

```

/*-----*
* File Name: FFTUsingNAG.c                                     *
* Creation: ER, 03/04/2005                                   *
* Purpose: Programming Example: Compute FFT using NAG library *
* Copyright (c) OriginLab Corp. 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 *
* All Rights Reserved                                       *
*                                                           *
* Modification Log:                                         *
*-----*/

#include <Origin.h>

// Note: Before compile and link this file, need run "run.LoadOC(Originlab\fft utils.c, 16)"
// in Command window to compile fft_utils.c file to current workspace.

////////////////////////////////////
// This function performs FFT computation using NAG library functions.
// By default, the FFT is computed with no padding or truncation of the data,
// using exactly the same number of points as are available in the signal.
// User can however specify the number of points. If user specifies a size
// less than signal size, the signal is truncated internally before computing FFT.
// If user specifies a size larger than the signal size, the signal is padded with
// zeroes internally. In either case, the original signal data in the worksheet is
// not affected.
//
// A result worksheet containing Freq, Re, Im, Amp, Phase, Power columns is created.
// Freq step is computed using:
//     delta-f = 1 / (delta-t * n)
// where delta-t is the spacing in time between the first two time values
// and n is the data size (default, or user-specified value).
// If no X dataset is found, then row number is used.
//
// Parameters:
//     strSignal:      Name of Y dataset containing the signal
//     nPts:           Number of points to use for FFT.
//                   If none specified, exact signal size is used.
//
// Example:
// Compile and link this function, and then go to the script window and issue
// command such as:
//     fft using nag data1_b      // perform fft data1_b using exact length
//     fft using nag data1 b 512 // pad or truncate to 512 points and perform fft
//
void fft using nag(string strSignal, int nPts = -1)
{
    // Declare curve object using signal dataset name and check validity
    Curve crvSignal(strSignal);
    if( !crvSignal )
    {
        out_str("Failed to declare signal curve.");
        return;
    }

    // Set up complex vectors, and copy signal curve to vector
    vector<complex> vecSignal, vecFFT;
    vecSignal = crvSignal;

    // If user specified nPts, then either pad or truncate
    if( -1 != nPts )
    {
        bool bRet = vecSignal.SetSize(nPts);
        if( !bRet )
        {
            out_str("Failed to set signal size to specified number of points.");
            return;
        }
    }
    else
        nPts = vecSignal.GetSize();

    // Call FFT() function.
    // This function internally calls NAG library functions to compute FFT
    int iRet = FFT(vecSignal, vecFFT);
    if( 0 != iRet )
    {
        printf("FFT routine returned error: %d\n", iRet);
        return;
    }

    // Scale the FFT result by sqrt(N) to undo normalization performed by NAG
    vecFFT *= sqrt(nPts);

    // Create result worksheet page
    WorksheetPage wpgFFT;
    wpgFFT.Create("Origin");
}

```

```

string strLabel;
strLabel.Format("NAG FFT of %s with %d Points", crvSignal.GetName(), nPts);
wpgFFT.Label = strLabel;
wpgFFT.TitleShow = WIN TITLE SHOW BOTH;

// Declare result worksheet and set up columns and datasets
Worksheet wksFFT = wpgFFT.Layers(0);
while( wksFFT.DeleteCol(0) ); // Delete all columns

int nIndex;
nIndex = wksFFT.AddCol("Frequency");
wksFFT.Columns(0).SetType(OKDATAOBJ DESIGNATION X);
Dataset dsFrequency(wksFFT, nIndex);

nIndex = wksFFT.AddCol("Real");
Dataset dsReal(wksFFT, nIndex);

nIndex = wksFFT.AddCol("Imaginary");
Dataset dsImaginary(wksFFT, nIndex);

nIndex = wksFFT.AddCol("Amplitude");
Dataset dsAmplitude(wksFFT, nIndex);

nIndex = wksFFT.AddCol("Phase");
Dataset dsPhase(wksFFT, nIndex);

nIndex = wksFFT.AddCol("Power");
Dataset dsPower(wksFFT, nIndex);

// Get Re, Im, Amp, Phase from FFT result1 and store in wks
vector vecTemp;
vecFFT.GetReal(vecTemp);
dsReal = vecTemp;

vecFFT.GetImaginary(vecTemp);
dsImaginary = vecTemp;

vecFFT.GetAmplitude(vecTemp);
dsAmplitude = vecTemp;

vecFFT.GetPhase(vecTemp);
dsPhase = vecTemp;
dsPower = dsAmplitude^2 / nPts;

// Compute frequency column
dsFrequency.SetSize(nPts);
// If X column exists for signal, use data in that column
double dFreqStep;
if( crvSignal.HasX() )
{
    Dataset dsTime;
    crvSignal.AttachX(dsTime);
    dFreqStep = 1.0 / (nPts * (dsTime[1] - dsTime[0]));
}
// else just use row numbers
else
    dFreqStep = 1.0 / nPts;
for(int ii = 0; ii < nPts; ii++)
    dsFrequency[ii] = ii * dFreqStep;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```