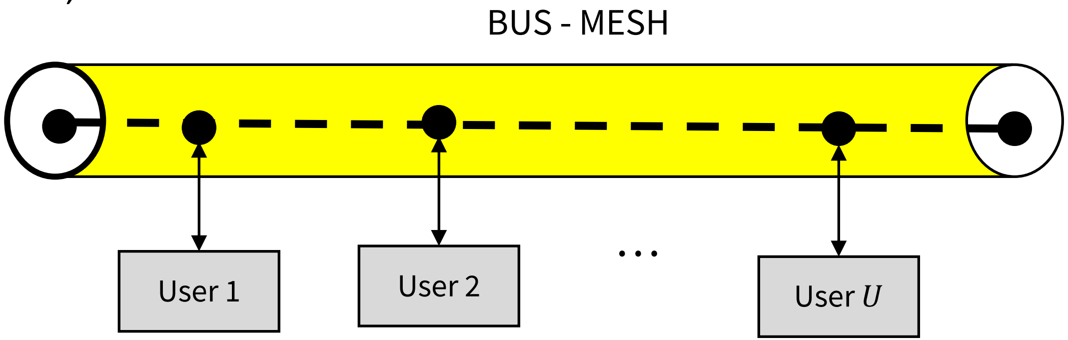
- Studied earlier and found an order-indexed set of vertices $\mathcal{I}_{min}(\boldsymbol{\Pi}, p_{xy})$.
- Then take convex combinations over the possible $(U^2!)^U$ orders of the $\mathcal{I}_{min}(\boldsymbol{\Pi}, p_{xy})$.



The achievable-region remains convex, and the Lagrangian $\boldsymbol{\theta}$ is equivalent to order (I pose even for the IC).

Another Interference Channel Form

- The BUS model;



- All connections are bi-directional and the transmitter/receiver are independent; $U' = U \cdot (U-1)$.
  - This is actually $U \cdot (U-1)$-user MACs with $U \cdot (U-1)$-user BCs. And then, we can consider subusers ….

- Restrictions to IC might include:
  - Only U messages are permitted (e.g., a "switch"), and there is a transmit/receive pair for each.
  - In any given symbol period → time-varying IC.

- The "yellow" might be a common wire(s) or air (common shared spectrum).

- Designers have headed consequently towards "ODM" (**orthogonal division multiplex** – all get their own dims.)

CIC Optimization

CIC's "Optimum" Spectrum Balancing (no GDFEs)

- **OSB** minimizes weighted energy sum for given $\boldsymbol{b}_{min}$.
  - There is no crosstalk cancellation

$$\max_{\{R_{\boldsymbol{xx}}(u,n)\}} \quad \sum_{u=1}^{U} w_u \cdot \mathcal{E}_u$$

$$ST: \quad 0 \leq \sum_{n} \text{trace}\left\{R_{\boldsymbol{xx}}(u,n)\right\} \leq \mathcal{E}_{u,max} \;\; u = 1, \ldots, U$$

- OSB relates $\boldsymbol{b}$ to $R_{\boldsymbol{xx}}(u,n)$.

$$b_u = \sum_{n} \log_2 \frac{\left| H_{uu,n} \cdot R_{\boldsymbol{xx}}(u,n) \cdot H^*_{uu,n} + \mathcal{R}_{noise}(u,n) \right|}{\left| \mathcal{R}_{noise}(u,n) \right|}$$

- $\mathcal{R}_{noise}(u,n) = I + \sum_{i \neq u} \widetilde{H}_{u,i,n} \cdot R_{\boldsymbol{xx}}(i,n) \cdot \widetilde{H}^*_{u,i,n}$.
  - **All** other users are crosstalk-noise additions.

- Tonal Lagrangian
  - Minimize each individually, and sum.
  - It's not convex (no sequential-differences transformation).

$$L_n(R_{\boldsymbol{xx}}(u,n), \boldsymbol{b}_n, \boldsymbol{w}, \boldsymbol{\theta}) = \sum_{u=1}^{U} w_u \cdot \mathcal{E}_{u,n} - \theta_u \cdot b_{u,n}$$

Still has minimum, exponential search

- $SNR(u,n) = \frac{|\tilde{H}_{u,u,n} \cdot R_{xx}(u,n) \cdot \tilde{H}^*_{u,u,n}|}{|R_{noise}(u,n)|}$

$$b_u = \sum_n \log_2\left(1 + \frac{\mathrm{SNR}(u,n)}{\Gamma}\right)$$

- Partition energy range for scalar case.

$$M = \frac{\max_u \boldsymbol{\mathcal{E}}(u)}{\Delta\mathcal{E}}$$

- **Energy step:** For each tone, search M^U possible energies to minimize tonal Lagrangian and add these tonals.
- **Constraint:** Use sub-gradient or ellipsoid descent method to update the $\boldsymbol{\theta}$ Lagrange multiplier for rate constraints.

$$\begin{aligned} \Delta \boldsymbol{b} &= \boldsymbol{b}_{min} - \sum_n \boldsymbol{b}_n \\ \Delta \boldsymbol{\mathcal{E}} &= \boldsymbol{\mathcal{E}}_{max} - \sum_n \boldsymbol{\mathcal{E}}_n \end{aligned}$$

$$\begin{aligned} \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \epsilon \cdot \Delta \boldsymbol{b} \\ \boldsymbol{w} &\leftarrow \boldsymbol{w} + \epsilon' \cdot \Delta \boldsymbol{\mathcal{E}} \end{aligned}$$

$\boldsymbol{w}$ update is only for admissibility search.

OSB.m

```
function [S1, S2, b1, b2] = osb(Hmag_sq, No, E, theta, mask, ...
  gap, bitcap, cb)

  osb and also finds w1 energy weight for USER 1
  A. Chowdhery ~2010 ; Updated by J. Cioffi in 2024.  It presently
handles
  only 2 users, so U=2.

  Inputs

  Hmag_sq is a N x 2 x 2 where N is FFT size. N inferred from this.
  No      is a 1 x U white-noise power spectra density matrix.
     If Hmag_sq is complex BB, then No should be the one-sided PSD.
  E       is a 1 x U energy vector.
  theta   is a 1 x U user-rate weighting vector.
  mask    is an N x U PSD maximum allowed.
  gap     is the (non-dB) linear gap (so 1 if 0 dB gap).
  bitcap  is a 1 x U maximum number of bits allowed per tone.
  cb      is 2 for real baseband and 1 for cplex bband

  Outputs

  S1      is user 1's Nx1 PSD
  S2      is user 2's Nx1 PSD
  b1      is user 1's Nx1 bit distribution
  b2      is user 2's Nx1 bit distribution

calls optimize_l2.m, which calls optimize_s.m
User order is reversed with respect to class convention.
```

Only tests integer bits/tone up to bitcap
Discards solutions that exceed user energy

```
>> H2
H2(:,:,1) =   0.6400   0.2500  % note this is squared mag each term
H2(:,:,2) =   0.4900   0.3600
>> Noise =  1.0e-04 * [   1.0000   1.0000];
>> Ex =[    1     1];
>> mask =[    1     1];
>> gap =    1;
>> bitcap =[   15    15];
>> [S1, S2, b1, b2] = osb(H2, Noise, Ex, [0.5 .5], mask, gap, bitcap,2)

S1 =   0.6398
S2 =    0
b1 =    6  % note < 6.3 for the GDFE based IC's maximum L11:16
b2 =    0
>> [S1, S2, b1, b2] = osb(H2, Noise, Ex, [0.01 .99], mask, gap, bitcap,2)
S1 =    0
S2 =   0.1419
b1 =    0
b2 =   4.5000 <5.9 for L11:16
```

- The OSB search can be very complex for U>3
- It also can have severe numerical issues (cause it to diverge) even in matlab double prec.

Multitone OSB

```
h = cat(3, [1  .8 ;  -1   1], [-.9  -.7 ;  0  1]  )*10;
He = fft(h, 8, 3);
>> H3=zeros(8,2,2);
>> H3(:,1,1)=He(1,1,:);
>> H3(:,2,1)=He(2,1,:);
>> H3(:,2,2)=He(2,2,:);
>> H3(:,1,2)=He(1,2,:);
>> Noise=ones(8,2);
>> mask=ones(8,2);
>> Ex=8*Ex;
>> [S1, S2, b1, b2] = osb(H3.*conj(H3), Noise, Ex, [0.5 .5], mask, gap,
bitcap,1);
>> S1' =    0        0       0.7017   0.8272   0   0.8272   0.7017     0
>> S2' =   0.6375   0.7469     0       0        0     0        0        0.7469

>> b1' =    0    0    7    8    0    8    7    0
>> b2' =    8    8    0    0    0    0    0    8
>> sum(b1) =   30
>> sum(b2) =   24
sum(b1+b2) =   54 % < ~116 that MAC, BC, single had for this channel
```

Same 2x2 channel As in L16.

OSB solution is often FDM

- GDFE's cancellation of crosstalk makes a large difference

IW_polite.m (integer bits like osb.m)

```
% function [b, E] = iw_polite(N, df, U, Hmag, No, Ex, mask, gap, mode,
b_target, bitcap,cb)
%
% Calculates data rates of M users and corresponding bit distributions and
Energy distributions
% using iterative waterfilling
%
% Inputs
% ------
% N: number of sub-channels
% M: number of users
% Hmag: squared channel transfer and crosstalk matrix (N x U x U matrix)
% Hmag(n,i,j) is the crosstalk transfer function from loop i to j at the
nth bin.
% No: noise energy/sample
% Ex: signal energy/SYMBOL
% mask: PSD mask - largest value N x U
% gap: gap (not in dB)
% mode: (U x 1 vector) each value is one of the followings
% 0 - rate adaptive
% 1 - fixed margin (power minimization)
% 2 - margin adaptive
% b_target: target bits on 1 DMT symbol for modes 1 and 2
% bitcap: maximum possible number of bits at each frequency bin
% cb =1 for cplx BB and =2 for real BB
%
% Outputs
% -------
% b: bit distribution (N x U matrix)
% E: energy distribution (N x U matrix)
%
% Remarks
% Iterate waterfiling for each user 10 times
% Youngjae(Sean) Kim - modified J. Cioffi, April 2024
```

>> [b, E] = iw_polite(8, 2, H3.*conj(H3), Noise, [8 8], mask, gap, zeros(8,1), [5 5], bitcap,1)

b =

0	8.0000
0	8.0000
2.0264	1.0000
8.0000	0
8.0000	0
8.0000	0
2.0264	1.0000
0	8.0000

>> [b1 b2] %(osb)

0	8
0	8
7	0
8	0
0	0
8	0
7	0
0	8

E =

0	0.6375
0	0.7469
0.6300	0.3609
0.8272	0
0.7064	0
0.8272	0
0.6300	0.3609
0	0.7469

>> [S1 S2] = %(osb)

0	0.6375
0	0.7469
0.7017	0
0.8272	0
0	0
0.8272	0
0.7017	0
0	0.7469

>> sum(b) = 28.0529 26.0000

>> sum([b1 b2]) = 30 24

- Sum is same, user 1 is better in osb.
- With continuous bit distribution, osb would be slightly better.

Energies < 8 because iw calls campello.m, which allows only integer bits (like osb.m).

IW.m (non-integer) – not in text, but at website

```
function function [b, E] = iw(N, U, Hmag, No, Ex, gap, mode, b_target,cb)

  Calculates data rates of M users and corresponding bit distributions and
Energy distributions
     using iterative waterfilling.

  Inputs
  ------
  N: number of sub-channels
  U: number of users
  Hmag: squared channel transfer and crosstalk matrix (N x U x U matrix)
         Hmag(n,i,j) is the crosstalk transfer function from loop i to j
at the nth bin.
  No: noise power spectrum per tone (N x U)
  Ex: signal energy/SYMBOL
  mask: PSD mask - largest value N x U
  gap: gap in dB
  mode: (U x 1 vector) each value is one of the followings
        0 - rate adaptive
        1 - fixed margin (power minimization)
        2 - margin adaptive
  b_target: target bits on 1 DMT symbol for modes 1 and 2
  bitcap: maximum possible number of bits at each frequency bin
  cb =1 for cplx BB and =2 for real BB

  Outputs
  -------
  b: bit distribution (N x U matrix)
  E: energy distribution (N x U matrix)

  Remarks
  Iterate waterfiling for each user 10 times
  Youngjae(Sean) Kim - modified J. Cioffi, April 2024
```

```
>> [b, E] = iw(8, 2, H3.*conj(H3), Noise, [8 8], gap, zeros(8,1), [5 5],1)

b =
       0    9.5897
       0    9.3612
  1.4972    1.4479
  9.2050         0
  9.4329         0
  9.2050         0
  1.4972    1.4479
       0    9.3612
E =
       0    1.9238
       0    1.9233
  1.1329    1.1148
  1.9112         0
  1.9118         0
  1.9112         0
  1.1329    1.1148
       0    1.9233

>> sum(b) % =  30.8374  31.2079
>> sum(E) % =   8   8
```

```
>> [b1 b2] %(osb)
   0   8
   0   8
   7   0
   8   0
   0   0
   8   0
   7   0
   0   8
```

```
>> [S1 S2] = %(osb)
     0      0.6375
     0      0.7469
  0.7017      0
  0.8272      0
     0        0
  0.8272      0
  0.7017      0
     0      0.7469
```

Energies = 8 now with fractional bits

minPIC = more "optimum"

- minPIC concept allows for each receiver u to cancel $i \in \mathcal{D}_u(\boldsymbol{\Pi}, p_{\boldsymbol{xy}}, \boldsymbol{b})$; the decodable set.
- Order has been restored 😀.
- The optimization is
  - $(i, u) = (RCVR, USER)$

$$\min_{\{R_{\boldsymbol{xx}}(i,u,n)\}} \sum_{n=0}^{\overline{N}-1} \sum_{u=1}^{U} w_u \cdot \text{trace} \underbrace{\left\{ \sum_{i=1}^{U} R_{\boldsymbol{xx}}(i,u,n) \right\}}_{\mathcal{E}_{u,n}}$$

$$ST: \quad b_u \geq b_{min,u}$$

$$R_{\boldsymbol{xx}}(i,u,n) \succeq \mathbf{0} \quad .$$

- $\boldsymbol{\theta}$ still has U terms, and they determine the U! "**sensible" orders $\boldsymbol{\Pi}$**.
- The achievable-region constraint remains convex already (like minPMAC).
- Implement GDFE at each receiver (no precoders).

3-User Order example

- Given a $\boldsymbol{\theta}$, say for example with $\theta_3 > \theta_1 > \theta_2$, they determine all receivers' order:

$FOR: \quad \theta_3 \quad > \quad \theta_1 \quad > \quad \theta_2 \quad > 0$

(RCVR, USER)	Receiver 3	Receiver 1	Receiver 2
	(1,1),(2,1),(1,2),(2,2)	(2,2),(3,2),(2,3),(3,3)	(1,1),(3,1),(1,3),(3,3)
	(3,3)	(1,3)	(2,3)
	(1,3)	**(3,1)**	(2,1)
	(2,3)	**(1,1)**	**(3,2)**
	(3,1)	**(2,1)**	**(1,2)**
	(3,2)	(1,2)	**(2,2)**

THESE ARE constant XTALK, AND CAN BE 0

Π

- Any other order is inconsistent with the Lagrangian multipliers' interpretation.

$$
\begin{aligned}
A &\triangleq |H_{3,1}|^2 \cdot (\mathcal{E}_{1,1} + \mathcal{E}_{2,1}) + |H_{3,2}|^2 \cdot (\mathcal{E}_{1,2} + \mathcal{E}_{2,2}) + I \\
B &\triangleq |H_{3,3}|^2 \cdot \mathcal{E}_3 + A \\
C &\triangleq |H_{3,1}|^2 \cdot \mathcal{E}_{3,1} + B \\
D &\triangleq |H_{3,2}|^2 \cdot \mathcal{E}_{3,2} + C
\end{aligned}
$$

RCVR 3

$$
\begin{aligned}
b_3 &= \log_2(B) - \log_2(A) \\
b_{3,1} &= \log_2(C) - \log_2(B) \\
b_{3,2} &= \log_2(D) - \log_2(C)
\end{aligned}
$$

$$\left\{ \sum_{u=1}^{3} \theta_u \cdot b_u \right\}_{RCVR3opt} = (\theta_3 - \theta_1) \cdot \log_2(B) + (\theta_1 - \theta_2) \cdot \log_2(C) + \theta_2 \cdot \log_2(D)$$

Do same for other 2 receivers

- RCVR 1 optimization of rate sum

RCVR 1

$$\begin{aligned} A &\triangleq |H_{1,3}|^2 \cdot (\mathcal{E}_{2,3} + \mathcal{E}_{3,3}) + |H_{1,2}|^2 \cdot (\mathcal{E}_{1,2} + \mathcal{E}_{2,2}) + I \\ B &\triangleq |H_{1,3}|^2 \cdot \mathcal{E}_{1,3} + A \\ C &\triangleq |H_{1,1}|^2 \cdot \mathcal{E}_1 + B \\ D &\triangleq |H_{1,2}|^2 \cdot \mathcal{E}_{1,2} + C \end{aligned}$$

$$\begin{aligned} b_{1,3} &= \log_2(B) - \log_2(A) \\ b_1 &= \log_2(C) - \log_2(B) \\ b_{1,2} &= \log_2(D) - \log_2(C) \quad . \end{aligned}$$

$$\left\{\sum_{u=1}^{3} \theta_u \cdot b_u\right\}_{RCVR1opt} = (\theta_3 - \theta_1) \cdot \log_2(B) + (\theta_1 - \theta_2) \cdot \log_2(C) + \theta_2 \cdot \log_2(D)$$

- RCVR 2 optimization of rate sum

RCVR 2

$$\begin{aligned} A &\triangleq |H_{2,3}|^2 \cdot (\mathcal{E}_{1,3} + \mathcal{E}_{3,3}) + |H_{2,1}|^2 \cdot (\mathcal{E}_{1,1} + \mathcal{E}_{3,1}) + I \\ B &\triangleq |H_{2,3}|^2 \cdot \mathcal{E}_{2,3} + A \\ C &\triangleq |H_{2,1}|^2 \cdot \mathcal{E}_{2,1} + B \\ D &\triangleq |H_{1,1}|^2 \cdot \mathcal{E}_2 + C \end{aligned}$$

$$\begin{aligned} b_{2,3} &= \log_2(B) - \log_2(A) \\ b_{2,1} &= \log_2(C) - \log_2(B) \\ b_2 &= \log_2(D) - \log_2(C) \quad . \end{aligned}$$

$$\left\{\sum_{u=1}^{3} \theta_u \cdot b_u\right\}_{RCVR2opt} = (\theta_3 - \theta_1) \cdot \log_2(B) + (\theta_1 - \theta_2) \cdot \log_2(C) + \theta_2 \cdot \log_2(D)$$

- Six energies repeat – select the smallest that has corresponding lowest rate for use.
- Outer $\boldsymbol{\theta}$ loop (e.g., Ellipsoid) remains the same as minPMAC.

Generalize – first order them to simplify

- Create order of users for each of (reordered) users

θ_U	$\dots$	θ_u	$\dots$	θ_1
$U^2 \setminus \{(1:U,U),(U,1:U-1)\}$	$\dots$	$U^2 \setminus \{(1:U,u),(u,1:U-1)\}$	$\dots$	$U^2 \setminus \{(U,1:U),(1,1:U-1)\}$
(U,U)	$\dots$	(u,U)	$\dots$	1,U-1
		$\vdots$		$\vdots$
$\vdots$	$\dots$	$(u,U-u+1)$	$\dots$	$(1,1)$
$(1,U)$	$\dots$	(U,u)	$\dots$	$(U,1)$
$(U,U-1)$	$\dots$	$\vdots$	$\dots$	$\vdots$
$\vdots$	$\dots$	$(1,u)$	$\dots$	$\vdots$
		$(u,U-u-1)$		
$\vdots$	$\dots$	$\vdots$	$\dots$	$\vdots$
$(U,1)$	$\dots$	$(u,1)$	$\dots$	(U,U)

Table 5.2: **Generalized of overall decoding order pairs given descending-order θ.**

Generalize A,B,C, D

$$
\begin{array}{llr}
K_{1,u} & \triangleq \sum_{i \neq u} H_{u,i} \cdot \left(\sum_{j \neq i} R_{\boldsymbol{xx}}(i,j) \right) \cdot H^*_{u,i} + I & \text{for } \; b_{u,U} \\
K_{2,u} & \triangleq H_u(2) \cdot R_{\boldsymbol{xx}}(u, 2U-3, 2) \cdot H^*_u(2^{nd}) + K_{1,u} & \text{for } \; b_{u,U-} \\
\vdots & \vdots & \\
K_{u,u} & \triangleq H_{u,2U-u+1}(2) \cdot R_{\boldsymbol{xx}}(u, 2U-u+1(2^{nd})) \cdot H^*_{u,2U-u+1}(2) + K_{u-1,u} & \text{for } \; b_u \\
\vdots & \vdots & \\
K_{2U-2,u} & \triangleq H_{u,1}(2) \cdot R_{\boldsymbol{xx}}(u, 1(2^{nd})) \cdot H^*_{u,1}(2) + K_{2U-3,u} & \text{for } \; b_{u,1}
\end{array}
$$

$$
\left\{ \sum_{u=1}^{U} \theta_u \cdot b_u \right\}_{RCV\,Ruopt} = (\theta_U - \theta_{U-1}) \cdot \log_2(K_{1,u}) + \ldots + (\theta_2 - \theta_1) \cdot \log_2(K_{2U-3,u}) + \theta_2 \cdot \log_2(K_{2U-2,u})
$$

- This is convex in those quantities optimized
- Need the outer subgradient loop on theta to drive IC rate vector to bmin.

Software awaits writing.

End Lecture 17