ERRATA

Adapted Wavelet Analysis from Theory to Software
Mladen Victor Wickerhauser

August 2, 2001

The following errors were found in the 9 May 1994 version:

- Chapter 1, page 10, line 31: replace
  
  The Schwartz class $S$

  with

  The Schwartz class $S$

- Chapter 1, page 11, line 6: replace

  For calculations, it is often useful to first define formulas on test functions or Schwartz functions,

  with

  For calculations, it is often useful to define formulas first on test functions or Schwartz functions,

- Chapter 1, page 21, line 7: replace

  $a(0) = c(0); \quad a(k) = \frac{1}{\sqrt{2}} [c(-k) + c(k)]; \quad b(k) = \frac{1}{\sqrt{2}} [c(-k) - c(k)].$ \hspace{1cm} (1.49)

  with

  $a(0) = c(0); \quad a(k) = \frac{1}{\sqrt{2}} [c(-k) + c(k)]; \quad b(k) = \frac{1}{i\sqrt{2}} [c(-k) - c(k)].$ \hspace{1cm} (1.49)
• Chapter 2, page 53, lines 23–31: replace

```
btnt2btn( ROOT, LEVEL, BLOCK ):  
    If LEVEL==0 || ROOT==NULL then
        Let NODE = ROOT
    Else
        If BLOCK is even then
            Let NODE = btnt2btn( ROOT.LEFT, LEVEL-1, BLOCK/2 )
        Else
            Let NODE = btnt2btn( ROOT.RIGHT, LEVEL-1, (BLOCK-1)/2 )
    Return NODE
```

with

```
btnt2btn( ROOT, LEVEL, BLOCK )  
    If LEVEL==0 || ROOT==NULL then
        Let NODE = ROOT
    Else
        If the LEVEL bit of BLOCK is 0 then
            Let NODE = btnt2btn( ROOT.LEFT, LEVEL-1, BLOCK )
        Else
            Let NODE = btnt2btn( ROOT.RIGHT, LEVEL-1, BLOCK )
    Return NODE
```

• Chapter 2, page 54, lines 17–27: replace

```
btn2branch( SELF, LEVEL, BLOCK ):  
    If LEVEL>0 then
        If BLOCK is even then
            If SELF.LEFT==NULL then
                Let SELF.LEFT = makebtn( NULL, NULL, NULL, NULL )
                Let SELF = btn2branch( SELF.LEFT, LEVEL-1, BLOCK/2 )
            Else
                If SELF.RIGHT==NULL then
                    Let SELF.RIGHT = makebtn( NULL, NULL, NULL, NULL )
                    Let SELF = btn2branch( SELF.RIGHT, LEVEL-1, (BLOCK-1)/2 )
        Return SELF
```

with

```
btn2branch( ROOT, LEVEL, BLOCK )  
    If ROOT==NULL then
        Let ROOT = makebtn( NULL, NULL, NULL, NULL )
    If the LEVEL bit of BLOCK is 0 then
        Let ROOT.LEFT = btn2branch( ROOT.LEFT, LEVEL-1, BLOCK )
    Else
        Let ROOT.RIGHT = btn2branch( ROOT.RIGHT, LEVEL-1, BLOCK )
    Return ROOT
```
• Chapter 2, page 54, line 28 to page 55, line 3: replace

This function expects as input a preallocated BTN data structure, regarded as the root of the binary tree, and valid level and block indices. It returns a pointer to the node corresponding to that level and block in the binary tree. It allocates any intermediate nodes along the branch, including the target node itself if that is necessary, and leaves null pointers to any unused children in the allocated nodes.

with

This function expects as input either a preallocated BTN data structure, regarded as the root of the binary tree, or a NULL pointer, plus valid level and block indices. It returns a pointer to the root of a BTN tree, either the same one as the input or a newly allocated one if ROOT was NULL, containing the complete branch to the node at the specified level and block. It allocates any intermediate nodes along the branch, including the target BTN itself if that is necessary, and leaves null pointers to any unused children in the allocated nodes.

• Chapter 2, page 59, line 32: replace

FRACTION += 1<<LEVEL

with

FRACTION += 1<<(MAXLEVEL-LEVEL)
• Chapter 2, page 60, lines 15–29: replace

```plaintext
hedge2btnt( ROOT, GRAPH ) :
Let MAXLEVEL = 0
Let FRACTION = 0
For I = 0 to GRAPH.BLOCKS-1
    Let LEVEL = GRAPH.LEVELS[I]
    If LEVEL>MAXLEVEL then
        FRACTION <<= LEVEL-MAXLEVEL
        Let MAXLEVEL = LEVEL
        Let BLOCK = FRACTION
    Else
        Let BLOCK = FRACTION>>(MAXLEVEL-LEVEL)
    Let NODE = btn2branch( ROOT, LEVEL, BLOCK )
    Let NODE.CONTENT = GRAPH.CONTENTSTEP[I]
    FRACTION += 1<<LEVEL
Return MAXLEVEL
```

with

```plaintext
hedge2btnt( GRAPH ) :
Let ROOT = 0
Let MAXLEVEL = 0
Let FRACTION = 0
For I = 0 to GRAPH.BLOCKS-1
    Let LEVEL = GRAPH.LEVELS[I]
    If LEVEL>MAXLEVEL then
        FRACTION <<= LEVEL-MAXLEVEL
        Let MAXLEVEL = LEVEL
        Let BLOCK = FRACTION
    Else
        Let BLOCK = FRACTION>>(MAXLEVEL-LEVEL)
    Let ROOT = btn2branch( ROOT, LEVEL, BLOCK )
    Let NODE = btn2btnt( ROOT, LEVEL, BLOCK )
    Let NODE.CONTENT = GRAPH.CONTENTSTEP[I]
    FRACTION += 1<<(MAXLEVEL-LEVEL)
Return ROOT
```

• Chapter 2, page 60, lines 32-33: delete

Notice that both `hedge2btnt()` and `btnt2hedge()` return the maximum depth of any node in their tree. In fact, the two functions differ in just two lines.
- Chapter 2, page 62, lines 13–18: replace
tfa12btnt( ROOT, ATOM ):
    Let NODE = btn2branch( ROOT, ATOM.LEVEL, ATOM.BLOCK )
    Let LEAST = min( NODE.CONTENT.LEAST, ATOM.OFFSET )
    Let FINAL = max( NODE.CONTENT.FINAL, ATOM.OFFSET )
    enlargeinterval( NODE.CONTENT, LEAST, FINAL )
    NODE.CONTENT.ORIGIN[ATOM.OFFSET] += ATOMAMPLITUDE

with

tfa12btnt( ROOT, ATOM ):
    Let NODE = btn2branch( ROOT, ATOM.LEVEL, ATOM.BLOCK )
    If NODE.CONTENT is NULL then
        Let NODE.CONTENT = makeinterval(NULL, ATOM.OFFSET, ATOM.OFFSET)
    Else
        Let LEAST = min( NODE.CONTENT.LEAST, ATOM.OFFSET )
        Let FINAL = max( NODE.CONTENT.FINAL, ATOM.OFFSET )
        enlargeinterval( NODE.CONTENT, LEAST, FINAL )
        NODE.CONTENT.ORIGIN[ATOM.OFFSET] += ATOMAMPLITUDE
    Return ROOT

- Chapter 2, page 62, lines 22–24: replace
tfa12btnt( ROOT, ATOMS, NUM ):
    For K = 0 to NUM-1
        tfa12btnt( ROOT, ATOMS[K] )

with

tfa12btnt( ROOT, ATOMS, NUM ):
    For K = 0 to NUM-1
        Let ROOT = tfa12btnt( ROOT, ATOMS[K] )
    Return ROOT

- Chapter 2, page 64, lines 11–12: replace

    Let NUM = abtbleftlength( LENGTH, LEVEL )
    array2tfais( ATOMS, NUM, GRAPH.CONTENT+J, BLOCK, LEVEL )

with

    Let NUM = abtbleftlength( LENGTH, LEVEL )
    START += NUM
    array2tfais( ATOMS, NUM, GRAPH.CONTENT+J, BLOCK, LEVEL )

- Chapter 2, page 64, line 31: replace

    FRACTION += 1<<LEVEL

with

    FRACTION += 1<<(MAXLEVEL-LEVEL)
• Chapter 2, page 65, line 31: replace

\[
\text{FRACTION} \leftarrow 1 \ll \text{LEVEL}
\]

with

\[
\text{FRACTION} \leftarrow 1 \ll (\text{MAXLEVEL}-\text{LEVEL})
\]

• Chapter 3, page 75, lines 26 and 27: replace

Let \( W[N].\text{RE} = \cos(K \cdot \text{FACTOR}) \)

Let \( W[N].\text{IM} = \sin(K \cdot \text{FACTOR}) \)

with

Let \( W[K].\text{RE} = \cos(K \cdot \text{FACTOR}) \)

Let \( W[K].\text{IM} = \sin(K \cdot \text{FACTOR}) \)

• Chapter 4, page 119, line 11: replace

\textbf{Lemma 4.7} Suppose that \( B_{c_0}(\alpha_0) \) and \( B_{c_1}(\alpha_1) \) are

with

\textbf{Lemma 4.7} Suppose that \( B_{d}(\alpha_0) \) and \( B_{e}(\alpha_1) \) are

• Chapter 4, page 119, lines 13–14: replace

\[
W(r,I,\epsilon)_1f = (1_I U(r,\alpha_0, \epsilon) U(r, \alpha_1, \epsilon)_1f)_I; \quad (4.43)
\]

\[
W^*(r,I,\epsilon)_1f = (1_I U^*(r,\alpha_0, \epsilon) U^*(r, \alpha_1, \epsilon)_1f)_I. \quad (4.44)
\]

with

\[
W(r,I,\epsilon)_1f = 1_I U(r,\alpha_0, \epsilon) U(r, \alpha_1, \epsilon)(1_I f)_I; \quad (4.43)
\]

\[
W^*(r,I,\epsilon)_1f = 1_I U^*(r,\alpha_0, \epsilon) U^*(r, \alpha_1, \epsilon)(1_I f)_I. \quad (4.44)
\]

• Chapter 4, page 139, line 29: replace

Let \( \text{R.\text{ORIGIN}[0]} = \sqrt{2.0} \)

with

Let \( \text{R.\text{ORIGIN}[0]} = \sqrt{0.5} \)

• Chapter 5, page 169, lines 4 and 5: replace

\[
\begin{array}{|c|c|c|c|}
\hline
C & 6 & H & 3.6160691415 \\ 
\hline
G & 1.3839308584 & 0.4990076823 \\
\hline
\end{array}
\]

with

\[
\begin{array}{|c|c|c|c|}
\hline
C & 6 & H & 2.0346814255 \\ 
\hline
G & 2.9653185745 & 0.0135212898 \\
\hline
\end{array}
\]

6
Chapter 5, page 175, lines 7 and 8: replace

\[
\begin{array}{ccc}
C & 6 & H \\
 & & .247013 \\
G & 6 & .268885 \\
& & .069768
\end{array}
\]

with

\[
\begin{array}{ccc}
C & 6 & H \\
 & & .219458 \\
G & 6 & .199186 \\
& & .038215
\end{array}
\]

Chapter 5, page 178, lines -7 to -5: replace

\[
\sum_k f(2k+1)e^{-2\pi ik\xi} \sum_n u(2n+1)e^{-2\pi in\xi}.
\]

Thus

\[
\hat{F}u(\xi) = \hat{f}(\xi) \hat{u}(\xi) + \hat{\hat{f}}(\xi) \hat{\hat{u}}(\xi),
\]  \hspace{1cm} (5.58)

with

\[
\sum_k f(2k+1)e^{-2\pi ik\xi} \sum_n u(2n-1)e^{-2\pi in\xi}.
\]

Thus

\[
\hat{F}u(\xi) = \hat{f}(\xi) \hat{u}(\xi) + e^{-2\pi i\xi} \hat{\hat{f}}(\xi) \hat{\hat{u}}(\xi),
\]  \hspace{1cm} (5.58)

Chapter 5, page 179, line 6: replace

\[
\hat{F}^*u(\xi) = \sum_n F^*u(n)e^{-2\pi in\xi} = \sum_k \sum_n \hat{f}(2k-n)u(n)e^{-2\pi in\xi}
\]

with

\[
\hat{F}^*u(\xi) = \sum_n F^*u(n)e^{-2\pi in\xi} = \sum_k \sum_n \hat{f}(2k-n)u(k)e^{-2\pi in\xi}
\]

Chapter 5, page 180, line 6: replace

matrix equation \( M^*M = 2I \)

with

matrix equation \( M^*M = M'M^* = 2I \)

Chapter 5, page 180, line 17: replace

\( M_0^*M_0' = 2I \)

with

\( M_0^*M_0' = M_0^*M_0 = 2I \)
• Chapter 5, page 186, line 3: replace
  support diameter less than $d/10$
  with
  support diameter less than $10d$

• Chapter 5, page 196, lines 13–14: replace
  If we wish to assign output values rather than to superpose them, we need only replace the increment operator in the $\text{OUT}[i]$ statement by an assignment operator.
  with
  If we wish to assign output values rather than to superpose them, we need to assign 0 to each successive element of $\text{OUT}[i]$ before superposing the result onto it.

• Chapter 5, page 198, lines 13–14: replace
  If we wish to assign output values rather than to superpose them, we need only replace the increment operator in the $\text{OUT}[i]$ statement by an assignment operator.
  with
  If we wish to assign output values rather than to superpose them, we need to assign 0 to each successive element of $\text{OUT}[i]$ before superposing the result onto it.

• Chapter 5, page 206, line 9: replace
  $\text{OUT}[\text{I*STEP}] += \text{FILTER}[\text{J*IN}[Q+2*I-J]]$
  with
  $\text{OUT}[\text{I*STEP}] += \text{FILTER}[\text{J*IN}[2*I-J-Q]]$

• Chapter 5, page 206, line 16: replace
  $\text{OUT}[\text{I*STEP}] += \text{FILTER}[\text{J*IN}[2*I-J-Q]]$
  with
  $\text{OUT}[\text{I*STEP}] += \text{FILTER}[\text{J*IN}[Q+2*I-J]]$

• Chapter 5, page 206, line 34: replace
  but we replace the increment operator with an assignment operator in all statements with $\text{OUT}[i]$ in the left-hand side.
  with
  but we assign 0 to each successive element of $\text{OUT}[i]$ before accumulating the result into it.
• Chapter 5, page 207, lines 21 through 26: replace

For I = 0 to J02-Q2
OUT[I*STEP] += FILTER[Q+2*I-J]*IN[I]
For I = max(0,JA2) to min(Q2-1,J02)
OUT[I*STEP] += FILTER[2*I-J]*IN[I]
For I = JA2+Q2 to Q2-1
OUT[I*STEP] += FILTER[2*I-J-Q]*IN[I]

with

For I = 0 to J02-Q2
OUT[J*STEP] += FILTER[Q+2*I-J]*IN[I]
For I = max(0,JA2) to min(Q2-1,J02)
OUT[J*STEP] += FILTER[2*I-J]*IN[I]
For I = JA2+Q2 to Q2-1
OUT[J*STEP] += FILTER[2*I-J-Q]*IN[I]

• Chapter 5, page 208, line 2: replace

but with assignments rather than increments in all statements with OUT[] in their left-hand sides.

with

but we assign 0 to each successive element of OUT[] before accumulating the result into it.

• Chapter 7, page 264, lines 15–25: replace

hedge2dwp spr( GRAPH, J, N, S, HQF, GQF, WORK ):
  If S < GRAPH.LEVELS[J] then
    Let J = hedge2dwp spr( GRAPH, J, N/2, S+1, HQF, GQF, WORK )
    Let LEFT = GRAPH.CONTENT S[J]
    Let J = hedge2dwp spr( GRAPH, J+1, N/2, S+1, HQF, GQF, WORK )
    Let RIGHT = GRAPH.CONTENT S[J]
    acdpe( WORK, 1, LEFT, N/2, HQF )
    acdf ( WORK, 1, RIGHT, N/2, GQF )
    For I = 0 to N-1
      Let LEFT[I] = WORK[I]
    Return J

with

hedge2dwp spr( GRAPH, J, N, S, HQF, GQF, WORK ):
  If S < GRAPH.LEVELS[J] then
    Let LEFT = GRAPH.CONTENT S[J]
    Let J = hedge2dwp spr( GRAPH, J, N/2, S+1, HQF, GQF, WORK )
    Let RIGHT = GRAPH.CONTENT S[J]
    Let J = hedge2dwp spr( GRAPH, J, N/2, S+1, HQF, GQF, WORK )
    acdpe( WORK, 1, LEFT, N/2, HQF )
    acdf ( WORK, 1, RIGHT, N/2, GQF )
    For I = 0 to N-1
      Let LEFT[I] = WORK[I]
  Else
    J += 1
  Return J
• Chapter 7, page 269, lines 8–18: replace

acdaparent( PARENT, CHILD, F ):
    If CHILD != NULL then
        If CHILD.Origin != NULL then
            Let LEAST = acdaleast( CHILD, F )
            Let LEAST = min( PARENT.LEAST, LEAST )
            Let FINAL = acdafinal( CHILD, F )
            Let FINAL = max( PARENT.FINAL, FINAL )
            Let PARENT = enlargeinterval( PARENT, LEAST, FINAL )
            acda( PARENT.Origin, 1, CHILD.Origin,
                CHILD.LEAST, CHILD.FINAL, F )
        Return PARENT

with

acdaparent( PARENT, CHILD, F ):
    If CHILD != NULL then
        If CHILD.Origin != NULL then
            Let LEAST = acdaleast( CHILD, F )
            Let FINAL = acdafinal( CHILD, F )
            If PARENT != NULL then
                Let LEAST = min( PARENT.LEAST, LEAST )
                Let FINAL = max( PARENT.FINAL, FINAL )
            Let PARENT = enlargeinterval( PARENT, LEAST, FINAL )
            acda( PARENT.Origin, 1, CHILD.Origin,
                CHILD.LEAST, CHILD.FINAL, F )
        Return PARENT

• Chapter 7, page 269, lines 23–27: replace

Next we link together whole branches from the root leading to nonempty nodes, using the utility function btn2branch(). This function returns a pointer to a target node and allocates any intermediate nodes along the branch, including the target node itself if that is necessary. It assigns null pointers to any unused children in the allocated nodes.

with

Next we link together a partial BTN tree containing branches to all the nonempty nodes, using the utility functions btn2branch() and btn2btn(). The first function allocates any intermediate nodes along the branch, including the root and the target node itself if necessary. It assigns NULL pointers to any unused children along the way. The latter function returns a pointer to the target BTN which we use to assign it by side effect.

• Chapter 8, page 284, line 25: replace

bottom nodes, as indicated by the asterisks in Figure 8.2. Their total infor-

with

bottom nodes, as indicated by the asterisks in Figure 8.3. Their total infor-
• Chapter 8, page 285, line 15: replace nodes, which constitutes a basis by Theorem 7.9. These best basis nodes are dis-
with nodes, which constitute a basis by Theorem 7.9. These best basis nodes are dis-

• Chapter 10, page 331, line 14: replace item 4 in Theorem 10.2:

4. The set \( \{ \phi_n : n = 1, 2, \ldots \} \) is dense in \( S \).

with

4. The linear span of \( \{ \phi_n : n = 1, 2, \ldots \} \) is dense in \( S \).

• Chapter 10, page 341, line 2: replace Equation 10.7:

\[
p' = 2^s p + (2^s - 1)c[h] + (c[g] - c[h])f''.
\]  \hspace{1cm} (10.7)

with

\[
p' = 2^s p - (2^s - 1)c[h] - (c[g] - c[h])f''.
\]  \hspace{1cm} (10.7)

• Chapter 10, page 341, line 14: replace Equation 10.8:

\[
p'' = \lfloor p'/2^s \rfloor = \lfloor p + (1 - 2^{-s})c[h] + 2^{-s}(c[g] - c[h])f'' \rfloor.
\]  \hspace{1cm} (10.8)

with

\[
p'' = \lfloor p'/2^s \rfloor = \lfloor p - (1 - 2^{-s})c[h] - 2^{-s}(c[g] - c[h])f'' \rfloor.
\]  \hspace{1cm} (10.8)

• Chapter 11, page 363, line 21: replace

Let \( \sigma(X) \subset \mathbb{R}^d \) be the vector

with

Let \( \sigma(X) \in \mathbb{R}^d \) be the vector

• Chapter 11, page 364, line -5: replace

The correlation coefficient \( C(X, Y) \) of two random variables is defined by the

with

The correlation coefficient \( C(X, Y) \) of two random variables \( X, Y \in \mathbb{R} \) is defined by the

• Chapter 11, page 364, line -2: replace

Here \( E(XY) = \frac{1}{N} \sum_{n=1}^{N} X_n Y_n \) denotes the expectation of the variable \( xy \), etc.

with

Here \( E(XY) = \frac{1}{N} \sum_{n=1}^{N} X_n Y_n \) denotes the expectation of the variable \( XY \), and so on.
• Chapter 11, page 365, lines 11–28: replace

Now suppose that \( x = \{x(n)\} \) is *numerically smooth*, which means that the following assumption is valid:

\[
0 \leq \delta \overset{\text{def}}{=} \max_{1 \leq n \leq N} \frac{|x(n+1) - x(n)|}{|x(n)| + |x(n+1)|} \ll 1. \tag{11.6}
\]

Roughly speaking, this hypothesis guarantees that \( \Delta x(n) \overset{\text{def}}{=} x(n+1) - x(n) \) remains small relative to \( |x(n)| \) for all \( n = 1, 2, \ldots, N \). Then we have the following lower estimate for the numerator of \( C(X, Y) \):

\[
x(n+1) = x(n) + \Delta x(n)
\Rightarrow x(n+1) \geq x(n) - \delta(|x(n)| + |x(n+1)|)
\Rightarrow x(n+1)x(n) \geq x(n)^2 - \delta(|x(n)|^2 + |x(n+1)||x(n)|); \\
x(n) = x(n+1) - \Delta x(n)
\Rightarrow x(n) \geq x(n+1) - \delta(|x(n)| + |x(n+1)|)
\Rightarrow x(n+1)x(n) \geq x(n+1)^2 - \delta(|x(n)||x(n+1)| + |x(n+1)|^2).
\]

We now average these two inequalities:

\[
x(n+1)x(n) \geq \frac{1}{2} \left( x(n)^2 + x(n+1)^2 \right) - \frac{\delta}{2} (|x(n)| + |x(n+1)|)^2
\geq \frac{1}{2} \left( x(n)^2 + x(n+1)^2 \right) - \delta \left( |x(n)|^2 + |x(n+1)|^2 \right)
\Rightarrow E(XY) \geq E(X^2) - 2\delta E(X^2).
\]

We can therefore estimate the correlation coefficient as follows:

\[
1 \geq C(X, Y) \geq \frac{E(X^2) - 2\delta E(X^2)}{E(X^2)} = 1 - 2\delta. \tag{11.7}
\]

with

Likewise, the denominator simplifies to \( E(XY) \).

Now put \( \Delta X \overset{\text{def}}{=} Y - X \), and suppose that \( x = \{x(n)\} \) is *numerically smooth*, namely, that there is some \( 0 < \delta < 1 \) for which we have:

\[
E(\Delta X^2) \leq 2\delta E(X^2). \tag{11.6}
\]

This hypothesis is a discrete analogue to the estimate \( \|f'\|^2 \leq 2\delta \|f\|^2 \), which is satisfied by square integrable functions with relatively small derivatives. It yields the following lower bound for the numerator of \( C(X, Y) \):

\[
E(\Delta X^2) = E(Y^2) + E(X^2) - 2E(XY) = 2E(X^2) - 2E(XY)
\Rightarrow E(XY) = E(X^2) - \frac{1}{2}E(\Delta X^2) \geq (1 - \delta)E(X^2).
\]

We can therefore estimate the correlation coefficient as follows:

\[
C(X, Y) = \frac{E(XY)}{E(X^2)} \geq 1 - \delta. \tag{11.7}
\]
• Chapter 11, page 366, lines 3–6: replace

Similarly, if we interpret \( x(N + n) \) as \( x(n) \) for \( n = 1, 2, \ldots, k \) and \( 1 \leq k \leq N \), then the correlations between more distant pixels will also be close to one:

**Proposition 11.1** If \( \frac{1}{N} \sum_{n=1}^{N} x(n) = 0 \) and \( x \) is numerically smooth, then for \( X_n = x(n) \) and \( Y_n = x(n + k) \) we have \( 1 \geq C(X, Y) \geq 1 - 2k\delta \).

Since \( C(X, Y) = C(Y, X) \), we can allow negative values of \( k \) just by substituting \( |k| \). Notice too that \( 1 - 2|k|\delta \approx (1 - \delta)^{|k|} \overset{\text{def}}{=} r^{|k|} \), where \( r = (1 - \delta)^2 \approx 1 \).

with

Similarly, the correlations between more distant pixel pairs \( x(n) \) and \( x(n + k) \) will also be close to one. Since \( C(X, Y) = C(Y, X) \), the estimate only depends on \( |k| \):

**Proposition 11.1** If \( x \) is a numerically smooth \( N \)-periodic sequence with \( \frac{1}{N} \sum_{n=1}^{N} x(n) = 0 \), then for \( X_n = x(n) \) and \( Y_n = x(n + k) \) we have \( 1 \geq C(X, Y) \geq 1 - k\delta \).

Notice that \( 1 - |k|\delta \approx (1 - \delta)^{|k|} \overset{\text{def}}{=} r^{|k|} \), where \( r = 1 - \delta \).

• Chapter 11, page 378: On line 35, replace \( \{Y_n : n = 1, \ldots, N\} \) with \( \{Y_n : n = 1, \ldots, d\} \). On line 36, replace \( Y_1, \ldots, Y_N \) with \( Y_1, \ldots, Y_d \).

• Chapter 11, page 381, line 27: replace

\[
\text{dist}(U, V) = \mathcal{H}(U^*V)
\]

with

\[
\text{dist}(U, V) = \text{dist}_G(U, V) \overset{\text{def}}{=} \sqrt{\mathcal{H}(U^*V)}
\]

• Chapter 11, page 383, line 21: replace

\[
M'_{ij} = \frac{1}{N} \sum_{n=1}^{N} U'_i \tilde{X}'_{n,j} U'_j \tilde{X}'_n,
\]

with

\[
M'_{ij} = \frac{1}{N} \sum_{n=1}^{N} U'_i \tilde{X}_n U'_j \tilde{X}_n.
\]

• Chapter 11, page 383, line 22: replace

Here \( \tilde{X}'_n = X'_n - E(X') \) is a vector in \( \mathbb{R}^d \), and \( E(\tilde{X}') = 0 \).

with

Here \( \tilde{X}_n = X_n - E(X) \) is a vector in \( \mathbb{R}^d \), and \( E(\tilde{X}) = 0 \).
• Chapter 11, page 408, line 4: replace

the sparsifiable Hilbert-Schmidt operators $\mathcal{S}$.

with

the sparsifiable Hilbert-Schmidt operators.

• Appendix C, page 445, lines 1 to 17: replace

**Coifman 6**

<table>
<thead>
<tr>
<th>Low-pass</th>
<th>High-pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>$((S_{R15}-3.0)/32.0)\times SR2$</td>
<td>$3.85807777478867490\ E^{-2}$</td>
</tr>
<tr>
<td>$((1.0-S_{R15})/32.0)\times SR2$</td>
<td>$-1.26969125396205200\ E^{-1}$</td>
</tr>
<tr>
<td>$((3.0-S_{R15})/16.0)\times SR2$</td>
<td>$-7.71615554957734980\ E^{-2}$</td>
</tr>
<tr>
<td>$((S_{R15}+3.0)/16.0)\times SR2$</td>
<td>$6.07491641385684120\ E^{-1}$</td>
</tr>
<tr>
<td>$((S_{R15}+13.0)/32.0)\times SR2$</td>
<td>$7.45687558934434280\ E^{-1}$</td>
</tr>
<tr>
<td>$((9.0-S_{R15})/32.0)\times SR2$</td>
<td>$2.26584265197068560\ E^{-1}$</td>
</tr>
</tbody>
</table>

with

<table>
<thead>
<tr>
<th>Coifman 6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Low-pass</strong></td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>$-7.2732619512526\ E^{-2}$</td>
</tr>
<tr>
<td>$3.3789766245748\ E^{-1}$</td>
</tr>
<tr>
<td>$8.5257202021160\ E^{-1}$</td>
</tr>
<tr>
<td>$3.8486484686486\ E^{-1}$</td>
</tr>
<tr>
<td>$-7.2732619512526\ E^{-2}$</td>
</tr>
<tr>
<td>$-1.5655728135792\ E^{-2}$</td>
</tr>
</tbody>
</table>