EE368B – Image and Video Compression

Solution to Homework Set #2

Prepared by Markus Flierl

1 Lloyd-Max Quantizer

To solve the problem, it is sufficient to design the quantizer with the pmf of the training set. For initialization, we choose a uniform codebook. We selected the quantizer output not to be integer-valued.

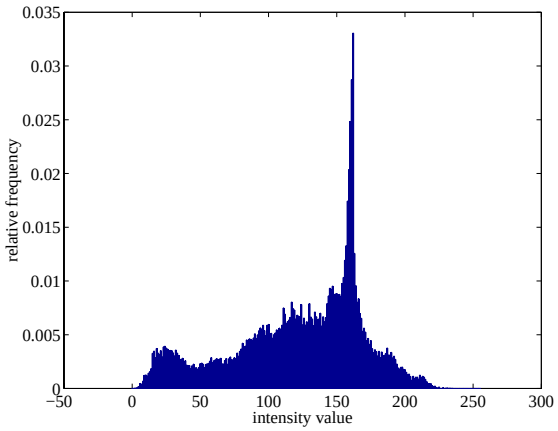

Figure 1: Marginal probabilities for the training set *boats, harbour,* and *peppers.*

During the iterative optimization process, one or more codeword(s) might end up having no samples mapped to it, forming empty quantization cell(s). One can choose to remove the codeword in an empty quantization cell which results in a smaller codebook, or even better, to replace the codeword in an empty cell with a new codeword, generated by splitting the most populated cell. The latter is usually better for fixed-length code since it does not waste any codeword.

The rate-PSNR pairs are plotted in Fig. 6. The Lloyd-Max quantizer with variable length code (VLC) applies an optimal entropy code for transmitting the indices. The slope of the Lloyd-Max quantizer with FLC is 5.2 dB/bit and with VLC 5.8 dB/bit. The 3-bit Lloyd-Max quantizer converges after 13 iterations, the 4-bit Lloyd-Max quantizer after 18 iterations.

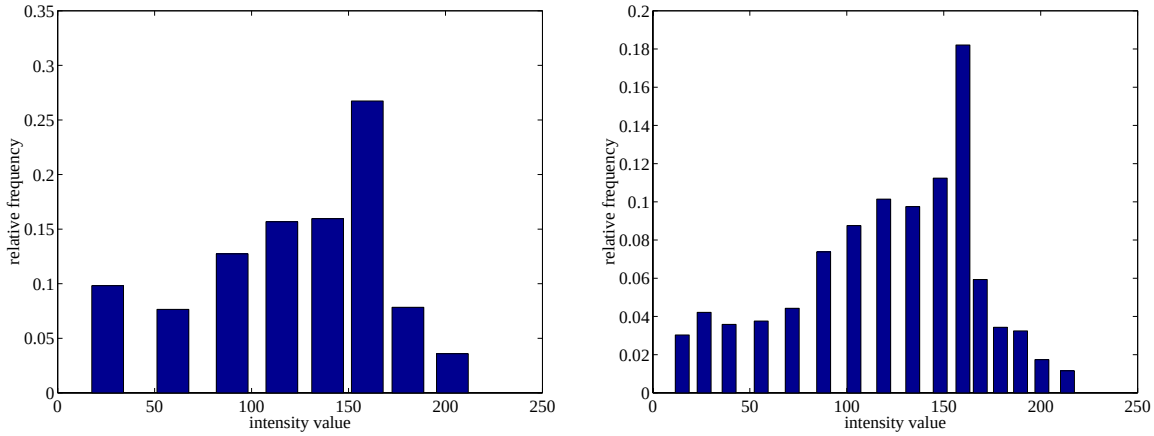

Figure 2: Marginal probabilities of the quantized training set *boats, harbour,* and *peppers*. The left plot shows the probabilities of 3-bit Lloyd-Max quantizer output, the right the probabilities of the 4-bit Lloyd-Max quantizer output.

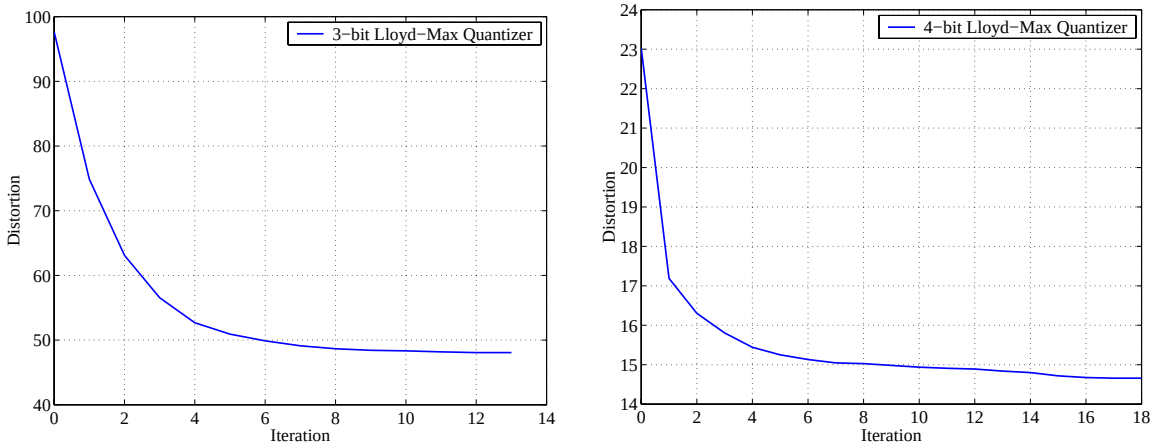

Figure 3: Convergence of the Lloyd-Max quantizer design. The left plot shows the 3-bit, the right the 4-bit Lloyd-Max quantizer.

2 Entropy-Constrained Scalar Quantization

The entropy-constrained scalar quantizer design algorithm incorporates the design of the variable length code into the design-loop. As we are optimizing the codebook with the entropy-constraint, each codeword is associated with an entropy code and thus a variable code length which minimizes entropy.

In the entropy-constrained algorithm, removing a codeword does not cause any waste in bits, whereas replacing a codeword by splitting another cell might actually increase entropy. So we simply remove the codeword in an empty cell.

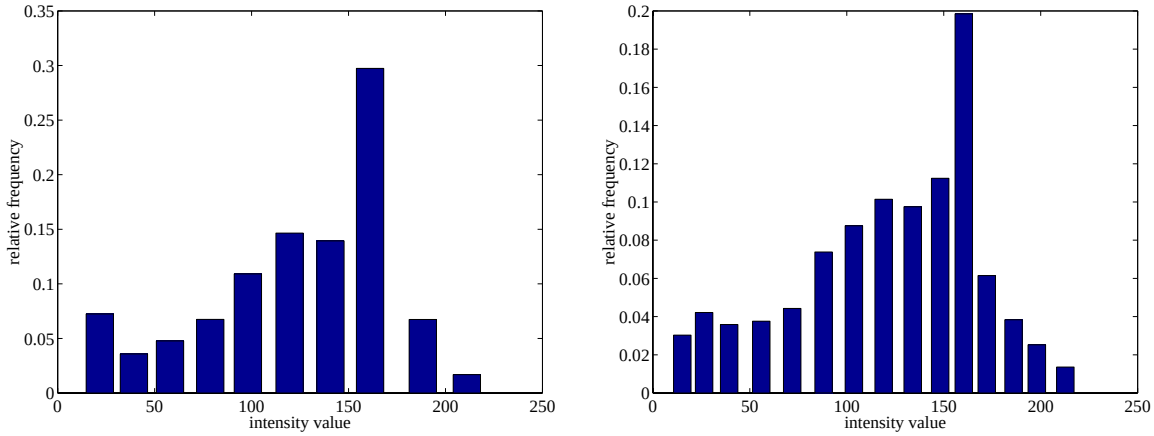

Figure 4: Marginal probabilities for the entropy constrained scalar quantizer output for the training set *boats, harbour,* and *peppers.* The left plot shows the probabilities for $\lambda = 70$, the right for $\lambda = 0$.

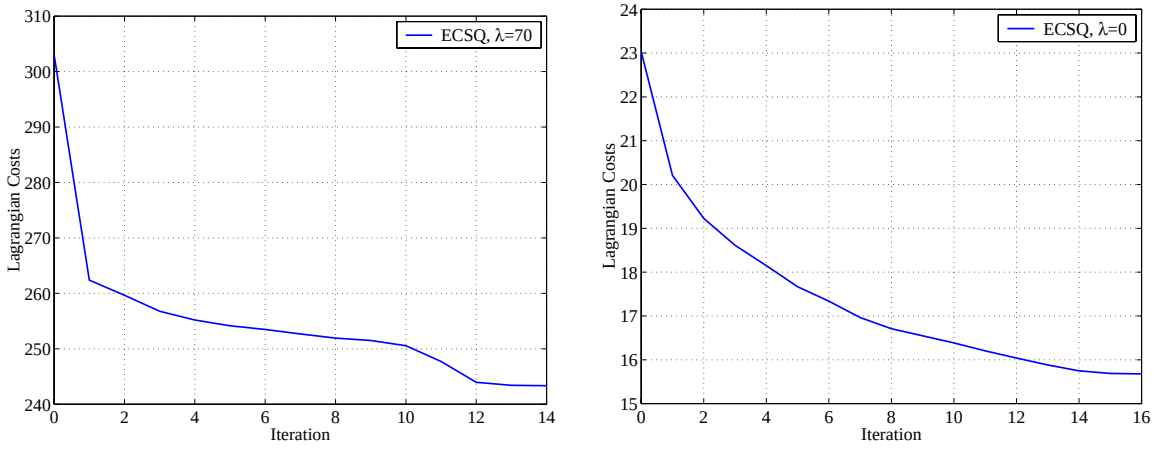

Figure 5: Convergence of the ECSQ design. The left plot shows the case $\lambda = 70$, the right $\lambda = 0$.

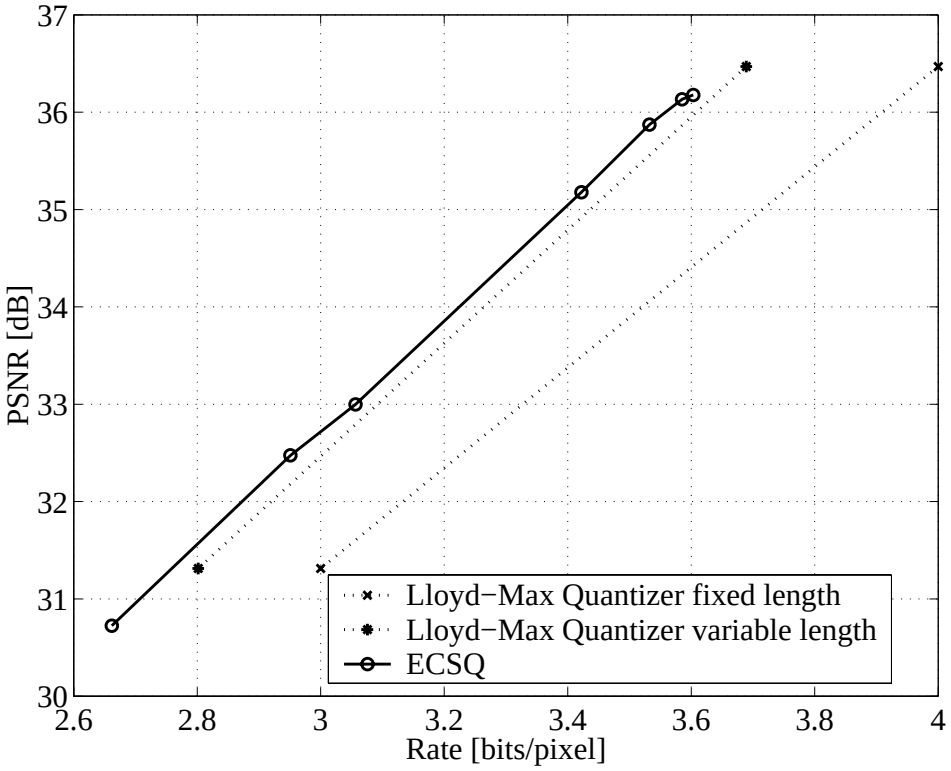

Figure 6: Rate distortion function for ECSQ and Lloyd-Max quantizer. The training set consists of the subsampled images *boats, harbour,* and *peppers.*

The rate-PSNR pairs for ECSQ are plotted in Fig. 6. Peaky distributions take advantage of this joint optimization. The slope of the ECSQ is about 5.9 dB/bit. The ECSQ design algorithm converges within 8 to 16 iterations. The individual numbers are depicted in Table 1.

λ	0	5	10	30	62	70	90
Iterations	16	15	10	8	14	14	14

Table 1: Number of steps ECSQ takes to convergence.

A Script for Problem 1 and 2

```
%  SHOWRD
%
%  EE368B - Image and Video Compression
%
%  Markus Flierl, 2000-10-11

lambda = [0 5 10 30 62 70 90]
n = length(lambda);
```

```
R    = zeros(1,n);
D    = zeros(1,n);
PSNR = zeros(1,n);
cnt  = zeros(1,n);
for i = 1:n
  lambda(i)
  [R(i), D(i), PSNR(i), cnt(i), costs{i}] = ecsq1(lambda(i));
end;

[R3, D3, PSNR3, H3, cnt3, costs3] = lloydmax1(3);
[R4, D4, PSNR4, H4, cnt4, costs4] = lloydmax1(4);

slopeFLC = (PSNR4-PSNR3)/(R4-R3)
slopeVLC = (PSNR4-PSNR3)/(H4-H3)
slopeEC  = (PSNR(3)-PSNR(7))/(R(3)-R(7))

plot(...
    [R4 R3], [PSNR4 PSNR3], ':x', ...
    [H4 H3], [PSNR4 PSNR3], ':*', ...
    R, PSNR, '-o', ...
    'linewidth', 1.5);

set(gca, 'fontname', 'Times');
set(gca, 'fontsize', 16);
xlabel('Rate [bits/pixel]');
ylabel('PSNR [dB]')

grid;
h = legend(...
    'Lloyd-Max Quantizer fixed length', ...
    'Lloyd-Max Quantizer variable length', ...
    'ECSQ', 4);

figure;
plot(0:cnt3, costs3, 'linewidth', 1.5);
set(gca, 'fontname', 'Times');
set(gca, 'fontsize', 16);
ylabel('Distortion');
xlabel('Iteration');
grid;
h = legend('3-bit Lloyd-Max Quantizer', 1);

figure;
plot(0:cnt4, costs4, 'linewidth', 1.5);
set(gca, 'fontname', 'Times');
set(gca, 'fontsize', 16);
ylabel('Distortion');
xlabel('Iteration');
grid;
h = legend('4-bit Lloyd-Max Quantizer', 1);

figure;
plot(0:cnt(4), costs{4}, 'linewidth', 1.5);
set(gca, 'fontname', 'Times');
set(gca, 'fontsize', 16);
ylabel('Lagrangian Costs');
xlabel('Iteration');
grid;
h = legend('ECSQ, \lambda=30', 1);

cnt
```

B Function lloydmax1.m

```
function [R, D, PSNR, HY, cnt, costs] = lloydmax1(rate)
%LLOYDMAX Lloyd-Max Quantizer
%       pX : input pmf
%       CB : Codebook:
%            1. Column    3. Column
%            Centroids    Length of Codeword [bit]
%                .           .
%               -1           3
%                0           1
%                1           3
%                .           .
%        I : Index-vector addressing valid centroids of codebook CB
%       pY : output pmf
%
%       See also

%       Author(s): M. Flierl, 2000-10-11

% trainingsset
noi   = 3;
name1 = 'Images/boats512x512.tif';
name2 = 'Images/harbour512x512.tif';
name3 = 'Images/peppers512x512.tif';

img = zeros(512,512, noi);
for i=1:noi
  file = eval(['name' num2str(i)]);
  img(:,:,i) = imread(file, 'tiff');
end;
img1 = double(img(1:4:512,1:4:512,:));
X = img1(:)';

% pmf of histogram
bins = 0:255;
pX = zeros(length(bins),2);
fX = hist(X,bins);
pX(:,1) = bins';
pX(:,2) = fX'/sum(fX);

% parameter
lambda = 0.0;
epsilon = 0.001;

% Entropy of X
HX = sum(-pX(:,2).*log2(pX(:,2)+eps));

if(0)
  figure;
  bar(pX(:,1),pX(:,2));
  set(gca, 'fontname', 'Times');
  set(gca, 'fontsize', 16);
  xlabel('intensity value');
  ylabel('relative frequency')
end;

% Initialize
CB = initialcodebook(0,255,2^rate);

% Encode
[pY,I,Jold] = encode1(pX,CB,lambda);

cnt = 0.0;
J   = Jold;
costs(cnt+1) = J;
error = 1.0;
```

```
% disp('      cnt       J       error');
% disp([cnt J error]);
while error>epsilon,
  % Update
  CB = updatecodebookLM1(pX,I,CB);

  % Encode
  cnt = cnt + 1;
  [pY,I,Jnew] = encode1(pX,CB,lambda);
  error=(Jold-Jnew)/Jold;
  Jold = Jnew;

  J = Jold;
  costs(cnt+1) = J;
%  disp([cnt J error]);
end;

if(0)
  figure;
  bar(pY(:,1),pY(:,2));
  set(gca, 'fontname', 'Times');
  set(gca, 'fontsize', 16);
  xlabel('intensity value');
  ylabel('relative frequency')
end;

% Entropy of Codebook
HCB = CB(:,2)'*(2.^(-CB(:,2)));

% Entropy of Reconstruction
HY = sum(-pY(:,2).*log2(pY(:,2)+eps));

% [HCB,HY]

R = HCB;

% Distortion
D  = 0.0;
for l = 1:length(pX)
  D = D + pX(l,2) * ((pX(l,1)-CB(I(l),1)).^2);
end;

SNR = 10*log10(var(X)/D);

PSNR = 10*log10(255*255/D);
```

C Function initialcodebook.m

```
function CB = initialcodebook(a,b,s);
%INITIALCODEBOOK Generates initial codebook
%       CB = INITIALCODEBOOK(a,b,s)
%
%       a: left border
%       b: right border
%       s: size of codebook
%       CB : Codebook:
%            1. Column    2. Column
%            Centroids    Lenth of Codeword [bit]
%                .           .
%               -1           3
%                0           1
%                1           3
%                .           .
%
```

```
%       See also UPDATECODEBOOK

%       Author(s): M. Flierl, 2000-10-11

CB = zeros([s,2]);

delta = (b-a)/s;
for i = 1:s
  CB(i,1) = a - delta/2 + i*delta;
end;

CB(:,2) = ones([s,1]);
CB(:,2) = -log2(CB(:,2)/sum(CB(:,2)));
```

D Function encode1.m

```
function [pY,I,J] = encode1(pX,CB,lambda);
%ENCODE Encodes pmf pX with Codebook CB and returns the index-vector I
%       and output pmf pY
%       [pY,I,J] = ENCODE(pX,CB,lambda)
%
%       pX : input pmf
%       CB : Codebook:
%            1. Column   2. Column
%            Centroids   Length of Codeword [bit]
%                .           .
%               -1           3
%                0           1
%                1           3
%                .           .
%       lambda : Lagrange-Parameter
%       I  : Index-vector addressing valid centroids of codebook CB
%       pY : pmf of quantizer output
%       J  : Lagragian costs
%
%       See also DECODE.

%       Author(s): M. Flierl, 2000-10-11

pY = zeros(size(CB));
I  = zeros(1,length(CB));
J  = 0.0;

pY(:,1) = CB(:,1);
n = size(CB(:,1));
for l = 1:length(pX)
  JJ = (pX(l,1)-CB(:,1)).^2+lambda*CB(:,2);
  [Jmin,I(l)] = min(JJ);
  J = J + pX(l,2)*Jmin;
  pY(I(l),2) = pY(I(l),2) + pX(l,2);
end;
```

E Function updatecodebookLM1.m

```
function CBK = updatecodebookLM1(pX,I,CB);
%UPDATECODEBOOKLM Generates updated codebook
%       CBK = UPDATECODEBOOKLM(pX,I,CB)
%
%       CBK : New Codebook
%       CB : Codebook containing centroids and codeword-length
```

```
%            1. Column    2. Column
%            Centroids    Lenth of Codeword [bit]
%                .           .
%               -1           3
%                0           1
%                1           3
%                .           .
%
%       See also INITIALCODEBOOK

%       Author(s): M. Flierl, 2000-10-11

C = zeros(size(CB));

for l = 1:length(pX)
  C(I(l),1) = C(I(l),1) + pX(l,1)*pX(l,2);
  C(I(l),2) = C(I(l),2) + pX(l,2);
end;

% remove empty cells
n = size(CB(:,1));
ii = 1;
for i = 1 : n
  if C(i,2) > eps
    CBK(ii,:) = C(i,:);
    ii = ii + 1;
  end;
end;

% centroids
CBK(:,1) = CBK(:,1)./CBK(:,2);

% split the most populated cell
sze = size(CBK(:,1)); lgh = sze(1);
while(lgh<n(1))
  disp('Split most populated cell');
  [cnt, idx] = max(CBK(:,2));
  sze = size(CBK(:,1)); lgh = sze(1);
  CBK((idx:lgh)+1,:) = CBK(idx:lgh,:);
  CBK(idx,1) = 0.75*CBK(idx,1)+0.25*CBK(idx-1,1);
  CBK(idx,2) = CBK(idx,2)/2;
  CBK(idx+1,1) = 0.75*CBK(idx+1,1)+0.25*CBK(idx+2,1);
  CBK(idx+1,2) = CBK(idx,2);
  sze = size(CBK(:,1)); lgh = sze(1);
end;

CBK(:,2) = CB(:,2);
```

F Function ecsq1.m

```
function [R, D, PSNR, cnt, costs] = ecsq1(lambda)
%ECSQ   Entropy contrained scalar quantization
%       pX : input pmf
%       CB : Codebook:
%            1. Column    3. Column
%            Centroids    Length of Codeword [bit]
%                .           .
%               -1           3
%                0           1
%                1           3
%                .           .
%       I  : Index-vector addressing valid centroids of codebook CB
%       pY : output pmf
%
```

```
%       See also

%       Author(s): M. Flierl, 2000-10-11

% trainingsset
noi   = 3;
name1 = 'Images/boats512x512.tif';
name2 = 'Images/harbour512x512.tif';
name3 = 'Images/peppers512x512.tif';

img = zeros(512,512, noi);
for i=1:noi
  file = eval(['name' num2str(i)]);
  img(:,:,i) = imread(file, 'tiff');
end;
img1 = double(img(1:4:512,1:4:512,:));
X = img1(:)';

% pmf of histogram
bins = 0:255;
pX = zeros(length(bins),2);
fX = hist(X,bins);
pX(:,1) = bins';
pX(:,2) = fX'/sum(fX);

% parameter
epsilon = 0.001;

% Entropy of X
HX = sum(-pX(:,2).*log2(pX(:,2)+eps));

if(0)
  figure;
  bar(pX(:,1), pX(:,2));
  set(gca, 'fontname', 'Times');
  set(gca, 'fontsize', 16);
  xlabel('intensity value');
  ylabel('relative frequency')
end;

% Initialize
CB = initialcodebook(0,255,16);

% Encode
[pY,I,Jold] = encode1(pX,CB,lambda);

cnt = 0.0;
J   = Jold;
costs(cnt+1) = J;
error = 1.0;
% disp('      cnt       J       error');
% disp([cnt J error]);
while error>epsilon,
  % Update
  CB = updatecodebook1(pX,I,CB);

  % Encode
  cnt = cnt + 1;
  [pY,I,Jnew] = encode1(pX,CB,lambda);
  error=(Jold-Jnew)/Jold;
  Jold = Jnew;

  J = Jold;
  costs(cnt+1) = J;
%  disp([cnt J error]);
end;

if(0)
```

```
  figure;
  bar(pY(:,1),pY(:,2));
  set(gca, 'fontname', 'Times');
  set(gca, 'fontsize', 16);
  xlabel('intensity value');
  ylabel('relative frequency')
end;

% Entropy of Codebook
HCB = CB(:,2)'*(2.^(-CB(:,2)));

% Entropy of Reconstruction
HY = sum(-pY(:,2).*log2(pY(:,2)+eps));

% [HCB,HY]

R = HCB;

% Distortion
D  = 0.0;
for l = 1:length(pX)
  D = D + pX(l,2) * ((pX(l,1)-CB(I(l),1)).^2);
end;

SNR = 10*log10(var(X)/D);

PSNR = 10*log10(255*255/D);
```

G Function updatecodebook1.m

```
function CBK = updatecodebook1(pX,I,CB);
%UPDATECODEBOOK Generates updated codebook
%       CBK = UPDATECODEBOOK(pX,I,CB)
%
%       CBK : New Codebook
%       CB : Codebook containing centroids and codeword-length
%            1. Column    2. Column
%            Centroids    Lenth of Codeword [bit]
%                .           .
%               -1           3
%                0           1
%                1           3
%                .           .
%
%       See also INITIALCODEBOOK

%       Author(s): M. Flierl, 2000-10-11

C = zeros(size(CB));

for l = 1:length(pX)
  C(I(l),1) = C(I(l),1) + pX(l,1)*pX(l,2);
  C(I(l),2) = C(I(l),2) + pX(l,2);
end;

% remove empty cells
n = size(CB(:,1));
ii = 1;
for i = 1 : n
  if C(i,2) > eps
    CBK(ii,:) = C(i,:);
    ii = ii + 1;
  end;
end;
```

```
% centroids
CBK(:,1) = CBK(:,1)./CBK(:,2);
CBK(:,2) = -log2(CBK(:,2));
```