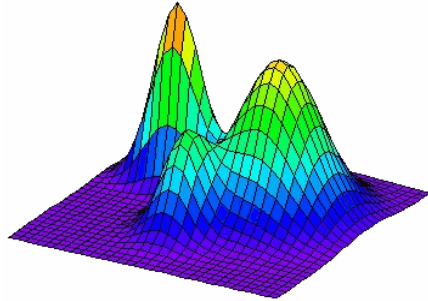




## MathLibX .NET Controls for Numerical Analysis



<b>1.0</b>	<b>INTRODUCTION</b> .....	<b>2</b>
<b>2.0</b>	<b>LANGUAGE SPECIFICS</b> .....	<b>2</b>
2.1	TYPE DEFINITIONS.....	2
<b>3.0</b>	<b>LEASTSQX</b> .....	<b>3</b>
3.1	SIMPLE POLYNOMIAL FITTING (POLYFIT) .....	3
3.2	LEAST SQUARES OF USER SUPPLIED FUNCTIONS (FITDATA).....	5
<b>4.0</b>	<b>FFTX</b> .....	<b>7</b>
4.1	REAL FFT.....	8
4.2	COMPLEX FFT .....	10
4.3	DATA WINDOWING.....	11
<b>5.0</b>	<b>SIGPROCX</b> .....	<b>12</b>
5.1	IIR (INFINITE IMPULSE RESPONSE) FILTER .....	12
5.2	FIR (FINITE IMPULSE RESPONSE) FILTER .....	14
5.3	EXPONENTIAL RUNNING AVERAGE FILTER .....	17
5.4	THE ALPHA BETA FILTER.....	19
5.5	INTERPOLATION/EXTRAPOLATION FUNCTIONS.....	21
<b>6.0</b>	<b>KALMANFTX</b> .....	<b>24</b>
<b>7.0</b>	<b>SORTX</b> .....	<b>29</b>
<b>8.0</b>	<b>ENCRYPTX</b> .....	<b>31</b>
<b>9.0</b>	<b>ROOTX</b> .....	<b>32</b>
<b>11.0</b>	<b>DISTRIBUTIONS</b> .....	<b>34</b>
<b>12.0</b>	<b>GETTING HELP</b> .....	<b>34</b>



## 1.0 Introduction

Thank you for your purchase of the Newcastle Scientific .NET Math Library, which includes controls for least squares fitting, fast Fourier transforms, digital signal processing, Kalman filtering, finding root/minimum/maximum, and numeric sorting.

**These controls were written as .NET Framework Components specifically for Visual Studio .NET applications, including Visual Basic .NET, and C# .NET.**

The goal of the MathLibX .NET library is to simplify the interface to a number of numerical functions; significantly minimizing the time a programmer will typically require developing code for complex numerical analyses. In most cases, the included functions have a one-minute learning curve.

Included with the controls are comprehensive source code examples in Visual Basic and C#, demonstrating many of the functions and properties. The source code will be found in the directory **c:\Program Files\MathLibX\MathLibX\_Demo\_VB (for VB)**, and **c:\Program Files\MathLibX\MathLibX\_Demo\_VS (for C#)**, unless you specified a different location during installation.

Please direct questions and comments to [info@mathfunctions.com](mailto:info@mathfunctions.com).

## 2.0 Language Specifics

Newcastle Scientific's MathlibX .NET controls are fully compatible with Visual Basic .NET and Visual C# NET. The controls may or may not be compatible with other languages/platforms.

### 2.1 Type Definitions

Unless otherwise stated, all integer types utilized in the math libraries are of type “long”. This is 64-bit (8-byte) integer.

Unless otherwise stated, all floating types utilized in the math libraries are of type “double”. This is 64-bit (8-byte) floating-point values



## 3.0 LeastSqX

LeastSqX provides powerful linear and non-linear least squares algorithms with an easy to use interface. Write your own function for fitting and pass the function's name to the control, or simply rely on the default general polynomial function.

### 3.1 Simple Polynomial Fitting (PolyFit)

Probably the most commonly used function in least squares is the simple polynomial ( $a + bx + cx^2 + dx^3...$ ), which is available through the **PolyFit** function. The call to **Polyfit** consists of passing the number of points, degrees of freedom to fit, array of independent variables (e.g., time), array of dependent variables (e.g., measurements to fit), The fitted coefficients are returned in `co`, and the standard deviation of the fit is returned in `stdev`.

```
Public Function polyfit(ByVal numpoints As Long, ByVal numco As Long,  
ByRef xd() As Double, ByRef yd() As Double, ByRef co() As Double, ByRef  
stdev As Double) As Long
```

Parameter Name	Type	Description
<code>numpoints</code>	long	Number of data points to fit
<code>numco</code>	Long	Number of coefficients to fit
<code>xd()</code>	Double()	Array of independent data (X values)
<code>yd()</code>	Double()	Array of dependent data (Y values)
<code>co()</code>	Double()	Array of coefficients
<code>stdev</code>	Double	Standard deviation (residuals) of fit

Blue parameters are inputs, red parameters are returned values. The function returns a -1 if the call fails, e.g., due to improper array size declarations.



See the following code VB and C# code fragments as examples.

```
'PolyFit example in Visual Basic .NET  
Sub demo_of_Polyfit()  
    Static xd(100), yd(100), co(4) As Double  
    Dim ma As Long = 3, npt As Long = 20, l as Long, stdev As Double  
  
    ' set up example data for fitting  
    For i = 0 To npt - 1  
        xd(i) = i  
        yd(i) = 3.0 + 2.0 * xd(i) - 1.0 * xd(i) ^ 2  
    Next i  
    i = LeastSqX1.polyfit(npt, ma, xd, yd, co, stdev)  
end sub
```

```
// PolyFit example in Visual C# .NET  
void demo_of_Polyfit()  
{  
    double[] xd = new double[100], yd = new double[100], co = new double[4];  
    long ma = 3, npt = 20, i;  
    double stdev=0.0;  
  
    // set up example data for fitting  
    for (i=0; i < npt; i++){  
        xd[i] = i;  
        yd[i] = 3.0 + 2.0 * xd[i] - 1.0 * xd[i]*xd[i];  
    }  
    i=leastSqX1.polyfit( npt, ma, ref xd, ref yd , ref co , ref stdev);  
}
```



### 3.2 Least Squares of User Supplied Functions (FitData)

It is often necessary to fit data to a function other than a simple polynomial. For instance fitting measured data to some sort of sine wave, or a Gaussian shape. The FitData function allows the user to easily create their own function for fitting.

```
Public Function FitData(ByVal numpoints As Long, ByVal numco As Long,
ByRef xd() As Double, ByRef yd() As Double, ByRef sigd() As Double,
ByVal userfunc As userfundel, ByRef co() As Double, ByRef P(,) As
Double, ByRef ChiSq As Double, ByRef stdev As Double) As Long
```

Parameter Name	Type	Description
<b>numpoints</b>	long	Number of data points to fit
<b>numco</b>	Long	Number of coefficients to fit
<b>xd()</b>	Double()	Array of independent data (X values)
<b>yd()</b>	Double()	Array of dependent data (Y values)
<b>sigd()</b>	Double()	Array of one sigma error estimates of Y values
<b>userfunc</b>	Userfundel	Supplied name of user function
<b>co()</b>	Double()	Array of coefficients
<b>P(,)</b>	Double(,)	2D Array of covariance
<b>ChiSq</b>	Double	Chisquare of fit
<b>stdev</b>	Double	Standard deviation (residuals) of fit

Blue parameters are inputs, red parameters are returned values. The function returns a -1 if the call fails e.g., due to improper array size declarations.

The call to **FitData** consists of passing the number of points, degrees of freedom to fit, array of independent variables (e.g., time), array of dependent variables (e.g., measurements to fit), array of uncertainties of these measurements (may be all set to 0 for non-weighted fitting), user function, and initial guess of the coefficients (may be all 0 for linear least squares). The function returns the coefficients are replaced by the fitted values, and the fitted covariance, chisquare, and standard deviation of the fit.

The control automatically determines whether the function you are fitting to is linear or non-linear, and thus automatically determines the optimal fitting method. The user does not have to feed the control partial derivatives of the fitting function, as is often the case for non-linear problems. While the input parameters include initial estimates of the final fitted coefficients, these may be all set to zero for linear functions.

Note, referring to a function as linear (or non-linear) has nothing to do with whether the function has powers greater than one. For instance “a + bx + cx<sup>2</sup> + dx<sup>3</sup>” is a linear function because each coefficient (a, b, c, d) is linear to the function. One way to tell if a function is linear is by taking the partial derivative of the function with respect to each coefficient to verify it is not dependent upon the other coefficients. For instance taking the partial derivative of the function “a + bx + cx<sup>2</sup> + dx<sup>3</sup>” with respect to “c” gives you simply x<sup>2</sup>, which is not dependent upon a, b, or d. So the function is linear. An example of a non-linear function is a\*exp(-(x-b)<sup>2</sup> / c). Taking the partial derivative with respect to a, b, or c gives you a function that is dependent on the other two coefficients. For non-linear functions it is necessary to include estimates of the coefficients. The required closeness of these estimates is dependent upon the specific function, so some level of iteration may be required.



See the following code VB and C# code fragments as examples.

```
' FitData example in Visual Basic .NET
Function Userfunc_Gauss (ByVal x As Double, ByVal co() As Double, ByVal numco As Long) As Double
    Return (co(0) + co(1) * Math.Exp(-(co(2) - x) ^ 2))
End Function

Sub demo_of_FitData()
    Static xd(100), yd(100), sigd(100), co(3), std(3) As Double
    Static P(3, 3) As Double
    Dim ma As Long = 3, npt = 20, i, chisq As Double, stdev As Double

    For i = 0 To npt - 1          ' set up example data for fitting
        xd(i) = i
        yd(i) = 3.0 + 25.0 * Math.Exp(-(11.3 - xd(i)) ^ 2)
        sigd(i) = 0.0;
    Next i

    ' initial guess of coeffs
    co(0) = 2.0
    co(1) = 5.2
    co(2) = 10.5
    i = LeastSqX1.FitData(npt, ma, xd, yd, sigd, AddressOf Userfunc_Gauss, co, P, chisq, stdev)
end sub
```

```
// FitData example in Visual C# .NET
Double Userfunc_Gauss(Double x, Double [] co, long numco) {
    Return(co[0]+co[1]*Math.Exp(-Math.Pow(co[2] - x),2));
}

void demo_of_FitData()
{
    // declare and assign a delegate for your user function
    MathLibX.LeastSqX.userfundel userfundeluse;
    userfundeluse = new MathLibX.LeastSqX.userfundel(Userfunc_Gauss);

    double[] xd = new double[100], yd = new double[100], sigd = new double[100];
    double[] co = new double[4], std = new double[4]; double[,] P = new double[3,3];
    long ma = 3, npt = 4, i;
    double chisq = 0.0, stdev = 0.0;

    for (i=0; i < npt; i++){          // set up example data for fitting
        xd[i] = i;
        yd[i] = 3.0 + 25.0 * Math.Exp(-Math.Pow(11.3 - xd[i],2));
        sigd[i] = 0.0;
    }

    co[0] = 2.0; co[1] = 5.2; co[2] = 10.5;    // initial guess of coeffs

    i=leastSqX1.FitData( npt, ma, ref xd, ref yd , ref sigd, userfundeluse , ref co, ref P,
        ref chisq, ref stdev);
}
```



## 4.0 FFTX

The FFT control allows one to quickly calculate the Fast Fourier Transform and the Inverse Fast Fourier Transform of a set of data. The data can be a real or complex data set. The control also includes the option of Welch windowing the data prior to FFT.

To FFT data, simply make a call to the FFT method, passing a vector of data, plus the number of data. Upon return, the data vector has been replaced by its FFT. To take the inverse, make a similar call to the IFFT method. To use Welch windowing prior to FFT, just set the Welch window property to true.

For complex data sets, make a call to the complex FFT method (CFFT) passing vectors of the real and imaginary components of the complex numbers. Use the complex IFFT (ICFFT) to take the complex inverse.

Most users actually find it easier to always use the complex FFT and simply set the imaginary data set to zero when requiring an FFT of real data set.

Please examine the included real and complex FFT/IFFT examples carefully to understand the correct data scaling.



## 4.1 Real FFT

To perform an FFT and IFFT on a set of real numbers just send a vector of the data to the FFT and IFFT function.

```
Public Function FFT(ByRef Data() As Double, ByVal number_of_data As Long) As Long
```

```
Public Function IFFT(ByRef Data() As Double, ByVal number_of_data As Long) As Long
```

Parameter Name	Type	Description
<b>dataY()</b>	Double()	This is the vector of data to be FFTed. Note that the data must be put in every other position ( <b>see example below</b> ), and that dataY must be dimensioned twice as large as the dataset .
<b>number_of_data</b>	Long	This is the number of data in dataY to be FFTed. This must be a 2 <sup>N</sup> number where N is an long (e.g., 256, 512, 1024, etc.)

Blue parameters are inputs, black parameters are both inputs and outputs. The function returns a -1 if the call fails.

Note on the preparation of dataY, where the input data must be given put in every other position:

```
for i = 0 to number_of_data-1
    dataY(i*2) = data(i)
next I
```

```
i = FFTX1.FFT(dataY, number_of_data)
```

To perform an IFFT send a vector of data to the control as follows.

```
i = FFTX1.IFFT(dataY, number_of_data)
```

The following example code fragment demonstrates how to take the FFT of a dataset, how to properly scale the resulting frequency domain data, and then how to take the IFFT to return back to the original time domain dataset. Please examine this example closely, especially with regard to the proper scaling.



**' FFT/IFFT example in Visual Basic .NET**

Sub Perform\_FFT()

**' dimension must be twice as large as the dataset**

Static dataX(2048) as double, dataY(2048) as double

Static dataXp(2048) as double, dataYp(2048) as double **' for plotting**

Dim i as long, number\_of\_data as long

**' some example data**

number\_of\_data=1024

For i = 0 To number\_of\_data - 1

    dataX(i)= i

    dataY(2\*i) = 300.0 \* math.Sin(2# \* Math.PI \* dataX(i) / 64) **' note every other position**

    dataY(2\*i+1) = 0.0

Next i

**' take FFT of data.**

i = FFTX1.FFT(dataY, number\_of\_data)

**' scale for plotting**

For i = 0 To number\_of\_data - 1

    dataXp(i) = dataX(i) \* 2.0 \* Math.PI / (number\_of\_data) **' plot x axis as frequency**

    dataYp(i) = Math.Sqrt(dataY(i \* 2) ^ 2 + dataY(i \* 2 + 1) ^ 2) \* 2 / number\_of\_data

Next i

plot(dataXp, dataYp) **' plot power vs. frequency**

**' take IFFT which will equal original dataY vector**

i = FFTX1.IFFT(dataY, number\_of\_data)

End Sub



## 4.2 Complex FFT

To FFT a complex set of numbers, put the real components into one vector and the complex components into a second vector. Then make a call to the complex FFT method called "CFFT". Note that this approach also works fine with a straight real vector, simply set the imaginary vector all to zero. In fact with a real data set, CFFT is probably easier to use than FFT, and the results are identical.

```
Public Function CFFT(ByRef Datareal() As Double, ByRef Dataimaginary()  
As Double, ByVal number_of_data As Long) As Long
```

```
Public Function ICFFT(ByRef Datareal() As Double, ByRef Dataimaginary()  
As Double, ByVal number_of_data As Long) As Long
```

Parameter Name	Type	Description
<b>datareal</b>	Double()	This is the vector of the real value components of the data to be FFTed.
<b>dataimaginary</b>	Double()	This is the vector of the imaginary value components of the data to be FFTed.
<b>number_of_data</b>	Long	This is the number of data in dataY to be FFTed. This must be a $2^N$ number where N is an long (e.g., 256, 512, 1024, etc.)

Blue parameters are inputs, black parameters are both inputs and outputs. The function returns a -1 if the call fails.

To take a complex FFT, make a call to the CFFT function, as follows.

```
i = FFTX1.CFFT(datareal, dataimaginary, number_of_data)
```

To take the inverse FFT of a set of complex numbers put the real components into one vector and the complex components into a second vector then make a call to the complex inverse FFT method called "ICFFT" as follows.

```
i = FFTX1.ICFFT(datareal, dataimaginary, number_of_data)
```

The following example code fragment demonstrates how to take the CFFT of a dataset, how to properly scale the resulting frequency domain data, and then how to take the ICFFT to return back to the original time domain dataset. Please examine this example closely, especially with regard to the proper scaling.



```
' Complex FFT/IFFT example in Visual Basic .NET
Sub Perform_ComplexFFT()

    Static dataX(1024) as double , datareal(1024) as double, dataimaginary(1024) as double
    Static dataarealp (1024) as double, dataimaginaryp(1024) as double
    Dim i as long, number_of_data as long

    ' some example data
    number_of_data=1024
    For i = 0 To number_of_data - 1
        dataX(i)=CDBL(i)
        datareal(i) = 300.0 * Sin(2# * PI * dataX(i) * 0.012)
        dataimaginary(i) = 0.0
    Next i

    ' take complex FFT of data.
    i = FFTX1.CFFT(datareal, dataimaginary, number_of_data)

    ' scale for plotting.
    For i = 0 To number_of_data
        dataXp(i) = dataX(i) * 2.0*Pi/ (number_of_data) ' plot x axis as frequency
        dataarealp (i) = datareal(i) * 2 / number_of_data
        dataimaginaryp (i) = dataimaginary(i) * 2 / number_of_data
    Next i

    plot(dataX, dataarealp)
    plot(dataX, dataimaginaryp)

    ' take IFFT which will equal original datareal dataimaginary vectors
    i = FFTX1.ICFFT(datareal, dataimaginary, number_of_data)

End Sub

    plot(dataX, dataarealp)
    plot(dataX, dataimaginaryp)

    ' take IFFT which will equal original datareal dataimaginary vectors
    i = FFTX1.ICFFT(datareal, dataimaginary, number_of_data)

End Sub
```

### **4.3 Data Windowing**

To window the data with a Welch style window just set the Window\_Welch property value to true before performing the FFT or CFFT.

```
FFTX1.Window_Welch = True
```



## 5.0 SigProcX

The SigProcX control offers a variety of time domain filtering functions, including an IIR (Infinite Impulse Response) filter, FIR (Finite Impulse Response) filter, Exponential Running Average filter, and an Alpha Beta filter. The control also includes a Cubic Spline and 2<sup>nd</sup> Order Polynomial interpolation routine. All of these functions are very easy to use, and should benefit anyone analyzing, e.g., time series data for trends, whether the data is of historical or real-time nature.

### 5.1 IIR (Infinite Impulse Response) Filter

An IIR filter is a digital filter that relies on both the current unfiltered data point as well as the previous filtered data for updating. In essence, an IIR has an infinite (but decaying) memory of past data. The form of an IIR is usually  $y = a*(x) + b*(x-)... + c*(y-)...$

The input parameters to the IIR\_Filter include the data sampling rate, the desired cutoff frequency (must be less than ½ the sampling rate), the data value to be filtered, and an initialization value which needs to be set to true for the first call.

The returned lowfilter is the filtered data with frequencies below the cutoff, and the highfilter is the filtered data with frequencies above the cutoff.

Important note, the actual results will show some “leakage” through the cutoff filter, which is expected with any digital time domain filter. If it is critical to minimize this leakage in an application, please consider utilizing a frequency domain filter built using the FFT control described in section 4.0 (and shown in the demo Visual Basic source code).



```
Public Function IIR_Filter(ByVal init As Boolean, ByVal x As Double,  
ByVal frequency_cutoff As Double, ByVal frequency_sample As Double,  
ByRef lowpass As Double, ByRef highpass As Double) As Long
```

Parameter Name	Type	Description
<b>Init</b>	boolean	Set to true for first call, then set to false. Tells the control to initialize the filter.
<b>dataIn</b>	double	Input unfiltered data value
<b>frequency_cutoff</b>	double	Input cutoff frequency scaled to frequency_sample. In otherwords, the true cutoff frequency in Hz is frequency_cutoff/frequency_sample
<b>frequency_sample</b>	double	Data sampling rate in Hz, e.g., if your data is taken at ½ second intervals then set frequency_sample to 2.
<b>lowfilter</b>	double	This is the filtered output data point with the frequencies above the frequency_cutoff removed (except for the leakage component).
<b>highfilter</b>	double	This is the filtered output data point with the frequencies below the frequency_cutoff removed (except for the leakage component).

Blue parameters are inputs, red parameters are returned values. The function returns a -1 if the call fails.

Below is a simple code fragment demonstrating a IIR filter.

**'IIR filter example in Visual Basic .NET**

```
Private Sub Perform_Filter()  
Dim lowfilter As Double, highfilter As Double, Dim i as long, k as long  
Dim frequency_cutoff As Double, frequency_sample As Double  
Static dataY(2000) as double  
Dim number_of_data=2000 as long, l as long, k as long  
  
for i=0 to number_of_data-1 ' example data  
    dataY(i)=100.0*sin(.1*i) + 30*rand;  
next i  
  
' Sampling Rate and Cutoff frequency  
frequency_sample = 1.0 ' Data points per second  
frequency_cutoff = 0.2 * frequency_sample 'Cycles per second (Hz)  
  
For i = 0 To number_of_data - 1  
    If i = 0 Then  
        k = SigProcX1.IIR_Filter(True, dataY(i), frequency_cutoff, frequency_sample, lowfilter, highfilter)  
    else  
        k = SigProcX1.IIR_Filter(False, dataY(i), frequency_cutoff, frequency_sample, lowfilter, highfilter)  
    End If  
    Print #1, dataY(i), lowfilter, highfilter  
Next i  
  
End sub
```



## 5.2 FIR (Finite Impulse Response) Filter

The FIR filter relies on only N numbers of data points, and thus does not have the infinite memory that the IIR filter has. The implementation of this FIR allows the user to set exact filter weights to each wavelength in order to create a custom low pass, mid pass, high pass, band pass, or notch filter.

The function internally, at initialization, takes an IFFT of the filter weights and convolves these weights in the time domain with the data to create the FIR filter.

The advantage of this FIR filter is that it has a much sharper response (less leakage) than the IIR filter shown in section 5.1. The disadvantage of this FIR filter is that there is a latency of the output that is 0.5 numweights long. For example, if numweights is set to 32, then the output filtered data corresponds to the input data point 16 points ago. If one made a plot of the unfiltered and filtered data, the filtered data will all be shifted to the left by 16 points.

The parameter “numweights” sets the kernel size of the filter. This must be set to a power of 2, e.g., 2, 4, 8, 16, 32, 64.... The larger the value is set the sharper the response (less leakage), however the slower the function will run. It is recommended that the user does not set this value above 64.

The frequency of each of the numweights filter weights are as follows:

**Frequency (i) = 0.5 \* i/numweights, where i = 0...numweights-1**

These are the primary frequencies of the filter, which means that these frequencies can be filtered out nearly completely (if desired) by setting the weights of the desired primary frequencies to zero. Note that frequencies in the actual raw data that are not a primary frequency will not be completely filtered out. Try increasing the number of weights (numweights) and widening the notch filter such that several primary frequencies are set to zero (band gap filter), until the desired performance is reached. The input parameters of the FIR filter are as follows.

```
Public Function FIR_Filter(ByVal init As Boolean, ByRef x As Double,  

ByVal num_filter_weights As Long, ByRef filter_weights() As Double,  

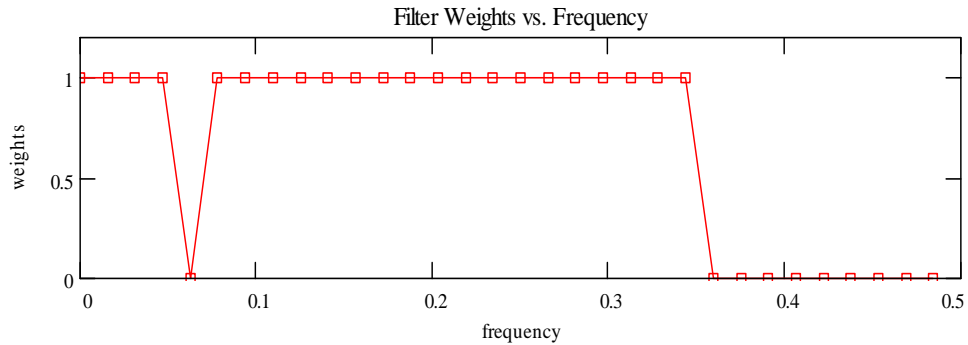
ByRef filter_data As Double)
```

Parameter Name	Type	Description
<b>Init</b>	Boolean	Set to true for first call, then set to false. Tells the control to initialize the filter.
<b>dataIn</b>	double	Input unfiltered data value
<b>numweights</b>	long	Number of filter components (must be a power of 2, e.g., 2, 4, 8, 16, 32...)
<b>filterweights()</b>	Double ( )	This “numweights” long vector contains the filter weights.
<b>Filter_data</b>	double	This is the Filter_data output.

Blue parameters are inputs, red parameters are returned values. The function returns a -1 if the call fails.

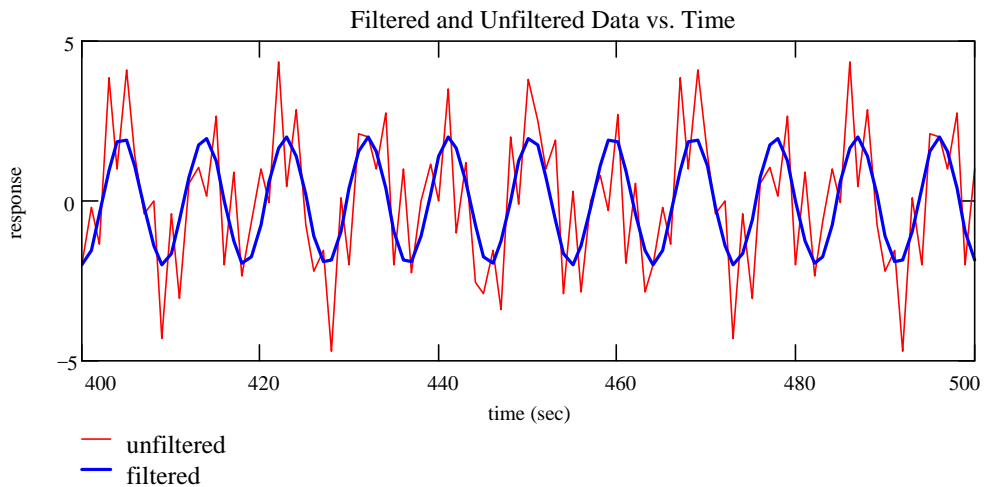


The following plot shows filter weights designed as a combined notch and low pass filter.



The next plot shows the raw unfiltered data (below equation) along with the filtered data. The filtered data nearly removes the first and third sinusoidal components of this equation (first from the notch filter and third from the low pass filter), leaving only  $\sin(2 \cdot \pi \cdot 0.109 \cdot i) \cdot 2$ .

$$d_i := \sin(2 \cdot \pi \cdot 0.063 \cdot i) + \sin(2 \cdot \pi \cdot 0.109 \cdot i) \cdot 2 + \cos(2 \cdot \pi \cdot 0.422 \cdot i) \cdot 2$$





Below is a simple code fragment demonstrating a FIR filter designed to notch at .2 Hz and noise above .3 Hz

```
' FIR filter example in Visual Basic .NET
Private Sub Perform_Filter()
Dim lowfilter As Double
Dim i as long, k as long, numweights as long
Dim frequency_cutoff As Double

Static filterweights(256) As Double
Static dataY(2000) as double
Dim number_of_data=2000 as long, numweights = 32 as long

' example data
for i=0 to number_of_data-1
  dataY(i)=100.0*sin(.1*2.0*pi*i)+ 50.0*sin(.2*2.0*pi*i)+rnd;
next i

' Set low pass (filter above .3 hz)
For i = 0 To numweights
  If i < CInt(0.3 * numweights) then
    filterweights(i) = 1#
  else
    filterweights(i) = 0#
  end if
Next i

' Set notch at .2 Hz
l = cint(0.2 * numweights)
filterweights(i) = 0#

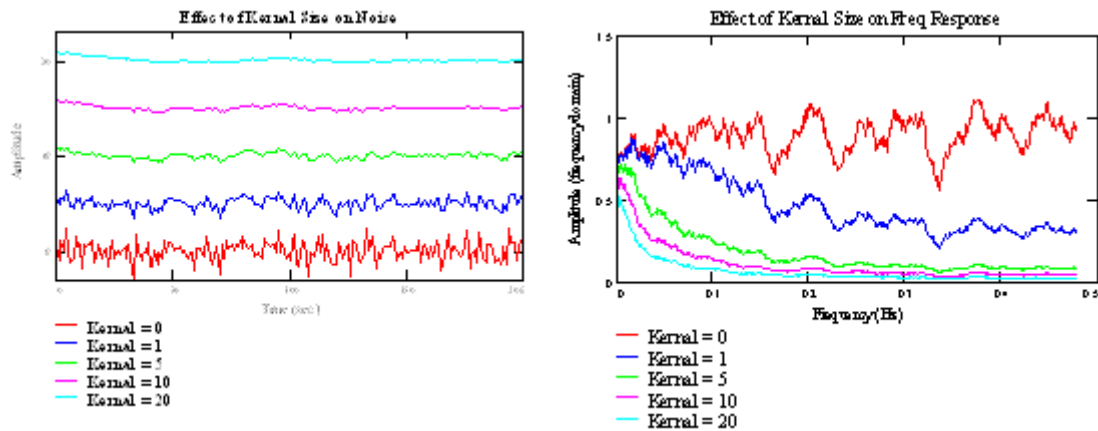
For i = 0 To number_of_data - 1
  If i = 0 Then
    k = SigProcX1.FIR_Filter(True, dataY(i), numweights, filterweights, lowfilter)
  Else
    k = SigProcX1.FIR_Filter(False, dataY(i), numweights, filterweights, lowfilter)
  End If
  Print #1, dataY(i), lowfilter
Next i

End sub
```



### 5.3 Exponential Running Average Filter

Probably the simplest filter to run is the basic low pass exponential running average filter (which actually is a simple type of IIR filter). Simply set kernel size property to the desired filter level, set the Init Filter property to true and start filtering. The two plots below demonstrate typical filter responses to the kernel settings (lowpass output), in the time (left) and the frequency (right) domains. The curves of the left plot have been vertically offset for clarity.



Since the filtering level (the kernel setting) is very dependent upon the noise level in the data as well as the function form of the underlying data, it is recommended that several trial and error iterations be made.

The relationship between the kernel size and the desired cutoff frequency is

**Kernel = 1 / (2.0 \* Pi \* Fc/Fs), where Fc = frequency cutoff, and Fs = data sampling frequency.**

The function Data creates an IIR lowpass filter and an IIR highpass filter, and returns the lowpass filtered output, highpass filter output, and midpass filter output (midpass is the delta between the high and low pass).

**Public Function Data**(ByVal init As Boolean, ByVal x As Double, ByRef lowpass As Double, ByRef midpass As Double, ByRef highpass As Double) As Long

Parameter Name	Type	Description
<b>Init</b>	Long	Set to true for first call, then set to false. Tells the control to initialize the filter.
<b>x</b>	Long	Data to filter
<b>lowpass</b>	Double	Lowpass output
<b>midpass</b>	Double	Midpass output
<b>highpass</b>	Double	Highpass output

Blue parameters are inputs, red parameters are returned values. The function returns a -1 if the call fails.



Below is a simple code fragment demonstrating a simple lowpass, midpass, and highpass filter.

```
' lowpass, midpass, and highpass filter example in Visual Basic .NET
Private Sub Perform_Filter()

Dim lowfilter As Double, midfilter As Double, highfilter As Double
dim i as long, k as long, number_of_data as long, number_of_data=2000 as long
Static dataY(2000) as double

' example data
for i=0 to number_of_data-1
    dataY(i)=100.0*sin(.1*i) + 30*rnd;
next i

' init filter (kernal =1 / (2.0 * π * Fc) , derive for desired Fc for low and high pass
' Set the control properties accordingly.
SigProcX1.High_pass_kernal = 1
SigProcX1.Low_pass_kernal = 2
SigProcX1.Init_Filter = True

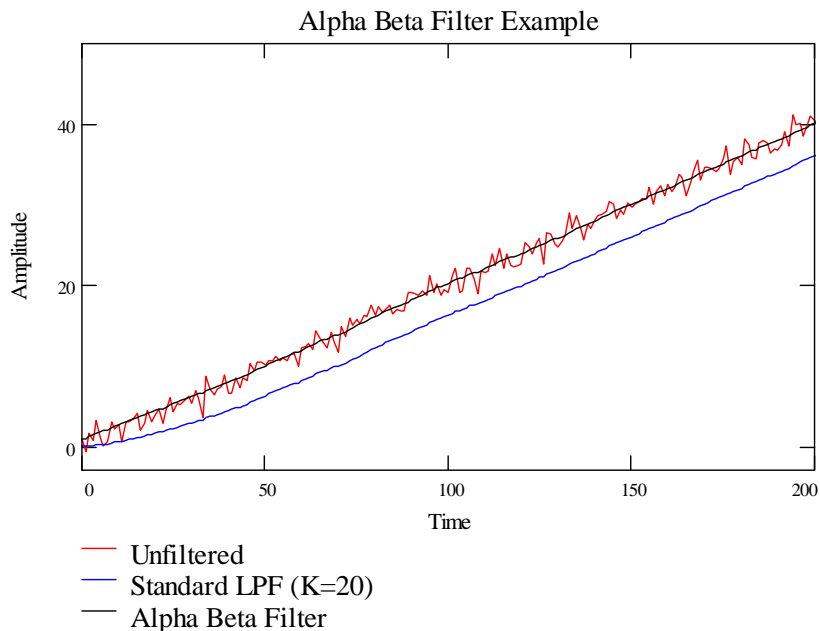
' filter data
For i = 0 To number_of_data - 1
    k = SigProcX1.Data(dataY(i), lowfilter, midfilter, highfilter)
    print #1, dataY(i), lowfilter, midfilter, highfilter    ' filtered data
next i

End Sub
```



## 5.4 The Alpha Beta Filter

An additional filter included in the SigProcX control is the Alpha Beta filter. This filter is designed for low pass filtering while also minimizing the “data lag” effect. If your dataset has a trend, e.g., upwards, the traditional low pass filter will tend to produce a filtered value below the actual value set. The 50-day running average of the stock market is a prime example of this. The Alpha Beta filter automatically performs a 1<sup>st</sup> order correction of the trend, and thus will have little or no data lag. For this reason, Alpha Beta filters are often used in applications such as navigation and targeting. The following plot shows an unfiltered data set with an upward trend (red), the same data filtered with a standard low pass filter (kernel = 20) demonstrating the data lag (blue), and the data set filtered with the Alpha Beta filter (output of Alpha) in black. The Alpha gain is set to  $2.0 * \pi * F_c / F_s$  ( $F_c$  = frequency cutoff, and  $F_s$  = data sampling frequency). This is the exact inverse of the kernel size for the exponential running average filter discussed in the previous section. The Beta Gain level is typically set to  $\sim 1/10^{\text{th}}$  to  $1/20^{\text{th}}$  the Alpha gain level.





```
Public Function alpha_beta(ByRef init As Boolean, ByVal x As Double,  
ByRef alpha As Double, ByRef beta As Double) As Long
```

Parameter Name	Type	Description
<b>Init</b>	Long	Set to true for first call, then set to false. Tells the control to initialize the filter.
<b>x</b>	Long	Data to filter
<b>alpha</b>	Double	alpha output
<b>beta</b>	Double	Beta output

Blue parameters are inputs, red parameters are returned values. The function returns a -1 if the call fails.

Simple Example of an Alpha Beta type filter.

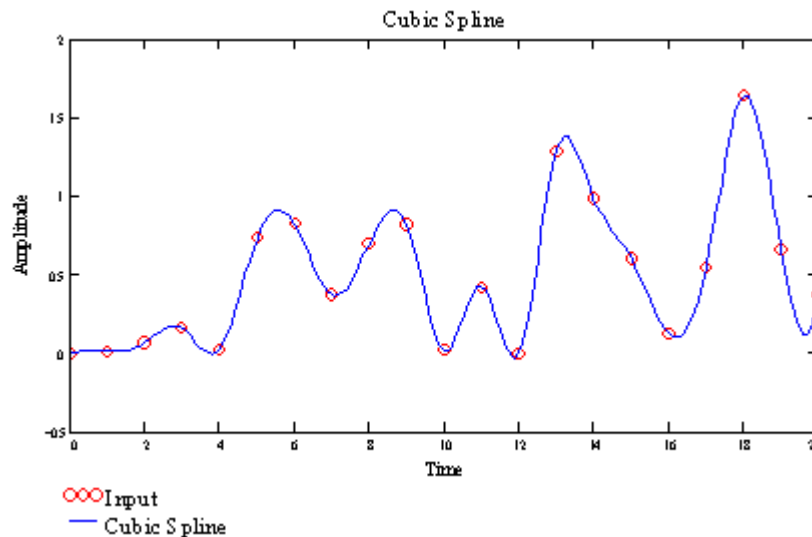
```
' Alpha Beta filter example in Visual Basic .NET  
Private Sub Perform_Alpha_Beta()  
  
Dim alpha As Double, beta As Double  
dim i as long, k as long, number_of_data = 2000 as long  
Static dataY(2000) as double  
  
' example data  
for i=0 to number_of_data-1  
    dataY(i)=100.0*sin(.1*i) + 30*rnd;  
next i  
  
' init filter  
SigProcX1.Alpha_Gain = .1  
SigProcX1.Beta_Gain = .01    ' Recommended Beta gain = ~ 1/10th Alpha gain  
SigProcX1.Init_Filter = True  
'filter data  
For i = 0 To number_of_data - 1  
    k = SigProcX1.alpha_beta(dataY(i), alpha, beta)  
    print #1, dataY(i), alpha , beta    ' filtered data  
next I  
  
End Sub
```



## 5.5 Interpolation/Extrapolation Functions

The SigProcX control includes two interpolation functions, the “Spline\_interpolate” for cubic spline interpolation, and the “interpolate” function for 2<sup>nd</sup> order polynomial interpolation. While either function can also be utilized for extrapolation, the 2<sup>nd</sup> order polynomial is probably more reliable for extrapolation of simple trends. Cubic Spline is usually the preferred method for most interpolation applications.

In either of these interpolation functions, pass an array of independent data and an array of dependent data, the desired X, and the function returns the corresponding Y. The Spline\_interpolate function also requires a Boolean for initialization. Set this to true whenever a new input dataset is used, and then set to false for subsequent calls with different desired X values (see example below).



```
Public Function Spline_interpolate(ByVal init As Boolean, ByVal numpoints As Long, ByRef x() As Double, ByRef y() As Double, _
ByVal xwant As Double) As Double
```

Parameter Name	Type	Description
<b>init</b>	Long	Set to true for first call, then set to false. Tells the control to initialize the filter.
<b>numpoints</b>	Long	Number of data points to “spline”
<b>x()</b>	Double ()	Independent (x) data
<b>y()</b>	Double ()	Dependent (y) data
<b>xwant</b>	Long	Desired location or “x” to derive “y”
<b>Spline_interpolate</b>	Double	Function returns results.

Blue parameters are inputs, red parameters are returned values.



### Example of a Cubic Spline function call

```
' Spline Interpolation filter example in Visual Basic .NET
Private Sub Perform_Spline()

    Static xd(10) As Double, yd(10) As Double
    Dim num As Long
    Dim i As Long

    ' example data
    num = 10
    For i = 0 To 9
        xd(i) = i
        yd(i) = 30# - 0.2 * i + 0.3 * i ^ 2
    Next i

    ' interpolate the data at 5.4 (note initialization is set to true)
    MsgBox SigProcX1.Spline_interpolate (True, num, xd, yd, 5.4)

    ' extrapolate the data at 9.7 (note initialization is set to false)
    MsgBox SigProcX1.Spline_interpolate (False, num, xd, yd, 9.7)

End Sub
```



A second function “interpolate” performs a second order polynomial fit on the three data points closest the desired x location, and uses this fit to extrapolate or interpolate. In the case of interpolation, these three data points will bracket the desired location. For extrapolation, the three data points will be the first or last three of the x(), y() set.

```
Public Function interpolate(ByVal numpoints As Long, ByRef x() As Double, ByRef y() As Double, ByVal xwant As Double) As Double
```

Parameter Name	Type	Description
<b>numpoints</b>	Long	Number of data points in set
<b>x()</b>	Double ()	Independent (x) data
<b>y()</b>	Double ()	Dependent (y) data
<b>xwant</b>	Long	Desired location or “x” to derive “y”
<b>interpolate</b>	Double	Function returns results.

Blue parameters are inputs, red parameters are returned values.

Simple Example of a Interpolation/Extrapolation function call with 2<sup>nd</sup> order Polynomial

```
' Interpolation/Extrapolation example in Visual Basic .NET
```

```
Private Sub Perform_Interpolation()
```

```
    Static xd(10) As Double, yd(10) As Double  
    Dim num As Long  
    Dim i As Long
```

```
    ' example data
```

```
    num = 10  
    For i = 0 To 9  
        xd(i) = i  
        yd(i) = 30# - 0.2 * i + 0.3 * i ^ 2  
    Next i
```

```
    ' interpolate the data at 5.4
```

```
    MsgBox SigProcX1.Interpolate(num, xd, yd, 5.4)
```

```
    ' extrapolate the data at 14.3
```

```
    MsgBox SigProcX1.Interpolate(num, xd, yd, 14.3)
```

```
End Sub
```



## 6.0 KalmanFtX

The KalmanFtX control consists of a standard linear Kalman filter, allowing for propagation and updates of the state vector and covariance matrix. Inputs consists of the "Z" vector of measurements, "R" measurement noise matrix, "H" matrix of partials, "X" state vector, " " propagation matrix, "P" current covariance matrix, and "Q" process noise matrix. The output consists of the updated state vector and updated covariance.

Kalman filters are similar to least squares fitting except that measurements can be incorporated into the fitting coefficients one or a few at a time, rather than "batch", or all at once as with least squares. This aspect is especially useful for real-time applications such as those found in navigation, guidance, and control.

This .NET control is not designed to serve as a substitute for a proper understanding of Kalman filters. The control will certainly be useful to users with a basic or advanced working knowledge of Kalman filters, as well as a beginner user, provided they have additional literature such as the reference book recommended on page one of this manual.

```
Public Function Kalman(ByRef Z() As Double, ByRef r(,) As Double,
ByRef x() As Double, ByRef P(,) As Double, ByRef H(,) As Double,
ByRef Q(,) As Double, ByRef Phi(,) As Double, ByVal Num_states As
Long, ByVal Num_measurements As Long) As Long
```

i = KalmanFtX1.Kalman(Z, R, X, P, H, Q, Phi, Nx, Nz)

Please see the included Kalman Filter example source code in the example included demo program.

Parameter Name	Type	Description
Z()	Double ()	vector of measurements, of length Nz.
R(,)	Double (,)	matrix of measurement noise (sigma ^2) of size Nz, Nz
X()	Double ()	vector of states, of length Nx.
P(,)	Double (,)	covariance matrix, of size Nx, Nx
H(,)	Double (,)	matrix of partials, of size Nz, Nx
Q(,)	Double (,)	matrix of process noise of size Nx, Nx.
Φ(,)	Double (,)	propagation or dynamics matrix and is of size Nx, Nx.
Nx	Double ()	number of states.
Nz	Long	number of measurements at each update.

Blue parameters are inputs, black parameters are both inputs and outputs. Function returns as a zero for success, and a -1 if the update failed (e.g., singular P matrix).



The input matrices and vectors are defined above. All are double precision. Nx and Nz are longs. **Please see the sample program Visual Basic Source Code for a demonstration on the declarations and calling sequence.** Upon return both the X state vector and the P covariance matrix have been propagated and updated.

The algorithms utilized by KalmanFtX are listed below.

**State Propagation - Propagates the state vector to the time of the current measurement.**

$$X(-)_k = \Phi_{k-1} \cdot X(+)_k$$

**Covariance Propagation - Propagates the covariance matrix to the time of the current measurement and adds process noise.**

$$P(-)_k = \Phi_{k-1} \cdot P(+)_k \cdot \Phi_{k-1}^T + Q_{k-1}$$

**Kalman Filter Gain - Derives the Gain "weighting" matrix.**

$$K_k = P(-)_k H_k^T [H_k P(-)_k H_k^T + R_k]^{-1}$$

**Covariance Update - Updates the covariance matrix.**

$$P(+)_k = [I - K_k \cdot H_k] P(-)_k$$

**State Vector Update - Updates the state vector with the current measurements, weighted by the gain matrix.**

$$X(+)_k = X(-)_k + K_k [Z_k - H_k X(-)_k]$$



Below is an example of a simple Kalman filter design utilized to update measurements of position and velocity to a three state system consisting of position, velocity, and acceleration. These measurements are obtained every  $\Delta T$  seconds.

The  $\Phi$  propagation matrix would be as shown here. The first row shows how the position changes (1 times the position plus  $\Delta T$  times the velocity +  $.5 \cdot \Delta T^2$  times the acceleration). The next row shows how the velocity changes (0 times the position plus 1 times the velocity plus  $\Delta T$  times the acceleration). The last row shows that the acceleration stays constant (0 times the position plus 0 times the velocity plus 1 times the acceleration). For this example, the measurements are obtained every 10 seconds, so  $\Delta T = 10$ .

$$\Phi = \begin{pmatrix} 1 & \Delta T & .5 \cdot \Delta T^2 \\ 0 & 1 & \Delta T \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 10 & 50 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{pmatrix}$$

Note that just because we are modeling the acceleration as a constant does not mean the acceleration state can not change measurement to measurement. The process noise ( $Q$ ) introduced to the covariance matrix at each update allows the acceleration state to "drift".

One of the most complex aspects of Kalman filters is how the process noise and the propagation matrix work together.

For this example we use a simple process noise of .1 (m/sec/sec) 1 sigma on the acceleration term only. Note that the value is squared in the matrix since the  $Q$  consists of variance terms (same for the  $R$  and  $P$  matrices described below).

$$Q = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & .1^2 \end{pmatrix}$$

The  $H$  matrix represents the partial derivative of the measurements with respect to the states, e.g.,  $H(j,k) = dZ_j/dX_k$ , where  $Z$  is the measurement vector and  $X$  is the state vector. In this case the measurements consist of one position and one velocity value (at each update). So the  $H$  matrix would consist of the following values. Recall that  $H$  has the dimension of number measurements by number of states. Since one of the measurements (1st row) consists of the position then  $dZ_0/dX_0 = 1$  (both  $Z$  and  $X$  are positions). Also  $dZ_0/dX_1 = 0$  and  $dZ_0/dX_2 = 0$  since change in position does not effect velocity and acceleration (in this model). Similarly for the velocity measurement (second row).



$$H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

The R matrix consists of the measurement noise. If the measurements are uncorrelated (probably the most common situation) then the matrix will be diagonal. In this example we will use the measurement noise one sigma of 2 meters for position and .5 m/sec for velocity. The R matrix would then be:

$$R = \begin{pmatrix} 2^2 & 0 \\ 0 & .5^2 \end{pmatrix}$$

Lets say that this is the first update, so the input X state vector consists of just a crude estimate of the position, velocity, and acceleration at time "0". In this case the covariance matrix has not yet been propagated or updated. In most situations like this the covariance matrix is initialized with large values on the diagonal, and zeros on the off diagonals. The values utilized are not so critical as long as they are at least as large as your uncertainty and not so large as to cause a singular matrix problem. If you feel your initial guess for the position is +/- 20 m and velocity is +/- 5 m/sec and acceleration is +/- .5 m/sec/sec then an example initial covariance for this problem could be something significantly larger then these. For example:

$$P = \begin{pmatrix} 100^2 & 0 & 0 \\ 0 & 10^2 & 0 \\ 0 & 0 & 1^2 \end{pmatrix}$$

For the purpose of this example take the initial estimate of the state vector (position, velocity, acceleration) as:

$$X = (0 \ 0 \ .2)$$



The measurements are received at 10-second intervals consisting of the following sets:

1st measurement set =

$$Z = (11.3 \quad 1.8)$$

After the call to KalmanFtX1 ( $i = \text{KalmanFtX1.Kalman}(Z, R, X, P, H, Q, \Phi, N_x, N_z)$ ) the covariance vector becomes

$$P = \begin{pmatrix} 3.9986 & 0.0007 & -0.0088 \\ 0.0007 & 0.2494 & 0.0166 \\ -0.0088 & 0.0166 & 0.4556 \end{pmatrix}$$

and the state vector becomes

$$X = (11.299 \quad 1.801 \quad 0.184)$$

Note that this represents the estimate of the position velocity and acceleration of the object 10 seconds after the time where the system was initialized at  $X = (0,0,.2)$ .

Now we can simply recall KalmanFtX1 with the new measurements 10 seconds later using the updated states and covariance from the previous measurement. The  $\Phi$  propagation matrix stays the same as long as the updates are every 10 seconds. The  $Q$ , and  $R$  matrix will stay the same (in most implementations). The  $H$  matrix will stay the same if this is a linear system. Many systems are non-linear and require recalculations of the  $H$  matrix using the most recent state vector estimates. If the update period changes then recalculate the propagation matrix with the new  $\Delta T$ . Also for non-linear systems the propagation matrix will change with the state vector.



## 7.0 SortX

The SortX control contains two types of functions, those utilized for sorting a vector (VectorSort), and those utilized for sorting an array of data based on a selected column (ColumnSort).

A property called SortOrder sets whether any future call to the various sorting functions will return the data sorted in ascending or descending order. The default is ascending order.

SortX1.SortOrder = 0 ' for Ascending order (default value)

SortX1.SortOrder = 1 ' for Descending order

The VectorSort functions include VectorSort\_Double, VectorSort\_Single, VectorSort\_Long, VectorSort\_Integer, and are used for vectors declared as double, single, long, and integer respectively.

For example:

```
Public Function VectorSort_Double(ByRef Data() As Double, ByVal num As Long)
```

Parameter Name	Type	Description
<b>Data()</b>	Double ()	vector of data to be sorted, of length num
<b>num</b>	long	Number of data in vector

Blue parameters are inputs, black parameters are both inputs and outputs.

```
Private Sub Perform_Sort()

    Static xx(num) As Double
    Dim num as long

    ' makeup data
    num=100
    For i = 0 To num - 1
        xx(i) = -10 + Rnd(1) * 20#
    Next i

    ' sort data, after call xx is sorted
    Call SortX1.VectorSort_Double(xx, num)

End Sub
```



The ColumnSort functions include ColumnSort\_Double, ColumnSort\_Single, ColumnSort\_Long, ColumnSort\_Integer, and are used for arrays declared as double, single, long, and integer respectively.

For example:

```
Public Function ColumnSort_Double(ByRef Data(,) As Double, ByVal numrows As Long, ByVal numcol As Long, ByVal SortCol As Long) As Long
```

Parameter Name	Type	Description
<b>Data(,)</b>	Double ( , )	array of data to be sorted, of size num
numrows	long	Number of rows
numcol	long	Number of columns
SortCol	long	Desired column to sort array by.

Blue parameters are inputs, black parameters are both inputs and outputs. Function returns as a zero for success, and a -1 if the update failed.

```
Private Sub Perform_Column_Sort()  
  
Dim numrow As Long, numcol As Long, selectedcol As Long  
  
numrow = 30  
numcol = 4  
selectedcol = 3  
  
' important, array must be dimensioned exactly by numrow x numcol  
Static xxx(numrow, numcol) As Double  
  
' makeup data  
For i = 0 To numrow - 1  
    For j = 0 To numcol - 1  
        xxx(i, j) = -10 + rnd() * 20#  
    Next j  
Next i  
  
' sort array row by row based on column "selectedcol", after call xxx is sorted  
Call SortX1.ColumnSort_Double(xxx, numrow, numcol, selectedcol)  
  
End Sub
```



## 8.0 EncryptX

The EncryptX control is utilized to encrypt a text field. The user can encrypt entire ASCII based documents with one simple call. If someone tries to decrypt the encrypted text with the wrong password, the text remains unchanged (remains encrypted).

```
Sub EncryptExample()  
Dim sdata As string  
Dim password as string  
  
Sdata="Hello, have a nice day" ' some example text  
Password="yourpassword" ' user selected password  
Call EncryptX1.encrypt ("Password ", Sdata, True)  
msgbox Sdata  
Call EncryptxX1.encrypt ("Password ", Sdata, false)  
msgbox Sdata  
  
End Sub
```



## 9.0 RootX

The RootX control is utilized to find a root (zero crossing) of a user supplied function. The control also has a function for finding the minimum or maximum of a function quickly and accurately. These are demonstrated below.

```
Public Function rootfind(ByVal initX1 As Double, ByVal initX2 As Double,
ByVal xacc As Double, ByRef Xfinal As Double, ByRef Yfinal As Double,
ByVal userfunc As roottestdel) As Long
```

Parameter Name	Type	Description
initX1	Double	Initial guess (initX1 and initX2 must bracket true answer)
initX2	Double	Initial guess (initX1 and initX2 must bracket true answer)
xacc	Double	Required accuracy
Xfinal	Double	Resulting X value where root occurs
Yfinal	Double	Resulting Y value where root occurs (should be close to zero).
userfunc	roottestdel	Delegate function type: Function roottest(ByVal x As Double) As Double

Blue parameters are inputs, red parameters are returned values. Function returns as a zero for success, and a -1 if the update failed.

```
Public Function minmaxfind(ByVal initX1 As Double, ByVal initX2 As
Double, ByVal xacc As Double, ByRef Xfinal As Double, ByRef Yfinal As
Double, ByVal userfunc As mintestdel) As Long
```

Parameter Name	Type	Description
initX1	Double	Initial guess (initX1 and initX2 must bracket true answer)
initX2	Double	Initial guess (initX1 and initX2 must bracket true answer)
xacc	Double	Required accuracy
Xfinal	Double	Resulting X value where root occurs
Yfinal	Double	Resulting Y value where min occurs (should be close to zero).
userfunc	mintestdel	Delegate function type: Function mintestdel(ByVal x1 As Double) As Double

Blue parameters are inputs, red parameters are returned values. Function returns as a zero for success, and a -1 if the update failed.

Note, to find a functions maximum, simply multiply the user function equation by -1.0 and call minmaxfind to search for the “minimum”.



Here are examples of root finding in Visual Basic and Visual C#.

```
' Root Finding example in Visual Basic .NET
Function roottest(ByVal x As Double) As Double
    roottest = (x - 17.45)
End Function

Sub Root_Finding()
    Dim ilong As Long, xfinal As Double, yfinal As Double
    Dim x1 As Double, x2 As Double, x3 As Double, i As Long

'initial guess (must bracket root or min or max)
    x1 = -1000.0#
    x2 = 1000.0#

' required accuracy
    x3 = 0.0001

' find root
    i = RootX1.rootfind(x1, x2, x3, xfinal, yfinal, AddressOf roottest)
    MsgBox("Root X Y = " & xfinal & " " & yfinal)

End Sub
```

```
// Root Finding example in Visual C# .NET
double roottest(double x ) {
    return x - 17.45;
}

void Root_Finding()
{
    double x1, x2, x3, xfinal=0.0, yfinal =0.0;
    long i;

    // declare and assign a delegate for your user function
    MathLibX.RootX.roottestdel userfun;
    userfun = new MathLibX.RootX.roottestdel(roottest);

    // initial guess (must bracket root or min or max)
    x1 = -1000.0, x2 = 1000.0;

    x3 = 0.0001; // required accuracy

// find root
    i = rootX1.rootfind(x1, x2, x3, ref xfinal, ref yfinal, userfun);
    textBox1.Text = ("root found at " + xfinal.ToString());
}
```



## 11.0 Distributions

To distribute .NET applications that contains the MathLibX .Net controls, you must include the following two files:

**MathLibX.dll** - Note: this is the MathLibX .NET control.

and

**mathdll.dll** - Note : this file must be in the directory of the application, or must be put into the windows\system32\ folder (recommended).

## 12.0 Getting Help

If you are experiencing any kind of trouble with installation, compilation, execution, etc, please contact us at [info@mathfunctions.com](mailto:info@mathfunctions.com). In the overwhelming majority of the cases we can respond within 24 hours. Please include your name and phone number.