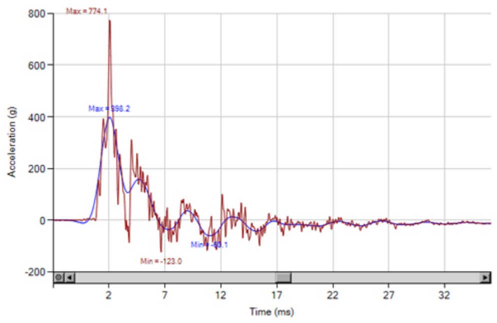
A Butterworth Filter in C#

Darryl Bryk, 22 Apr 2016

★★★★★ 5.00 (11 votes) Rate this:

C# code for a low-pass Butterworth filter is presented



Graph showing same data unfiltered and filtered

Introduction

This post presents C# code for a fourth order zero-phase shift low-pass Butterworth filter function. The code was converted to C# from code originally written in Visual Basic for Applications (VBA) by Sam Van Wassenbergh (University of Antwerp, 2007). I was using the VBA code for an application, and when I decided to convert the application to C#, I found there were not many resources for a Butterworth filter written in C#, so I converted the VBA code to C#, and hence I thought an article could be useful to others.

Using the Code

The function is shown below. The unfiltered data is passed to the function as a `double[]` array, along with the time step in seconds, and the desired cutoff frequency in Hz. The filtered data is returned, unless the input data `indata` is `null`, in which case, `null` is returned, or if the cutoff frequency is `0` then the original data is returned unfiltered. The time step variable `deltaTimeinsec`, which is the time between one data point and the next in seconds, is needed to calculate the sampling rate (inverse of the time step). The input data is prepared in a slightly larger array (4 extra points - two extra points at each end of the data), so that the real endpoints will have neighbors for the calculations. After the variables are assigned, the two loops perform the backward and forward traversal of the filter calculations. Then the output data is put back into an array the same size as the original for return.

Hide Shrink ▲ Copy Code

```
//--------------------------------------------------------------------------
// This function returns the data filtered. Converted to C# 2 July 2014.
// Original source written in VBA for Microsoft Excel, 2000 by Sam Van
// Wassenbergh (University of Antwerp), 6 june 2007.
//--------------------------------------------------------------------------
public static double[] Butterworth(double[] indata, double deltaTimeinsec, double CutOff) {
    if (indata == null) return null;
    if (CutOff == 0) return indata;

    double Samplingrate = 1 / deltaTimeinsec;
    long dF2 = indata.Length - 1;        // The data range is set with dF2
    double[] Dat2 = new double[dF2 + 4]; // Array with 4 extra points front and back
    double[] data = indata; // Ptr., changes passed data

    // Copy indata to Dat2
    for (long r = 0; r < dF2; r++) {
        Dat2[2 + r] = indata[r];
    }
    Dat2[1] = Dat2[0] = indata[0];
    Dat2[dF2 + 3] = Dat2[dF2 + 2] = indata[dF2];

    const double pi = 3.14159265358979;
    double wc = Math.Tan(CutOff * pi / Samplingrate);
    double k1 = 1.414213562 * wc; // Sqrt(2) * wc
    double k2 = wc * wc;
    double a = k2 / (1 + k1 + k2);
    double b = 2 * a;
    double c = a;
    double k3 = b / k2;
    double d = -2 * a + k3;
    double e = 1 - (2 * a) - k3;

    // RECURSIVE TRIGGERS - ENABLE filter is performed (first, last points constant)
    double[] DatYt = new double[dF2 + 4];
    DatYt[1] = DatYt[0] = indata[0];
    for (long s = 2; s < dF2 + 2; s++) {
        DatYt[s] = a * Dat2[s] + b * Dat2[s - 1] + c * Dat2[s - 2]
                   + d * DatYt[s - 1] + e * DatYt[s - 2];
    }
    DatYt[dF2 + 3] = DatYt[dF2 + 2] = DatYt[dF2 + 1];

    // FORWARD filter
    double[] DatZt = new double[dF2 + 2];
    DatZt[dF2] = DatYt[dF2 + 2];
    DatZt[dF2 + 1] = DatYt[dF2 + 3];
    for (long t = -dF2 + 1; t <= 0; t++) {
        DatZt[-t] = a * DatYt[-t + 2] + b * DatYt[-t + 3] + c * DatYt[-t + 4]
                    + d * DatZt[-t + 1] + e * DatZt[-t + 2];
    }

    // Calculated points copied for return
    for (long p = 0; p < dF2; p++) {
        data[p] = DatZt[p];
```