LAGARIAS-DAUBECHIES' ALGORITHM IN ACTION

Splus Implementation by Brani Vidakovic

1 A few Daubechies' filters

```
> Daub2
[1]  0.4829629  0.8365163  0.2241439 -0.1294095
> Daub3
[1]  0.33267055  0.80689151  0.45987750 -0.13501102 -0.08544127  0.03522629
> Daub4
[1]  0.23037781  0.71484657  0.63088077 -0.02798377 -0.18703481  0.03084138
[7]  0.03288301 -0.01059740
> Daub5
 [1]  0.160102398  0.603829270  0.724308528  0.138428146 -0.242294887
 [6] -0.032244870  0.077571494 -0.006241490 -0.012580752  0.003335725
> Daub6
 [1]  0.1115407434  0.4946238904  0.7511339080  0.3152503517 -0.2262646940
 [6] -0.1297668676  0.0975016056  0.0275228655 -0.0315820393  0.0005538422
[11]  0.0047772575 -0.0010773011
> Daub7
 [1]  0.0778520541  0.3965393195  0.7291320908  0.4697822874 -0.1439060039
 [6] -0.2240361850  0.0713092193  0.0806126092 -0.0380299369 -0.0165745416
[11]  0.0125509986  0.0004295780 -0.0018016407  0.0003537138
> Daub8
 [1]  0.0544158422  0.3128715909  0.6756307363  0.5853546837 -0.0158291053
 [6] -0.2840155430  0.0004724846  0.1287474266 -0.0173693010 -0.0440882539
[11]  0.0139810279  0.0087460940 -0.0048703530 -0.0003917404  0.0006754494
[16] -0.0001174768
> Daub9
 [1]  3.807795e-02  2.438347e-01  6.048231e-01  6.572881e-01  1.331974e-01
 [6] -2.932738e-01 -9.684078e-02  1.485407e-01  3.072568e-02 -6.763283e-02
[11]  2.509471e-04  2.236166e-02 -4.723205e-03 -4.281504e-03  1.847647e-03
[16]  2.303858e-04 -2.519632e-04  3.934732e-05
> Daub10
 [1]  2.667006e-02  1.881768e-01  5.272012e-01  6.884590e-01  2.811723e-01
 [6] -2.498464e-01 -1.959463e-01  1.273693e-01  9.305736e-02 -7.139415e-02
[11] -2.945754e-02  3.321267e-02  3.606554e-03 -1.073318e-02  1.395352e-03
[16]  1.992405e-03 -6.858567e-04 -1.164669e-04  9.358867e-05 -1.326420e-05
> Daub15
 [1]  4.538537e-03  4.674339e-02  2.060239e-01  4.926318e-01  6.458131e-01
 [6]  3.390025e-01 -1.932041e-01 -2.888826e-01  6.528295e-02  1.901467e-01
[11] -3.966618e-02 -1.111209e-01  3.387714e-02  5.478055e-02 -2.576701e-02
[16] -2.081005e-02  1.508392e-02  5.101000e-03 -6.487735e-03 -2.417565e-04
[21]  1.943324e-03 -3.734824e-04 -3.595652e-04  1.558965e-04  2.579270e-05
[26] -2.813330e-05  3.362987e-06  1.811270e-06 -6.316882e-07  6.133360e-08
> Daub20
 [1]  7.799532e-04  1.054939e-02  6.342375e-02  2.199420e-01  4.726959e-01
 [6]  6.104929e-01  3.615021e-01 -1.392120e-01 -3.267866e-01 -1.672698e-02
```

```
[11]  2.282910e-01  3.985033e-02 -1.554586e-01 -2.471675e-02  1.022917e-01
[16]  5.632273e-03 -6.172285e-02  5.874686e-03  3.229429e-02 -8.789319e-03
[21] -1.381052e-02  6.721624e-03  4.420540e-03 -3.581492e-03 -8.315618e-04
[26]  1.392559e-03 -5.349756e-05 -3.851045e-04  1.015328e-04  6.774277e-05
[31] -3.710584e-05 -4.376142e-06  7.241244e-06 -1.011993e-06 -6.847076e-07
[36]  2.633923e-07  2.014326e-10 -1.814842e-08  4.056125e-09 -2.998835e-10
```

2 Dyad.S

```
# The function gives the binary decimal for the
# input number $x$ from [0,1]
# $n$ is the precision, the number of decimal
# places.
#----------------------------------------------
# Use
#  >  Dyad(0.76, 12)
#   [1] 1 1 0 0 0 0 1 0 1 0 0 0
# Indeed: 0.76 dec = 0.11000010100...bin
#----------------------------------------------
# This function is needed for Psi.S and Phi.S
#----------------------------------------------

Dyad  <- function(x, n)
{
        dyad <- NULL
        y <- x
        for(i in 1:n) {
                y <- 2 * y
                dyad <- c(dyad, floor(y))
                y <- y - floor(y)
        }
        return(dyad)
}
#----------------------------------------------
```

3 T0 and T1 Matrices

```
#----------------------------------------------------
# This function forms the Daubechies-Lagarias matrix
# T0 given the filter. This function is called by
# Phi.S and Psi.S.
# Use:
#> Daub2
```

```
#[1]  0.4829629  0.8365163  0.2241439 -0.1294095
#> T0(Daub2)
#          [,1]       [,2]      [,3]
#[1,] 0.6830127  0.0000000 0.0000000
#[2,] 0.3169873  1.1830127 0.6830127
#[3,] 0.0000000 -0.1830127 0.3169873
#--------------------------------------------------
 T0 <- function(filter)
{
        N <- length(filter)
        domain <- N - 1
        T0 <- matrix(0, domain, domain)
        for(i in 1:domain) {
                for(j in 1:domain) {
                        if(2 * i - j > N || 2 * i - j < 1)
                                next
                        T0[i, j] <- sqrt(2) * filter[2 * i - j]
                }
        }
        return(T0)
}
#---------------------------------------------------

#--------------------------------------------------
# This function forms the Daubechies-Lagarias matrix
# T1 given the filter. This function is needed in
# Phi.S and Psi.S
# Use:
#> Daub2
#[1]  0.4829629  0.8365163  0.2241439 -0.1294095
#> T1(Daub2)
#           [,1]      [,2]       [,3]
#[1,]  1.1830127 0.6830127  0.0000000
#[2,] -0.1830127 0.3169873  1.1830127
#[3,]  0.0000000 0.0000000 -0.1830127
#---------------------------------------------------
 T1 <- function(filter)
{
        N <- length(filter)
        domain <- N - 1
        T1 <- matrix(0, domain, domain)
        for(i in 1:domain) {
                for(j in 1:domain) {
                        if(2 * i - j + 1 > N || 2 * i - j + 1 < 1)
```

```
                                next
                        T1[i, j] <- sqrt(2) * filter[2 * i - j + 1]
                }
        }
        return(T1)
}
#-----------------------------------------------------
```

4 Evaluating ϕ and ψ

```
#---------------------------------------------------
# Returns the value of the scaling function connected
# with the $filter$  evaluated at the input point $x$.
# The precision of $x$ as a binary is $n$.
# Use:
# > Phi(0.9, Daub2, n=25)
#[1] 1.266163
# or
# > xx_seq(0,4, length=129)
# > yy_rep(0,129)
# > for (i in 1:129) {
# +   yy[i] _ yy[i] + Phi(xx[i], Daub2, n=22)
# + }
# (It takes some time...)
# > plot(xx, yy, type="l")
# should give the familiar object (See figure 1)
#-----------------------------------------------------
 Phi <- function(x, filter = Daub2, n = 20)
{
        daun <- length(filter)/2        #daubechies number
        N <- length(filter) - 1         #[0,N] is domain of Phi
        if(x <= 0 || x >= N)
                return(0)       #gives value 0 outside [0,N]
        int <- floor(x) # splitting x on the integer part and
        dec <- x - int  # on the decimal part
        dy <- Dyad(dec, n)      # dyadic repr. of the decimal part
        t0 <- T0(filter)        #forming T0
        t1 <- T1(filter)        #and T1 matrices
        prod <- diag(N) #initializing prod
        for(i in 1:n) {
                if(dy[i] == 1)
                        prod <- prod %*% t1
```

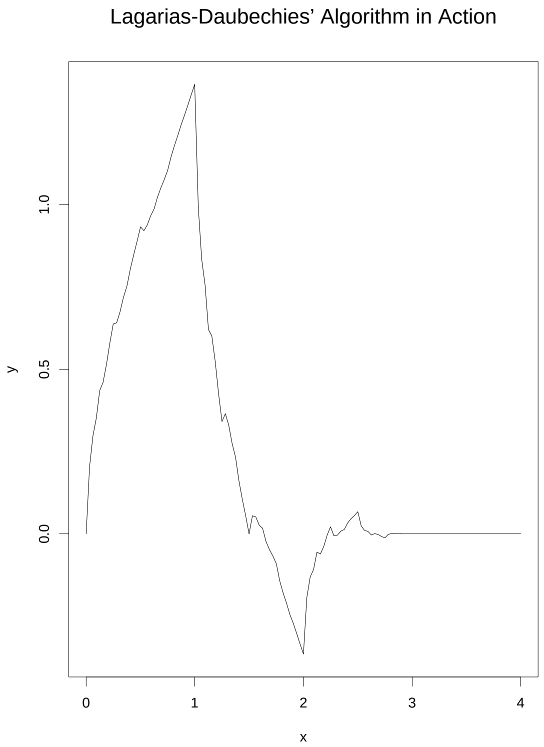

Figure 1: Scaling function for Daubechies' $N = 2$ wavelet

```
                else prod <- prod %*% t0
        }
        onevec <- rep(1, N)
        phis <- prod %*% onevec/N        #getting the averages of rows
        return(phis[int + 1, 1])         #returning the right component
}
#-----------------------------------------------------
```

```
#---------------------------------------------------
# Returns the value of the mother wavelet connected
# with the $filter$ evaluated at the input point $x$.
# The precision of $x$ as a binary is $n$.
# Use:
# > Psi(0.9, Daub2, n=25)
# [1] -0.9902929
# See also Phi.S.
#-----------------------------------------------------
```

```
 Psi <- function(x, filter = Daub2, n = 20)
{
	daun <- length(filter)/2	#daubechies number
	N <- length(filter) - 1 #[daun-N,daun] is the domain of Psi
	if(x <= daun - N || x >= daun)
		return(0)	#gives value 0 outside [daun-N,daun]
	xx <- 2 * x	#doubling argument
	int <- floor(xx)	# splitting xx on the integer part and
	dec <- xx - int #  the decimal part
	dy <- Dyad(dec, n)	#dyadic representation
	t0 <- T0(filter)	#forming T0
	t1 <- T1(filter)	#and T1 matrices
	prod <- diag(N) #initializing prod
	for(i in 1:n) {
		if(dy[i] == 1)
			prod <- prod %*% t1
		else prod <- prod %*% t0
	}
	onevec <- rep(1, N)
	v <- prod %*% onevec/N  #getting the averages of rows
	u <- NULL
	mirror <- c(-1, 1) * filter
	for(i in 1:N) {
		if(i + 1 - int > 0 && i + 1 - int < N + 2)
			u <- c(u, mirror[i + 1 - int])
		else u <- c(u, 0)
	}
	return(sqrt(2) * sum(u * v))
}
#----------------------------------------------------------
```