JOURNAL OF OBJECT TECHNOLOGY

Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2010**

Vol. 9, No. 2, March - April 2010

The Discrete Fourier Transform, Part 6: Cross-Correlation

By Douglas Lyon

Abstract

This paper is part 6 in a series of papers about the Discrete Fourier Transform (DFT) and the Inverse Discrete Fourier Transform (IDFT). The focus of this paper is on correlation. The correlation is performed in the time domain (slow correlation) and in the frequency domain using a Short-Time Fourier Transform (STFT). When the Fourier transform is an FFT, the correlation is said to be a "fast" correlation. The approach requires that each time segment be transformed into the frequency domain after it is windowed. Overlapping windows temporally isolate the signal by amplitude modulation with an apodizing function. The selection of overlap parameters is done on an ad-hoc basis, as is the apodizing function selection.

This report is a part of project *Fenestratus*, from the skunk-works of DocJava, Inc. Fenestratus comes from the Latin and means, "to furnish with windows".

1 INTRODUCTION TO CROSS-CORRELATION

Cross-Correlation (also called cross-covariance) between two input signals is a kind of template matching. Cross-correlation can be done in any number of dimensions. For the purpose of this presentation, we define one-dimensional normalized cross-correlation between two input signals as:

$$r_d = \frac{\sum_i [(x[i] - \bar{x}) \cdot (y[i-d] - \bar{y})]}{\sqrt{\sum_i (x[i] - \bar{x})^2} \sqrt{\sum_i (y[i-d] - \bar{y})^2}} \quad (1)$$

The coefficient, *r*, is a measurement of the size and direction of the linear relationship between variables *x* and *y*. If these variables move together, where they both rise at an identical rate, then *r* = +1. If the other variable does not budge, then *r* = 0. If the other variable falls at an identical rate, then *r* = -1. If *r* is greater than zero, we have positive correlation. If *r* is less than zero, we have negative correlation.

The sample non-normalized cross-correlation of two input signals requires that *r* be computed by a sample-shift (time-shifting) along one of the input signals. For the numerator, this is called a sliding dot product or sliding inner product. The dot product is given by:

$$\mathbf{X} \bullet \mathbf{Y} = \sum_i x_i y_i \tag{2}$$

When (1) is computed, for all delays, then the output is twice that of the input. It is common to use the pentagon notation when showing a cross correlation:

$$(x \bigstar y)_d = \sum_i x_i^* y_{i+d} \tag{3}$$

Where the asterisk indicates the complex conjugate (a negation of the imaginary part of the number). Input signals should either have the same length, or there should be a policy in place to make them the same (perhaps by zero padding or data replication). If the input signals are real-valued, then we can write:

$$(x \bigstar y)_d = \sum_{i=-\infty}^{\infty} x_i y_{i+d} \tag{4}$$

Comparing (4) with the convolution:

$$x_n * y_n = \sum_{i=-\infty}^{\infty} x_i y_{n-i} \tag{5}$$

Shows that Y is time-reversed before shifting by *n*. In comparison, correlation has shifting without the time reversal.

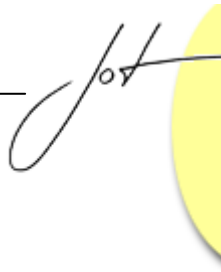
2 AN EXAMPLE, IN EXCEL

Suppose you are given two signals that have already had their means subtracted. Correlate the signals, without dividing by the standard deviation. The signals are: X=1,2,3,4 with Y = 3, 2, 0, 1.

x	y	y1	y2	y3	y4	y5	y6	y7
1	3	0	0	0	3	2	0	2
2	2	0	0	3	2	0	2	0
3	0	0	3	2	0	2	0	0
4	2	3	2	0	2	0	0	0
corr		12	17	12	15	8	4	2

Figure 1. Y is shifted 7 times

Figure 1 demonstrates that a moving cross correlation requires that the kernel of the signal be shifted so that its leading edge appears and then is shifted until only the trailing edge can be seen. After each shift, the rest of the signal is padded with zeros. The row labeled *corr* contains the correlation and results from the dot product of X with Y. For example X•Y1=12, X•Y2=17, etc.



3 A SLOW CROSS CORRELATION

We implement the slow cross correlation using a sliding dot product:

```
public static double[] shift(double d[], int s) {
        double c[] = new double[d.length];
        for (int i = 0; i < c.length; i++) {
            if ((s + i >= 0)&&(s+i < d.length))
                c[i] = d[s + i];
        }
        return c;
    }

    public static void main(final String[] args) {
        double x[] = {1,2,3,4};
        double y[] = {3,2,0,2};
        PrintUtils.print(x);
        PrintUtils.print(y);
        for (int i = -y.length+1; i < y.length; i++){
            double slidingY[] = shift(y, i);
            PrintUtils.print(slidingY);
            System.out.println("dot
product:"+Mat1.dot(x,slidingY));
        }
    }
```

Where the dot product is implemented using:

```
static public double dot(final double[] a,
                                final double[] b) {
        final int aLength = a.length;
        if (aLength != b.length) {
            System.out.println(
                    "ERROR: Vectors must be of equal length in
dot product.");
            return 0;
        }
        double sum = 0;
        for (int i = 0; i < aLength; i++) {
            sum += a[i] * b[i];
        }
        return sum;
    }
```

The output matches that given in Figure 1:

```
1.0 2.0 3.0 4.0
3.0 2.0 0.0 2.0
0.0 0.0 0.0 3.0
dot product:12.0
0.0 0.0 3.0 2.0
dot product:17.0
0.0 3.0 2.0 0.0
dot product:12.0
3.0 2.0 0.0 2.0
dot product:15.0
2.0 0.0 2.0 0.0
```

```
dot product:8.0
0.0 2.0 0.0 0.0
dot product:4.0
2.0 0.0 0.0 0.0
```

The implementation is clearly not optimized, but it is correct and serves to illustrate the sliding dot product nature of the cross correlation.

4 A FAST CORRELATION

A slow implementation of the moving cross correlation algorithm, as shown in Figure 1, will take O(N**2) time. Further, the unoptimized implementation, shown in section 3, has high constant time overhead.

Using the FFT and the correlation theorem, we accelerate the correlation computation. The correlation theorem says that multiplying the Fourier transform of one function by the complex conjugate of the Fourier transform of the other gives the Fourier transform of their correlation. That is, take both signals into the frequency domain, form the complex conjugate of one of the signals, multiply, then take the inverse Fourier transform. This is expressed by:

$$f(x) \star g(x) \leftrightarrow F^*(u)G(u) \tag{6}$$

We now compare the slow correlation with the fast correlation:

```
public static void testCorrelation() {

        final double x[] = {1,2,3,4};
        final double y[] = {3,2,0,2};
        PrintUtils.print(x);
        PrintUtils.print(y);

        System.out.println("cross cor (slow):");
        PrintUtils.print(Mat1.slowCorrelation(x, y));

        System.out.println("Fast xcor:");
        PrintUtils.print(SigProc.correl(x, y, 0));
    }
```

The output follows:

```
1.0 2.0 3.0 4.0
3.0 2.0 0.0 2.0
cross cor (slow):
12.0 17.0 12.0 15.0 8.0 4.0 2.0
Fast xcor:
12.0 17.0 12.0 15.0 8.0 4.0 2.0
```

The fast correlation makes use of the FFT, and gets identical results to the slow correlation. So how much faster is the FFT than the slow correlation for small sized arrays? The testing code follows:

```
    public static void main(String[] args) {
            for (int i = 1; i < 5; i++) {
                System.out.println("size:" + i*256);
                speedTestCorrelation(256 * i);
            }
            //main2(args);
        }

        public static void speedTestCorrelation(int n) {

            final double x[] = new double[n];
            final double y[] = new double[n];
            StopWatch sw = new StopWatch();
            sw.start();

            Mat1.slowCorrelation(x, y);
            sw.stop();
            sw.print("slow correlation is done");
            sw.start();
            SigProc.correl(x, y, 0);
            sw.stop();
            sw.print("fast correlation is done");
        }
```

Even for modest arrays, we see substantial speedup (between 2.5 and 12 times faster, for small array sizes):

```
size:256
slow correlation is done 0.01 seconds
fast correlation is done 0.0040 seconds
size:512
slow correlation is done 0.0060 seconds
fast correlation is done 0.0020 seconds
size:768
slow correlation is done 0.013 seconds
fast correlation is done 0.0030 seconds
size:1024
slow correlation is done 0.025 seconds
fast correlation is done 0.0020 seconds
```

5 SUMMARY

This paper shows how the FFT can be used to speed up cross correlation. Further, it shows that even for small array sizes, substantial speed up can be obtained by using the fast cross correlation. Arguments for using the FFT to accelerate the cross correlation are often not supported with specific data on computation time (a situation, which this paper remedies) [Lyon 97].

The cross correlation has uses in many fields of scientific endeavor (music, identification of blood flow, astronomical event processing, speech processing, pattern recognition, financial engineering, etc.).

One of the basic problems with the term *normalization* when applied to the cross-correlation is that it is defined in different places differently. For example, Pratt suggests that the number of elements in the normalization (an even a square root) is

not needed [Pratt]. Lewis suggests using both the square root and the average [Lewis]. Therefore the question of which normalization to use is application-specific.

REFERENCES

[Lewis] "Fast Normalized Cross-Correlation" by J.P. Lewis, http://www.idiom.com/~zilla/Papers/nvisionInterface/nip.html last accessed 8/24/09.

[Lyon 97] *Java Digital Signal Processing*, Douglas A. Lyon and H. Rao, M&T Press (an imprint of Henry Holt). November 1997.

[Pratt] W. Pratt, *Digital Image Processing*, John Wiley, New York, 1978.

About the author

Douglas A. Lyon (M'89-SM'00) received the Ph.D., M.S. and B.S. degrees in computer and systems engineering from Rensselaer Polytechnic Institute (1991, 1985 and 1983). Dr. Lyon has worked at AT&T Bell Laboratories at Murray Hill, NJ and the Jet Propulsion Laboratory at the California Institute of Technology, Pasadena, CA. He is currently the co-director of the Electrical and Computer Engineering program at Fairfield University, in Fairfield CT, a senior member of the IEEE and President of DocJava, Inc., a consulting firm in Connecticut. Dr. Lyon has authored or co-authored three books (Java, Digital Signal Processing, Image Processing in Java and Java for Programmers). He has authored over 40 journal publications. Email: lyon@docjava.com. Web: http://www.DocJava.com.