Quantizador Vetorial com Restrição de Entropia

José Gabriel R. C. Gomes

UFRJ / COPPE

CPE718 – Aula #9 – Parte I

1. Codificador de Fonte Discreta (VLC)

- Função $\gamma : \{1, \ldots, K\} \longrightarrow [0, 1, \times]^L$

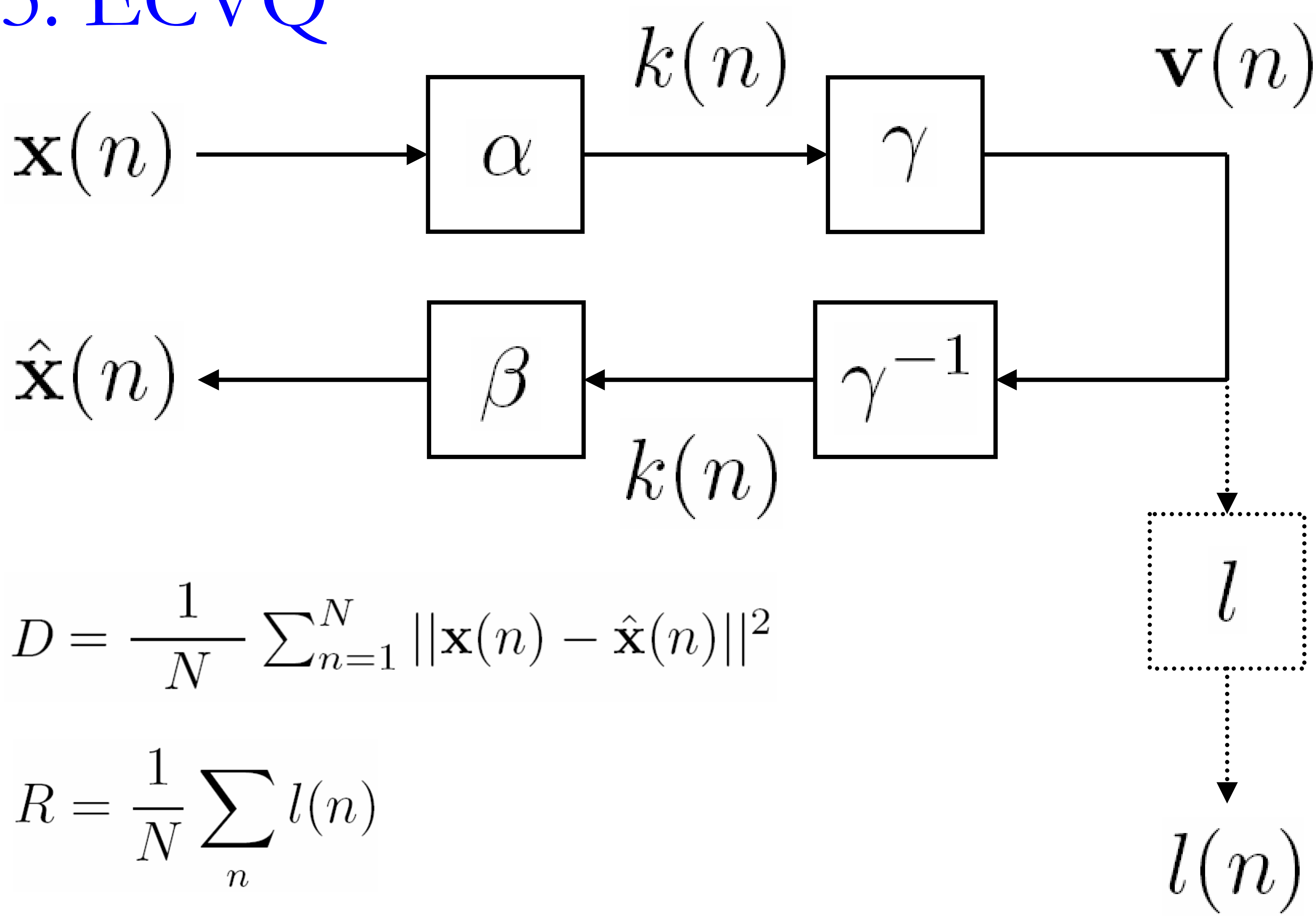
$$k(n) \longrightarrow \boxed{\gamma} \longrightarrow \mathbf{v}(n)$$

2. Decodificador de Fonte Discreta

- Função $\gamma^{-1} : [0, 1, \times]^L \longrightarrow \{1, \ldots, K\}$

$$\mathbf{v}(n) \longrightarrow \boxed{\gamma^{-1}} \longrightarrow k(n)$$

3. ECVQ



$$D = \frac{1}{N} \sum_{n=1}^{N} ||\mathbf{x}(n) - \hat{\mathbf{x}}(n)||^2$$

$$R = \frac{1}{N} \sum_{n} l(n)$$

4. Entropia

$$p_c = \frac{1}{N} \text{card}\{k(n) | k(n) = c\}, \; c = 1, \ldots, K$$

- O VLC γ é parametrizado pelo vetor $\mathbf{p}$.

$$H = -\sum_k p_k \log_2 p_k$$

$$H < R < H + \frac{1}{m} \qquad\qquad m = 1$$

4. Entropia e VLC – Exemplo

```
>> p = [1/2 1/4 1/8 1/8]

p =

    0.5000    0.2500    0.1250    0.1250

>> -sum(p.*log2(p))

ans =

    1.7500

>> l = HuffLen(p)

l =

     1     2     3     3

>> gamma = HuffCode(l)

gamma =

     0     0     0
     1     0     0
     1     1     0
     1     1     1
```

5. Problema (Projeto do ECVQ)

- Minimizar D, considerando-se uma restrição (valor máximo aceitável) em H, ou vice-versa. Isso equivale a minimizar a função custo:

$$J = D + \lambda H$$

- Problema: dados **X** e λ, encontrar **Y** tal que $J = J_{min}$.

- Exemplo (MATLAB): Slides #5 e #6 da aula passada.

6. Considerações Básicas

6.1. Condição da Partição (Codificador)

Fixando **Y**, calcular divisão de **X** em *K* sub-conjuntos

6.2. Condição do Centróide (Decodificador)

Fixando divisão de **X**, calcular **Y**

6.1. Condição da Partição

- Considerando que $\mathbf{y}_1$, $\mathbf{y}_2$, ..., $\mathbf{y}_K$ são dados, seja j($\mathbf{x}$,$\mathbf{y}_k$):

$$j(\mathbf{x}, \mathbf{y}_k) = ||\mathbf{x} - \mathbf{y}_k||^2 + \lambda l_k$$

- Partição ótima (sem demonstração aqui):

$$R_i = \{\mathbf{x} \mid j(\mathbf{x}, \mathbf{y}_i) < j(\mathbf{x}, \mathbf{y}_m) \ \forall m \neq i\}$$

- Neste caso: $j(\mathbf{x}, \hat{\mathbf{x}}) = \min_{k \in \{1,2,\ldots,K\}} j(\mathbf{x}, \mathbf{y}_k)$

6.1. Condição da Partição (MATLAB)

- Na primeira iteração, não sabemos valores de l_k.

```
>> X

X =

  Columns 1 through 12

    0.9285    0.9564    0.8928    1.0096    0.9665    0.5975    0.5774    0.6000    0.6602    0.6021    0.9060    0.9270
    0.1479    0.2455    0.2907    0.2293    0.2399    0.5223    0.5951    0.4917    0.4889    0.4444    0.6953    0.8433

  Columns 13 through 20

    0.8567    0.9540    0.8193    0.4365    0.4972    0.5210    0.5160    0.4555
    0.8050    0.6824    0.7907    0.0530    0.0541    0.0519   -0.0416    0.0107

>> L

L =

     2     2     2     2

>> lambda = 0.05;
>> J = 0;
>> for n=1:20, j = sum((repmat(X(:,n),1,size(Y,2)) - Y).^2,1) + lambda*L; k(n) = min(find(j==min(j))); J = J + min(j); end;
>> J = J/20

J =

    0.1899

>> k

k =

     4     4     4     4     4     1     1     1     1     1     3     1     1     3     1     2     2     2     2     2
```

6.2. Condição do Centróide

- É a mesma do Slide #12 da aula passada:

$$\mathbf{y}_k = \frac{1}{N_k} \sum_{\mathbf{x}(n) \in R_k} \mathbf{x}(n)$$

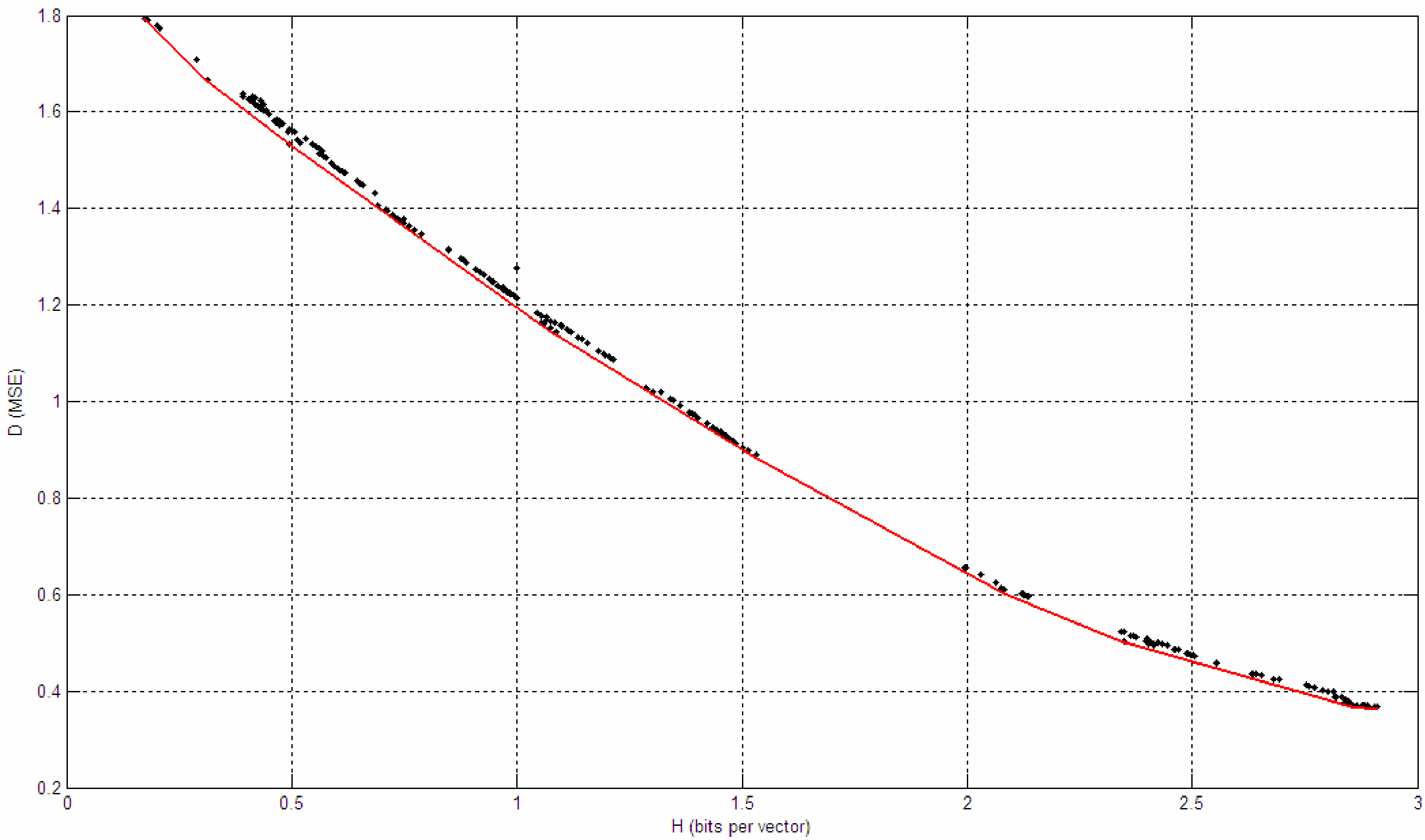
6.2. Condição do Centróide (MATLAB)

```
>> k

k =

     4     4     4     4     4     1     1     1     1     1     3     1     1     3     1     2     2     2     2     2

>> p = zeros(K,1); Y = zeros(size(Y)); for n=1:20, Y(:,k(n)) = Y(:,k(n)) + X(:,n); p(k(n)) = p(k(n)) + 1; end;
>> p

p =

     8
     5
     2
     5

>> for j=1:K, Y(:,j) = Y(:,j)/p(j); end;
>> Y

Y =

    0.7050    0.4852    0.9300    0.9508
    0.6227    0.0256    0.6888    0.2306

>> plot(Y(1,:),Y(2,:),'g.');
>> D = 0; for n=1:20, D = D + sum((X(:,n)-Y(:,k(n))).^2);
end;
>> D = D/20

D =

    0.0178
```

6.2. E logo após a avaliação de **p**:

```
>> p = p/sum(p)

p =

    0.4000
    0.2500
    0.1000
    0.2500

>> L = HuffLen(p)

L =

     1
     2
     3
     3
```

6.3. Exemplo ECVQ



6.3. Exemplo ECVQ

```
clear all; close all; S = 0.01; BKJ = [];

for s = 1:400,

    lambda = S*(s-1);
    randn('state',0); rand('state',0); M = 2; N = 800; K = 8; e = 0.5;
    X = randn(M,N);
    Y = 0.5*randn(M,K);
    l = log2(K)*ones(1,size(Y,2));
    F = 200; BK = zeros(F,4);

    for i=1:F,
        % Partition
        J = 0; for n=1:N, j = sum((repmat(X(:,n),1,size(Y,2)) - Y).^2,1) + lambda*l;
                   k(n) = min(find(j==min(j))); J = J + min(j); end; J = J/N;
        % Centroid
        p = zeros(K,1); Y = zeros(size(Y)); for n=1:N, Y(:,k(n)) = Y(:,k(n)) + X(:,n);
                                               p(k(n)) = p(k(n)) + 1; end;
        for j=1:K, if p(j)~=0, Y(:,j) = Y(:,j)/p(j); end; end;
        % Cost Evaluation
        D = 0; for n=1:N, D = D + sum((X(:,n)-Y(:,k(n))).^2); end; D = D/N;
        Y = Y(:,find(p~=0)); p = p(find(p~=0));
        p = p/sum(p); H = -sum(p.*log2(p)); BK(i,:) = [D H D+lambda*H J];
        % Codeword Length Update
        l = HuffLen(p)';
    end;

    BKJ = [BKJ ; [lambda D H D+lambda*H size(Y,2)]]; [s lambda D H D+lambda*H size(Y,2)]

end;

plot(BKJ(:,3),BKJ(:,2),'k.'); grid on; xlabel('H (bits per vector)'); ylabel('D (MSE)');
```