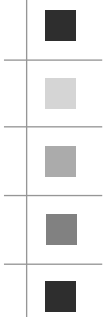
SAS Publishing

TECHNICAL REPORT

SAS/IML® Software: Changes and Enhancements, Release 8.2



The Power to Know™

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/IML® Software: Changes and Enhancements, Release 8.2*, Cary, NC: SAS Institute Inc., 2001

ISBN 1-58025-867-0

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, January 2001

Table of Contents

Chapter 1
Wavelet Analysis

Chapter Table of Contents

Chapter 1
Wavelet Analysis

Overview

Wavelets are a versatile tool for understanding and analyzing data, with important applications in nonparametric modeling, pattern recognition, feature identification, data compression, and image analysis. Wavelets provide a description of your data that localizes information at a range of scales and positions. Moreover, they can be computed very efficiently, and there is an intuitive and elegant mathematical theory to guide you in applying them.

Some Brief Mathematical Preliminaries

The discrete wavelet transform decomposes a function as a sum of basis functions called wavelets. These basis functions have the property that they can be obtained by dilating and translating two basic types of wavelets known as the *scaling function* or *father wavelet* ϕ, and the *mother wavelet* ψ. These translates and dilations are defined as follows:

$$\begin{aligned} \phi_{j,k}(x) &= 2^{j/2}\phi(2^j x - k) \\ \psi_{j,k}(x) &= 2^{j/2}\psi(2^j x - k) \end{aligned}$$

The index j defines the dilation or *level* while the index k defines the translate. Loosely speaking, sums of the $\phi_{j,k}(x)$ capture low frequencies and sums of the $\psi_{j,k}(x)$ represent high frequencies in the data. More precisely, for any suitable function $f(x)$ and for any j_0,

$$f(x) = \sum_k c_k^{j_0}\phi_{j_0,k}(x) + \sum_{j \geq j_0}\sum_k d_k^j \psi_{j,k}(x)$$

where the c_k^j and d_k^j are known as the scaling coefficients and the detail coefficients respectively. For orthonormal wavelet families these coefficients can be computed by

$$\begin{aligned} c_k^j &= \int f(x)\phi_{j,k}(x)\,dx \\ d_k^j &= \int f(x)\psi_{j,k}(x)\,dx \end{aligned}$$

The key to obtaining fast numerical algorithms for computing the detail and scaling coefficients for a given function $f(x)$ is that there are simple recurrence relationships

that enable you to compute the coefficients at level $j-1$ from the values of the scaling coefficients at level j. These formulae are

$$\begin{aligned} c_k^{j-1} &= \sum_i h_{i-2k} c_i^j \\ d_k^{j-1} &= \sum_i g_{i-2k} c_i^j \end{aligned}$$

The coefficients h_k and g_k that appear in these formulae are called *filter coefficients.* The h_k are determined by the father wavelet and they form a low-pass filter; $g_k = (-1)^k h_{1-k}$ and form a high-pass filter. The preceding sums are formally over the entire (infinite) range of integers. However, for wavelets that are zero except on a finite interval, only finitely many of the filter coefficients are non-zero and so in this case the sums in the recurrence relationships for the detail and scaling coefficients are finite.

Conversely, if you know the detail and scaling coefficients at level $j-1$ then you can obtain the scaling coefficients at level j using the relationship

$$c_k^j = \sum_i h_{k-2i} c_i^{j-1} + \sum_i g_{k-2i} d_i^{j-1}$$

Suppose that you have data values

$$y_k = f(x_k), \qquad k = 0, 1, 2, \cdots, N-1$$

at $N = 2^J$ equally spaced points x_k. It turns out that the values $2^{-J/2} y_k$ are good approximations of the scaling coefficients c_k^J. Then using the recurrence formula you can find c_k^{J-1} and d_k^{J-1}, $k = 0, 1, 2, \cdots, N/2 - 1$. The discrete wavelet transform of the y_k at level $J-1$ consists of the $N/2$ scaling and $N/2$ detail coefficients at level $J-1$. A technical point that arises is that in applying the recurrence relationships to finite data, a few values of the c_k^J for $k < 0$ or $k \geq N$ may be needed. One way to cope with this difficulty is to extend the sequence c_k^J to the left and right using some specified boundary treatment.

Continuing by replacing the scaling coefficients at any level j by the scaling and detail coefficients at level $j-1$ yields a sequence of N coefficients

$$\{c_0^0, d_0^0, d_0^1, d_1^1, d_0^2, d_1^2, d_2^2, d_3^2, d_1^3, \ldots, d_7^3, \ldots, d_0^{J-1}, \ldots, d_{N/2-1}^{J-1}\}$$

This sequence is the finite discrete wavelet transform of the input data $\{y_k\}$. At any level j_0 the finite dimensional approximation of the function $f(x)$ is

$$f(x) \approx \sum_k c_k^{j_0} \phi_{j_0,k}(x) + \sum_{j=j_0}^{J-1} \sum_k d_k^j \psi_{j,k}(x)$$

Getting Started

Fourier Transform Infrared (FT-IR) spectroscopy is an important tool in analytic chemistry. This example demonstrates wavelet analysis applied to an FT-IR spectrum of quartz (Sullivan 2000). The following DATA step creates a data set containing the spectrum, expressed as an absorbance value for each of 850 wave numbers.

```
data quartzInfraredSpectrum;
   WaveNumber=4000.6167786 - _N_ *4.00084378;
   input Absorbance @@;
datalines;
  4783  4426  4419  4652  4764  4764  4621  4475  4430  4618
  4735  4735  4655  4538  4431  4714  4738  4707  4627  4523
  4512  4708  4802  4811  4769  4506  4642  4799  4811  4732
  4583  4676  4856  4868  4796  4849  4829  4677  4962  4994
  4924  4673  4737  5078  5094  4987  4632  4636  5010  5166
  5166  4864  4547  4682  5161  5291  5143  4684  4662  5221
  5640  5640  5244  4791  4832  5629  5766  5723  5121  4690
  5513  6023  6023  5503  4675  5031  6071  6426  6426  5723
  5198  5943  6961  7135  6729  5828  6511  7500  7960  7960
  7299  6484  7257  8180  8542  8537  7154  7255  8262  8898
  8898  8263  7319  7638  8645  8991  8991  8292  7309  8005
  9024  9024  8565  7520  7858  8652  8966  8966  8323  7513
  8130  8744  8879  8516  7722  8099  8602  8729  8726  8238
  7885  8350  8600  8603  8487  7995  8194  8613  8613  8408
  7953  8236  8696  8696  8552  8102  7852  8570  8818  8818
  8339  7682  8535  9038  9038  8503  7669  7794  8864  9163
  9115  8221  7275  8012  9317  9317  8512  7295  7623  9021
  9409  9338  8116  6860  7873  9282  9490  9191  7012  7392
  9001  9483  9457  8107  6642  7695  9269  9532  9246  7641
  6547  8886  9457  9457  8089  6535  7537  9092  9406  9178
  7591  6470  7838  9156  9222  7974  6506  7360  8746  9057
  8877  7455  6504  7605  8698  8794  8439  7057  7202  8240
  8505  8392  7287  6634  7418  8186  8229  7944  6920  6829
  7499  7949  7831  7057  6866  7262  7626  7626  7403  6791
  7062  7289  7397  7397  7063  6985  7221  7221  7199  6977
  7088  7380  7380  7195  6957  6847  7426  7570  7508  6952
  6833  7489  7721  7718  7254  6855  7132  7914  8040  7880
  7198  6864  7575  8270  8229  7545  7036  7637  8470  8570
  8364  7591  7413  8195  8878  8878  8115  7681  8313  9102
  9185  8981  8283  8197  8932  9511  9511  9101  8510  8670
  9686  9709  9504  8944  8926  9504  9964  9964  9627  9212
  9366  9889 10100  9939  9540  9512  9860 10121 10121  9828
  9567  9513  9782  9890  9851  9510  9385  9339  9451  9451
  9181  9076  9015  8960  9014  8957  8760  8760  8602  8584
  8584  8459  8469  8373  8279  8327  8282  8341  8341  8155
  8260  8260  8250  8350  8245  8358  8403  8355  8490  8490
  8439  8689  8689  8621  8680  8661  8897  9028  8900  8873
  8873  9187  9377  9377  9078  9002  9147  9635  9687  9535
  9127  9242  9824  9928  9775  9200  9047  9572 10102 10102
  9631  9024  9209 10020 10271  9830  9062  9234 10154 10483
 10453  9582  9011  9713 10643 10701 10372  9368  9857 10865
 10936 10572  9574  9691 10820 11452 11452 10623  9903 10787
```

```
 11931 12094 11302 10604 11458 12608 12808 12589 11629 11795
 12863 13575 13575 12968 12498 13268 14469 14469 13971 13727
 14441 15334 15515 15410 14986 15458 16208 16722 16722 16618
 17061 17661 18089 18089 18184 18617 19015 19467 19633 19830
 20334 20655 20947 21347 21756 22350 22584 22736 22986 23412
 24126 24498 24501 24598 24986 25729 26356 26356 26271 26754
 27624 28162 28162 28028 28305 29223 30073 30219 30185 30308
 31831 32699 32819 32793 33320 34466 35600 36038 36086 36518
 37517 38765 39462 39681 40209 41243 42274 42772 42876 43172
 43929 44842 45351 45395 45551 46035 46774 47353 47353 47362
 47908 48539 48936 48978 49057 49497 50101 50670 50914 51134
 51603 52276 53007 53399 53769 54281 54815 54914 55365 55874
 56180 56272 56669 57076 57422 57458 57525 57681 57679 57318
 57318 57181 57417 57409 57144 57047 56377 56551 56483 56098
 56034 55598 55364 55364 55146 54904 54990 55501 55533 55362
 54387 55340 55240 54748 53710 55346 55795 55795 55060 55945
 55945 55753 56759 56859 57509 56741 56273 56961 58566 58566
 58104 59275 59275 59051 59090 59461 60362 60560 61103 61272
 61380 61878 62067 62237 62214 61182 61532 62173 62253 60473
 61346 63143 63378 61519 61753 63078 63841 63841 62115 61227
 63237 63237 61338 63951 63951 63604 63633 64625 65135 64976
 63630 63494 63834 63338 63218 62324 64131 64234 65122 64551
 64127 64415 64621 64621 63142 65344 65585 65476 65074 64714
 63803 65085 65085 65646 65646 64851 65390 65390 64997 65541
 65587 65682 65952 65952 65390 65702 65846 65734 65734 65628
 65509 65571 65636 65636 65620 65487 65544 65547 65738 65758
 65711 65360 65362 65362 65231 65333 65453 65473 65435 65302
 65412 65412 65351 65242 65242 65170 65221 65297 65297 65202
 65177 65183 65184 65179 65209 65209 65144 65134 65113 65009
 64919 64945 64988 64988 64856 64686 64529 64370 64282 64233
 64169 63869 63685 63480 63373 63349 63307 63131 63017 62885
 62736 62736 62706 62666 62622 62671 62781 62853 62950 63106
 63135 63141 63220 63263 63489 63807 63966 64132 64294 64612
 64841 64985 65159 65204 65259 65540 65707 65749 65732 65719
 65820 65895 65925 65925 65888 65937 66059 66109 66109 66078
 66007 65897 65897 65747 65490 64947 64598 64363 64140 63801
 63571 63395 63333 63442 63442 63339 63196 62911 62118 61795
 61454 61456 61607 62025 62190 62190 62023 61780 61502 61482
 61458 61320 61015 60852 60708 60684 60522 60488 60506 60640
 60797 60995 61141 61141 61036 60664 60522 60017 59681 59129
 58605 58035 57192 56137 54995 53586 52037 50283 48565 45419
 43341 41111 36131 35377 34431 31679 29237 26898 24655 22417
 19876 17244 15176 12575 10532  8180  6040  4059  2210   575
;
```

The following statements produce the line plot of these data displayed in Figure 1.1.

```
symbol1 c=black i=join v=none;
proc gplot data=quartzInfraredSpectrum;
   plot Absorbance*WaveNumber/
        hminor = 0   vminor = 0
        vaxis  = axis1
        hreverse frame;
```

```
        axis1 label = ( r=0 a=90 );
run;
```

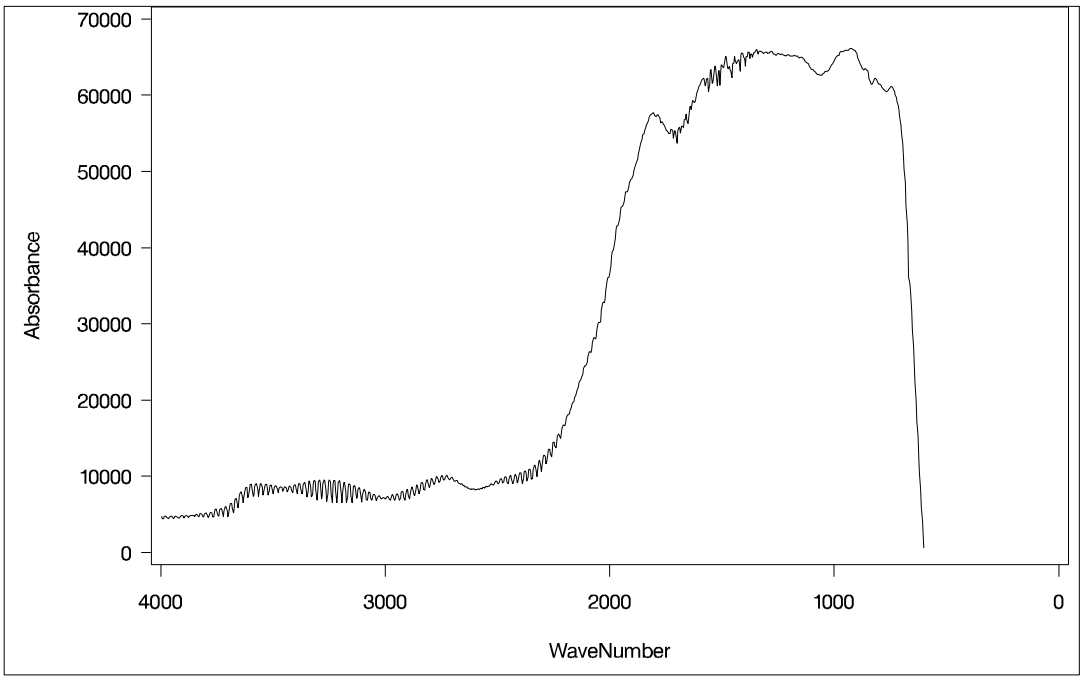


Figure 1.1. FT-IR Spectrum of Quartz

This data contains information at two distinct scales, namely a low frequency underlying curve superimposed with a high frequency oscillation. Notice that the oscillation is not uniform but that it occurs in several distinct bands. Wavelet analysis is an appropriate tool for providing insight into this type of data as it enables you to identify the frequencies present in the absorbance data as the wave number changes. This property of wavelets is known as "time frequency localization"; in this case the role of time is played by WaveNumber. Also note that the dependent variable Absorbance is measured at equally spaced values of the independent variable WaveNumber. This condition is necessary for the direct use of the discrete wavelet transform that is implemented in the SAS/IML wavelet functions.

Creating the Wavelet Decomposition

The following SAS code starts the wavelet analysis:

```
%wavginit;
proc iml;
   %wavinit;
```

Notice that the previous code segment includes two SAS macro calls. You can use the IML wavelet functions without using the WAVGINIT and WAVINIT macros. The macros are called to initialize and load IML modules that you can use to produce several standard wavelet diagnostic plots. These macros have been provided as autocall macros that you can invoke directly in your SAS code.

The WAVGINIT macro must be called prior to invoking PROC IML. This macro defines several macro variables that are used to adjust the size, aspect ratio, and font size for the plots produced by the wavelet plot modules. This macro can also take several optional arguments that control the positioning and and size of the wavelet diagnostic plots. See "Obtaining Help for Wavelet Modules and Macros" on page 32 for details on getting help about this macro call.

The WAVINIT macro must be called from within PROC IML. It loads the IML modules that you can use to produce wavelet diagnostic plots. This macro also defines symbolic macro variables that you can use to improve the readability of your code.

The following statements read the absorbance variable into an IML vector:

```
use quartzInfraredSpectrum;
read all var{Absorbance} into absorbance;
```

You are now in a position to begin the wavelet analysis. The first step is to set up the options vector that specifies which wavelet and what boundary handling you want to use. You do this as follows:

```
optn              =  &waveSpec;   /* optn=j(1,4,.); */
optn[&family]     =  &daubechies; /* optn[3] = 1;   */
optn[&member]     =  3;           /* optn[4] = 3;   */
optn[&boundary]   =  &polynomial; /* optn[1] = 3;   */
optn[&degree]     =  &linear;     /* optn[2] = 1;   */
```

These statements use macro variables that are defined in the WAVINIT macro. The equivalent code without using these macro variables is given in the adjacent comments. As indicated by the suggestive macro variable names, this options vector specifies that the wavelet to be used is the third member of the Daubechies wavelet family and that boundaries are to be handled by extending the signal as a linear polynomial at each endpoint.

The next step is to create the wavelet decomposition with the following call:

```
call wavft(decomp,absorbance,optn);
```

This call computes the wavelet transform specified by the vector optn of the input vector absorbance. The specified transform is encapsulated in the vector decomp. This vector is not intended to be used directly. Rather you use this vector as an argument to other IML wavelet subroutines and plot modules. For example, you use the WAVPRINT subroutine to print the information encapsulated in a wavelet decomposition. The following code produces output in Figure 1.2.

```
call wavprint(decomp,&summary);
call wavprint(decomp,&detailCoeffs,1,4);
```

Decomposition Summary

Decomposition Name	DECOMP
Wavelet Family	Daubechies Extremal Phase
Family Member	3
Boundary Treatment	Recursive Linear Extension
Number of Data Points	850
Start Level	0

Wavelet Detail Coefficients for DECOMP

Translate	Level 1	Level 2	Level 3	Level 4
0	-1.70985E-9	1.31649E-10	-8.6402E-12	5.10454E-11
1	1340085.30	-128245.70	191.084707	4501.36
2		62636.70	6160.27	-1358.23
3		-238445.36	-54836.56	-797.724143
4			39866.95	676.034389
5			-28836.85	-5166.59
6			223421.00	-6088.99
7				-5794.67
8				30144.74
9				-3903.53
10				638.063264
11				-10803.45
12				33616.35
13				-50790.30

Figure 1.2. Output of WAVPRINT CALLS

Usually such displayed output is of limited use. More frequently you will want to represent the transformed data graphically or use the results in further computational routines. As an example, you can estimate the noise level of the data using a robust measure of the standard deviation of the highest level detail coefficients, as demonstrated in the following statements:

```
call wavget(tLevel,decomp,&topLevel);
call wavget(noiseCoeffs,decomp,&detailCoeffs,tLevel-1);

noiseScale=mad(noiseCoeffs,"nmad");
print "Noise scale = " noiseScale;
```

The result is shown in Figure 1.3;

NOISESCALE

Noise scale = 169.18717

Figure 1.3. Scale of Noise in the Absorbance Data

The first WAVGET call is used to obtain the top level number in the wavelet decomposition decomp. The highest level of detail coefficients are defined at one level below the top level in the decomposition. The second WAVGET call returns these coefficients in the vector noiseCoeffs. Finally, the MAD function computes a robust estimate of the standard deviation of these coefficients.

Wavelet Coefficient Plots

Diagnostic plots greatly facilitate the interpretation of a wavelet decomposition. One standard plot is the detail coefficients arranged by level. Using a module included by the WAVINIT macro call, you can produce the plot shown in Figure 1.5 as follows:

```
call coefficientPlot(decomp, , , , ,"Quartz Spectrum");
```

The first argument specifies the wavelet decomposition and is required. All other arguments are optional and need not be specified. You can use the WAVHELP macro to obtain a description of the arguments of this and other wavelet plot modules. The WAVHELP macro is defined in autocall the WAVINIT macro. For example, invoking the WAVHELP macro as follows writes the calling information shown in Figure 1.4 to the SAS log.

```
%wavhelp(coefficientPlot);
```

```
      coefficientPlot Module

Function: Plots wavelet detail coefficients

Usage: call coefficientPlot(decomposition,
                            threshopt,
                            startLevel,
                            endLevel,
                            howScaled,
                            header);

Arguments:
   decomposition - (required) valid wavelet decompostion produced
                              by the IML subroutine WAVFT
   threshopt     - (optional) numeric vector of 4 elements
                              specifying thresholding to be used
                              Default: no thresholding
   startLevel    - (optional) numeric scalar specifying the lowest
                              level to be displayed in the plot
                              Default: start level of decomposition
   endLevel      - (optional) numeric scalar specifying the highest
                              level to be displayed in the plot
                              Default: end level of decomposition
   howScaled     - (optional) character: 'absolute' or 'uniform'
                              specifies coefficients are scaled uniformly
                              Default: independent level scaling
   header        - (optional) character string specifying a header
                              Default: no header
```

Figure 1.4. Log Output Produced by %wavhelp(coefficientPlot) Call

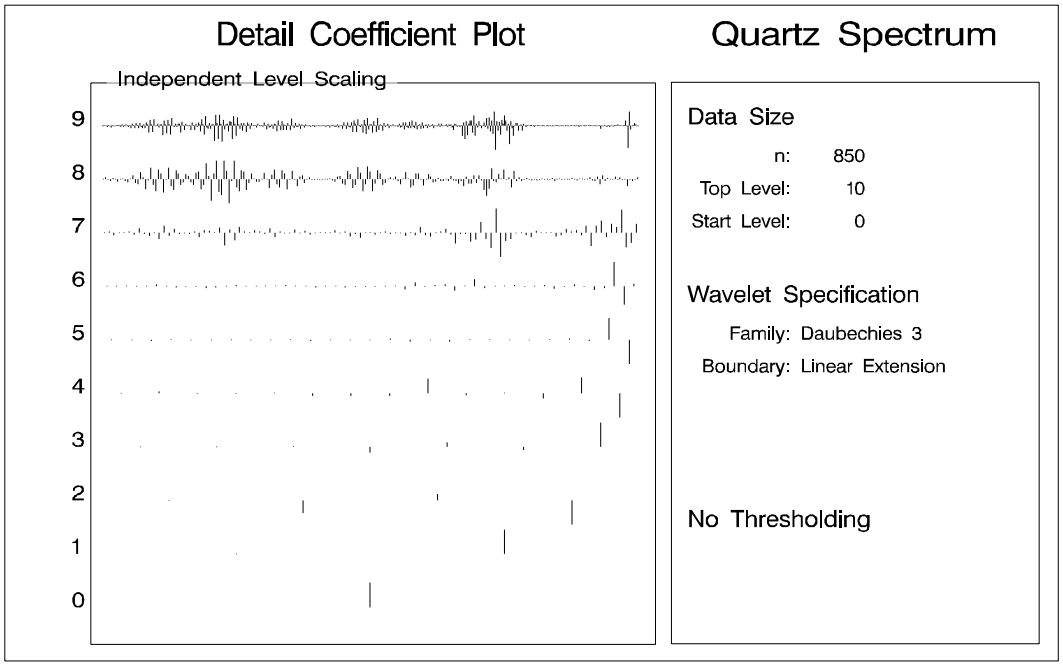


Figure 1.5. Detail Coefficients Scaled by Level

In this plot the detail coefficients at each level are scaled independently. The oscillations present in the absorbance data are captured in the detail coefficients at levels 7, 8, and 9. The following statement produces a coefficient plot of just these higher level detail coefficients and shows them scaled uniformly.

```
call coefficientPlot(decomp, ,7, ,
                     'uniform',"Quartz Spectrum");
```

The plot is shown in Figure 1.6.

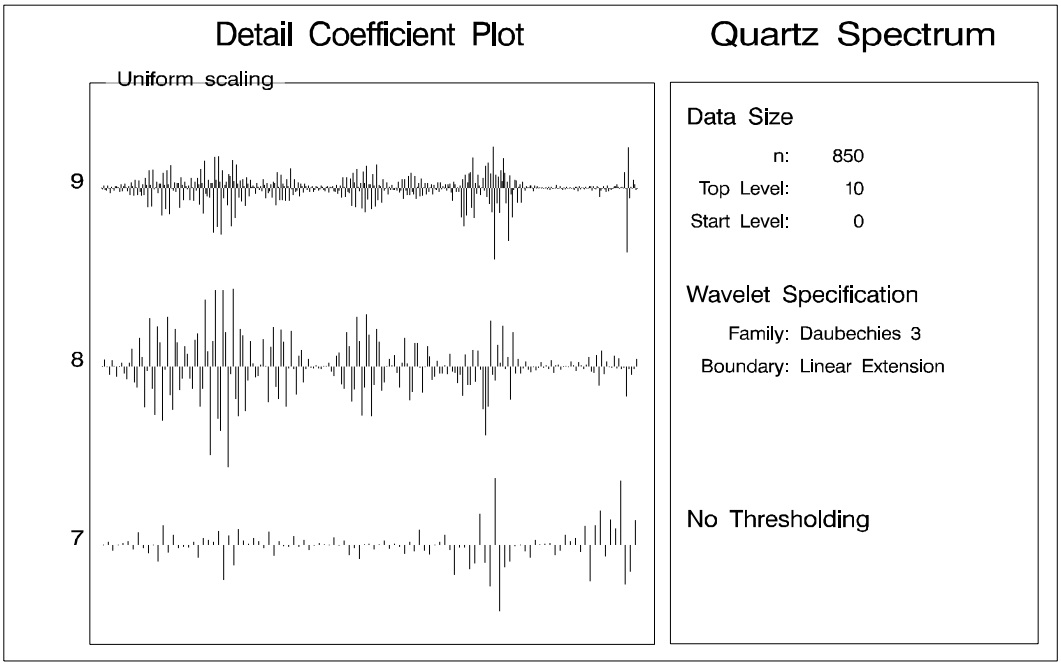


Figure 1.6. Uniformly Scaled Detail Coefficients

As noted earlier, noise in the data is captured in the detail coefficients, particularly in the small coefficients at higher levels in the decomposition. By zeroing or shrinking these coefficients, you can get smoother reconstructions of the input data. This is done by specifying a threshold value for each level of detail coefficients and then zeroing or shrinking all the detail coefficients below this threshold value. The IML wavelet functions and modules support several policies for how this thresholding is performed as well as for selecting the thresholding value at each level. See the "WAVIFT Call" on page 26 for details.

An options vector is used to specify the desired thresholding; several standard choices are predefined as macro variables in the WAVINIT module. The following statements produce the detail coefficient plot with the "SureShrink" thresholding algorithm of Donoho and Johnstone (1995).

```
call coefficientPlot(decomp,&SureShrink,6,, ,
                     "Quartz Spectrum");
```

The plot is shown in Figure 1.7.

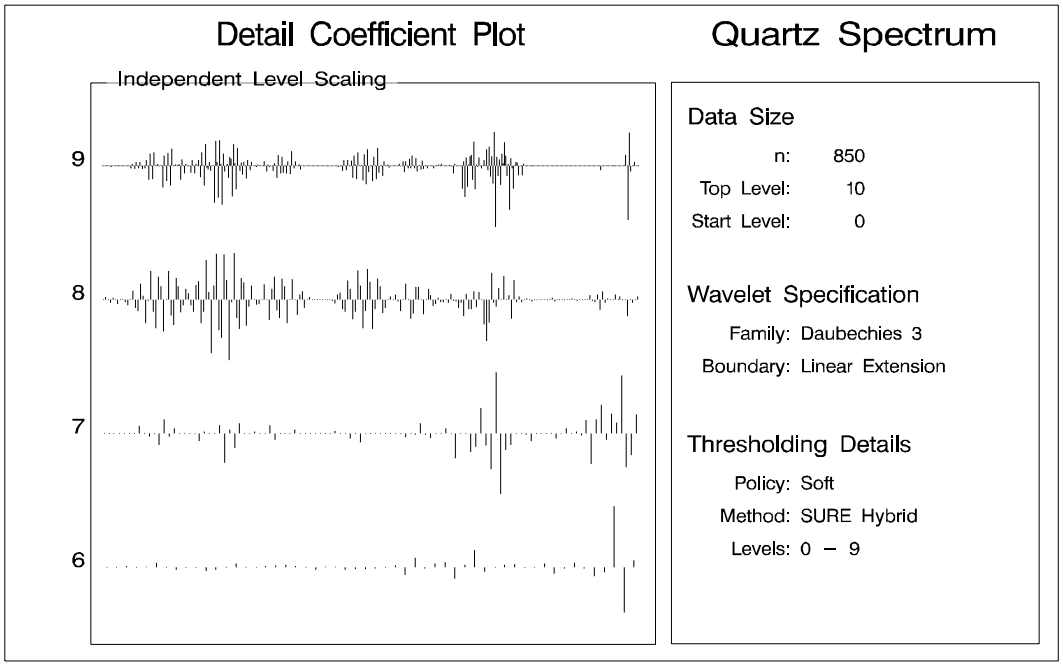


Figure 1.7. Thresholded Detail Coefficients

You can see that "SureShrink" thresholding has zeroed some of the detail coefficients at the higher levels but the larger coefficients that capture the oscillation in the data are still present. Consequently, reconstructions of the the input signal using the thresholded detail coefficients will still capture the essential features of the data, but will be smoother as much of the very fine scale detail has been eliminated.

Multiresolution Approximation Plots

One way of presenting reconstructions is in a multiresolution approximation plot. In this plot reconstructions of the input data are shown by level. At any level the reconstruction at that level uses only the detail and scaling coefficients defined below that level.

The following statement produces such a plot, starting at level 3:

```
call mraApprox(decomp, ,3, ,"Quartz Spectrum");
```

The results are shown in Figure 1.8.

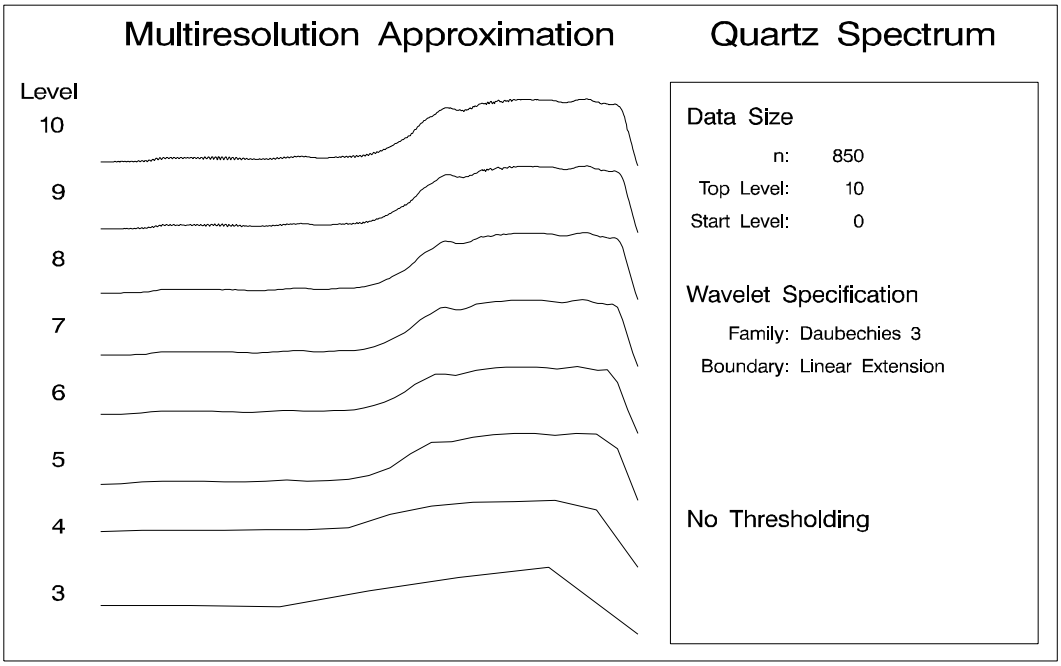


Figure 1.8. Multiresolution Approximation

You can see that even at level 3, the basic form of the input signal has been captured. As noted earlier, the oscillation present in the absorbance data is captured in the detail coefficients above level 7. Thus, the reconstructions at level 7 and below are largely free of these oscillation since they do not use any of the higher detail coefficients. You can confirm this observation by plotting just this level in the multiresolution analysis as follows:

```
call mraApprox(decomp, ,7,7,"Quartz Spectrum");
```

The results are shown in Figure 1.9.

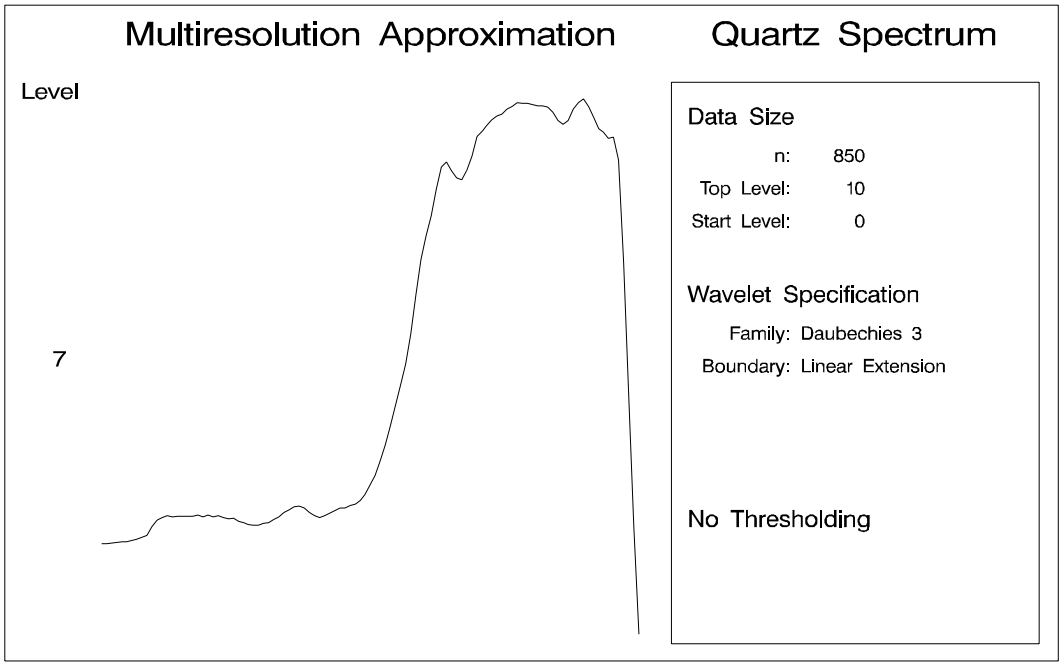


Figure 1.9. Level 7 of the Multiresolution Approximation

You can also plot the multiresolution approximations obtained with thresholded detail coefficients. For example, the following statement plots the top level reconstruction obtained using the "SureShrink" threshold:

```
call mraApprox(decomp,&SureShrink,10,10,
               "Quartz Spectrum");
```

The results are shown in Figure 1.10.

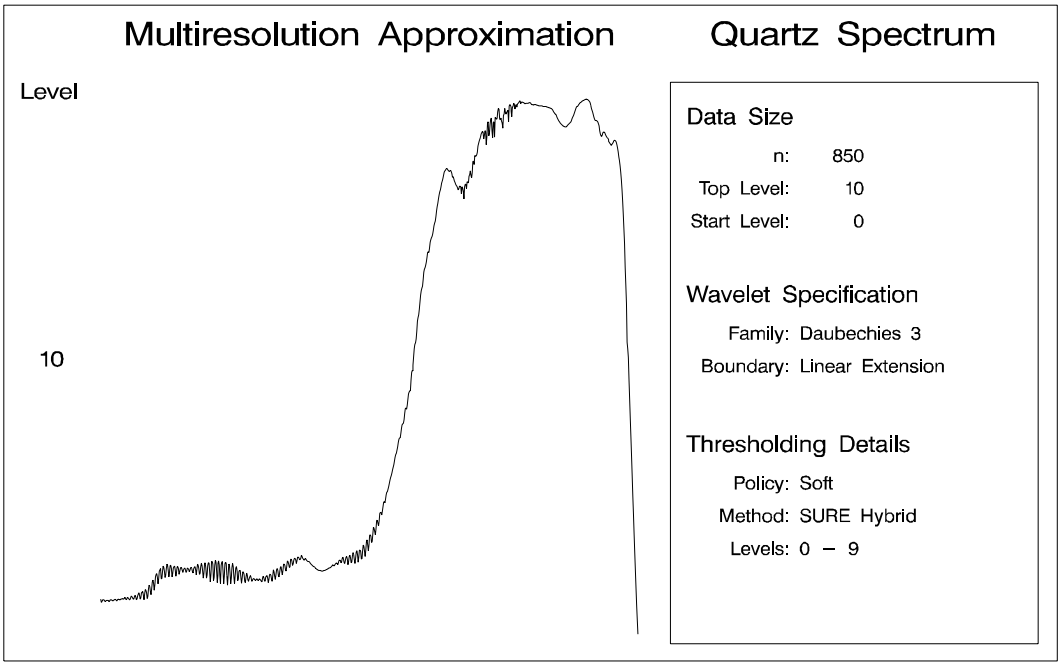


Figure 1.10. Top Level of Multiresolution Approximation with SureShrink Thresholding Applied

Note that the high frequency oscillation is still present in the reconstruction even with "SureShrink" thresholding applied.

Multiresolution Decomposition Plots

A related plot is the multiresolution decomposition plot, which shows the detail coefficients at each level. For convenience, the starting level reconstruction at the lowest level of the plot and the reconstruction at the highest level the plot are also included. Adding suitably scaled versions of all the detail levels to the starting level reconstruction recovers the final reconstruction. The following statement produces such a plot, yielding the results shown in Figure 1.11.

```
call mraDecomp(decomp, ,5, , ,"Quartz Spectrum");
```

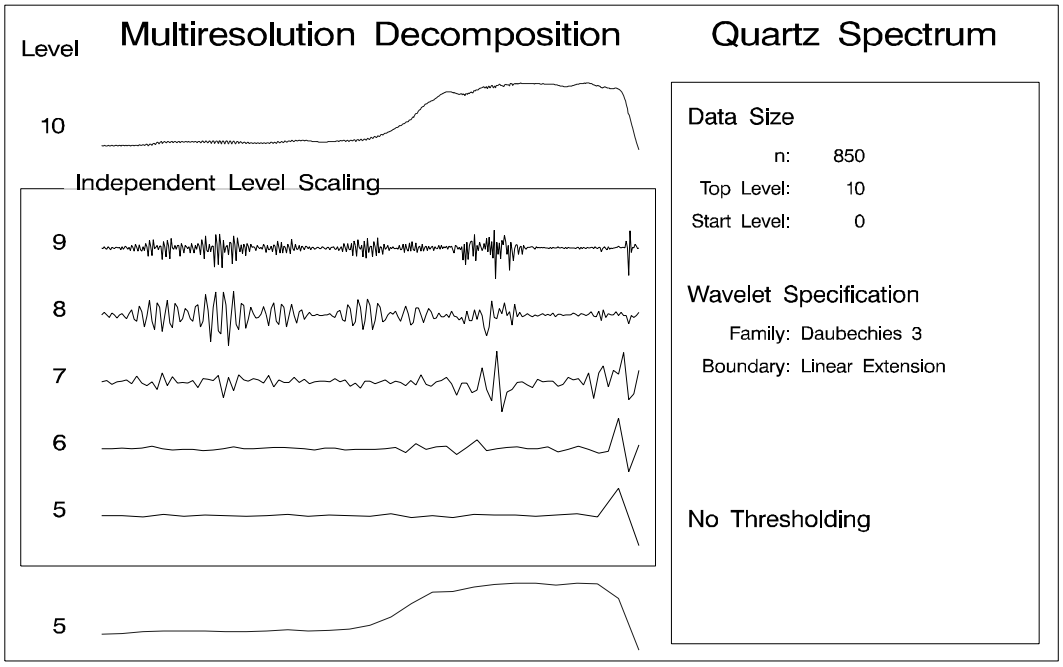


Figure 1.11. Multiresolution Decomposition

Wavelet Scalograms

Wavelet scalograms communicate the time frequency localization property of the discrete wavelet transform. In this plot each detail coefficient is plotted as a filled rectangle whose color corresponds to the magnitude of the coefficient. The location and size of the rectangle are related to the time interval and the frequency range for this coefficient. Ccoefficients at low levels are plotted as wide and short rectangles to indicate that they localize a wide time interval but a narrow range of frequencies in the data. In contrast, rectangles for coefficients at high levels are plotted thin and tall to indicate that they localize small time ranges but large frequency ranges in the data. The heights of the rectangles grow as a power of 2 as the level increases. If you include all levels of coefficients in such a plot, the heights of the rectangles at the lowest levels are so small that they will not be visible. You can use an option to plot the heights of the rectangles on a logarithmic scale. This results in rectangles of uniform height but requires that you interpret the frequency localization of the coefficients with care.

The following statement produces a scalogram plot of all levels with “SureShrink” thresholding applied:

```
call scalogram(decomp,&SureShrink, , ,0.25,
               'log',"Quartz Spectrum");
```

The sixth argument specifies that the rectangle heights are to be plotted on a logarithmic scale. The role of the fifth argument (0.25) is to amplify the magnitude of the small detail coefficients. This is necessary since the detail coefficients at the lower

levels are orders of magnitude larger than those at the higher levels. The amplification is done by first scaling the magnitudes of all detail coefficients to lie in the interval $[0, 1]$ and then raising these scaled magnitudes to the power 0.25. Note that smaller powers yield larger amplification of the small detail coefficient magnitudes. The default amplification is $1/3$.

The results are shown in Figure 1.12.

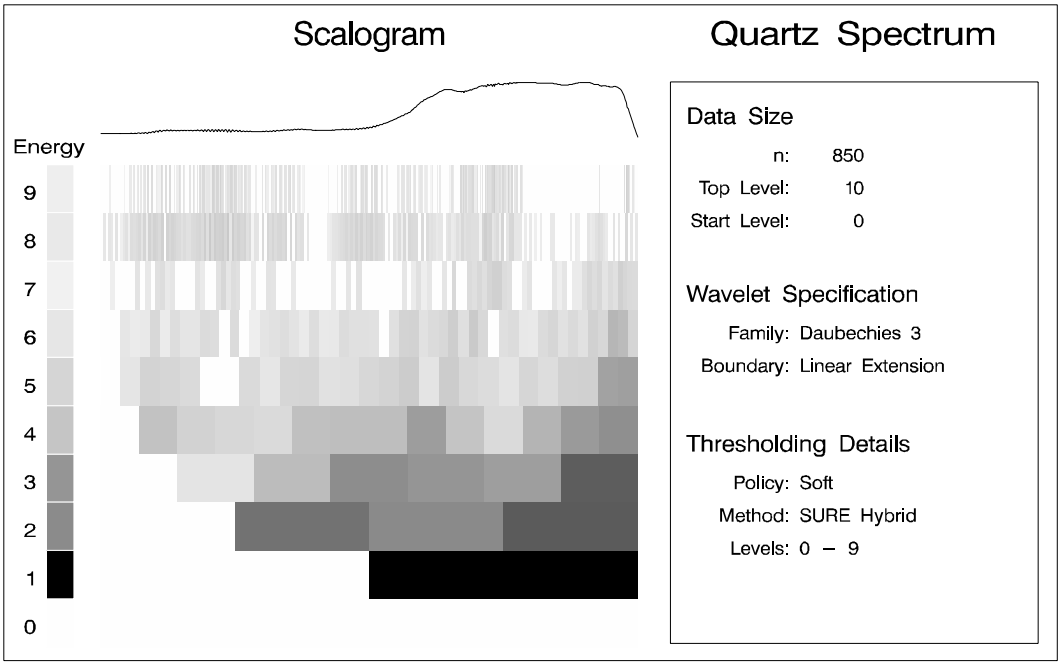


Figure 1.12. Scalogram Showing All Levels

The bar on the left-hand side of the scalogram plot indicates the overall energy of each level. This energy is defined as the sum of the squares of the detail coefficients for each level. These energies are amplified using the same algorithm for amplifying the detail coefficient magnitudes. The energy bar in Figure 1.12 shows that higher energies occur at the lower levels whose coefficients capture the gross features of the data. In order to interpret the finer-scale details of the data it is helpful to focus on just the higher levels. The following statement produces a scalogram for levels 6 and above without using a logarithmic scale for the rectangle heights, and using the default coefficient amplification.

```
call scalogram(decomp,&SureShrink,6, , , ,
               "Quartz Spectrum");
```

The result is shown in Figure 1.13.

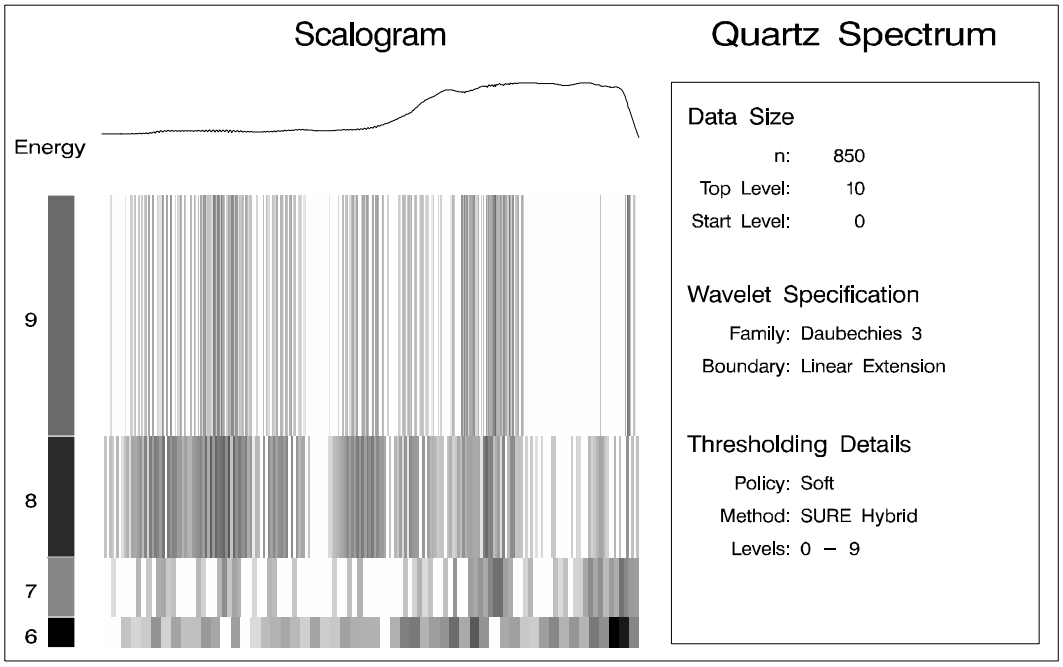


Figure 1.13. Scalogram of Levels 6 and Above Using SureShrink Thresholding

The scalogram in Figure 1.13 reveals that most of the energy of the oscillation in the data is captured in the detail coefficients at level 8. Also note that many of the coefficients at the higher levels are set to zero by "SureShrink" thresholding. You can verify this by comparing Figure 1.13 with Figure 1.14, which shows the corresponding scalogram except that no thresholding is done. The following statement produces Figure 1.14:

```
call scalogram(decomp, ,6, , , ,"Quartz Spectrum");
```

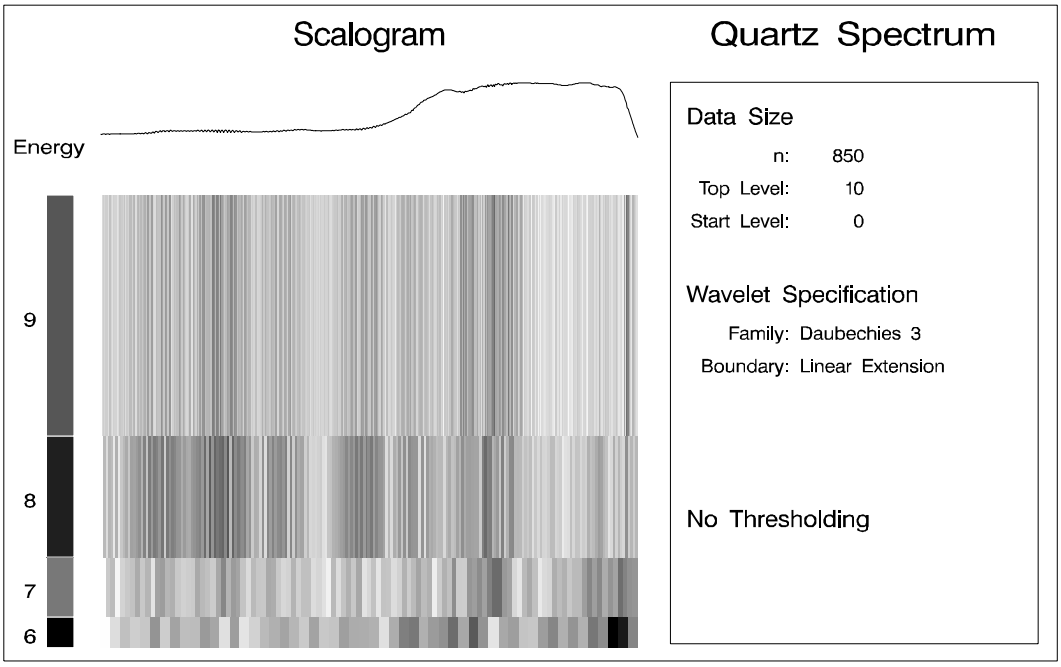


Figure 1.14. Scalogram of Levels 6 and Above Using No Thresholding

Reconstructing the Signal from the Wavelet Decomposition

You can use the WAVIFT subroutine to invert a wavelet transformation computed using the WAVFT subroutine. If no thresholding is specified, then up to numerical rounding error this inversion is exact. The following statements provide an illustration of this:

```
call wavift(reconstructedAbsorbance,decomp);
errorSS=ssq(absorbance-reconstructedAbsorbance);
print "The reconstruction error sum of squares = " errorSS;
```

The output is shown in Figure 1.15.

```
                                                    ERRORSS

The reconstruction error sum of squares =  1.288E-16
```

Figure 1.15. Exact Reconstruction Property of WAVIFT

Usually you use the WAVIFT subroutine with thresholding specified. This yields a smoothed reconstruction of the input data. You can use the following statements to create a smoothed reconstruction of absorbance and add this variable to the QuartzInfraredSpectrum data set.

```
call wavift(smoothedAbsorbance,decomp,&SureShrink);
create temp from smoothedAbsorbance[colname='smoothedAbsorbance'];
   append from smoothedAbsorbance;
```

```
   close temp;
quit;

data quartzInfraredSpectrum;
  set quartzInfraredSpectrum;
  set temp;
run;
```

The following statements produce the line plot of the smoothed absorbance data shown in Figure 1.16:

```
symbol1 c=black i=join v=none;
proc gplot data=quartzInfraredSpectrum;
   plot smoothedAbsorbance*WaveNumber/
        hminor = 0   vminor = 0
        vaxis  = axis1
        hreverse frame;
        axis1 label = ( r=0 a=90 );
run;
```

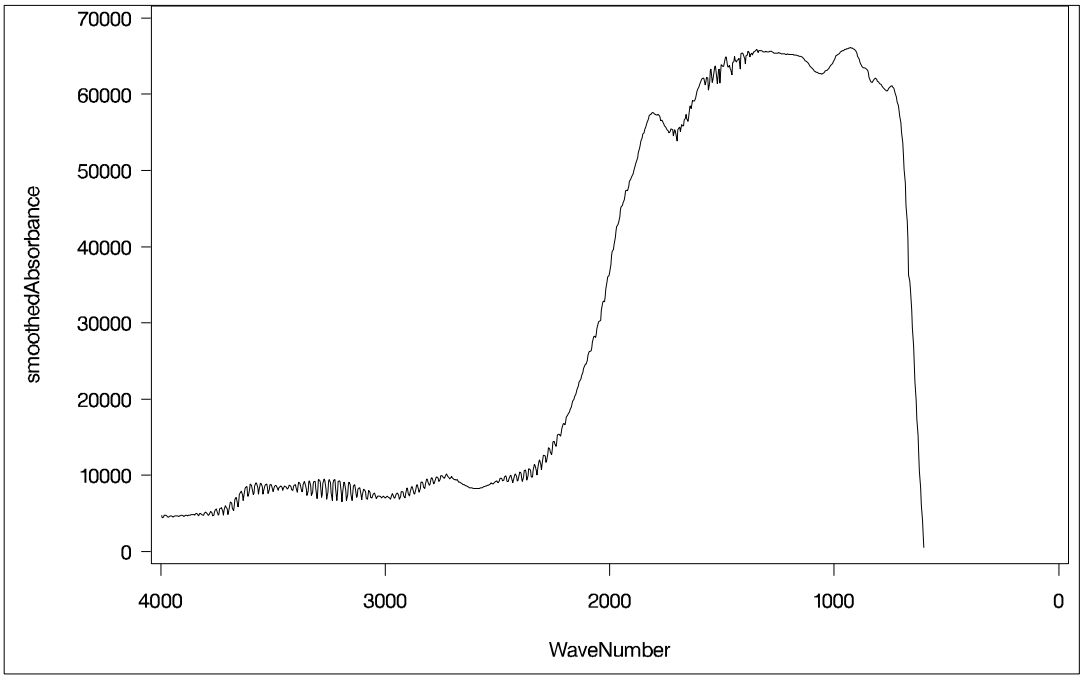


Figure 1.16. Smoothed FT-IR Spectrum of Quartz

You can see by comparing Figure 1.1 with Figure 1.16 that the wavelet smooth of the absorbance data has preserved all the essential features of this data.

Syntax

Wavelet Analysis Calls

WAVFT Call	computes a specified wavelet transform of one-dimensional data
WAVGET Call	returns requested information encapsulated in a wavelet transform
WAVIFT Call	inverts a wavelet transform after applying specified thresholding to the detail coefficients
WAVPRINT Call	displays requested information encapsulated in a wavelet transform
WAVTHRSH Call	applies specified thresholding to the detail coefficients of a wavelet transform

WAVFT Call

computes fast wavelet transform

CALL WAVFT(*decomp, data, opt* <, *levels*> **);**

The Fast Wavelet Transform (WAVFT) subroutine computes a specified discrete wavelet transform of the input data, using the algorithm of Mallat (1989). This transform decomposes the input data into sets of detail and scaling coefficients defined at a number of scales or "levels."

The input data are used as scaling coefficients at the top level in the decomposition. The fast wavelet transform then recursively computes a set of detail and a set of scaling coefficients at the next lower level by respectively applying "low pass" and "high pass" conjugate mirror filters to the scaling coefficients at the current level. The number of coefficients in each of these new sets is approximately half the number of scaling coefficients at the level above them. Depending on the filters being used, a number of additional scaling coefficients, known as *boundary coefficients*, may be involved. These boundary coefficients are obtained by extending the sequence of interior scaling coefficients using a specified method.

Details of the discrete wavelet transform and the fast wavelet transformation algorithm are available in many references, including Mallat (1989), Daubechies (1992), and Ogden (1997).

The inputs to the WAVFT subroutine are as follows:

data specifies the data to transform. This data must be either a row or column vector.

opt refers to an options vector with the following components:

opt[1] specifies the boundary handling used in computing the wavelet transform. At each level of the wavelet decomposition, neces-

sary boundary scaling coefficients are obtained by extending the interior scaling coefficients at that level as follows:

opt[1]=0 specifies extension by zero.
opt[1]=1 specifies periodic extension.
opt[1]=2 specifies polynomial extension.
opt[1]=3 specifies extension by reflection.
opt[1]=4 specifies extension by anti-symmetric reflection.

opt[2] specifies the polynomial degree that is used for polynomial extension. The value of *opt*[2] is ignored if *opt*[1]$\neq 2$.

opt[2]=0 specifies constant extension.
opt[2]=1 specifies linear extension.
opt[2]=2 specifies quadratic extension.

opt[3] specifies the wavelet family.

opt[3]=1 specifies the Daubechies Extremal phase family (Daubechies 1992).
opt[3]=2 specifies the Daubechies Least Asymmetric family (also known as the Symmlet family) (Daubechies 1992).

opt[4] specifies the wavelet family member. Valid values are

opt[4]=1 through 10, if *opt*[3]=1
opt[4]=4 through 10, if *opt*[3]=2

Some examples of wavelet specifications are

opt={1 . 1 1}; specifies the first member (more commonly known as the Haar system) of the Daubechies extremal phase family with periodic boundary handling.

opt={2 1 2 5}; specifies the fifth member of the Symmlet family with linear extension boundary handling.

levels is an optional scalar argument that specifies the number of levels from the top level to be computed in the decomposition. If you do not specify this argument, then the decomposition terminates at level 0. Usually, you will not need to specify this optional argument. You use this option to avoid unneeded computations in situations where you are interested in the detail and scaling coefficients at only higher levels.

The WAVFT subroutine returns

decomp a row vector that encapsulates the specified wavelet transform. The information that is encoded in this vector includes:

- the options specified for computing the transform
- the number of detail coefficients at each level of the decomposition
- all detail coefficients
- the scaling coefficients at the bottom level of the decomposition

- boundary scaling coefficients at all levels of the decomposition

Note: *decomp* is a private representation of the specified wavelet transform and is not intended to be interpreted in its raw form. Rather, you should use this vector as an input argument to the WAVIFT, WAVPRINT, WAVGET, and WAVTHRSH subroutines.

WAVGET Call

extracts wavelet information

CALL WAVGET(*result, decomp, request* <, *options*> **);**

The WAVGET subroutine is used to return information that is encoded in a wavelet decomposition.

The required inputs are

decomp specifies a wavelet decomposition that has been computed using a call to the WAVFT subroutine.

request specifies a scalar indicating what information is to be returned.

You can specify different optional arguments depending on the value of *request*:

request=1 requests the number of points in the input data vector.

result returns as a scalar containing this number.

request=2 requests the detail coefficients at a specified level. Valid syntax is

CALL WAVGET(*result, decomp, 2, level* <, *opt*> **);**

where the argument

level is the level at which the detail coefficients are requested.

opt is an optional vector that specifies the thresholding to be applied to the returned detail coefficients. See the WAVIFT subroutine call for details. If you omit this argument, no thresholding is applied.

result returns as a column vector containing the specified detail coefficients.

request=3 requests the scaling coefficients at a specified level. Valid syntax is

CALL WAVGET(*result, decomp, 3, level* <, *opt*> **);**

where the argument

level is the level at which the scaling coefficients are requested.

opt is an optional vector that specifies the thresholding to be applied. See the WAVIFT subroutine call for a description of this vector. The scaling coefficients at the requested level are obtained by using the inverse wavelet transform, after applying the specified thresholding. If you omit this argument, no thresholding is applied.

result returns as a column vector containing the specified scaling coefficients.

request=4 requests the thresholding status of the detail coefficients in *decomp*.

result returns as a scalar whose value is

0, if the detail coefficients have not been thresholded.

1, otherwise.

request=5 requests the wavelet options vector that you specified in the WAVFT subroutine call to compute *decomp.*

result returns as a column vector with four elements containing the specified options vector. See the WAVFT subroutine call for the interpretation of the vector entries.

request=6 requests the index of the top level in *decomp*.

result returns as a scalar containing this number.

request=7 requests the index of the lowest level in *decomp*.

result returns as a scalar containing this number.

request=8 requests a vector evaluating the father wavelet used in *decomp*, at an equally spaced grid spanning the support of the father wavelet. The number of points in the grid is specified as a power of 2 times the support width of the father wavelet. For wavelets in the Daubechies extremal phase and least asymmetric families, the support width of the father wavelet is $2m - 1$, where m is the family member. Valid syntax is

CALL WAVGET(*result, decomp, 8 <, power>* **);**

where the optional argument

power is the exponent of 2 determining the number of grid points used. *power* defaults to 8 if you do not specify this argument.

result returns as a column vector containing the specified evaluation of the father wavelet.

WAVIFT Call

computes inverse fast wavelet transform

CALL WAVIFT(*result, decomp* <, *opt* <, *level*>> **);**

The Inverse Fast Wavelet Transform (WAVIFT) subroutine computes the inverse wavelet transform of a wavelet decomposition computed using the WAVFT subroutine. Details of this algorithm are available in many references, including Mallat (1989), Daubechies (1992), and Ogden (1997).

The inverse transform yields an exact reconstruction of the original input data, provided that no smoothing is specified. Alternatively, a smooth reconstruction of the input data can be obtained by thresholding the detail coefficients in the decomposition prior to applying the inverse transformation. Thresholding, also known as shrinkage, replaces the detail coefficient $d_j^{(i)}$ at level i by $\delta_{T_i}(d_j^{(i)})$, where the $\delta_T(x)$ is a shrinkage function and T_i is the threshold value used at level i. The SAS/IML wavelet subroutines support hard and soft shrinkage functions (Donoho and Johnstone 1994) and the non-negative garrote shrinkage function (Breiman 1995). These functions are defined as follows:

$$\delta_T^{\text{hard}}(x) = \begin{cases} 0 & |x| \le T \\ x & |x| > T \end{cases}$$

$$\delta_T^{\text{soft}}(x) = \begin{cases} 0 & |x| \le T \\ x - T & x > T \\ x + T & x < -T \end{cases}$$

$$\delta_T^{\text{garrote}}(x) = \begin{cases} 0 & |x| \le T \\ x - T^2/x & |x| > T \end{cases}$$

You can specify several methods for choosing the threshold values. Methods in which the threshold T_i varies with the level i are called *adaptive.* Methods where the same threshold is used at all levels are called *global.*

The inputs to the WAVIFT subroutine are as follows:

decomp specifies a wavelet decomposition that has been computed using a call to the WAVFT subroutine.

opt refers to an options vector that specifies the thresholding algorithm. If this optional argument is not specified, then no thresholding is applied.

The options vector has the following components:

opt[1] specifies the thresholding policy.

opt[1]=0 specifies that no thresholding be done. If *opt*[1]=0 then all other entries in the options vector are ignored.

opt[1]=1 specifies hard thresholding.
opt[1]=2 specifies soft thresholding.
opt[1]=3 specifies garrote thresholding.

opt[2] specifies the method for selecting the threshold.

opt[2]=0 specifies a global user-supplied threshold.
opt[2]=1 specifies a global threshold chosen using the minimax criterion of Donoho and Johnstone (1994).
opt[2]=2 specifies a global threshold defined using the universal criterion of Donoho and Johnstone (1994).
opt[2]=3 specifies an adaptive method where the thresholds at each level i are chosen to minimize an approximation of the L^2 risk in estimating the true data values using the reconstruction with thresholded coefficients (Donoho and Johnstone 1995).
opt[2]=4 specifies a hybrid method of Donoho and Johnstone (1995). The universal threshold as specified by *opt*[2]=2 is used at levels where most of the detail coefficients are essentially zero. The risk minimization method as specified by *opt*[2]=4 is used at all other levels.

opt[3] specifies the value of the global user-supplied threshold if *opt*[2]=1. It is ignored if *opt*[2] $\neq 1$.

opt[4] specifies the number of levels starting at the highest detail coefficient level at which thresholding is to be applied. If this value is negative or missing, thresholding is applied at all levels in *decomp*.

Some common examples of threshold options specifications are:

opt={1 3 . -1}; specifies hard thresholding with a minimax threshold applied at all levels in the decomposition. This threshold is named "*RiskShrink*" in Donoho and Johnstone (1994).

opt={2 2 . -1}; specifies soft thresholding with a universal threshold applied at all levels in the decomposition. This threshold is named "*VisuShrink*" in Donoho and Johnstone (1994).

opt={2 4 . -1}; specifies soft thresholding with level-dependent thresholds that minimize the Stein Unbiased Estimate of Risk (SURE). This threshold is named "*SureShrink*" in Donoho and Johnstone (1995).

level is an optional scalar argument that specifies the level at which the reconstructed data are to be returned. If this argument is not specified then the reconstructed data are returned at the top level defined in *decomp*.

The WAVIFT subroutine returns

result a vector obtained by inverting, after thresholding the detail coefficients, the discrete wavelet transform encoded in *decomp*. The row or column orientation of *result* is the same as that of the input data specified in the corresponding WAVFT subroutine call. If you specify the optional *level* argument, *result*

contains the reconstruction at the specified level, otherwise the reconstruction corresponds to the top level in the decomposition.

WAVPRINT Call

displays wavelet information

CALL WAVPRINT(*decomp, request* <, *options*> **);**

The WAVPRINT subroutine is used to display the information that is encoded in a wavelet decomposition.

The required inputs are

decomp specifies a wavelet decomposition that has been computed using a call to the WAVFT subroutine.

request specifies a scalar indicating what information is to be displayed.

You can specify different optional arguments depending on the value of *request*:

request=1 displays information about the wavelet family used to perform the wavelet transform. No additional arguments need to be specified.

request=2 displays the detail coefficients by level. Valid syntax is

CALL WAVPRINT(*decomp, 2* <, *lower* <, *upper*>> **);**

where the argument

lower is optional and specifies the lowest level to be displayed. The default value of *lower* is the lowest level in *decomp.*

upper is optional and specifies the upper level to be displayed. The default value of *upper* is the highest detail level in *decomp*.

request=3 displays the scaling coefficients by level. Valid syntax is

CALL WAVPRINT(*decomp,3* < , *lower* <, *upper*>> **);**

where the argument

lower is optional and specifies the lowest level to be displayed. The default value of *lower* is the lowest level in *decomp.*

upper is optional and specifies the upper level to be displayed. The default value of *upper* is the top level in *decomp.*

request=4 displays thresholded detail coefficients by level. Valid syntax is

CALL WAVPRINT(*decomp, 4, opt <, lower <, upper>>* **);**

where the argument

opt is a required options vector that specifies the thresholding algorithm used. See the WAVIFT subroutine call for a description of this options vector.

lower is optional and specifies the lowest level to be displayed. The default value of *lower* is the lowest level in *decomp.*

upper is optional and specifies the upper level to be displayed. The default value of *upper* is the highest detail level in *decomp*.

WAVTHRSH Call

thresholds wavelet detail coefficients

CALL WAVTHRSH(*decomp, opt* **);**

The Wavelet Threshold (WAVTHRSH) subroutine thresholds the detail coefficients in a wavelet decomposition.

The required inputs are

decomp specifies a wavelet decomposition that has been computed using a call to the WAVFT subroutine.

opt refers to an options vector that specifies the thresholding algorithm used. See the WAVIFT subroutine call for a description of this options vector.

On return, the detail coefficients encoded in *decomp* are replaced by their thresholded values. Note that this action is not reversible. If you want to retain the original detail coefficients, you should not use the WAVTHRSH subroutine to do thresholding. Rather, you should supply the thresholding argument where appropriate in the WAVIFT, WAVGET, and WAVPRINT subroutine calls.

Details

Using Symbolic Names

Several of the wavelet subroutines take arguments that are options vectors that specify user input. For example, the third argument in a WAVFT subroutine call is an options vector that specifies which wavelet and which boundary treatment are used in computing the wavelet transform. Typical code that defines this options vector is

```
optn    = j(1, 4, .);
optn[1] = 0;
optn[3] = 1;
optn[4] = 3;
```

A problem with such code is that it is not easily readable. By using symbolic names readability is greatly enhanced. SAS macro variables provide a convenient mechanism for creating such symbolic names. For example, the previous code could be replaced by

```
optn            = &waveSpec;
optn[&family]   = &daubechies;
optn[&member]   = 3;
optn[&boundary] = &zeroExtension;
```

where the symbolic macro variables (names with a preceding ampersand) resolve to the relevant quantities. Another example where symbolic names improve code readability is to use symbolic names for an integer argument that controls what action a multipurpose subroutine performs. An illustration is replacing code such as

```
call wavget(n,decomposition,1);
call wavget(fWavelet,decompostion,8);
```

by

```
call wavget(n,decomposition,&numPoints);
call wavget(fWavelet,decompostion,&fatherWavelet);
```

A set of symbolic names is defined in the autocall WAVINIT macro. The following tables list the symbolic names that are defined in this macro:

Table 1.1. Macro Variables for Wavelet Specification

Position		Admissible Values	
Name	**Value**	**Name**	**Value**
&boundary	1	&zeroExtension	0
		&periodic	1
		&polynomial	2
		&reflection	3
		&antisymmetricReflection	4
&degree	2	&constant	0
		&linear	1
		&quadratic	2
&family	3	&daubechies	1
		&symmlet	2
&member	4	1 - 10	

Table 1.2. Macro Variables for Threshold Specification

Position		Admissible Values	
Name	**Value**	**Name**	**Value**
&policy	1	&none	0
		&hard	1
		&soft	2
		&garrote	3
&method	2	&absolute	0
		&minimax	1
		&universal	2
		&sure	3
		&sureHybrid	4
		&nhoodCoeffs	5
&value	3	*positive real*	
&levels	4	&all	-1
		positive integer	

Table 1.3. Symbolic Names for the Third Argument of WAVGET

Name	**Value**
&numPoints	1
&detailCoeffs	2
&scalingCoeffs	3
&thresholdingStatus	4
&specification	5
&topLevel	6
&startLevel	7
&fatherWavelet	8

Table 1.4. Macro Variables for the Second Argument of WAVPRINT

Name	Value
&summary	1
&detailCoeffs	2
&scalingCoeffs	3
&thresholdedDetailCoeffs	4

Table 1.5. Macro Variables for Predefined Wavelet Specifications

Name		&boundary	&degree	&family	&member	
&waveSpec	{	.	.	.	.	}
&haar	{	&periodic	.	&daubechies	1	}
&daubechies3	{	&periodic	.	&daubechies	3	}
&daubechies5	{	&periodic	.	&daubechies	5	}
&symmlet5	{	&periodic	.	&symmlet	5	}
&symmlet8	{	&periodic	.	&symmlet	8	}

Table 1.6. Macro Variables for Predefined Threshold Specifications

Name		&policy	&method	&value	&levels	
&threshSpec	{	.	.	.	.	}
&RiskShrink	{	&hard	&minimax	.	&all	}
&VisuShrink	{	&soft	&universal	.	&all	}
&SureShrink	{	&soft	&sureHybrid	.	&all	}

Obtaining Help for the Wavelet Macros and Modules

The WAVINIT macro that you call to define symbolic macro variables and wavelet plot modules also defines a macro WAVHELP that you can call to obtain help for the wavelet macros and plot modules. The syntax for calling the WAVHELP macro is

%WAVHELP< *(name)*>;

where *name* is one of wavginit, wavinit, coefficientPlot, mraApprox, mraDecomp, or scalogram. This macro displays usage and argument information for the specified macro or module. If you call the WAVHELP macro with no arguments, it lists the names of the macros and modules for which help is available. Note that you can obtain help for the built-in IML wavelet subroutines using the SAS Online Help.

References

Daubechies, I. (1992), *Ten Lectures on Wavelets*, Volume 61, CBMS-NSF Regional Conference Series in Applied Mathematics, Philadelphia, PA: Society for Industrial and Applied Mathematics.

Donoho, D.L. and Johnstone, I.M. (1994), "Ideal Spatial Adaptation via Wavelet

Shrinkage," *Biometrika*, 81, 425–455.

Donoho, D.L. and Johnstone, I.M. (1995), "Adapting to Unknown Smoothness via Wavelet Shrinkage," *Journal of the American Statistical Association*, 90, 1200–1224.

Mallat, S. (1989), "Multiresolution Approximation and Wavelets," *Transactions of the American Mathematical Society*, 315, 69–88.

Ogden, R.T. (1997), *Essential Wavelets for Statistical Applications and Data Analysis*, Boston: Birkhäuser.

Sullivan, D. (2000), "FT-IR Library," [http://www.che.utexas.edu/~dls/ir/ir_dir.html], accessed 16 October 2000.

Chapter 2

Fractionally Integrated Time Series Analysis

Chapter Table of Contents

Chapter 2
Fractionally Integrated Time Series Analysis

Overview

This chapter describes SAS/IML subroutines related to fractionally integrated time series analysis.

The following subroutines are supported:

FARMACOV	computes the auto-covariance function for a fractionally integrated ARMA model
FARMAFIT	estimates the parameters for a fractionally integrated ARMA model
FARMALIK	computes the log-likelihood function for a fractionally integrated ARMA model
FARMASIM	generates a fractionally integrated ARMA process
FDIF	computes a fractionally differenced process

Getting Started

Fractionally Integrated Time Series

The fractional differencing enables the degree of differencing d to take any real value rather than being restricted to integer values. The fractionally differenced processes are capable of modeling long-term persistence. The process

$$(1 - B)^d y_t = \epsilon_t$$

is known as a fractional Gaussian noise process or an ARFIMA$(0, d, 0)$ process, where $d \in (-1, 1)\backslash\{0\}$, ϵ_t is a white noise process with mean zero and variance σ_ϵ^2, and B is the backshift operator such that $B^j \mathbf{y}_t = \mathbf{y}_{t-j}$. The extension of an ARFIMA$(0, d, 0)$ model combines fractional differencing with an ARMA(p, q) model, known as an ARFIMA(p, d, q) model.

Consider an ARFIMA$(0, 0.2, 0)$ represented as $(1 - B)^{0.2} y_t = \epsilon_t$ where $\epsilon_t \sim NID(0, 1)$. With the following statements you can

- compute the auto-covariance function
- generate the simulated data

- compute the log-likelihood function
- fit a fractionally integrated time series model to the data
- obtain the fractionally differenced data

```
d = 0.2;
call farmacov(cov, d); print cov;
call farmasim(yt, d); print yt;
call farmalik(lnl, yt, d); print lnl;
call farmafit(d, ar, ma, sigma, yt); print d sigma;
call fdif(zt, yt, d); print zt;
```

FARMACOV Call

computes the auto-covariance function for an ARFIMA(p, d, q) process

CALL FARMACOV(*cov, d <, phi, theta, sigma, p, q, lag>***);**

The inputs to the FARMACOV subroutine are as follows:

d specifies a fractional differencing order. The value of d must be in the open interval $(-0.5, 0.5)$ excluding zero. This input is required.

phi specifies an m_p-dimensional vector containing the autoregressive coefficients, where m_p is the number of the elements in the subset of the AR order. The default is zero. All the roots of $\phi(B) = 0$ should be greater than one in absolute value, where $\phi(B)$ is the finite order matrix polynomial in the backshift operator B, such that $B^j y_t = y_{t-j}$.

theta specifies an m_q-dimensional vector containing the moving-average coefficients, where m_q is the number of the elements in the subset of the MA order. The default is zero.

p specifies the subset of the AR order. The quantity m_p is defined as the number of elements of *phi*.

If you do not specify *p*, the default subset is *p*= $\{1, 2, \ldots, m_p\}$.

For example, consider *phi*=0.5.

If you specify *p*=1 (the default), the FARMACOV subroutine computes the theoretical auto-covariance function of an ARFIMA$(1, d, 0)$ process as $y_t = 0.5\ y_{t-1} + \epsilon_t$.

If you specify *p*=2, the FARMACOV subroutine computes the auto-covariance function of an ARFIMA$(2, d, 0)$ process as $y_t = 0.5\ y_{t-2} + \epsilon_t$.

q specifies the subset of the MA order. The quantity m_q is defined as the number of elements of *theta*.

If you do not specify *q*, the default subset is *q*= $\{1, 2, \ldots, m_q\}$.

The usage of *q* is the same as that of *p*.

lag specifies the length of lags, which must be a positive number. The default is $lag = 12$.

The FARMACOV subroutine returns the following value:

cov is a $lag + 1$ vector containing the auto-covariance function of an ARFIMA(p, d, q) process.

To compute the auto-covariance of an ARFIMA($1, 0.3, 1$) process

$$(1 - 0.5B)(1 - B)^{0.3} y_t = (1 + 0.1B)\epsilon_t$$

where $\epsilon_t \sim NID(0, 1.2)$, you can specify

```
d    = 0.3;
phi  = 0.5;
theta= -0.1;
sigma= 1.2;
call farmacov(cov, d, phi, theta, sigma) lag=5;
print cov;
```

For $d \in (0.5, 0.5)\backslash\{0\}$, the series y_t represented as $(1 - B)^d y_t = \epsilon_t$ is a stationary and invertible ARFIMA($0, d, 0$) process with the auto-covariance function

$$\gamma_k = \mathrm{E}(y_t y_{t-k}) = \frac{(-1)^k \Gamma(-2d + 1)}{\Gamma(k - d + 1)\Gamma(-k - d + 1)}$$

and the auto-correlation function

$$\rho_k = \frac{\gamma_k}{\gamma_0} = \frac{\Gamma(-d + 1)\Gamma(k + d)}{\Gamma(d)\Gamma(k - d + 1)} \sim \frac{\Gamma(-d + 1)}{\Gamma(d)} k^{2d-1}, \quad k \to \infty$$

Notice that ρ_k decays hyperbolically as the lag increases, rather than showing the exponential decay of the auto-correlation function of a stationary ARMA(p, q) process.

The FARMACOV subroutine computes the auto-covariance function of an ARFIMA(p, d, q) process.

For $d \in (0.5, 0.5)\backslash\{0\}$, the series y_t is a stationary and invertible ARFIMA(p, d, q) process represented as

$$\phi(B)(1 - B)^d y_t = \theta(B)\epsilon_t$$

where $\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p$ and $\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q$ and ϵ_t is a white noise process; all the roots of the characteristic AR and MA polynomial lie outside the unit circle.

Let $x_t = \theta(B)^{-1}\phi(B) y_t$, so that x_t follows an ARFIMA($0, d, 0$) process; let $z_t = (1 - B)^d y_t$, so that z_t follows an ARMA(p, q) process; let γ_k^x be the auto-covariance function of $\{x_t\}$ and γ_k^z be the auto-covariance function of $\{z_t\}$.

Then the auto-covariance function of $\{y_t\}$ is as follows:

$$\gamma_k = \sum_{j=-\infty}^{j=\infty} \gamma_j^z \gamma_{k-j}^x$$

The explicit form of the auto-covariance function of $\{y_t\}$ is given by Sowell (1992, p. 175).

FARMAFIT Call

estimate the parameters of an ARFIMA(p, d, q) model

CALL FARMAFIT(*d, phi, theta, sigma, series <, p, q, opt>***);**

The inputs to the FARMAFIT subroutine are as follows:

series specifies a time series (assuming mean zero).

p specifies the set or subset of the AR order. If you do not specify *p*, the default is *p=0*.

If you specify *p*=3, the FARMAFIT subroutine estimates the coefficient of the lagged variable y_{t-3}.

If you specify *p*=$\{1, 2, 3\}$, the FARMAFIT subroutine estimates the coefficients of lagged variables y_{t-1}, y_{t-2}, and y_{t-3}.

q specifies the subset of the MA order. If you do not specify *q*, the default is *q=0*.

If you specify *q*=2, the FARMAFIT subroutine estimates the coefficient of the lagged variable ϵ_{t-2}.

If you specify *q*=$\{1, 2\}$, the FARMAFIT subroutine estimates the coefficients of lagged variables ϵ_{t-1} and ϵ_{t-2}.

opt specifies the method of computing the log-likelihood function.

- *opt=0* requests the conditional sum of squares function. This is the default.
- *opt=1* requests the exact log-likelihood function. This option requires that the time series be stationary and invertible.

The FARMAFIT subroutine returns the following values:

d is a scalar containing a fractional differencing order.

phi is a vector containing the autoregressive coefficients.

theta is a vector containing the moving-average coefficients.

sigma is a scalar containing a variance of the innovation series.

To estimate parameters of an ARFIMA$(1, 0.3, 1)$ model

$$(1 - 0.5B)(1 - B)^{0.3} y_t = (1 + 0.1B)\epsilon_t$$

where $\epsilon_t \sim NID(0, 1)$, you can specify

```
d    = 0.3;
phi  = 0.5;
theta= -0.1;
call farmasim(yt, d, phi, theta);
call farmafit(d, ar, ma, sigma, yt) p=1 q=1;
print d ar ma sigma;
```

The FARMAFIT subroutine estimates parameters d, $\phi(B)$, $\theta(B)$, and σ_ϵ^2 of an ARFIMA(p, d, q) model. The log-likelihood function needs to be solved by iterative numerical procedures such as the quasi-Newton optimization. The starting value d is obtained by the approach of Geweke and Poter-Hudak (1983); the starting value of the AR and MA parameters are obtained from the least squares estimates.

FARMALIK Call

computes the log-likelihood function of an ARFIMA(p, d, q) model

CALL FARMALIK(*lnl, series, d <, phi, theta, sigma, p, q, opt>***);**

The inputs to the FARMALIK subroutine are as follows:

series specifies a time series (assuming mean zero).

d specifies a fractional differencing order. This argument is required; the value of d should be in the open interval $(-1, 1)$ excluding zero.

phi specifies an m_p-dimensional vector containing the autoregressive coefficients, where m_p is the number of the elements in the subset of the AR order. The default is zero.

theta specifies an m_q-dimensional vector containing the moving-average coefficients, where m_q is the number of the elements in the subset of the MA order. The default is zero.

sigma specifies a variance of the innovation series. The default is one.

p specifies the subset of the AR order. See the FARMACOV subroutine for additional details.

q specifies the subset of the MA order. See the FARMACOV subroutine for additional details.

opt specifies the method of computing the log-likelihood function.

- *opt=0* requests the conditional sum of squares function. This is the default.

opt=1 requests the exact log-likelihood function. This option requires that the time series be stationary and invertible.

The FARMALIK subroutine returns the following value:

lnl is 3-dimensional vector. *lnl*[1] contains the log-likelihood function of the model; *lnl*[2] contains the sum of the log determinant of the innovation variance; and *lnl*[3] contains the weighted sum of squares of residuals. The log-likelihood function is computed as $-0.5\times$ (*lnl*[2]+*lnl*[3]). If the *opt=0* is specified, only the weighted sum of squares of residuals returns in *lnl*[1].

To compute the log-likelihood function of an ARFIMA$(1, 0.3, 1)$ model

$$(1 - 0.5B)(1 - B)^{0.3}y_t = (1 + 0.1B)\epsilon_t$$

where $\epsilon_t \sim NID(0, 1.2)$, you can specify

```
d    = 0.3;
phi  = 0.5;
theta= -0.1;
sigma= 1.2;
call farmasim(yt, d, phi, theta, sigma);
call farmalik(lnl, yt, d, phi, theta, sigma);
print lnl;
```

The FARMALIK subroutine computes a log-likelihood function of the ARFIMA(p, d, q) model. The exact log-likelihood function is worked by Sowell (1992); the conditional sum of squares function is worked by Chung (1996).

The exact log-likelihood function only considers a stationary and invertible ARFIMA(p, d, q) process with $d \in (-0.5, 0.5)\backslash\{0\}$ represented as

$$\phi(B)(1 - B)^d y_t = \theta(B)\epsilon_t$$

where $\epsilon_t \sim NID(0, \sigma^2)$.

Let $Y_T = [y_1, y_2, \ldots, y_T]'$ and the log-likelihood function is as follows without a constant term:

$$\ell = -\frac{1}{2}(\log|\Sigma| + Y_T'\Sigma^{-1}Y_T)$$

where $\Sigma = [\gamma_{i-j}]$ for $i, j = 1, 2, \ldots, T$.

The conditional sum of squares function does not require the normality assumption. The initial observations y_0, $y_{-1}, \ldots$ and ϵ_0, $\epsilon_{-1}, \ldots$ are set to zero.

Let y_t be an ARFIMA(p, d, q) process represented as

$$\phi(B)(1 - B)^d y_t = \theta(B)\epsilon_t$$

then the conditional sum of squares function is

$$\ell = -\frac{T}{2}\log\left(\frac{1}{T}\sum_{t=1}^{T}\epsilon_t^2\right)$$

FARMASIM Call

generates an ARFIMA(p, d, q) process

CALL FARMASIM(*series, d <, phi, theta, mu, sigma, n, p, q, initial, seed>***);**

The inputs to the FARMASIM subroutine are as follows:

d specifies a fractional differencing order. This argument is required; the value of d should be in the open interval $(-1, 1)$ excluding zero.

phi specifies an m_p-dimensional vector containing the autoregressive coefficients, where m_p is the number of the elements in the subset of the AR order. The default is zero.

theta specifies an m_q-dimensional vector containing the moving-average coefficients, where m_q is the number of the elements in the subset of the MA order. The default is zero.

mu specifies a mean value. The default is zero.

sigma specifies a variance of the innovation series. The default is one.

n specifies the length of the series. The value of n should be greater than or equal to the AR order. The default is $n = 100$ is used.

p specifies the subset of the AR order. See the FARMACOV subroutine for additional details.

q specifies the subset of the MA order. See the FARMACOV subroutine for additional details.

initial specifies the initial values of random variables. The initial value is used for the nonstationary process. If $initial = a_0$, then $y_{-p+1}, \ldots, y_0$ take the same value a_0. If the *initial* option is not specified, the initial values are set to zero.

seed specifies the random number seed. If it is not supplied, the system clock is used to generate the seed. If it is negative, then the absolute value is used as the starting seed; otherwise, subsequent calls ignore the value of *seed* and use the last seed generated internally.

The FARMASIM subroutine returns the following value:

series is an n vector containing the generated ARFIMA(p, d, q) process.

To generate an ARFIMA$(1, 0.3, 1)$ process

$$(1 - 0.5B)(1 - B)^{0.3}(y_t - 10) = (1 + 0.1B)\epsilon_t$$

where $\epsilon_t \sim NID(0, 1.2)$, you can specify

```
d    = 0.3;
phi  = 0.5;
theta= -0.1;
mu   = 10;
sigma= 1.2;
call farmasim(yt, d, phi, theta, mu, sigma, 100);
print yt;
```

The FARMASIM subroutine generates a time series of length n from an ARFIMA(p, d, q) model. If the process is stationary and invertible, the initial values $y_{-p+1}, \ldots, y_0$ are produced using covariance matrices obtained from FARMACOV. If the process is nonstationary, the time series is recursively generated using the user-defined initial value or the zero initial value.

To generate an ARFIMA(p, d, q) process with $d \in [0.5, 1)$, x_t is first generated for $d' \in (-0.5, 0)$, where $d' = 1 - d$ and then y_t is generated by $y_t = y_{t-1} + x_t$.

To generate an ARFIMA(p, d, q) process with $d \in (-1, -0.5]$, a two-step approximation based on a truncation of the expansion $(1 - B)^d$ is used; the first step is to generate an ARFIMA$(0, d, 0)$ process $x_t = (1 - B)^{-d}\epsilon_t$, with truncated moving-average weights; the second step is to generate $y_t = \phi(B)^{-1}\theta(B)x_t$.

FDIF Call

obtain a fractionally differenced process

CALL FDIF(*out, series, d***);**

The inputs to the FDIF subroutine are as follows:

series specifies a time series with n length.

d specifies a fractional differencing order. This argument is required; the value of d should be in the open interval $(-1, 1)$ excluding zero.

The FDIF subroutine returns the following value:

out is an n vector containing the fractionally differenced process.

Consider an ARFIMA$(1, 0.3, 1)$ process

$$(1 - 0.5B)(1 - B)^{0.3}y_t = (1 + 0.1B)\epsilon_t$$

Let $z_t = (1 - B)^{0.3}y_t$, that is, z_t follows an ARMA(1,1). To get the filtered series z_t, you can specify

```
d    = 0.3;
phi  = 0.5;
theta= -0.1;
call farmasim(yt, d, phi, theta) n=100;
call fdif(zt, yt, d);
print zt;
```

References

Chung, C.F. (1996), "A Generalized Fractionally Integrated ARMA Process," *Journal of Time Series Analysis*, 2, 111–140.

Geweke, J. and Porter-Hudak, S. (1983), "The Estimation and Application of Long Memory Time Series Models," *Journal of Time Series Analysis*, 4, 221–238.

Granger, C.W.J. and Joyeux, R. (1980), "An Introduction to Long Memory Time Series Models and Fractional Differencing," *Journal of Time Series Analysis*, 1, 15–39.

Hosking, J.R.M. (1981), "Fractional Differencing," *Biometrika*, 68, 165–176.

Li, W.K. and McLeod, A.I. (1986), "Fractional Time Series Modeling," *Biometrika*, 73, 217–221.

Sowell, F. (1992), "Maximum Likelihood Estimation of Stationary Univariate Fractionally Integrated Time Series Models," *Journal of Econometrics*, 53, 165–88.

Subject Index

Syntax Index

Your Turn

If you have comments or suggestions about *SAS/IML® Software: Changes and Enhancements, Release 8.2*, please send them to us on a photocopy of this page or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Institute Inc.
SAS Publishing
SAS Campus Drive
Cary, NC 27513
E-mail: yourturn@sas.com

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
E-mail: suggest@sas.com

*Welcome * Bienvenue * Willkommen * Yohkoso * Bienvenido*

SAS® Institute Publishing Is Easy to Reach

Visit our Web page located at www.sas.com/pubs

You will find product and service details, including

- **sample chapters**
- **tables of contents**
- **author biographies**
- **book reviews**

Learn about

- **regional user-group conferences**
- **trade-show sites and dates**
- **authoring opportunities**
- **custom textbooks**

Explore all the services that SAS Institute Publishing has to offer!

Your Listserv Subscription Automatically Brings the News to You

Do you want to be among the first to learn about the latest books and services available from SAS Institute Publishing? Subscribe to our listserv **newdocnews-l** and, once each month, you will automatically receive a description of the newest books and which environments or operating systems and SAS release(s) that each book addresses.

To subscribe,

1. Send an e-mail message to **listserv@vm.sas.com**.
2. Leave the "Subject" line blank.
3. Use the following text for your message:

   subscribe NEWDOCNEWS-L *your-first-name your-last-name*

   For example: subscribe NEWDOCNEWS-L John Doe

Create Customized Textbooks Quickly, Easily, and Affordably

SelecText® offers instructors at U.S. colleges and universities a way to create custom textbooks for courses that teach students how to use SAS software.

For more information, see our Web page at **www.sas.com/selectext**, or contact our SelecText coordinators by sending e-mail to **selectext@sas.com**.

You're Invited to Publish with SAS Institute's User Publishing Program

If you enjoy writing about SAS software and how to use it, the User Publishing Program at SAS Institute offers a variety of publishing options. We are actively recruiting authors to publish books, articles, and sample code. Do you find the idea of writing a book or an article by yourself a little intimidating? Consider writing with a co-author. Keep in mind that you will receive complete editorial and publishing support, access to our users, technical advice and assistance, and competitive royalties. Please contact us for an author packet. E-mail us at **sasbbu@sas.com** or call 919-677-8000, then press 1-6479. See the SAS Institute Publishing Web page at **www.sas.com/pubs** for complete information.

See *Observations*®, Our Online Technical Journal

Feature articles from *Observations*®: *The Technical Journal for SAS® Software Users* are now available online at **www.sas.com/obs**. Take a look at what your fellow SAS software users and SAS Institute experts have to tell you. You may decide that you, too, have information to share. If you are interested in writing for *Observations*, send e-mail to **sasbbu@sas.com** or call 919-677-8000, then press 1-6479.

Book Discount Offered at SAS Public Training Courses!

When you attend one of our SAS Public Training Courses at any of our regional Training Centers in the U.S., you will receive a 15% discount on book orders that you place during the course. Take advantage of this offer at the next course you attend!

SAS Institute
SAS Campus Drive
Cary, NC 27513-2414
Fax 919-677-4444

E-mail: sasbook@sas.com
Web page: www.sas.com/pubs
To order books, call Fulfillment Services at 800-727-3228*
For other SAS Institute business, call 919-677-8000*

*** Note:** Customers outside the U.S. should contact their local SAS office.