Jurik Research Limited Use Software License Agreement

CONCERNING THE SOFTWARE AND DOCUMENTATION, THIS LICENSE AGREEMENT IS THE ENTIRE AGREEMENT BETWEEN JURIK RESEARCH AND YOU. IN ORDER TO HAVE COMPLETED INSTALLATION OF THIS SOFTWARE, YOU GAVE CONSENT TO BE BOUND BY THE FOLLOWING TERMS AND CONDITIONS. KEEP THIS AGREEMENT WITH YOUR PERMANENT RECORDS.

PREFACE -- This manual (the "Documentation") refers to commercial software products (the "Software") provided by Jurik Research ("JR"). This Software will not operate unless you purchase or already own a fully paid license from JR. JR licenses its use under the terms set forth herein.

LICENSE GRANT -- If you are a fully paid license holder, Jurik Research, as licensor, grants to you, and you accept, a non-exclusive license to use the enclosed program Software and accompanying Documentation, only as authorized in this agreement. You acquire no right, title or interest in or to the Documentation or Software, whose copyrights are owned by Jurik Research (JR). All rights are reserved. You agree to respect and to not remove or conceal from view any copyright notices appearing on the Software or Documentation. You may not sublicense this license. You may not rent, lease, modify, translate, disassemble, decompile, or reverse engineer the software, nor create derivative works based on the software without written permission from JR. The software may be used only on a single computer at a single location at any one time. JR permits you to make one archival copy of the software. No other copies or any portions thereof may be made by you or by any persons under your control. No part of this manual may be transmitted or reproduced in any form, by any means, for any purpose other than the purchaser's personal use without written permission of JR.

TERM -- This license is effective until terminated. You may terminate it at any time. It will also terminate upon conditions set forth elsewhere in this Agreement if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the Software and Documentation together with all copies, modifications and merged portions of the software expressed in any media form, including archival copies.

LIMITED WARRANTY -- The information in the user guide and on the diskette is subject to change without notice and does not represent a commitment on the part of JR. JR warrants the diskette and physical document to be free of defects in materials and workmanship. The user's sole remedy, in the event a defect is found, is that JR will replace the defective diskette or documentation. JR warrants that the Software, if properly installed and operated on software for which it is designed, will perform substantially as described in the Documentation. JR also warrants the software to be operationally compatible with the software platforms and operating systems as specified on the JR website, whose software version numbers for each relevant software product are also specified at said website. You recognize and accept that there is the possibility that a software platform developer or operating system developer may significantly change their product so as be incompatible with Jurik tools. Although JR may create a revised version of its tools to re-establish compatibility, no warranty is expressed or implied to that effect. In the case said incompatibility does occur, JR is under no obligation to provide a refund, product exchange, revision or upgrade. The above express warranty is the only warranty made by JR. It is in lieu of any other warranties, whether expressed or implied, including, but not limited to, any implied warranty of merchantability of fitness for a particular purpose. This warranty commences on the date of delivery to the Licensee and expires sixty (60) days thereafter. Any misuse or unauthorized modification of the Software will void this limited warranty. No JR dealer, distributor, agent or employee is authorized to make any modification or addition to this warranty. This warranty gives you specific legal rights, and you may have other rights that vary from state to state. Product may not be returned for a refund after warranty has expired.

LIMITATION OF LIABILITY -- Because computer software is inherently complex and may not be completely free of errors, you are advised to test the software thoroughly before relying upon it. JR will not be responsible for your failure to do so. You assume full responsibility and risk for Software installation, application and results. Do not use the Software in any case where an error may cause significant damage or injury to persons, property or business. In no event shall JR be liable for any indirect, special, incidental, tort, economic, cover, consequential, exemplary damages or other damages, regardless of type, including, without limitation, damages or costs relating to the loss of profits, business, goodwill, data, or computer programs, arising out of the use of or inability to use JR products or services, even if the company or its agent has been advised of the possibility of such damages or of any claim by any other party. JR's total liability to you or any other party for any loss or damages resulting from any claims, demands or actions arising out of or related to this agreement shall not exceed the license fee paid to JR for use of this software. Some states or provinces do not allow the exclusion or limitation of implied warranties or limitation of liability for incidental or consequential damages, so the above exclusion or limitation may not apply to you.

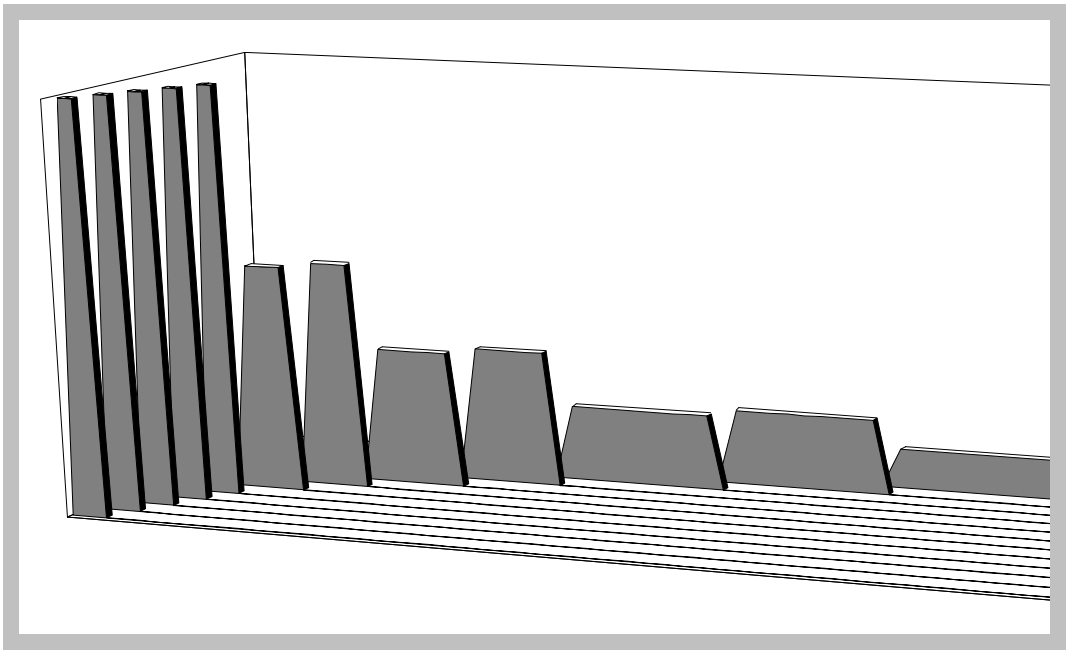
GOVERNING LAW and COST OF LITIGATION -- The license agreement shall be construed and governed in accordance with the laws of California. In any legal action regarding the subject matter hereof, venue shall be in the state of California, and the prevailing party shall be entitled to recover, in addition to any other relief granted, reasonable attorney fees and expenses of litigation. The export of JR products is governed by the U.S. Department of Commerce under the export administration regulations. It is your responsibility to comply with all such regulations.

NO WAIVER -- The failure of either party to enforce any rights granted hereunder shall not be deemed a waiver by that party as to subsequent enforcement of rights in the event of future breaches.

FEES -- A new password is required for installing Software into each additional computer. This license entitles you up to two passwords, one for each computer that you own. There is a fee for each additional password beyond the first two. Violation of this restriction is a direct copyright infringement to which Jurik Research shall seek legal remedy. Future upgrades may require a fee. Prices may change without notice.

WAV 2.0

Historical Sampling Filter DLL module for Windows® Application Developers



USER'S GUIDE

Requirements

- Windows 95, 98, 2000 or NT 4.
- Application software that can access DLL functions.

 PO 460669, Aurora, CO 80046 303-680-4409 fax: 303-680-4605

Installing the 32 bit DLL module

1. Execute the Installer, JRS_DLL.EXE. It will analyze your computer and give you a computer identification number. Write it down.
2. Get your access PASSWORD from Jurik Research Software. You can do so by calling 323-258-4860 (USA), faxing 323-258-0598 (USA), e-mailing support@nfsmith.net, or writing Jurik Research Software at 686 South Arroyo Parkway, Suite 237, Pasadena, California 91105. Be sure to give your full name, mailing address and computer identification number. You will then be given a password.
3. Rerun the installer JRS_DLL.EXE, this time entering the password when asked. Also enter **all the Jurik Research modules that you currently are licensed to run**. It will copy the latest version of these modules to any directory you specify.

You may now code your software to access the DLL as described below.

ABOUT NEW PASSWORDS

If you upgrade to a new computer, you will need a new password to run WAV. If you want to run WAV on additional computers, you will need additional passwords. Call 323-258-4860 for details.

Why Use WAV ?

Brief Description

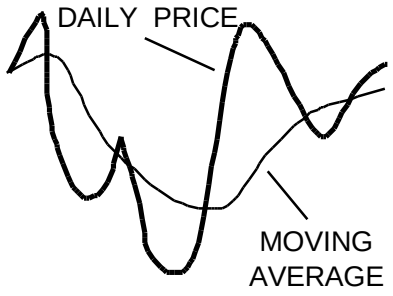
If you are building a model whereby each data fact-record needs historical values of a time series and you can arrange the time series in a spreadsheet, then the Historical Data Wavelet Sampler, **WaveSamp**, is for you. When the original time series is arranged as a single column on a spreadsheet, WaveSamp adds additional columns such that within any row the additional cells look farther into the past as you progress from left to right. The wavelet sampling algorithm filters and samples the time series to efficiently squeeze short, medium and long term information into a very small number of columns for your forecasting or financial trading system.

Background

Let's suppose the time series involves numbers from the stock market's ticker tape. If you buy stock at today's market price, the price will already reflect long term fundamental factors. Variations in price (the source of profit making) are to a large degree due to changes in its perceived value. However, perception is partly emotional and thus partly unpredictable, giving trade prices a chaotic appearance. Despite this, a certain amount of predictability does exist provided you have all the necessary information to make such a prediction.

Exactly what information is relevant is not obvious. Sometimes an economic measurement that is useless for making forecasts all by itself becomes useful when used in conjunction with other measurements. For example, the long term price ratio of gold-to-oil has been fairly predictable for hundreds of years. Thus, if you know the long term average price of oil you can estimate the long term average price of gold. Another example would be to quantify a trend as the change between today's closing price and that of 5, 20 or 60 days ago. Better yet, you could evaluate change between today's price and an *average* price centered 5, 20 or 60 days ago. No matter how you slice it, past behavior of a time series bears information that should not be ignored.

In order to predict future price movements, investors use mathematical equations to model parts of the financial world. They first select important aspects of the data, such as the price of corn. They may then opt to modify that data in order to make it more useful. For example, one simple modification to a series of corn prices is to produce its moving average. The moving average serves to filter out chaotic "noise" in the day-to-day prices, leaving a smooth trend line. See diagram at right. A simple short term buy/sell policy might be to buy immediately when the price rises above its moving average (and sell later on) or to sell immediately when the price falls below its moving average (and buy later on).

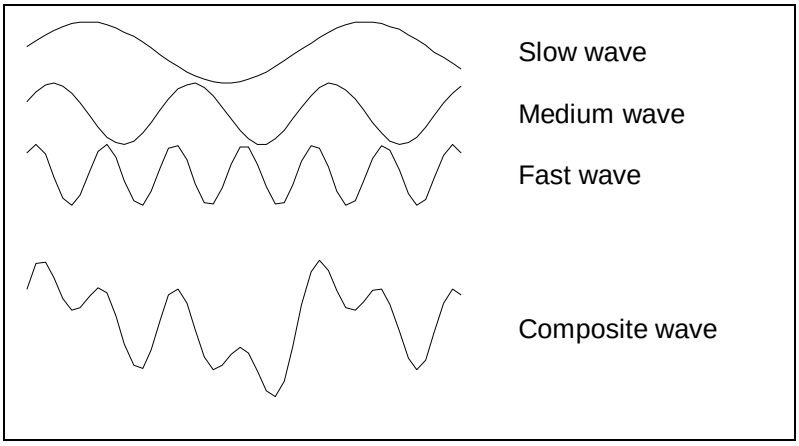


To proceed, we first need to cover some definitions. Let's define **raw data** as the numbers coming right off some live data feed. Any specific category, such as the price of gold, will be called a **raw feature**. Raw features could have their meaningful data hidden in noise or in some other way. The act of cleaning up raw features is called **preprocessing**. An example of preprocessing would be to attenuate noise in a raw feature by using a moving average.

Suppose you believe that a good forecasting model of the S&P 500 futures requires, as input, data spanning the last 300 days of S&P closing prices. You certainly do not want to create a model with 300 input variables! To do so will require an enormous amount of data for training the model. You would therefore like to reduce the number of columns from 300 to around 15. But taking the easy method of sampling every 20th day (300/15=20 days) is potentially useless to the model. Why? The method of selecting which daily readings of the S&P should be used for making a forecast is actually based on two thoughts, one is intuitive while the other is not.

THE KEY ISSUE

Market oscillations between being overbought and oversold are due, in part, to a kind of psychological momentum that tends to persist despite changes in market conditions. It is the prime reason why 90% of all traders lose money. Each market has its own time-varying momentum (TVM), and these TVM influence each other. Many TVMs, with different cycle lengths, may be driving the market you wish to forecast. Their cumulative effect contributes to the market price's complex waveform.



Many TVM's oscillate up and down at varying rates. For example, the seasons of the year cause annual cycles in food commodity prices; our country seems to need one good war every decade to keep the military industrial complex in good shape; the housing market has a mad rush about every 6 years; and so on. These cyclic factors come together and, depending on their individual effects, can create a combined composite driving force like that shown in the diagram.

Now let's go off on two tangential thoughts for a moment and return to this one later. Suppose you are in a town where the air quality changes very slowly. If you paid $1 to get the air quality report, would you need to spend another dollar the next hour? Not really. Since the raw parameter (air quality) is changing slowly, you only need to pay $1 once per day. You save money that way. However, if the air quality changed hourly you would have to ask every hour, and so it goes. The faster the changes, the more frequent the samples need to be taken. This is intuitive.

The other thought runs as follows. Suppose the daily reports on the S&P for the year 1993 produce a chart like the "composite wave" shown above. If this curve was entirely based upon a combination of cycles of different frequencies, then we could recreate any point on the chart by just knowing what those underlying cycles were. Or to put it another way, if we sample the S&P in just the right way so as to detect all the underlying cycles, then those samples are all we need to know "everything" about that curve. (Remember, this method will really not tell us everything since most financial curves are driven by cyclic and non-cyclic driving forces, some of which are random and thus unpredictable. However, capturing all the cyclic information present in a time series is a major accomplishment nonetheless.)

So for a trading system to work properly, it must have the right historical samples for detecting all possible TVMs affecting the market you wish to trade. Although slow cycle TVMs may be sampled slowly (once per month), fast cycles must be sampled quickly (once per day). So the big question is: what is the best spread of samples in a financial time series when you do not have a clue about which TVMs are driving the market?

BREAKTHROUGH: WAVELET SAMPLING

We now put all the above thoughts together. We need to sample a price time series in order to detect any underlying cycles and we know that cycles with faster frequencies need to be sampled more frequently. The simple and *inefficient* way to accomplish this is to sample the price every day for the last year. We would no doubt capture all the cycle information, but we would also require the model to accept over 200 input variables!

A more efficient way is to follow the concept of wavelet analysis, whereby only the smallest number of samples is taken for each cycle size. To see how this works, note that along the bottom of figure 1 is a hypothetical price time-series whose current value is designated by the "today" marker. The zigzag waveforms above it represent TVMs of various cycles that might be driving that market.

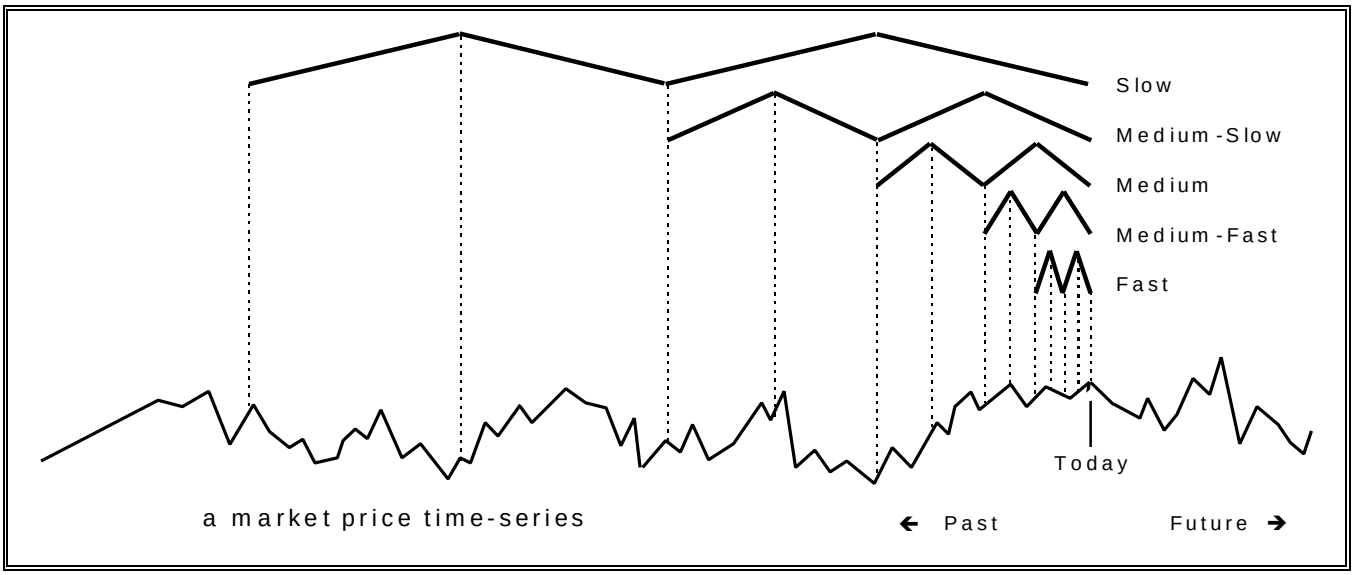


FIGURE 1

Proper sampling of the market line requires getting just enough samples for detecting presence of the slow waveform, and just enough for detecting the medium-slow waveform, and just enough for all the other waveforms too. The dotted lines show where these samplings would occur. Note that the dotted lines get increasingly farther apart the further into history you sample.

Fundamentals of WAV operation

To see precisely how WaveSamp works, consider figure 2 below. There are 5 horizontal rows of little squares, the rows labeled A through E. Let the last square in row A represent the S&P daily value on the last day of the year 1993. The second-to-last square in row A represents the S&P reading on the second-to-last day of the same year, and so on, each preceding square representing the preceding daily report in the same year. Thus row A is a time-series of prices, ending on the last day of 1993. Let rows B, C, D, E and F also represent the same time series as row A.

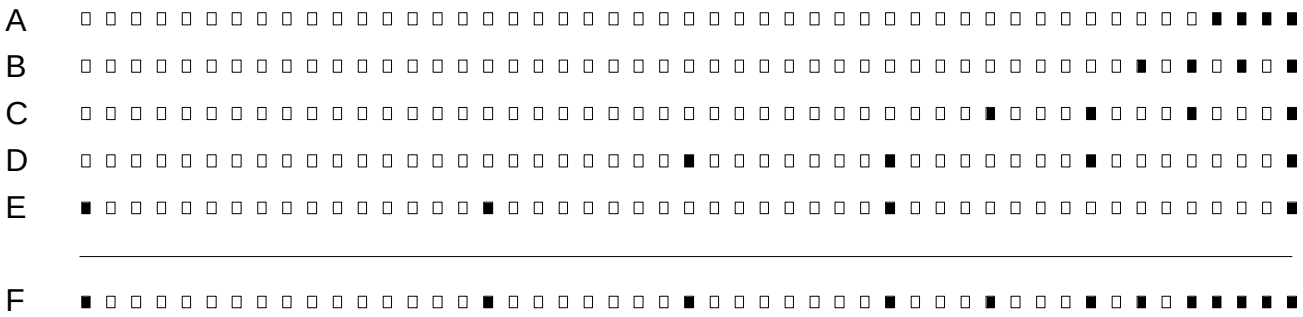


FIGURE 2

To capture any fast cycles that may exist in the S&P, every day's price must be sampled for the last four days, as shown by the black squares in row A. To capture a slower cycle, every other day is sampled as shown in row B. Slower cycles are sampled successively in rows C, D and E. Now, combine all the days that need to be sampled into one row, as shown in row F. This is a very efficient way to determine on which days the S&P price needs to be sampled in order to cover all possible cycle speeds with the least number of samples. The longer the series, the more efficient this sampling scheme becomes, since the sampling gets increasingly farther apart the further back into history one samples.

WaveSamp does more than merely sample historical prices. A single sample 32 days ago, for example, would not be very meaningful since such a sample would ignore data on all the days surrounding it. That's why WaveSamp filters the time-series before sampling it. The special filtering method forces every sample to contain information about a block of data points on both sides of the sample.

For example, in Figure 3 below, let P1 through P16 be the S&P price on day 1 through day 16. Suppose P16 is the current price. The historical sample WaveSamp creates is a combination of several prices. In the figure, we see prices P8 through P10 combined to form a composite sample whose center is located (16 minus 9) or 7 "days" before that of P16. If we define N to be the temporal distance between the "current" time slot and the center of the historical sample, then in this example, N = 7.

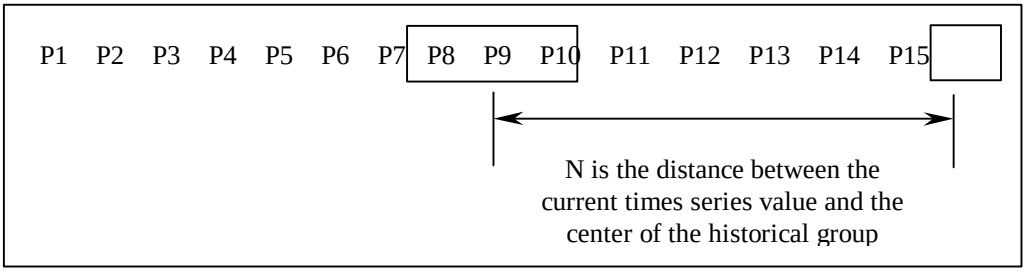


FIGURE 3

The INDEX TABLE

The N values used in WaveSamp are given in the table to the right. The column labeled “max. distance” indicates the farthest distance into the historical past that WaveSamp will sample for a specified value of N.

An example: if you decide that an accurate forecast of tomorrow’s S&P requires sampling up to the previous 135 days of activity of the S&P, then you would select the next highest number from the "max. distance" column which would be 139. The corresponding N value would be 123.

In your own project, you select the maximum distance you want WAV to consider by referencing the “Index” value in column 3 of the table. For example, to select max distance = 139, you would set WAV’s INDEX parameter to 15.

max. distance	N	index
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
8	7	6
11	10	7
16	14	8
21	19	9
30	26	10
39	35	11
56	48	12
73	65	13
106	90	14
139	123	15
204	172	16
269	237	17
398	334	18

TABLE 1

The OUTPUT ARRAY

The WAV DLL accepts the original time series as input. The DLL returns an array of data, whose number of rows is the same as the number of elements in the original time series. The number of columns in the output array depends on two factors: the user-selectable INDEX into Table #1 above, and the user-selectable MODE of operation.

There are three MODES to choose from: Standard, Detrend, Normalized. Their properties will be explained later. To grasp the effect INDEX has on the number of columns produced, let’s begin by assuming the user set MODE of operation to “Standard”. That is the simplest and most common setting.

For the rest of this discussion, please refer to Table 2, which displays a column of input data and an array of WAV output data. In addition, the rows and columns are enumerated for purposes of this discussion only. WAV output is restricted to cells inside the heavily outlined box.

When MODE is set to “Standard”, the total number of columns in the output array is the same as the user-selectable INDEX value. Table 2 is the result of the user setting MODE = “Standard” and INDEX = 7. Note that each column is assigned a value for N, starting with N=1 and increasing in value for each successive column. The values of N applied are the same as those in Table 1.

For each element in the input time series, WAV outputs a row of data into the output array. Consider the data going down output column 1. Data in this column is attained from appropriate historical (prior) cells in the input column. When N=1, the maximum distance into the past required by the WAV algorithm, as given in Table 1, is 1 row. Therefore, output is delayed by one row and starts on row 2. For column 2, we see N=2 which means maximum distance into the past is 2 rows, so data in this row is delayed 2 rows and therefore begins at row 3. This process continues for each column in the output array. The last column has N=10 and according to Table 1, the maximum distance into the past as

required by the WAV algorithm is 11 rows; therefore, output in this column is delayed 11 rows causing data to begin at row 12.

	Input		*Col 1*	*Col 2*	*Col 3*	*Col 4*	*Col 5*	*Col 6*	*Col 7*
	Time								
	Series		N=1	N=2	N=3	N=4	N=5	N=7	N=10
Row 1	18.11		0	0	0	0	0	0	0
Row 2	18.00		18.11	0	0	0	0	0	0
Row 3	18.12		18.10	18.11	0	0	0	0	0
Row 4	18.30		18.12	18.10	18.11	0	0	0	0
Row 5	18.43		18.30	18.12	18.00	18.11	0	0	0
Row 6	18.60		18.43	18.30	18.12	18.00	18.11	0	0
Row 7	18.55		18.60	18.43	18.30	18.12	18.00	0	0
Row 8	18.53		18.55	18.60	18.43	18.30	18.12	0	0
Row 9	18.64		18.53	18.55	18.60	18.43	18.30	18.08	0
Row 10	18.56		18.64	18.53	18.55	18.60	18.43	18.14	0
Row 11	18.56		18.56	18.64	18.53	18.55	18.60	18.28	0
Row 12	18.48		18.56	18.56	18.64	18.53	18.55	18.44	18.07
Row 13	18.55		18.48	18.56	18.56	18.64	18.53	18.53	18.14
Row 14	18.68		18.55	18.48	18.56	18.56	18.64	18.56	18.28
Row 15	18.78		18.68	18.55	18.48	18.56	18.56	18.57	18.44

TABLE 2

In table 2, we see in each successive column data starting later than the preceding column. Those rows that are not completely filled in with valid data are useless for purposes of data modeling. Therefore, just prior to returning the output array, WAV zeros out all those rows that were not completely filled in. The resulting array that WAV actually returns is shown in table 3. Although it may seem a lot of data was lost, remember that in a typical application the output array would contain thousands of rows. With this perspective, zeroing a small number of early rows poses no loss significant data loss.

Row 1		0	0	0	0	0	0	0
Row 2		0	0	0	0	0	0	0
Row 3		0	0	0	0	0	0	0
Row 4		0	0	0	0	0	0	0
Row 5		0	0	0	0	0	0	0
Row 6		0	0	0	0	0	0	0
Row 7		0	0	0	0	0	0	0
Row 8		0	0	0	0	0	0	0
Row 9		0	0	0	0	0	0	0
Row 10		0	0	0	0	0	0	0
Row 11		0	0	0	0	0	0	0
Row 12		18.56	18.56	18.64	18.53	18.55	18.44	18.07
Row 13		18.48	18.56	18.56	18.64	18.53	18.53	18.14
Row 14		18.55	18.48	18.56	18.56	18.64	18.56	18.28
Row 15		18.68	18.55	18.48	18.56	18.56	18.57	18.44

TABLE 3

ESTIMATING the INDEX value

When selecting N, we recommend the following heuristic: if you want to feed WAV data into a forecasting module, set WAV's maximum historical distance to be at least four times the forecast horizon. For example, if you want your forecast module to predict 8 rows into the future, then you need to feed that module historical information that spans 8x4 or 32 rows into the past. According to Table 1, getting the smallest historical distance that is at least as large as 32 requires setting INDEX to 11.

SELECTING the APPROPRIATE COLUMNS

One elegant property of wavelet sampling is its invariance to scale. That is, whether you want to forecast 4 days ahead or 40 days ahead, we recommend using only 8 of the columns produced by WaveSamp. The only difference is the actual columns selected. For example, Table 4 shows all the columns WaveSamp can produce.

As mentioned earlier, for a forecast model predicting 8 rows ahead (forecast horizon = 8), the recommended INDEX value is 11. This produces 11 columns, of which you really only need to feed the last 8 to your forecast model. The table below shows which columns of WAV's output array is most useful for forecasting purposes.

INDEX	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Maximum lookback	1	2	3	4	5	8	11	16	21	30	39	56	73	106	139	204	269	398
Forecast Horizon																		
1 - 4	☺	☺	☺	☺	☺	☺	☺	☺										
5		☺	☺	☺	☺	☺	☺	☺	☺									
6 - 7			☺	☺	☺	☺	☺	☺	☺	☺								
8 - 9				☺	☺	☺	☺	☺	☺	☺	☺							
10 - 14					☺	☺	☺	☺	☺	☺	☺	☺						
15 - 18						☺	☺	☺	☺	☺	☺	☺	☺					
19 - 26							☺	☺	☺	☺	☺	☺	☺	☺				
27 - 34								☺	☺	☺	☺	☺	☺	☺	☺			
35 - 51									☺	☺	☺	☺	☺	☺	☺	☺		
52 - 67										☺	☺	☺	☺	☺	☺	☺	☺	
69 - 100											☺	☺	☺	☺	☺	☺	☺	☺

The user is not constrained to employ only 8 columns. As many or as few as desired may be used (provided there is enough historical data to support the lookbacks).

The reason you get to specify INDEX parameter N rather than let WaveSamp simply give you all the columns it can produce is because the rightmost columns require a large amount of historical data. This delays the input by hundreds of rows, causing hundreds of rows in the output array to be zeroed out. Under some circumstances, that can amoun to a large percentage of the data. On the other hand, when you specify n=2, for example, only the first two rows are blank, thereby giving you more usable rows of data for further processing.

MODES OF OPERATION

WAV offers three different modes to process your data: Standard, Detrend, and Detrend+Normalize.

1. **Standard** - this mode is designed to sample signals that do not have any long term trend. These signals include oscillators, stochastics, and Nth order derivatives. The total number of columns produced by this mode equals INDEX.

2. **Detrend** - this mode produces INDEX+1 columns. The first of these additional columns is a detrended version of the input time series. The remaining INDEX columns are the result of applying WAV to this detrended time series.

   This mode is designed to sample signals that wander over long periods of time. These signals include prices, which tend to increase by way of market inflation. In this mode WaveSamp cancels out the long term (or medium term) trend, causing the resulting values to fluctuate around zero. This lets your models process many years of financial data, regardless of how the data may have drifted during that time period.

   Available INDEX values this mode are 10-18, although *we strongly recommend going no lower than 12*. Detrending a signal using a smaller INDEX will produce correct results, but it may detrend out important medium-short term trending action in the input time series.

3. **Detrend & Normalize** - this mode produces INDEX+1 columns. The first of these additional columns is a detrended and normalized version of the input time series. The remaining INDEX columns are the result of applying WAV to this modified time series.

   In this mode the detrended input signal's amplitude is continuously rescaled so as to attain uniform strength over the entire time series. For example, if during the past 5 years, the price of T-Bonds was especially volatile during a 2-year period, automatic scale normalization would scale down the price activity during that time to make it more in line with the rest of the 5-year period. This way, price activity patterns are still preserved and can be accepted by your model without fear of overdriving your model's inputs.

   Available INDEX values this mode are 12-18, although *we strongly recommend going no lower than 14*. Detrending a signal using a smaller INDEX will produce correct results, but they may appear visually unintuitive.

TOTAL COLUMNS PRODUCED

The total number of columns produced by each MODE of operation is as follows:

MODE	Number of Columns
Standard	INDEX
Detrend	INDEX + 1
Detrend/Normalize	INDEX + 1

C Programming the 32 bit WAV DLL

The file **JRS_32.DLL** contains the functions **WAV** and **WAVcols**. In your C code, you should declare both functions as externally defined and, if using MS VC++, use the _declspec(dllimport) keywords. The function is exported as a C function, so if you are using C++, you should insert "C" (with the quotes) between the words "extern" and "_declspec". Also, you should link with JRS_32.LIB, which we provide.

WAV function

```
extern _declspec(dllimport) int WINAPI WAV( double *pdData, double *pdOut, DWORD dwLength,
DWORD dwINDEX, int iMode ) ;
```

PARAMETERS

pdData, a pointer to double specifying the memory location of the first cell in the input array to be processed.

pdOut, a pointer to double specifying the memory location of the first cell in the output array. The memory space must be allocated by the caller, and it must be sized so that it can contain exactly as many double precision numbers as will occupy an array equal to the product of the number of rows in the input array times the number of columns returned by the function **WAVcols**. Do not pass the name of a two dimensional array (a pointer to a pointer to double); the function expects the address of the first element in the array, so that the function can be called by a program written in C or VB.

dwLength is a DWORD indicating the number of rows of data in the input array.

dwINDEX is a DWORD specifying the INDEX value as described above in this user manual.

iMode is an INT (32 bit signed integer) specifying the desired type of output mode as described in the manual. The valid values for iMode are:

iMode = 1	WAV applied to actual input
iMode = 2	WAV applied to detrended input
iMode = 3	WAV applied to detrended and normalized input

NOTES

WAV can encounter problems. It will return error values indicating the nature of the error.
The error codes are defined on the next page.

WAVCols function

```
extern _declspec(dllimport) int WINAPI WAVCols( DWORD dwINDEX, int iMode ) ;
```

PARAMETERS

dwINDEX is a DWORD specifying the INDEX value as described in the user manual.

iMode is an INT (32 bit signed integer) specifying the desired type of output mode as described in the manual. The valid values for iMode are:

iMode = 1	WAV applied to actual input
iMode = 2	WAV applied to detrended input
iMode = 3	WAV applied to detrended and normalized input

NOTES

Function returns the number of columns required for WAV's output array.

WAVCols can encounter problems. It will return error values indicating the nature of the error.
The error codes are defined on the next page.

Detailed Instructions

INPUT ARRAY

First you will need to allocate memory for your original data array, which should then be filled with double precision numbers. The array should be a vector, a single column of numbers. The following line of code is an example of allocating memory for your input array when you have 1200 rows of data in a column.

```
int length = 1200 ;
DataPtr = (double *) GlobalAllocPtr( GHND, sizeof(double) * length ) ;
```

OUTPUT ARRAY

Your output array will contain multiple columns of double precision numbers. It should have the same number of rows as your input data vector, and also the number of columns that WAV2 will require. You can determine this required number of columns by calling the following function prior to allocating memory for WAV's output:

```
columns = WavCols ( dwINDEX, iMode ) ;
```

The size of the output array is then the product: (columns * length) * sizeof(double).

The following code is an example of allocating memory for your output array when you have 1200 rows of data, and you want Detrend Mode and INDEX = 12.

```
iMode = 2 ;
length = 1200 ;
dwINDEX = 12 ;
OutCols = WAVCols ( dwINDEX , iMode ) ;
OutPtr = (double *) GlobalAllocPtr ( GHND, length*OutCols * sizeof(double) ) ;
```

HEADER FILE

Both functions described above should have their prototypes listed in your application's header file, and all should have the extern keyword in front of them.

LIB FILE

We provide a LIB file which should be added to your C project. By default it's placed in C:\JRS_DLL\LIB.

ERROR CODES

0	no error conditions met
-1	problem with password/installation
10001	pointer to data is NULL
10002	pointer to output memory is NULL
10003	Mode parameter not between 1 and 3 inclusive
10004	Index parameter outside min and max values
10005	not enough data rows for selected value of Index
10006	Index must be at least 10 if Mode is 2
10007	Index must be at least 12 if Mode is 3

VISUAL BASIC PROGRAMMING EXAMPLE

INTRODUCTION

In your Jurik Research DLL installation directory (eg., `C:\JRS_DLL`) the workbook WAV_DLL.XLS contains a working example of how to use Excel's VBA to operate WAV automatically. The workbook includes the following:

- Worksheet "DLL VBA Results" -- where you can apply the Visual Basic macro that calls the DLL
- Worksheet "Excel add-in Results" – containing the results of running the WAV for Excel add-in product
- Visual Basic Module – containing the VB macro code

In this example, you will run the VBA macro called "**WAV_Test**" on the worksheet titled "**DLL VBA Results**". The macro gets the data in column 1 and sends it to the WAV DLL function. The output array produced by WAV is then written back onto the spreadsheet, column by column.

VBA MACRO DESCRIPTION

The VBA subroutine **WAV_Test** is shown below. This code will read data from column 1 of the active worksheet, call the WAV DLL, and output its results back to the worksheet, column by column. Note that function **WAV** is the main DLL function that analyzes the data and produces the output array. Function **WAVCols** determines the total number of columns in WAV's output array. This is needed to properly size the array destined to received the results from WAV; though you could provide your own code to make this determination based on the information provided earlier in the manual.

```
Declare Function WAV Lib "JRS_32.dll" ( _
                            ByRef daData As Double, _
                            ByRef pdOut As Double, _
                            ByVal dwLength As Long, _
                            ByVal dwINDEX As Long, _
                            ByVal iMode As Long) As Long

Declare Function WAVCols Lib "JRS_32.dll" ( _
                            ByVal dwINDEX As Long, _
                            ByVal iMode As Long) As Long


Sub WAV_Test()

    Dim k As Long
    Dim i As Long
    Dim iResult As Long
    Dim dwLength As Long
    Dim dwNMIndex As Long
    Dim TotCols As Long
    Dim OutCells As Long
    Dim RowOffset As Long
    Dim iMode As Long
    Dim calctype As Long

    ' iMode determines output type
    '       1   standard
    '       2   detrended
    '       3   detrended/normalized
```

```
Dim InputData(1 To 199) As Double
Dim OutputData() As Double

'disable automatic calculation
calctype = Application.Calculation
Application.Calculation = xlManual

dwLength = 199        ' input has 199 elements
dwNMIndex = 12          ' set INDEX = 12

' Read Data from spreadsheet into array
' Input data is in column 1

For k = 1 To dwLength
    InputData(k) = Cells(k + 1, 1)
Next k

' *** Create the standard mode output ***

iMode = 1

' redimension output array to be large enough to contain
' Call WAV using pointers to first elements of arrays

TotCols = WAVCols(dwNMIndex, iMode)
OutCells = TotCols * dwLength
ReDim OutputData(1 To OutCells) As Double
iResult = WAV(InputData(1), OutputData(1), dwLength, dwNMIndex, iMode)

'----- error codes ------
'     0       no error conditions met
'    -1       problem with password/installation
' 10001       pointer to data NULL
' 10002       pointer to output memory NULL
' 10003       Mode parameter not between 1 and 3 inclusive
' 10004       N and M Index parameter outside min and max values
' 10005       not enough data rows for N and M Index
' 10006       N and M Index must be at least 10 if Mode is 2
' 10007       N and M Index must be  at least 12 if Mode is 3

If iResult <> 0 Then
    ' Post Error Message and HALT
    Call Error_handler(iResult, calctype)

Else

    ' Show results in columns 5+ on spreadsheet

    For i = 1 To TotCols
        RowOffset = 0
        For k = 1 To dwLength
            Cells(1 + k, 4 + i).FormulaR1C1 = OutputData(RowOffset + i)
            RowOffset = RowOffset + TotCols
        Next k
    Next i
End If

'enable automatic calculation

Application.Calculation = calctype

End Sub
```

```
' The following subroutine is a simple way to handle run-time errors that may occur
' It is good practice to handle each error type mentioned in the user manual.

Private Sub Error_handler(ByVal error_code As Long, ByVal calctype As Long)
    Dim result As Long
    result = MsgBox("Error number " & Str(error_code) & " was returned by WAV.", , "WAV
                    Error")
    Application.Calculation = calctype
    End   ' this END command will halt execution of the VBA code.
End Sub
```

IF YOU FIND A BUG . . . YOU WIN

If you discover a legitimate bug in any of our preprocessing tools, please let us know! We will try to verify it on the spot. If you are the first to report it to us, you will receive the following two coupons redeemable toward your acquisition of any of our preprocessing tools:

- a $50 discount coupon
- a free upgrade coupon

You may collect as many coupons as you can.
You may apply more than one discount coupon toward the purchase of your next tool.

$$$ Anti-Piracy Reward Policy $$$

Jurik tools are world renown for excellence and value. We manage to keep costs down with large sales volume, maintained in part by protecting our copyrights with the following anti-piracy policy…

1. We have on permanent retainer one of the best intellectual property law firms in the U.S.
2. We do not perform cost-benefit analysis when it comes to litigation. We prosecute all offenders.
3. We register portions of our software with the U.S. Copyright office, entitling us to demand the offender compensate Jurik Research for all legal costs, which is typically over $10,000 per lawsuit.
4. We offer up to $5000 reward for information leading to the prosecution of any offender(s).

Risk & Liability

Hypothetical or simulated performance results have certain inherent limitations. Simulated performance is subject to the fact that they are designed with the benefit of hindsight.

We must also state here that, due to the frequently unpredictable nature of the marketplace, past performance of any trading system is never a guarantee of future performance. In addition, all trading systems have risk and commodities trading leverages that risk. We advise you never to place at risk more than you can afford to lose. It's only common sense.

The user is advised to test the software thoroughly before relying upon it. The user agrees to assume the entire risk of using the software. In no event shall JRC be responsible for any special, consequential, actual or other damages, regardless of type, and any lost profit resulting from the use of this software.