Short introduction to the Scilab toolbox "wavelib"

Alexander Stoffel

Institute of Communication Engineering
Faculty of Information, Media and Electrical Engineering
Cologne University of Applied Sciences
alexander.stoffel@fh-koeln.de

July 25, 2013

Note: This introduction essentially relies on section 24 of the german introduction "Kurzeinführung in Scilab" available at
`http://alex.nt.fh-koeln.de/mapdf/scilabein.pdf`

Contents

1 Getting started

The toolbox “wavelib” is available from

```
http://alex.nt.fh-koeln.de/wavelib.html
```

You will find there the link for downloading a archive file as well as a detailed instruction how to install the toolbox. Note that you have to load the toolbox each time you restart Scilab by executing the program “loader.sce” (the “builder.sce” has only to be executed once). You can avoid manual loading using a startup file - you only have to put there the command exec(.....loader.sce). For the detail, see the hints in the help pages chapter “startup” (use the search facility of the help browser or type “help startup” in the console).

You get a list of the functions in the toolbox from the content of the corresponding chapter in the left column of the help browser. Furthermore, you will find an example program (file name “function name+ex.sce”) for each toolbox function in the toolbox directory “examples” and there in the subdirectory corresponding to the chapter in the help browser. For example you will find a sample program using the function `psihut` in the file `psihutex.sce` in the subdirectory `examples/functions`. The path to the toolbox directory is available as the content of the variable `WLHOME`. This means that you may execute the example program typing

```
exec(WLHOME+'/examples/functions/psihutex.sce')
```

in the console.

A detailed description for each function is available in the help browser; you may also get this description typing `help` *function name* in the console. Here only a short overview on the most frequently needed functions of the toolbox is given.

2 Scaling functions and wavelets

You get the function values of the wavelet “CDF(3,5)” by `y=psicdf35(x)` and `x` may be a vector. A plot of this function may be generated by

```
x=-3:0.01:4; y=psicdf35(x); plot(x,y);
```

You get the function values of the corresponding scaling functions by `y=phicdf35(x)` Analogously the function values of the Haar are available by `psihaar` and the values of the hat or CDF(2,2) wavelet by `psihut`. And you get the function values of the corresponding scaling functions replacing „`psi`“ by „`phi`“.

3 Wavelet transforms for one dimensional signals

3.1 One step transforms

By `y=sigwt(x,'Haar')` you get the normalized Haar wavelet transform of the vector `x`. You will find the high pass coefficients $d(k)$ in the “upper” half of the vector `y`. This means, if the input vector `x` has N components, the high pass coefficients $d(k)$ will be the components of `y` with index $\frac{N}{2}+1$ to N, and the low pass coefficients will be the components of `y` with index 1 bis $\frac{N}{2}$. This is shown in figure 1, where the numbering is corrected to the usual range from 0 to $N-1$; remember that Scilab indices go always from 1 to N. Note that N has to be even, the transform is not implemented for an odd number of input components of `x`. You obtain the original signal values from the coefficients `y` by the inverse transform `xr=sigwtinv(y,'Haar')`. Note that you should not expect in general that `xr` agrees exactly with the original input `x` due to roundoff errors.

You may choose the name of another wavelet as the second input parameter of `sigwt` , for insctance you get the wavelet transform with respect to the hat or CDF(2,2) wavelet by `y=sigwt(x,'hat')`, analogously the inverse transform is done by `xr=sigwtinv(y,'Hut')`. Table 1 shows which wavelet transforms are actually available.

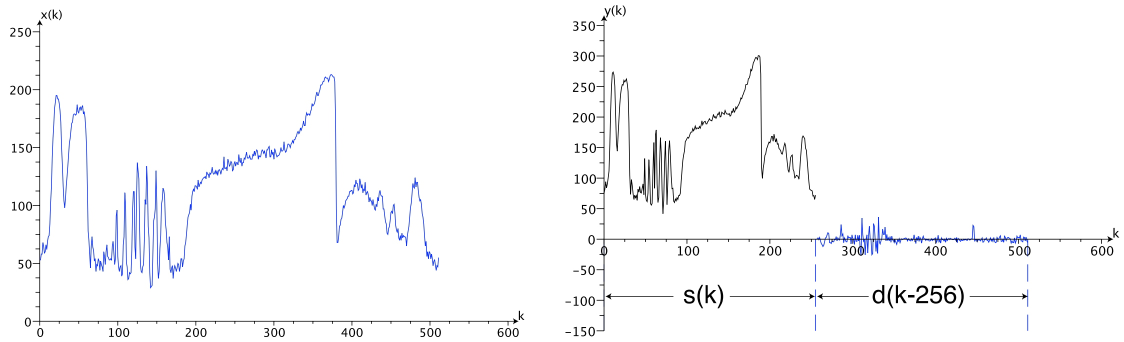


Figure 1: One dimensional signal `x` (at the left row 500 of the test image "Lena") and the result of the wavelet transform `y=sigwt(x,'Haar')` (at the right)

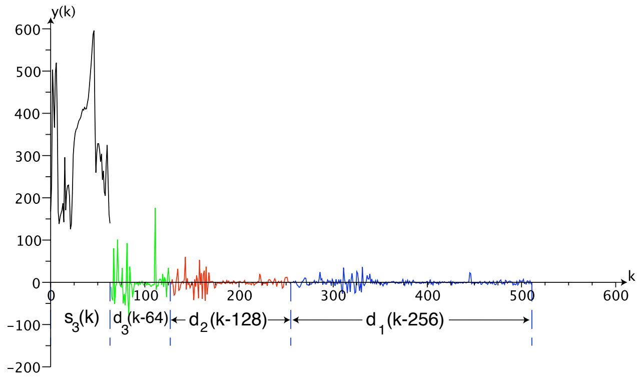


Figure 2: Result of the 3 step der filter bank transform `y=sigwt(x,'Haar',3)` appliedt to the same input signal as in figure 1

3.2 Cascaded transforms

You may use the low pass output of the first step (for example the first half of the vector +y=sigwt(x,'Haar')+) as an input of the same transform and even repeat this procedure several times. This may be done simply by `y=sigwt(x,'Haar',p)` where `p` denotes the number of cascades. The output of the high pass filter is always saved in the remaining "upper"half (with the higher indices) of the vector. The low pass output will be in the lowest part (smallest indices), it will be used as input in the next step. This makes it

wavelet name	second parameter	options	default
Haar wavelet	Haar	n,u	n
CDF(2,2), Hut	CDF(2,2), CDF22, Hut, hat	rr, zp, vm	rr
CDF(3,5)	CDF(3,5), CDF35	rr, zp	rr
DD(4,2), binlet	DD(4,2), DD42, bin	rr, zp, vm	rr
DD(4,4)	DD(4,4), DD44	rr, zp	rr
DD(8,6)	DD(8,6), DD86	rr, zp	rr
AI(3,3) (av. int.)	AI(3,3), AVI(3,3), AI33, AVI33	rr, zp	rr
AI(5,3) (av. int.)	AI(5,3), AVI(5,3), AI53, AVI53	rr, zp	rr
AI(5,5) (av. int.)	AI(5,5), AVI(5,5), AI55, AVI55	rr, zp	rr
FBI, 9/7	FBI	rr, zp	rr
D4 (orthogonal)	D4, Daub4, D(4)	rr, zp, pe, gs(1d)	gs(1d), rr(2d)
D6 (orthogonal)	D6, Daub6, D(6)	rr, zp, pe	pe
symlet	sym	pe, rr, zp	pe
QN(4), quincunx	quisu, quincunx	is ignored	

Table 1: Available wavelet transforms, the second and eventually the fourth parameter (column "options") have to be given as a string between single or double quotes. The case is ignored, instead of 'DD42' you may use 'dd42'. Note also the remarks in table 2.

necessary that the number of components of the input vector `x` has to be a multiple of 2^p (p, the third parameter, the number of cascades). Otherwise you will get an error message. Tghis leads to the following scheme of the output values in the output vector `y`:

type	Scilab indices		
$d_1(k)$	$\frac{N}{2}+1$	$\cdots$	N
$d_2(k)$	$\frac{N}{4}+1$	$\cdots$	$\frac{N}{2}$
$d_3(k)$	$\frac{N}{8}+1$	$\cdots$	$\frac{N}{4}$
$\vdots$		$\vdots$	
$d_p(k)$	$\frac{N}{2^p}+1$	$\cdots$	$\frac{N}{2^{p-1}}$
$s_p(k)$	1	$\cdots$	$\frac{N}{2^p}$

Note that $d_m(k)$ are the high pass output values and $s_p(k)$ the remaining low pass values. An increasing index m signifies a more coarse scale. This is shown in figure 2. The Scilab numbering starts at 1, but in the figure, the numbering is corrected to the usual convention starting at 0.

The inverse Transformation `xr=sigwtinv(y,'Haar',p)` furnishes the original signal values but only up to anavoidable roundoff errors.

Replacing the parameter `'Haar'` by `'hat'` one obtains the cascaded wavelet transform (and its inverse) with respect to the hat or CDF(2,2) wavelet. Analogously one gets the transform with respect with the other wavelet types shown in table 1. Remember that the case for the wavelet type is ignored, you may indicate `'fbi'` as well as `'FBI'` as the second parameter.

A remark is in order concerning the fourth parameter for `sigwt` and `sigwtinv` denoted "option" in table 1. It determines in general the treatment of the data at the boundary. It is strongly recommended not to specify this parameter, in that case the default value is used for which the best results are expected in the applications. You should only specify this parameter where you are explicitly asked for, for instance where experiments with the boundary conditions are intended. For the Haar wavelet this option determines whether the normalized version (default, filter constants $\pm\frac{1}{2}\sqrt{2}$) or the unnormalized version (filter constants $\pm\frac{1}{2}$ resp. ± 1) is used. Note that if you want to specify the fourth parameter, you always have to give explicitly the third parameter, even if you want to set the number of cascades to its default value 1.

For those who are interested in the details for the fourth parameter, here an explicit list of the signification of the different options.

'rr' (Reflection with repetition) Symmetric continuation by reflection with repetition at the boundary. This is the default option for most wavelet types.

'zp' (Zero padding) Continuation by 0 at the boundary (expressions in the formulae referring to an index outside the used range as $s(-1)$ are replaced by 0.

'pe' Periodic continuation of the data at the boundary. This option is actually only implemented for the orthogonal Daubechies wavelets D4, D6 and the symlet, it is the default option for D6 and the symlet.

'vm' (Vanishing moments) If $H(z)$ has a p-fold zero at $z = -1$, special boundary filters lead to a high pass output zero even at the boundary; analogously they lead to zero low pass output even at the boundary. This option is only implemented for the hat and the DD(4,2) wavelet and leads to numerical problems if the transform is cascaded (third parameter in `sigwt` or `sigwtinv` greater than 1).

'n' normalized transform, only admitted for the one dimensional Haar wavelet where this option is the default. It leads to filter constants $\pm\frac{1}{2}\sqrt{2}$.

'u' not normalized transform, only admitted for the one dimensional Haar wavelet, it leads to filter constants $\pm\frac{1}{2}$ and ± 1.

If you specify an invalid fourth paramter, it will be replaced by the default value, in general with a warning. For the symlet you will get a warning specifying other parameter values as the default `'pe'`, as they may cause numerical problems. You may also consult table 2 for the possible options.

4 Treatment of image data

4.1 Wavelet transforms for image data

Analogously to the transformation of one dimensional images you get the one step Haar wavelet transform for image data by `W=imwt(A,'Haar')` and the inverse transform by `Ar=imwtinv(W,'Haar')`. The image data should be in the matrix `A` whose number of rows and columns has to be even. The wavelet coefficients are given by the matrix `W` in the usual order, i.e. above at the left the HH subband as a result of a low pass filtering in raw and colum direction. If N_r denotes the number of raws and N_c the number of columns of `A`, the subband HH is the submatrix of matrix elements A_{ik} with $1 \leq i \leq \frac{N_r}{2}$ and $1 \leq k \leq \frac{N_c}{2}$. You obtain this submatrix by the command `HH=A(1:Nr/2,1:Nc/2)`.

wavelet type	options	default	remarks
Haar	n,u	n	for images only n
CDF(2,2), hat	rr, zp, vm	rr	
CDF(3,5)	rr, zp	rr	
DD(4,2), binlet	rr, zp, vm	rr	
DD(4,4) and DD(8,6)	rr, zp	rr	
AI(n,m) (average interpolating)	rr, zp	rr	
FBI, 9/7	rr, zp	rr	
D4 (orthogonal)	rr, zp, pe, gs (1d)	gs (1d), rr (2d)	
D6 (orthogonal)	rr, zp, pe	pe	
symlet	rr, zp, pe	pe	only pe recommended
QN(4), Quincunx-Neville	ignored		only for images

Table 2: Possible values of the otional fourth input parameter, the “option”, which may be specified for the functions `sigwt, sigwtinv, imwt` and `imwtinv`, compare also with table 1.

By `W=imwt(A,'Haar',p)` you get the cascaded Haar wavelet transform with p steps in scale and you get the corresponding inverse transform by `Ar=imwtinvm(W,'Haar',p)`. The number of rows and columns has to be a mutiple of 2^p. Analogously to the one dimensional case you get the wavelet transform for other wavelets replacing the string `'Haar'` by the string belonging to the corresponding wavelet as is indicated in table 1.

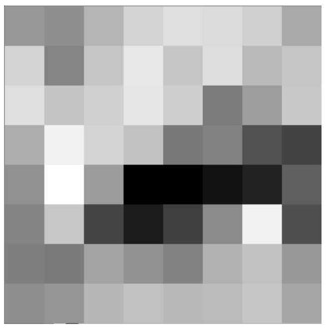
For images there are wavelet transforms which have no one dimensional counterparts, one of them is implemented and is chosen by `'Quisu'` or `'Quincunx'` as second parameter of `imwt` and `imwtinv`. Note that this wavelet type may only be chosen if the number p of cascades is even and it has to be specified as third parameter.

4.2 In- and output of image data

`[A,head] = reimppm('`*filename*`');` reads the image data of a file in PBM-, PGM- or PPM file format, the result will be in the matrix `A`, the output variable `head` contains the header of the file as matrix of strings. Note that the image data in `A` will be in the format `uint8` (positive integers with 8 bit). This may lead to unexplainable results or error messages if you apply the usual arithmetic operations. If you want to do calculations with the data in `A`, it is strongly recommended to transform the data into the usual type by `A=double(A)` . You may specify the path to your image file if this is not in the actual working directory, for instance by `[A,head]=reimppm('wavedata/boats512.pgm');`note that you have to specify the complete filename with the extension “`.pgm`”.

If only one return parameter is given, it contains the image data. Figure 3 shows a gray scale picture and the corresponding matrix obtained by `reimppm`.

`writepgm('`*filename*`',A)` writes the data of the matrix `A` interpreted as gray scale values in an image file with name *filename*`.pgm`. The string '*filename*' shuold therefore not contain the extension „`.pgm`“, but it may contain a path if the file should not be in the actual working directory. The matrix elements of `A` should not be outside the



$$\begin{pmatrix} 151 & 142 & 181 & 211 & 222 & 219 & 207 & 169 \\ 210 & 132 & 197 & 232 & 197 & 221 & 185 & 197 \\ 222 & 196 & 208 & 230 & 206 & 123 & 156 & 200 \\ 173 & 242 & 211 & 193 & 121 & 130 & 81 & 67 \\ 144 & 255 & 155 & 1 & 0 & 18 & 34 & 96 \\ 131 & 198 & 68 & 27 & 63 & 138 & 240 & 79 \\ 127 & 122 & 164 & 145 & 130 & 178 & 193 & 152 \\ 142 & 149 & 182 & 192 & 184 & 186 & 197 & 165 \end{pmatrix}$$

Figure 3: Reading a gray scale image (at the left) using `reimppm`, at the right the obtained matrix (0 corresponding to black, 255 to white)

allowed range from 0 to 255.

`imview(A)` shows the data of the matrix `A` interpreted as gray scale values in a graphic window. The matrix elements of `A` should not be outside the allowed range from 0 to 255. A string as second optional input parameter as in `imview(A,'my title')` will add this string as title above the image. Note that `imview` changes the parameters of the graphic window (for instance the color set). You may reset those parameters using `sdf();` to its default values (for instance if you want to use the usual colors). Note also that `imview` uses the whole actual graphic window (and changes its size and position). Therefore it may not be used in a subwindow defined by subplot. The following code example

```
A=zeros(512,512);
for i=1:512;for k=1:512;
     A(i,k)=255*abs(sin(i*%pi/256)*cos(k*%pi/256));
end;end;
imview(A)
```

generates the image shown in figure 4.

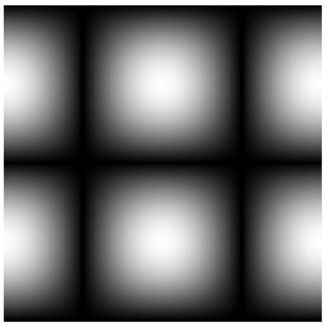


Figure 4: (512×512)-matrix with matrix elements $a_{ik} = \left|255 \cdot \sin\left(\frac{i\pi}{256}\right)\cos\left(\frac{k\pi}{256}\right)\right|$ interpreted as gray scale values

`B=grvalues(A)` transforms the data in the matrix `A` into the interval $[0, 255]$ such that the minimal value is mapped to 0 and the maximal to 255. This is useful before applying `imview` or `writepgm` where the data have to be in this interval. If $d_{\min}$ is the valuze of the smallest, $d_{\max}$ the value of the largest matrix element of `A`, the transformation is given by $c := \frac{1}{d_{\max}-d_{\min}}$, $h := -d_{\min} \cdot c$ and

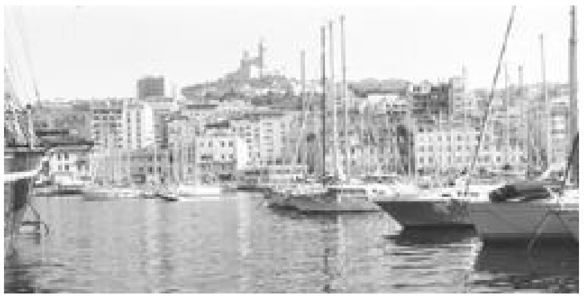
$$b_{ik} = 255(c \cdot a_{ik} + h)$$

where the matrix elements of `A` are denoted by a_{ik} and the elements of `B` by b_{ik}. In the particular case that $d_{\max} - d_{\min}$ is near to 0, all matrix elements b_{ik} are put to 127. This will be the case if all elements of `A` are equal.

The following example shows the application of `grvalues`. The wavelet coefficients resulting from a Haar wavelet transform will be transformed into the range of gray scale values. This is done for each subband separately.

```
[A,header]=reimppm('Testbilder/Marseille.pgm');
W=imhaarwt(A);
bdi=size(W);
nrh=bdi(1)/2; nch=bdi(2)/2;
// gray values separately for the subbands:
HH=grvalues(W(1:nrh,1:nch));
GG=grvalues(W(nrh+1:2*nrh,nch+1:2*nch));
HG=grvalues(W(1:nrh,nch+1:2*nch));
GH=grvalues(W(nrh+1:2*nrh,1:nch));
B=[HH,HG;GH,GG];
clf(); imview(B);
```

The resulting submatrices have been put together into one matrix. The final result and the original image are shown in figure 5.



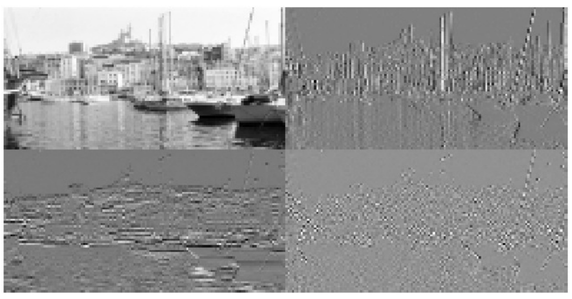


Figure 5: Original image at the left, wavelet coefficients (using the Haar wavelet) transformed into gray scale values by `grvalues` and shown by `imview` at the right

`B=grayclip(A)` also puts the values of the matrix elements of `A` into the range $[0, 255]$, but it cuts the values above 255 and setting them to 255 and it cuts the negative values setting them to 0. Values which are already inside the desired interval $[0, 255]$ remain unchanged. This procedure is recommended if you consider the values outside $[0, 255]$ as exceptions (for instance due to the Gibbs phenomenon) which should not change the representation of the "normal" values.

5 Treating filters

Scilab permits to use polynomials and rational functions in a very intuitive way. You have first to specify how you denote your polynomial variable. For filters this is usual z, so you should start with

```
z=poly(0,'z');
```

which initializes `z` as polynomial variable. Then you may define polynomials simply by

```
F=1-z+3*z^2-5*z^3
```

and the result will be shown in a way easy to read at the console:

```
F  =

                2    3
    1 - z + 3z  - 5z
```

and `G=(z-1)^5` is shown in the standard way

```
G  =

                   2      3     4    5
  - 1 + 5z - 10z  + 10z  - 5z  + z
```

In the context of wavelets we have to treat expressions like $H(z) = -z + 3 + 3z^{-1} - z^{-2}$. Trying the input `H=-z+3+3*z^(-1)-z^(-2)` we may be a little disapointed by the output

```
H  =

                  2    3
  - 1 + 3z + 3z  - z
    --------------
              2
             z
```

Scilab is treating the result as a rational function and brings it to the standard form as a quotient of two polynomials. But this is undesired for filters, so `showfilt(H,'H')` helps to get the usual representation

```
            1            -1     -2
H(z)= - z   + 3 + 3z    - z
```

The toolbox `wavelib` contains the following functions for calculations with filters and their output:

`F = ztrans(f,fkmin,z)` performs the z-transform. `f` is a vector containing the filter constants and `fkmin` indicates the true index of the filter constant `f(1)`. `z` is the polynomial variable, usually initialized as indicated at the beginning of the section. For the example of the FIR filter given by $f(\pm 2) = 1$, $f(\pm 1) = 4$, $f(0) = 6$, one has to put the input parameters to `f=[1 4 6 4 1]` und `fkmin=-2`. The result `F` will be a Scilab rational function. The command `showfilt(F,'F')` will show the result as

```
          2      1             -1     -2
F(z)=z   + 4z   + 6 + 4z    + z
```

`[f,fkmin] = iztrans(F)` performs the inverse z-transform. The input parameter `F` has to be a rational function which represents a FIR filter, i.e. which has only one power of z in the denominator. The return parameters are as should be the input parameters of `ztrans`. The vector `f` contains the filter constants and `fkmin` indicates the true index of the filter constant `f(1)`.

`H = compos(F,G)` calculates the compostion $F(G(Z))$ of two Laurent polynomials. It will be mainly used for expressions like $H(-z^{-1})$. This may be generated by `compos(H,-z^(-1))` if `H` is a properly defined Laurent polynomial (FIR filter) with `z` as polynomial variable.

`table = tabfiltco(F)` furnishes a table of the filter coefficients for the Laurent polynomial (FIR) filter given by `F`. Here the output of `tabfiltco(z^2*(1+z^(-1))^5)` (compare to the example for `showfilt` below)

```
ans  =

 - 2.    1.
 - 1.    5.
   0.    10.
   1.    10.
   2.    5.
   3.    1.
```

`showfilt(F,'F')` has been mentioned already and furnsihes the output of the filter `F` as this is desired for filters. The second parameter is optional, if omitted only the part at the right of the equal sign is shown as for `showfilt(z^2*(1+z^(-1))^5)` (compare to the example for `tabfiltco` above)

```
 2      1              -1      -2     -3
z  + 5z  + 10 + 10z    + 5z    + z
```

6 Coding experiments

The following functions are available

`x = scanWL(W,p)` puts the wavelet coefficients of image data which are assembled in a matrix `W` as they may be obtained for example by `W = imwt(A,'FBI',steps);` into a column vector. This is done in such a way that one gets an optimal data reduction using runlength coding. First the HH subband is scanned row-wise. Then the values of the HG subbands are put column-wise into the output vector `x`, starting with the subband belonging to the most coarse scale. Afterwards the values of the GH subbands are put row-wise to `x`. Finally the values of the GG subbands are transferred in the zigzag order indicated in figure 6, first the values of the subbands belonging to a more coarse scale.

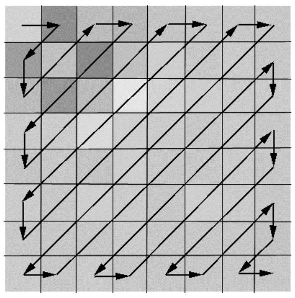


Figure 6: Scanning of GG subbands by the macro `scanWL`

`[y,mfv] = runlength(x,mfv)` performs a runlength coding of the numerical data in the vector `x`. The optional parameter `mfv` (“most frequent value”) indicates which

value is coded by its runlength. This should be a very frequent value. If the second input parameter is not specified, the value of the parameter `mfv` is determined as the most frequent value in the data vector `x`. If several values have the same frequency, the largest one is chosen. The result of the run length coding is put in the matrix `y` which has two columns. In each row the value in the first column indicates the number of values `mfv`, the value in the second column gives the next value different from `mfv` in the input data. There may be an exception in the last row of `y`: if `mfv` is the last value in `x`, then the first column there is the runlength of `mfv` reduced by 1 and in the second column the value `mfv` itself.
Example:

```
x=[0,0,0,4,0,0,,5,0,0,0,0,2,0,0];y=runlength(x)
```

furnishes the following vector `y`

$$\begin{matrix} 3 & 4 \\ 2 & 5 \\ 4 & 2 \\ 1 & 0 \end{matrix}$$

B = deadzone(A,d,mfv) simulates a dead zone quantization. The optional parameter `d` is the value of the threshold which is half the length of the dead zone. The default value of `d` is 1. The parameter `mfv` ("most frequent value") determines the center of the dead zone. For most applications this will be 0. If it is not specified, it will be calculated as the most frequent value in the input data in `A`, if several values have the same frequency, the largest one is chosen. `A` is the vector or matrix containing the input data. All matrix elements `A(i,k)` or vector components fulfilling

$$|\texttt{A(i,k)-mfv}| \leq \texttt{d}$$

are set to `mfv`, the others remain unchanged. The result will be saved in `B` which is therefore a vector or matrix of the same size as `A`. Such a dead zone quantization increases the effectivity of a subsequent runlength coding, as it increases the frequency of the most frequent value. In many cases the heigth of the quantization step is used as threshold `d`. This causes a doubling of the quantization step at `mfv`.

en = entrop(x) calculates the entropy of the data in the input vector or matrix `x`. If p_k denotes the relative frequency of the value indexed by k, the entropy is

$$-\sum_k p_k \log_2(p_k)$$

where the sum extends to all values in `x`. The logarithm to base 2 may be calculated by $\log_2(p_k) = \ln(p_k)/\ln(2)$. If $p_k = 0$, then the undefined value $p_k \log_2(p_k)$ has to be replaced by 0 which is the limit $\lim_{x\to 0} x \cdot \log_2(x)$. The calculation of the entropy permits an estimation of the need for memory for the data in `x`. The entropy indicates the theoretical need for minimal memory per bit using optimal coding. The practical need for memory for the data in `x` will therefore be larger than `length(x)*entrop(x)`.

A Appendix

A.1 Treating audio data

The functions described here belong to Scilab and are not part of the toolbox `wavelib`. This section is added to give a short review[1] how audio data in the Waveform Audio File Format (WAVE) with the filename extension `.wav` may be treated in the context of applications of the wavelet transform. Reading audio data is done by

`[x,sf,bits]=wavread('`*filename*`.wav');`

where the string *filename* may contain a path. The sampled audio data are available in the matrix `x` which will have two rows in the case of stereo data or be a row vector for mono. `sf` gives the sampling frequency, a usual value is 44 100 and `bits` indicates the number of bits per sample, 16 is a frequently used value.

Writing audio data may be done by

`wavwrite(x,sf,bits,'`*filename*`.wav');`

where the string *filename* may contain a path. The parameters `sf` und `bits` have the same signification as for reading by `wavread`. If the parameter `bits` is omitted, it will be replaced by its default value 16, which is quite usual. But the parameter `sf` should be specified because otherwise it gets the default value 22 050 which is quite unusual today and which may lead to surprises playing the resulting audio file. Therefore if you read some data, treat them and write the modified data, it is recommended to specify explicitly `sf` by the value obtained by `wavread`. Note that reading and subsequent rewriting of the unchanged audio data may lead to roundoff errors. The resulting relative error may attain 0.00009 which is quite large.

To get informations on further functions for treating audio data see the chapter "Sound file handling" in the help pages.

[1]This section essentially relies on section 23 of the german introduction "Kurzeinführung in Scilab", see the link on the title page.