Supplementary Lecture 12A

Probabilistic Amplitude Shaping Codes

February 17, 2026

JOHN M. CIOFFI

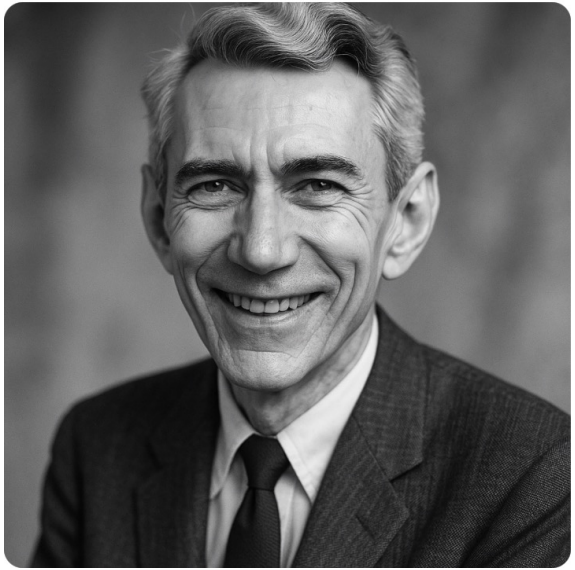
Hitachi Professor Emeritus (recalled) of Engineering

Instructor EE379A – Winter 2026

Announcements & Agenda

- Shaping Gain and Limits
- Shells, Shaping, and Entropy
- PAS
- CCDM Encoder and Decoder

The last (up to) 1.5 dB on the AWGN



Claude Shannon (smiling, thank you ChatGPT)

Shaping Gain and Limits

Section 8.4.2

Shaping Gain Review

- Coding gain is

$$\gamma \triangleq \frac{\left(d^2_{\min}(\boldsymbol{x})/\bar{\mathcal{E}}_{\boldsymbol{x}}\right)}{\left(d^2_{\min}(\breve{\boldsymbol{x}})/\bar{\mathcal{E}}_{\breve{\boldsymbol{x}}}\right)} = \underbrace{\frac{\left(\frac{d^2_{\min}(\boldsymbol{x})}{V^{2/N}(\Lambda)}\right)}{\left(\frac{d^2_{\min}(\breve{\boldsymbol{x}})}{V^{2/N}(\breve{\Lambda})}\right)}}_{\gamma_f} \cdot \underbrace{\frac{\left(\frac{V^{2/N}(\Lambda)}{\bar{\mathcal{E}}_{\boldsymbol{x}}}\right)}{\left(\frac{V^{2/N}(\breve{\Lambda})}{\bar{\mathcal{E}}_{\breve{\boldsymbol{x}}}}\right)}}_{\gamma_s}$$

γ_f: **fundamental gain**

γ_s: **shaping gain**

Shaping Gain is This S12A focus

- Shaping gain relative to $\mathbb{Z}^N$ (SQ QAM) with $d = 1$ in reference simplifies to

$$\gamma_s = \frac{V^{2/N}(\Lambda) \cdot \left(2^{2\bar{b}} - 1\right)}{\tilde{\mathcal{E}}_{\boldsymbol{x}} \cdot 6 \cdot \tilde{N}}$$

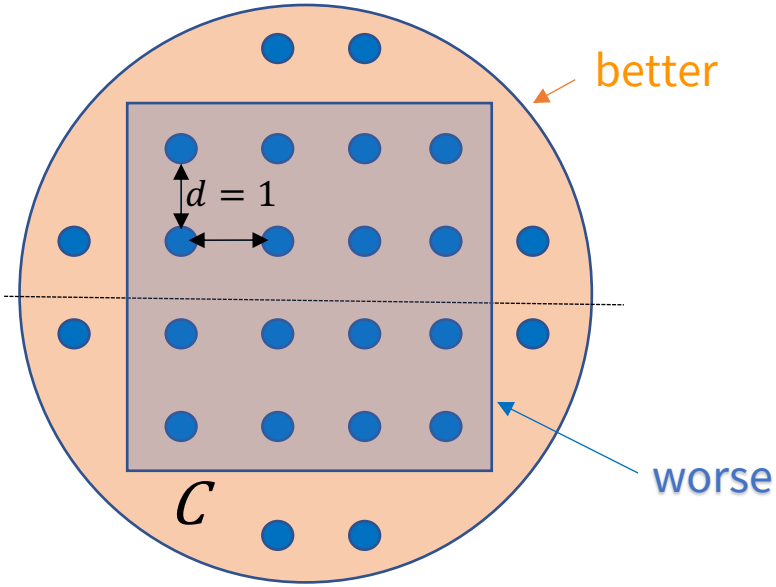
- When both constellations use the same number of 2D constellation points, this further simplifies to

$$\gamma_s = \frac{\left(2^{2\bar{b}} - 1\right)}{6 \cdot \tilde{\mathcal{E}}_{\boldsymbol{x}}}$$

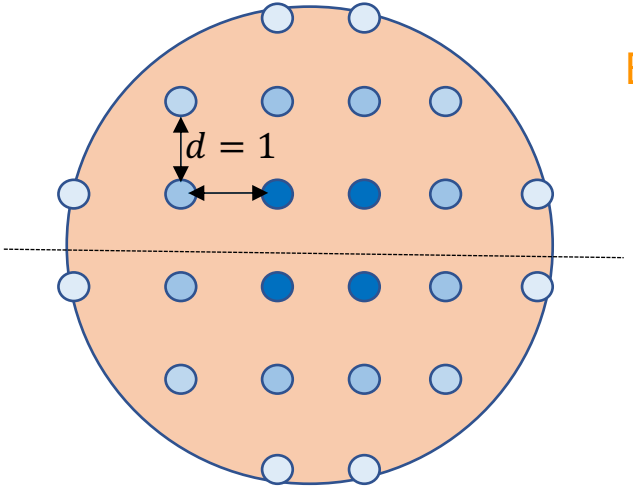
So shaping is basically the energy ratio to send same # of points.

Hypersphere boundaries are good (Gaussian)

Subsymbol (ss)



- As N grows, harder to visualize but uniform spacing with fixed average energy leads to Gaussian marginals



- So, a better boundary shape MARGINAL ~ non-equal probability (marginal) distribution
  - (NES= Non Equiprobable Signaling – or "Shaping")

Asymptotic Shaping-Gain Results

- Volume of $2n$-dimensional hypersphere is $\frac{\pi^n \cdot r^{2n}}{n!}$.
- Energy of $2n$-dimensional hypersphere is $\frac{1}{2} \cdot \left[\frac{r^2}{n+1}\right]$.

$$\gamma_s \leq \frac{\pi r^2 (n!)^{-\frac{1}{n}} / \frac{r^2}{2(n+1)}}{12\left(1-2^{-2\bar{b}}\right)} = \frac{\pi}{6} \frac{(n!)^{-\frac{1}{n}} \cdot (n+1)}{1-2^{-2\bar{b}}}$$

- Large n

$\bar{b}$	$\gamma_s(n \to \infty)$
∞	1.53 dB
4	1.52 dB
3	1.46 dB
2	1.25 dB
1	0.28 dB
0	0 dB

- Large $\bar{b}$

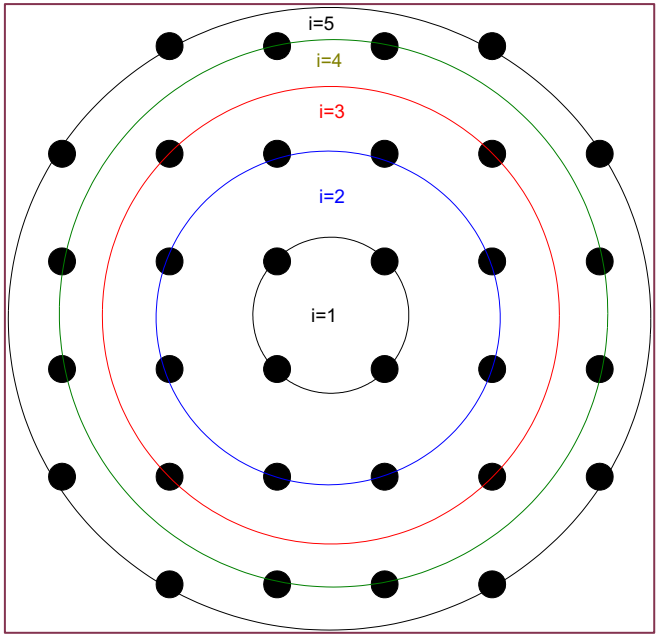
$n(N)$	$\gamma_s(\bar{b} \to \infty)$
1 (2)	0.20 dB
2 (4)	0.46 dB
3 (6)	0.62 dB
4 (8)	0.73 dB
12 (24)	1.10 dB
16 (32)	1.17 dB
24 (48)	1.26 dB
32 (64)	1.31 dB
64 (128)	1.40 dB
100 (200)	1.44 dB
500 (64)	1.50 dB
1000 (2000)	1.52 dB
10000 (20000)	1.53 dB

- infinite $\bar{b}$ first, then infinite n

$$\lim_{n\to\infty} \gamma_s \leq \lim_{n\to\infty} \frac{\pi}{6} \cdot \frac{n+1}{(n!)^{\frac{1}{n}}} = \frac{\pi}{6} \lim_{n\to\infty} \frac{n+1}{\frac{n}{e}} = \frac{\pi \cdot e}{6} = 1.53 \text{ dB}$$

Shells from 36SQ

- 32 points, 5 shells, corner points omitted (zero probability)



- With uniform probability of each point (1/32)

$$\mathcal{E}_{\boldsymbol{x}} = \sum_{i=1}^{G} p_i \cdot \mathcal{E}_i = \left[\frac{2}{8} + \frac{10}{4} + \frac{18}{8} + \frac{26}{4} + \frac{34}{4}\right] = 20$$

- Shaping gain is $\gamma_s = \dfrac{(1/6) \cdot 31 \cdot d^2}{20} = \dfrac{31}{30} = 0.14 \text{ dB}$

- How about if the data rate is $\tilde{b} = 4$?
  - But we keep this constellation and reduce outer points' probabilities?
  - It can increase to 1.2 dB (over 16QAM).
  - With $n \geq 256$.

Shells, Shaping, and Entropy

Section 8.4.2

Basic Entropy

- The number of bits per subsymbol is

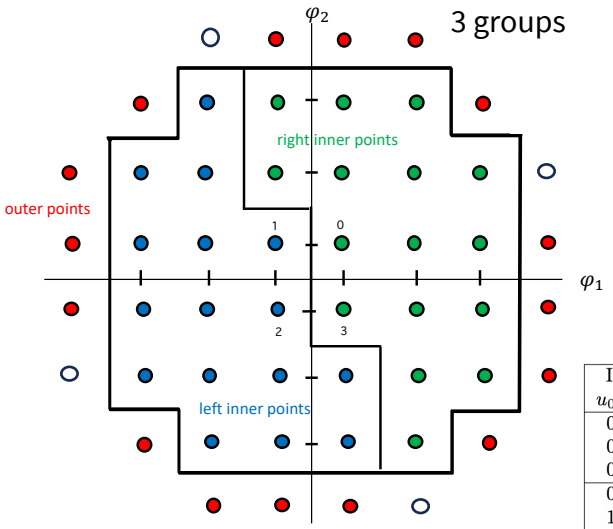
$$\tilde{b} \le \mathcal{H}_{\bar{\boldsymbol{x}}} = \underbrace{\sum_{i=1}^{G}}_{\text{shells}} \underbrace{\sum_{j=1}^{|C|_i}}_{\text{pts in shell}} p_{ij} \cdot \log_2\left(\frac{1}{p_{ij}}\right)$$

- This rearranges to

$$\mathcal{H}_{\boldsymbol{x}} = \underbrace{\sum_{i=1}^{G} p_i \cdot \log_2\left(\frac{1}{p_i}\right)}_{\substack{group\ rate \\ component}} + \underbrace{\sum_{i=1}^{G} p_i \cdot b_i}_{\substack{equally\ likely \\ in\ group}} = \mathcal{H}(\boldsymbol{p}) + \sum_{i=1}^{G} p_i \cdot b_i$$

- Good design minimizes energy $\mathcal{E}_{\bar{x}} = \sum_{g=1}^{G} \mathcal{E}_g$ while retaining $\mathcal{H}_{\bar{x}}$ rate/bound.

Groups do not have to be shells



a). 48CR $\tilde{b} = 5.5$

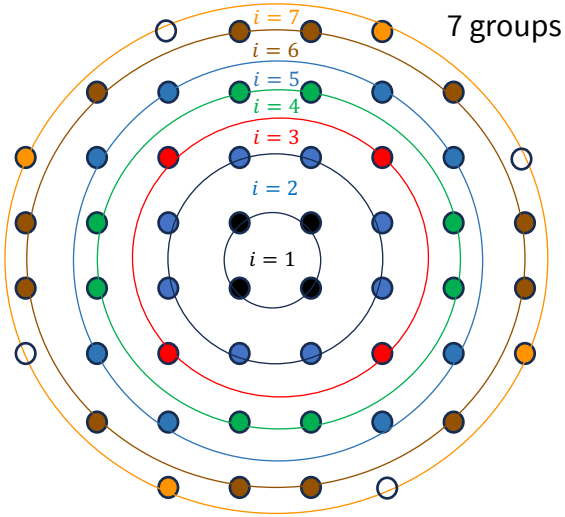
$p = p = 2p'$

(or 52CR o, with $\tilde{b} = 5.625$)

$\gamma_s = .23\ dB\ (48) \rightarrow .61\ dB\ (52)$

Input Bits			1^{st} 2D subsym.			2^{nd} 2D subsym.		
u_0	u_1	u_1	ss_1	ss_0	position	ss_3	ss_2	position
0	0	0	0	0	left-inner	0	0	left-inner
0	0	1	0	0	left-inner	0	1	right-inner
0	1	0	0	0	left-inner	1	0	outer
0	1	1	0	1	right-inner	0	0	left-inner
1	0	0	0	1	right-inner	0	1	right-inner
1	0	1	0	1	right-inner	1	0	outer
1	1	0	1	0	outer	0	0	left-inner
1	1	1	1	0	outer	0	1	right-inner

nonlinear map look-up table See S3 modulation codes



b). 48SH or 52SH

$\gamma_s = 1.3\ dB$ with optimum p_i

- **Left side, a), is improvement over 32CR , but still less than b).**

PAS
(probabilistic amplitude shaping)

Section 8.4.2.1

PAS Optimizes Shell Probabilities

- Optimization Statement

$$
\begin{aligned}
\min_{\mathcal{E}_{iin[1:G]}} \quad & \mathcal{E}_{\tilde{\boldsymbol{x}}} = && \sum_{i=1}^{G} p_i \cdot \mathcal{E}_i \\
\text{ST:} \quad & \mathcal{H}_{\tilde{\boldsymbol{x}}} = && -\sum_{i=1}^{G} p_i \cdot \log_2\left(\frac{p_i}{|C|_i}\right) \\
& 1 = && \sum_{i=1}^{G} p_i \\
& \mathcal{E}_i = && (2i+1)^2 \\
& 0 \leq && p_i \leq 1 \; \forall \; i \in [1:G] \quad .
\end{aligned}
$$

- Solution (exponential distribution) is:
  - $Z(\nu)$ is a constant function of $\mathcal{H}_{\tilde{\boldsymbol{x}}}$ target.

$$
p_i(\nu) = |C|_i \cdot \frac{e^{-\nu \cdot \mathcal{E}_i}}{Z(\nu)} \geq 0
$$

$$
Z(\nu) \triangleq \sum_{i=1}^{G} |C|_i \cdot e^{-\nu \cdot \mathcal{E}_i}
$$

8 groups from 7 rings, 1 with 2 arcs

- Ring 6 has 12 points, so divide into arcs of size 8 and size 4 (which are powers of 2.

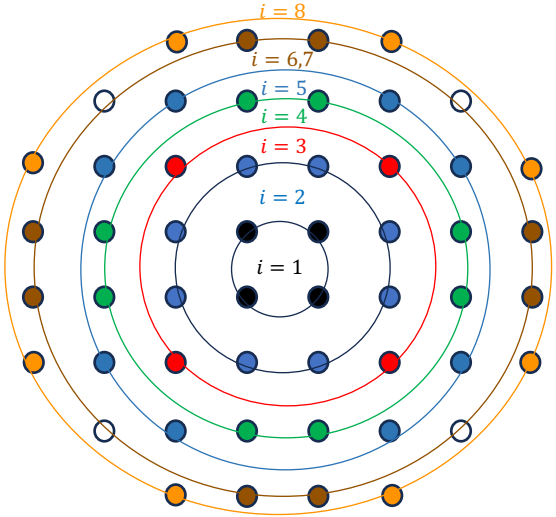
```
% function out = qam_shell_shaping(M, G, H_target_bits, opts)
qam_shell_shaping  Optimal probabilities for SQ-QAM under an entropy constraint,
                   aggregated into G groups/shells.
  J. Cioffi with chatGPT, October 2, 2025

 Features:
  grouping:
   - 'rings_only'  : each shell is exactly one constant-energy ring.
                     If G < #rings, keeps the G lowest-energy rings (outer points dropped).
   - 'equal_points': merges adjacent rings to ~equal #points per shell.
   - 'equal_rings' : merges adjacent rings to equal (±1) #rings per shell.
  normalization: 'dmin1' (default; Es_uniform=(M-1)/6) or 'dmin2' (energies ×4).
  mode:
   - 'microstate'    : q_i ∝ 2^{-λ e_i} on the kept support; entropy on q  (default)
   - 'shell_uniform' : p_g over shells with q_i = p_g/|S_g|; entropy H(p)+Σ p_g log2|S_g|
    (In 'rings_only', both modes coincide because all points in a ring have the same energy.)

 Shaping gain reported = Forney space-filling gain:
  10*log10( E_square_eqvolume(H) / Es_shaped )  (energy-only, equal-volume square reference).
  No indexing/volume factor is multiplied into the gain (we report it separately).

 Example:
  opts = struct('grouping','rings_only','normalization','dmin1','mode','shell_uniform');
  out  = qam_shell_shaping(64, 7, 5.5, opts);
  disp(out.shaping_gain_dB)
```

52 of 64 points in 64 SQ (8 groups with 2 or 3 bits each)



8 shells (shell 6 with 2 arcs)

Example Use

- Use 64SQ (but corner points have 0 prob)
  - 8 shells and arcs
  - **7** shells (see program input)
  - Rate/H is **5.5** bits/2D subsymbol
- Lots of outputs, but shaping gain is main one.
- Also, of interest are
  - The 7 → 8 **probabilities**
- The issue is the probabilities are not yet readily implemented. We'd like them to approximate integer/n so that we have integer occurrence of each shell/arc within a group-selection codeword.

```
>> out = qam_shell_shaping(64, 7, 5.5, opts);
SQ-64 QAM | kept 52/64 points | H_target=5.500000 (achieved 5.500000)
mode=shell_uniform | norm=dmin1 | Es_shaped=5.765538 |
Es_eqvol=7.375806 | gain=1.0697 dB | index K/2^H=1.1490
>> out =
          shell_prob: [7x1 double]
         shell_sizes: [7x1 double]
      shell_avg_energy: [7x1 double]
           Es_shaped: 5.7655
     Es_square_uniform: 10.5000
     Es_reference_eqvolume: 7.3758
    shaping_gain_dB: 1.0697
       H_achieved_bits: 5.5000
            lambda: 0.1688
      indexing_factor: 1.1490
             rings: [9x3 table]

 >> temp=out.shell_prob' % =
 0.1637   0.2591   0.1025   0.1622   0.1284   0.1206   0.0636
>> shellprob2=[temp(1:5),(2/3)*temp(6) , (1/3)*temp(6) , temp(7)] % =
   0.1637   0.2591   0.1025   0.1622   0.1284   0.0804   0.0402   0.0636>>
out.shell_sizes % =
 >> out.shell_sizes' % =
    4    8    4    8    8   12    8
>> shellsize2=[temp(1:5),(2/3)*temp(6) , (1/3)*temp(6) , temp(7)] % =
    4    8    4    8    8    8    4    8
```

Use larger constellation (and more groups)

- Can increase rate to 6.5, and then groups to 15→ 16
- Again ring 6 → 2 arcs
- The design then needs to trim the probabilities.

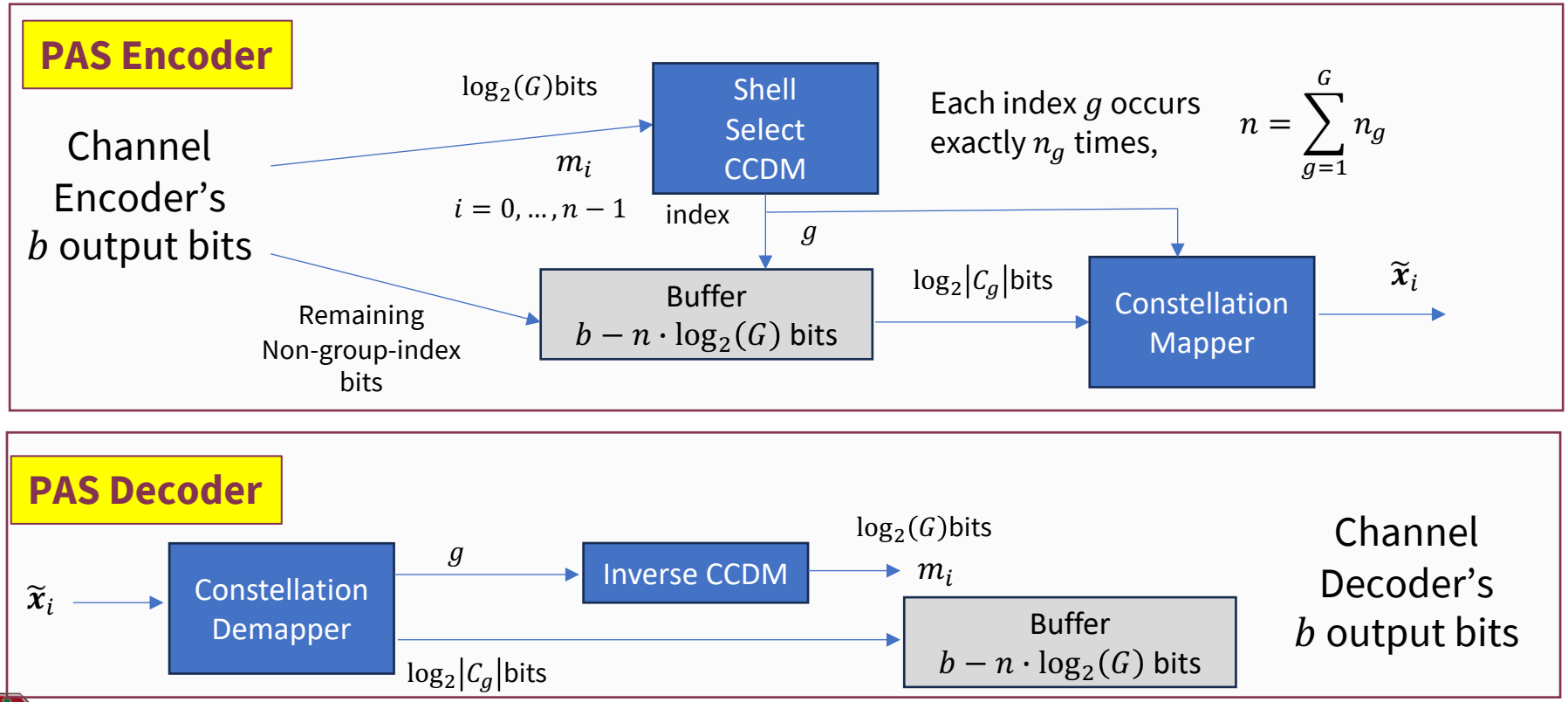
```
out = qam_shell_shaping(256, 15, 6.5, opts);
>> out
shaping_gain_dB: 1.3061
       H_achieved_bits: 6.5000
             lambda: 0.1081
       indexing_factor: 1.3325800
              rings: [32x3 table]
 > out.shell_prob' % =
   0.0974   0.1676   0.0722   0.1242   0.1069   0.1189   0.0682   0.0506
   0.0435   0.0375   0.0161   0.0278   0.0206   0.0354   0.0131
>>out.shell_sizes %=
>> out.shell_sizes'
    4   8   4   8   8  12   8   8   8   8   4   8   8  16   8   4
```

CCDM Encoder and Decoder

Section 8.4.2.2

Need to map input bits to rings

- Assumes initially that $n = \sum_{g=1}^{G} n_g$, and that all n_g are integers (we'll return to this "rounding" later).



Constant Composition Distribution Matching

- Given a (trimmed) probability distribution with $n_g = n \cdot p_g \in \mathbb{Z}^+$, compose codeword
  - that has exactly n_i occurrences of group g in each codeword of group indices, $n = \sum_{g=1}^{G} n_g$
- This relies heavily on reduction of the number and denominator of the multinomial integer:

$$\frac{n!}{\prod_{g=1}^{G} n_g}$$

- The CCDM algorithm iteratively creates a series of $r_g = n_g - i$ remaining index instances within codeword until all are zero.
- With time $t = 0, ., ., n-1$, the multinomial becomes time dependent, and one group is selected and then reduced by 1 in remaining occurences for transmission. This creates the time-varying multinomial

$$C(t, \boldsymbol{r}_t) \triangleq \frac{(n-t)!}{\prod_{i=1}^{G} (r_{i,t})!}$$

- Proceeding sequentially in time, up to G possible next multinomial values are tested/computed

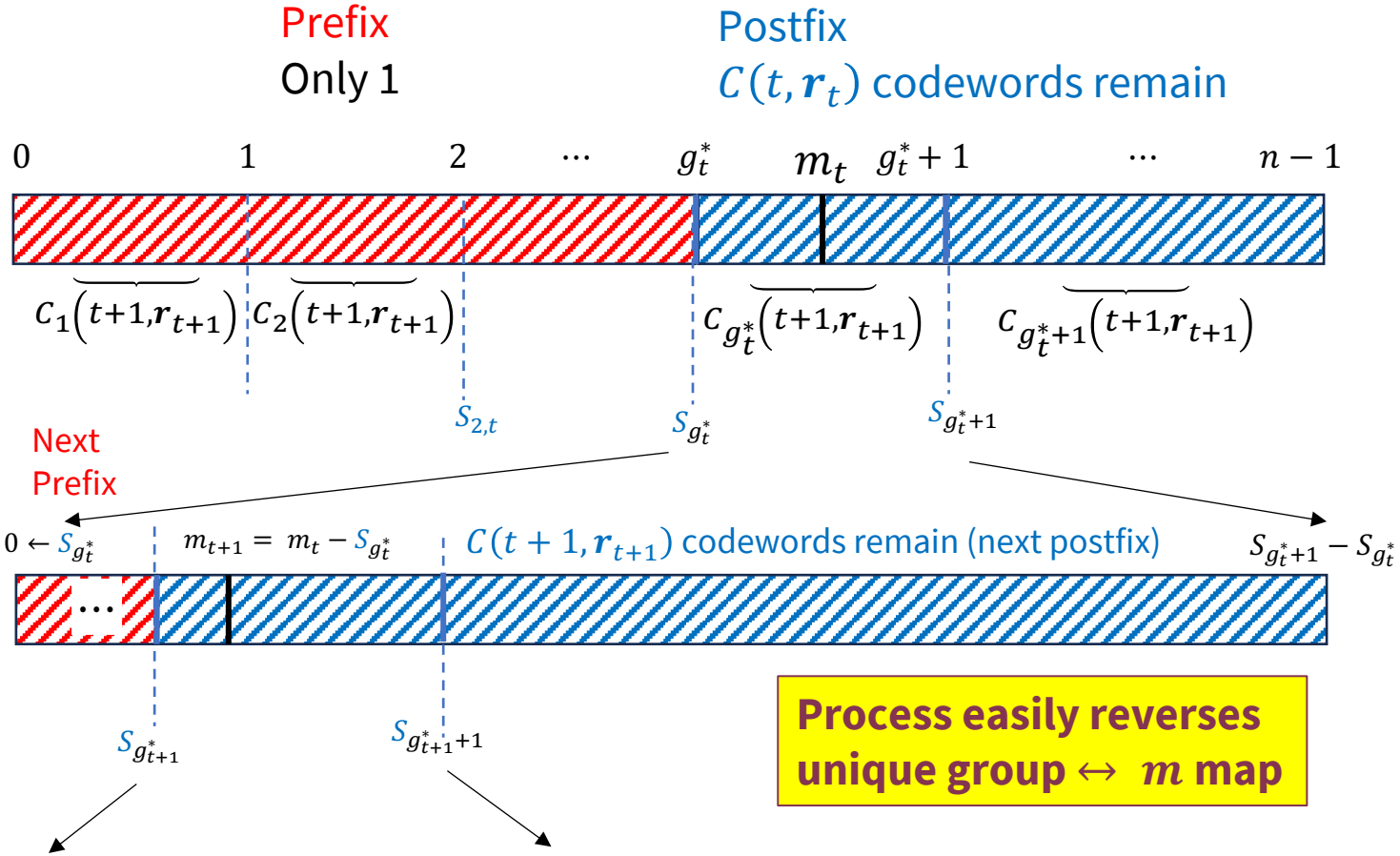
$$C_g(t+1, \boldsymbol{r}_{t+1}) \triangleq \frac{(n-t-1)!}{(r_g - 1) \cdot \prod_{i \neq g} (r_{i,t})!}$$

Timeline of Code Generation

- Time t

$$S_g \triangleq \sum_{i=1}^{g} C_i(t+1, \boldsymbol{r}_{t+1})$$

$0 \le m_0 = m < 2^n$
Map m to codeword
Invertibly maps
to group indices.



Process easily reverses unique group $\leftrightarrow$ m map

Encoder example with $G = 3$ groups, $n = 8$

- **Initial 8-bit input:** $m = 182$, converted from unsigned binary 8 bits (10110110) ; [.5 .25 .25] → [4 2 2]
  - $G = 3$ is not power of 2 (example only) so there is some additional rate loss. $G = 3$ is presented only to facilitate CCDM algorithm description.

1. $t = 0$, $\boldsymbol{r}_0 = [4, 2, 2]$, $C(0, \boldsymbol{r}_0) = 420$

$$C_1(1, \boldsymbol{r}_0 - \boldsymbol{e}_1) = \frac{7!}{3! \cdot 2! \cdot 2!} = 210 \quad (8.200)$$
$$C_2(1, \boldsymbol{r}_0 - \boldsymbol{e}_2) = \frac{7!}{4! \cdot 2!} = 105 \quad (8.201)$$
$$C_3(1, \boldsymbol{r}_0 - \boldsymbol{e}_3) = \frac{7!}{4! \cdot 2!} = 105 \quad (8.202)$$

$S_{[1:3]} = [210, 315, 420]$. Since 182 is between cumulative sums 0 and 1, then $\boldsymbol{x}_0 = 1$. Further, $\boldsymbol{r}_1 = [3, 2, 2]$ while $m_1 = 182 - 0 = 0$.

2. $t = 1$, $\boldsymbol{r}_1 = [3, 2, 2]$, $C(1, \boldsymbol{r}_1) = \frac{6!}{3! \cdot 2! \cdot 2!} = 210 = S_1$ from step $t = 0$.

$$C_1(1, \boldsymbol{r}_1 - \boldsymbol{e}_1) = \frac{6!}{2! \cdot 2! \cdot 2!} = 90 \quad (8.203)$$
$$C_2(1, \boldsymbol{r}_1 - \boldsymbol{e}_2) = \frac{6!}{3! \cdot 2!} = 60 \quad (8.204)$$
$$C_3(1, \boldsymbol{r}_1 - \boldsymbol{e}_3) = \frac{6!}{3! \cdot 2!} = 60 \quad (8.205)$$

$S_{[1:3]} = [90, 150, 210]$. Since 182 is between cumulative sums 2 and 3, then $\boldsymbol{x}_1 = 3$. Further, $\boldsymbol{r}_2 = [3, 2, 1]$ while $m_2 = 182 - 150 = 32$.

3. $t = 2$, $\boldsymbol{r}_2 = [3, 2, 1]$, $C(2, \boldsymbol{r}_2) = \frac{5!}{3! \cdot 2! \cdot 1!} = 60 = S_3$ from step $t = 1$.

$$C_1(2, \boldsymbol{r}_2 - \boldsymbol{e}_1) = \frac{5!}{2! \cdot 2!} = 30 \quad (8.206)$$
$$C_2(2, \boldsymbol{r}_2 - \boldsymbol{e}_2) = \frac{5!}{3! \cdot 1!} = 20 \quad (8.207)$$
$$C_3(2, \boldsymbol{r}_2 - \boldsymbol{e}_3) = \frac{5!}{3! \cdot 2!!} = 10 \quad (8.208)$$

$S_{[1:3]} = [30, 50, 60]$. Since 32 is between cumulative sums 1 and 2, then $\boldsymbol{x}_2 = 2$. Further, $\boldsymbol{r}_3 = [3, 1, 1]$ while $m_2 = 32 - 30 = 2$.

4. $t = 3$, $\boldsymbol{r}_3 = [3, 1, 1]$, $C(3, \boldsymbol{r}_3) = \frac{5!}{3!} = 20 = S_2$ from step $t = 2$.

$$C_1(3, \boldsymbol{r}_3 - \boldsymbol{e}_1) = \frac{4!}{2!} = 12 \quad (8.209)$$
$$C_2(3, \boldsymbol{r}_3 - \boldsymbol{e}_2) = \frac{4!}{3!} = 4 \quad (8.210)$$
$$C_3(3, \boldsymbol{r}_3 - \boldsymbol{e}_3) = \frac{4!}{3!} = 4 \quad (8.211)$$

$S_{[1:3]} = [12, 16, 20]$. Since 2 is between cumulative sums 0 and 1, then $\boldsymbol{x}_3 = 1$. Further, $\boldsymbol{r}_4 = [2, 1, 1]$ while $m_3 = 2 - 0 = 2$.

5. $t = 4$, $\boldsymbol{r}_4 = [2, 1, 1]$, $C(4, \boldsymbol{r}_4) = \frac{4!}{2!} = 12 = S_1$ from step $t = 3$.

$$C_1(4, \boldsymbol{r}_4 - \boldsymbol{e}_1) = \frac{3!}{1!} = 6 \quad (8.212)$$
$$C_2(4, \boldsymbol{r}_4 - \boldsymbol{e}_2) = \frac{3!}{2!} = 3 \quad (8.213)$$
$$C_3(4, \boldsymbol{r}_4 - \boldsymbol{e}_3) = \frac{3!}{2!} = 3 \quad (8.214)$$

$S_{[1:3]} = [6, 9, 12]$. Since 2 is between cumulative sums 0 and 1, then $\boldsymbol{x}_4 = 1$. Further, $\boldsymbol{r}_4 = [1, 1, 1]$ while $m_4 = 2 - 0 = 2$.

6. $t = 5$, $\boldsymbol{r}_5 = [1, 1, 1]$, $C(5, \boldsymbol{r}_5) = \frac{3!}{1!} = 6 = S_1$ from step $t = 4$.

$$C_1(5, \boldsymbol{r}_5 - \boldsymbol{e}_1) = 2! = 2 \quad (8.215)$$
$$C_2(5, \boldsymbol{r}_5 - \boldsymbol{e}_2) = 2! = 2 \quad (8.216)$$
$$C_3(5, \boldsymbol{r}_5 - \boldsymbol{e}_3) = 2! = 2 \quad (8.217)$$

$S_{[1:3]} = [2, 4, 6]$. Since 2 is between cumulative sums 1 and 2, then $\boldsymbol{x}_5 = 2$. Further, $\boldsymbol{r}_5 = [1, 0, 1]$ while $m_5 = 2 - 2 = 0$.

7. $t = 6$, $\boldsymbol{r}_6 = [1, 0, 1]$, $C(6, \boldsymbol{r}_6) = \frac{2!}{1!} = 2 = S_2$ from step $t = 5$.

$$C_1(6, \boldsymbol{r}_6 - \boldsymbol{e}_1) = 1! = 1 \quad (8.218)$$
$$C_2(6, \boldsymbol{r}_6 - \boldsymbol{e}_2) = \emptyset \quad (8.219)$$
$$C_3(6, \boldsymbol{r}_6 - \boldsymbol{e}_3) = 1! = 1 \quad (8.220)$$

$S_{[1:3]} = [1, 1, 1]$. Since 0 is between cumulative sums 0 and 0, then $\boldsymbol{x}_6 = 1$. Further, $\boldsymbol{r}_6 = [0, 0, 1]$ while $m_6 = 2 - 2 = 0$.

8. clearly $\boldsymbol{x}_7 = 3$ as it is the only choice left.

$$\boldsymbol{x} = [1\ 3\ 2\ 1\ 1\ 2\ 1\ 3]$$

Corresponding Decoder Example

- $\boldsymbol{x} = [1\ 3\ 2\ 1\ 1\ 2\ 1\ 3]$

1. $t = 0$, $\boldsymbol{r}_0 = [4, 2, 2]$, $C(0, \boldsymbol{r}_0) = 420$, $g^* = \boldsymbol{x}_0 - 1 = 1 - 1 = 0$

$$\begin{aligned} C_1(1, \boldsymbol{r}_0 - \boldsymbol{e}_1) &= \frac{7!}{3! \cdot 2! \cdot 2!} = 210 \\ C_2(1, \boldsymbol{r}_0 - \boldsymbol{e}_2) &= \frac{7!}{4! \cdot 2!} = 105 \\ C_3(1, \boldsymbol{r}_0 - \boldsymbol{e}_3) &= \frac{7!}{4! \cdot 2!} = 105 \end{aligned}$$

Further, $m = 0 + 0$. $r_1 = 4 - 1 = 3$

2. $t = 1$, $\boldsymbol{r}_1 = [3, 2, 2]$, $g^* = \boldsymbol{x}_1 - 1 = 3 - 1 = 2$

$$\begin{aligned} C_1(1, \boldsymbol{r}_1 - \boldsymbol{e}_1) &= \frac{6!}{2! \cdot 2! \cdot 2!} = 90 \\ C_2(1, \boldsymbol{r}_1 - \boldsymbol{e}_2) &= \frac{6!}{3! \cdot 2!} = 60 \\ C_3(1, \boldsymbol{r}_1 - \boldsymbol{e}_3) &= \frac{6!}{3! \cdot 2!} = 60 \end{aligned}$$

Further, $m = 0 + S_2 = 150$. $r_3 = 2 - 1 = 1$

3. $t = 2$, $\boldsymbol{r}_2 = [3, 2, 1]$, $g^* = \boldsymbol{x}_2 - 1 = 2 - 1 = 1$

$$\begin{aligned} C_1(2, \boldsymbol{r}_2 - \boldsymbol{e}_1) &= \frac{5!}{2! \cdot 2!} = 30 \\ C_2(2, \boldsymbol{r}_2 - \boldsymbol{e}_2) &= \frac{5!}{3! \cdot 1!} = 20 \\ C_3(2, \boldsymbol{r}_2 - \boldsymbol{e}_3) &= \frac{5!}{3! \cdot 2!!} = 10 \end{aligned}$$

Further, $m = 150 + 30 = 180$. $r_2 = 2 - 1 = 1$

4. $t = 3$, $\boldsymbol{r}_3 = [3, 1, 1]$, $g^* = \boldsymbol{x}_3 - 1 = 1 - 1 = 0$

$$\begin{aligned} C_1(3, \boldsymbol{r}_3 - \boldsymbol{e}_1) &= \frac{4!}{2!} = 12 \\ C_2(3, \boldsymbol{r}_3 - \boldsymbol{e}_2) &= \frac{4!}{3!} = 4 \\ C_3(3, \boldsymbol{r}_3 - \boldsymbol{e}_3) &= \frac{4!}{3!} = 4 \end{aligned}$$

Further, $m = 180 + 0 = 180$. $r_1 = 3 - 1 = 2$

5. $t = 4$, $\boldsymbol{r}_4 = [2, 1, 1]$, $g^* = \boldsymbol{x}_4 - 1 = 1 - 1 = 0$

$$\begin{aligned} C_1(4, \boldsymbol{r}_4 - \boldsymbol{e}_1) &= \frac{3!}{1!} = 6 \\ C_2(4, \boldsymbol{r}_4 - \boldsymbol{e}_2) &= \frac{3!}{2!} = 3 \\ C_3(4, \boldsymbol{r}_4 - \boldsymbol{e}_3) &= \frac{3!}{2!} = 3 \end{aligned}$$

Further, $m = 180 + 0 = 180$. $r_1 = 1 - 1 = 1$

6. $t = 5$, $\boldsymbol{r}_5 = [1, 1, 1]$, $g^* = \boldsymbol{x}_5 - 1 = 2 - 1 = 1$

$$\begin{aligned} C_1(5, \boldsymbol{r}_5 - \boldsymbol{e}_1) &= 2! = 2 \\ C_2(5, \boldsymbol{r}_5 - \boldsymbol{e}_2) &= 2! = 2 \\ C_3(5, \boldsymbol{r}_5 - \boldsymbol{e}_3) &= 2! = 2 \end{aligned}$$

7. $t = 6$, $\boldsymbol{r}_6 = [1, 0, 1]$, $g^* = \boldsymbol{x}_6 - 1 = 1 - 1 = 0$.

$$\begin{aligned} C_1(6, \boldsymbol{r}_6 - \boldsymbol{e}_1) &= 1! = 1 \\ C_2(6, \boldsymbol{r}_6 - \boldsymbol{e}_2) &= \emptyset \\ C_3(6, \boldsymbol{r}_6 - \boldsymbol{e}_3) &= 1! = 1 \end{aligned}$$

Further, $m = 182 + 0 = 182$. $r_2 = 1 - 1 = 0$

8. Because $\boldsymbol{x}_7 = 3$ and $r_3 \to 0$, $m = 182$.

Probability Rounding & index-swapping

Section 8.4.2.2

Integer power of 2 multinomial?

- **Wastes no bandwidth:** The input bits map to each and every possible multinomial combination.
  - This requires some further tuning of the shell probabilities,
  - which is possible for reasonable n with very little rate loss.

$$M(\boldsymbol{n}) \triangleq \frac{n!}{\prod_{g=1}^{G} n_g} \qquad L \triangleq \log_2 M(\boldsymbol{n}) \qquad k \triangleq \lfloor L \rfloor$$

- **Deficit:** The amount of power-of-2 miss $\Delta = n - k$.
  - $n_i \to n_i + 1$ for the original $p_i \cdot n$ that are closest to the upper integer (recall these all truncated earlier).
  - This provides a sum that adds exactly to n (so the multinomial works), but reduces the entropy/rate/
- **Index swapping:** Then pairs of (i, j) are found where $n_i \to n_i + 1$ and $n_j \to n_j - 1$
- The improvement in entropy is $\Delta_{ij} = \log_2\left(\frac{n_j}{n_i + 1}\right)$ see proof in Section 8.4.2.2
- **Greedy Selection (like bit swapping, Chap 4):** swap in positions with largest entropy benefit first.

$$\Delta L(\boldsymbol{n}) = \left| L(\boldsymbol{n}) - \underbrace{\left\lfloor \log_2(\boldsymbol{n}) - \sum_{g=1}^{G} \log_2(n_i) \right\rceil}_{rounded\ value} \right|;$$

Program to index swap

- Power2_multinomial_round.m

```
[r, info] = power2_multinomial_round(p, n, max_moves)
   p        : probability vector (will be normalized to sum=1)
   n        : blocklength (integer)
   max_moves  : optional cap on unit-transfer moves (default 8)

 Outputs:
   r        : integer composition (sum(r)=n)
   info     : struct with fields
     .q_input, .q_rounded, .q_final
     .H_input_bits, .H_rounded_bits, .H_final_bits
     .deltaH_rounded, .deltaH_final
     .L0, .L, .k
     .moves   : [i j delta] rows (unit transfers j->i with delta added to log2 M)

 Notes:
  - Each unit transfer updates log2 M by delta = log2(r_j / (r_i+1)).
  - Entropies are base-2; 0*log2(0) is treated as 0.
```

```
>> out=qam_shell_shaping(256,15,5.6)
shaping_gain_dB: 1.4375
       H_achieved_bits: 5.6000

>>  temp=out.shell_prob'
>> shellprob2=[temp(1:5),(2/3)*temp(6) , (1/3)*temp(6) , temp(7:15)]
>>  [r,info]=power2_multinomial_round(shellprob2,256)
>> r' % =
  53   73   26   37   26   13    6    9    4    3    2    1    1    1    1    0

info =   struct with fields:
        q_input: [16x1 double]
      q_rounded: [16x1 double]
        q_final: [16x1 double]
    H_input_bits: 2.9304
   H_rounded_bits: 2.9219
    H_final_bits: 2.9238
   deltaH_rounded: -0.0085
     deltaH_final: -0.0066
             L0: 712.5086
              L: 712.9901
              k: 713
          moves: [1 2 0.4815]
```

- Very small loss (.02 dB) in shaping gain for the rounded replacement probabilities.
- Retains shaping gain of 1.43 dB, which is near the limit

End Lecture S12A