Proceedings of the World Congress on Engineering 2010 Vol III
WCE 2010, June 30 - July 2, 2010, London, U.K.

Line Segment Convolution

Jeffrey A. Dunne, Kevin M. Ligozio

***Abstract*— An alternate method for the convolution of two functions is presented in which those functions can be described in terms of completely arbitrarily, non-uniform sampled ranges. The approach treats the functions as being defined by the linear segments between sampling points rather than the points themselves, and calculates a result using an analytic solution of the convolution of two finite line segments. It is observed that the approach can provide computational savings in circumstances where the non-uniform range specifications enable the functions to be adequately described with fewer sampling points.**

***Index Terms*— Convolution, distribution, linear, piecewise, probability density.**

I. Introduction

The demand for increasing accuracy and applicability of computer simulations and calculations to physical systems has resulted in a corresponding need for greater sophistication and precision in the digital encoding and manipulation of measurements. Where the representation of a distribution using single values or a limited subset of statistical moments (mean, median, standard deviation, and so forth) was once the only feasible approach, current expectations often require embracing more careful and exacting treatment. Unfortunately, even comparatively modest calculations involving arbitrary distributions (i.e. those distributions that cannot be represented analytically) can be demanding.

This work resulted from a need to utilize and manipulate measurements within a tool for predictive modeling of physical systems. In that analysis, as in many cases, the system's capabilities were driven by tails of the distributions of measurements and were largely invariant to the means or medians. In some instances, the model would need to incorporate multimodal measurements of physical quantities, where representing the parameter with a single number, or even a limited number moments of the distribution, would produce unrealistic results.

To address the problem, the above-mentioned code was rewritten to use distributions in place of single numbers. This resulted in the need to do numerous convolutions, one for each addition or subtraction of quantities. Because the quantities were taken from measurements, there was no assurance of a uniform grid of possible outcomes for any given random variable, and with the various quantities coming from different sources, there was no single "grid" spacing that was natural to imposed on the measurements. Random variable (RV) X, for example, might have ten values ranging from 3 to 4, while Y could have ten values ranging from 1000 to 2000. Imposing a single grid meant that in order to retain the relevant resolution for X, the representation of Y would require 10,000 points, 9990 of which were interpolations that carried no information content.

The typical approach for efficient convolution uses Fast Fourier Transforms (FFTs). While such implementations can be extremely efficient, there comes a limit where the increase in the number of points resulting from establishing a uniform grid – whether it is 10,000 or 100 million – outweighs the efficiency of the algorithm. While algorithms have been developed that utilize multiple grid/sampling rate resolutions [1], this is not a uniformly practical approach when working with measured data where there are potentially few, if any, portions of data that share a common spacing of RV values, or where algorithmically finding an appropriate grid might be more computationally intensive than doing the convolution calculation itself.

This motivated the search for an approach to convolution that could be carried out on fully arbitrary functions, i.e. without requiring any uniformity in the gridding by which the functions were characterized. The approach reported herein is noteworthy for two reasons. First, for the reasons outlined above, there are certain circumstances (e.g. when significant upsampling is required of the component functions to enable the standard approach) where its ability to compute convolutions using completely arbitrary function sampling enables it to run faster than the standard algorithms. Second, from a more academic perspective, it illustrates an alternate perspective from which to approach working with sampled functions. There are occasions where it is advantageous to consider a set of points as defining a function not as instantaneous snapshots of the function values, but rather by the segments connecting those points. As a tractable starting position with an easily derived analytic solution, the approach outlined in this paper treats the points as connected by straight line segments. However, just as Simpson's rule extends the trapezoidal rule, more precise algorithms could be developed if the required accuracy so demanded.

Manuscript received December 18, 2009.
J. A. Dunne is with the Johns Hopkins University Applied Physics Laboratory, Laurel, MD 20723 USA (phone: 443-778-4722; fax: 443-778-5950; e-mail: jeffrey.dunne@jhuapl.edu).
K. M. Ligozio is with the Johns Hopkins University Applied Physics Laboratory, Laurel, MD 20723 USA (kevin.ligozio@jhuapl.edu).

In summary, this paper describes an approach for calculating the sum of two arbitrary, continuous RVs represented by a sampling of their associated probability densities that is finite and potentially possesses no uniformity. It does not rely on FFTs, but rather utilizes an analytic solution for the convolution of finite linear segments. The approach can be less computationally demanding than standard convolution in cases where a) the two functions have substantially different resolutions, b) the functions do not have any regular grid by which they can be defined, or c) only a small subset of the distributions require fine sampling to capture the nature of those distributions, i.e. where non-uniformly spaced representations of the probability distributions are preferable.

II. Algorithm Overview

Typically, a distribution represented by a finite number of samples of the probability density is viewed as being represented by the samples themselves, and so the convolution process requires a uniformly sampled axis of the RV range in order to produce a meaningful result. The motivation for this algorithm is to consider the finite samples as not representing the distribution directly, but rather through the straight line segments that connect those samples, i.e. where n_s samples are used to represent $(n_s - 1)$ straight line segments. These $(n_s - 1)$ segments are then "convolved" in order to obtain the solution. The computational savings does not result from the minimal decrement of n_s to $(n_s - 1)$, but rather from the reduction of n_s that arises from being able to utilize a non-uniformly sampled RV range, as illustrated in Figure 1.

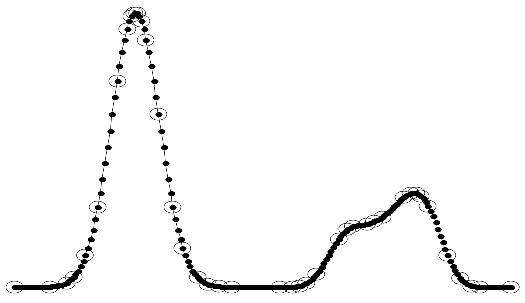
Fig. 1. Non-uniform sampling (circles) as compared with uniformly sampled representations (dots) typically used with convolution

In the algorithm, the probability is the area under the function (the probability density) defined by connecting straight line segments. A simple extension permits handling discontinuous functions as well. As with any sampled function, representations can be made arbitrarily exact by including additional sample points, although as described earlier, the primary value of the algorithm is realized in cases where an exact representation is not necessary to obtain the accuracy required of the result.

The approach utilizes the analytic solution of the convolution of two arbitrary but finite line segments to break up the total convolution into $(m_s - 1)(n_s - 1)$ individual operations, where m_s and n_s are the total numbers of data points in the two distributions to be convolved. Each comprising calculation combines two non-normalized analytical densities (the straight line segments) to provide a non-normalized contribution to the final solution. The summation of all such solutions is automatically normalized to the product of the normalizations of the individual probability densities.

III. Straight Line Segment Convolution

Consider two functions for straight line segments,

$$f(x) = (m_f x + b_f)h(x - \alpha_1)h(\alpha_2 - x) \tag{1}$$
$$g(x) = (m_g x + b_g)h(x - \beta_1)h(\beta_2 - x) \tag{2}$$

where *h(x)* represents a step function at *x* = 0, m_f and m_g are the slopes of the lines, and b_f and b_g are the y-intercepts of the lines defined by these segments. α_1 and α_2 represent the starting and ending range values of the line segment defined by *f(x)*, just as β_1 and β_2 define those limits for *g(x)*.

Without loss of generality we can set α_1 and β_1 to 0, and define $a_f \equiv \alpha_2 - \alpha_1$ and $a_g \equiv \beta_2 - \beta_1$ as the length of each line segment, giving:

$$f(x) = (m_f x + b_f)h(x)h(a_f - x) \tag{3}$$
$$g(x) = (m_g x + b_g)h(x)h(a_g - x). \tag{4}$$

For the sake of this derivation, we will assume, also without loss of generality (because convolution is commutative), that $a_f \leq a_g$.

The convolution of *f(x)* and *g(x)* is given by:

$$c(\bar{x}) = \int_{-\infty}^{\infty} f(x)g(\bar{x} - x)dx. \tag{5}$$

The step functions from f(x) can be easily applied to the limits of the integral to give:

$$c(\bar{x}) = \int_{0}^{a_f} (m_f x + b_f)(m_g(\bar{x} - x) + b_g) \cdot \\ h(\bar{x} - x)h(a_g - (\bar{x} - x))dx. \tag{6}$$

The application of the remaining step functions to the limits of integration is most easily implemented by recognizing three distinct conditions of integration:

Region	Definition	Integration Limits
I	$0 \le \bar{x} < a_f$	$0 \to \bar{x}$
II	$a_f \le \bar{x} < a_g$	$0 \to a_f$
III	$a_g \le \bar{x} \le a_f + a_g$	$\bar{x} - a_g \to a_f$

No regions of consideration are required for $\bar{x} < 0$ or $a_f + a_g < \bar{x}$ as there is no overlap of *f(x)* and $g(\bar{x} - x)$ under these conditions, i.e. the regions do not contribute to the integral.

With these limits γ_1 and γ_2 established, the integral itself results in a simple polynomial:

$$c(\bar{x}) = \left[(b_f b_g + b_f m_g \bar{x})x + (m_f m_g \bar{x} + m_f b_g - b_f m_g)\frac{x^2}{2} - m_f m_g \frac{x^3}{3} \right]_{\gamma_1}^{\gamma_2}. \quad (7)$$

Writing the solutions c_I, c_{II}, and c_{III}, corresponding to these three conditions gives:

$$c_I(\bar{x}) = b_f b_g \bar{x} + (m_f b_g + b_f m_g)\frac{\bar{x}^2}{2} + m_f m_g \frac{\bar{x}^3}{6} \quad (8)$$

$$c_{II}(\bar{x}) = \left(b_f b_g a_f + (m_f b_g - b_f m_g)\frac{a_f^2}{2} - (m_f m_g)\frac{a_f^3}{3} \right) + \left(m_f m_g \frac{a_f^2}{2} + b_f m_g a_f \right)\bar{x} \quad (9)$$

$$c_{III}(\bar{x}) = (b_f m_g \bar{x} + b_f b_g)\tau_1 + \frac{1}{2}(m_f m_g \bar{x} + m_f b_g - b_f m_g)\tau_2 - \frac{m_f m_g}{3}\tau_3. \quad (10)$$

The terms τ_1, τ_2, and τ_3 are given by:

$$\tau_1 = a_f - (\bar{x} - a_g) \quad (11)$$

$$\tau_2 = a_f^2 - (\bar{x} - a_g)^2 \quad (12)$$

$$\tau_3 = a_f^3 - (\bar{x} - a_g)^3. \quad (13)$$

Assigning a value of zero to each of these components for values of $\bar{x}$ outside the solutions' boundaries of applicability, one could also write the solution in terms of a single equation $c(\bar{x}) = c_I(\bar{x}) + c_{II}(\bar{x}) + c_{III}(\bar{x})$, which could then be integrated from 0 to $a_f + a_g$ to show that this result possesses the same net probability as the product of total probabilities in *f(x)* and *g(x)*, i.e.:

$$\frac{a_f a_g}{4}(m_f a_f + 2b_f)(m_g a_g + 2b_g). \quad (14)$$

IV. Algorithm Implementation

The implementation of the line segment convolution method is straightforward, and illustrated by the following steps:

1. Given two inputted functions with m_s and n_s point pairs (e.g. value/probability density pairs in the example outlined in section I), calculate the slopes and horizontal lengths of the contained $\mu \equiv (m_s - 1)$ and $\nu \equiv (n_s - 1)$ straight line segments.
2. Establish an output axis vector of interest. To simulate the output of a standard convolution this could be a vector of range values spanning min(*f(x)*) + min(*g(x)*) through max(*f(x)*) + max(*g(x)*) and spaced according to the smallest interval in either of the inputted functions. However, it is equally feasible to include only selected portions of the resulting distribution, or to sample more finely in certain regions than others[1].
3. Establish an empty (i.e. zeroed) output value vector of matching size.
4. Loop through the $N = \mu\nu$ segment pairings between inputted functions. For each pairing of segments (μ_i, ν_j):
   a. Define $x_{\mu 1}$ and $x_{\mu 2}$ as the minimum and maximum domain values for the μ_i segment, and similarly $x_{\nu 1}$ and $x_{\nu 2}$ for ν_j.
   b. Select the points *P* in the output axis vector that fall within $x_{\mu 1} + x_{\nu 1} \rightarrow x_{\mu 2} + x_{\nu 2}$.
   c. Temporarily consider those *P* points to represent the range 0 to $(a_f + a_g)$, where a_f is the lesser of $(x_{\mu 2} - x_{\mu 1})$ or $(x_{\nu 2} - x_{\nu 1})$ and a_g is the greater of the two differences. In other words, any value in *P* is considered to have a temporary value of its actual value less $(x_{\mu 1} + x_{\nu 1})$.
   d. Select the appropriate slope and intercept for a_f and a_g. If $a_f = (x_{\mu 2} - x_{\mu 1})$ then m_f is the slope of μ_i, $b_f = x_{\mu 1}$, m_g is the slope of ν_j, and $b_g = x_{\nu 1}$. If $a_f = (x_{\nu 2} - x_{\nu 1})$ then m_f is the slope of ν_j, $b_f = x_{\nu 1}$, m_g is the slope of μ_i, and $b_g = x_{\mu 1}$.
   e. Calculate the values corresponding to each axis value in *P* using the previously described $c(\bar{x})$, and increment the associated points in output vector by those values.

[1] The steps outlined here are only one implementation. A more complex implementation might establish an axis but intentionally not fill in points that do not carry unique content, while a still more complex approach might not pre-establish an output vector of the result at all, but rather build one dynamically during the calculation. The choice of implementations is entirely dependant on the needs of the system. The steps used here were selected because they could be described simply.

V. Run-Time Analysis

In order to compare the running times of the FFT convolution and line segment convolution algorithms, we implemented the recursive Cooley-Tukey FFT algorithm using source code from [2]. The performance of this algorithm is known to be $O(N \log_2 N)$ [3]. In order to understand where the line segment convolution algorithm has performance advantages over traditional FFT convolution, it is important to understand how N varies with various inputs. The Cooley-Tukey implementation used for these experiments operates only on powers of two, and requires that both input vectors are of the same size. Further, the computations must be performed at the smallest increase in x values in either input. For example, if we have $x_1 = [1,2,3,\ldots 1000]$ and $x_2 = [1,1.1,50]$, the resulting output x vector would be the 10^6 elements of $[1,1.1,1.2,\ldots 1000]$. Since the algorithm works only on powers of two, we need to pad the input vectors with zeros to get a total $N = 2^{20} = 1{,}048{,}576$. It is this type of inflation that led to the development of the line segment convolution approach.

The line segment convolution algorithm is $O(n \cdot m \cdot p)$ where $n = |x_1|$ and $m = |x_2|$. p varies with the x values in the input arrays such that at each iteration of n and m, p is the length of the resulting line segment range. In order to understand this more clearly, consider the following pseudocode:

```
function doLineSegmConv (Segment[] ls1,
                         Segment[] ls2)
  n = length(ls1)
  m = length(ls2)
  loop i from 1 to n
    loop j from 1 to m
      startAt = ls1[i].getMinX() +
                ls2[j].getMinX();
      endsAt = ls1[i].getMaxX() +
                ls2[j].getMaxX();
      loop k from startsAt to endsAt
         computeConv (ls1[i], ls2[j], k)
      end loop
    end loop
  end loop
end function
```

In the case of the x_1 and x_2 above, the first line segments would range from [1..2] and [1..1.1], respectively. Hence the inner most loop would range from [2..3.1] with a step size of 0.1. Understanding the variable p helps to show why the benefit of line segment convolution is situational.

The following empirical results illustrate a simple run-time comparison between the two algorithms. The x_1 vector is held constant throughout as $x_1 = [1,2,\ldots 10000]$, and $x_2 = [1,2,\ldots m]$ where $5 \leq m \leq 2000$ at steps of 5. We expect that varying the size of x_2 will not affect the FFT convolution run time because we do not increase the size to be greater than x_1 (and so FFT convolution will always operate as though x_2 were as large as x_1). We further expect a linear increase in the line segment convolution algorithm because n is held constant and m is increased linearly. The following chart shows the run-time results as a function of m.

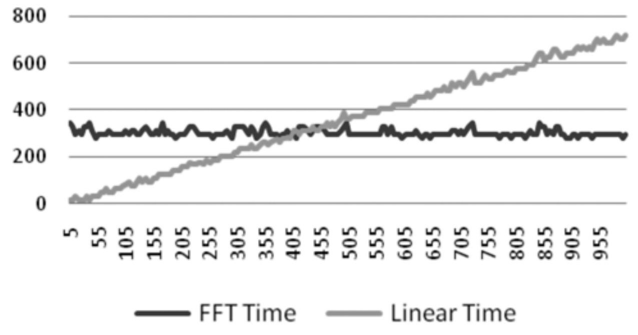

Fig. 2. Run time comparison of FFT and Line Segment Convolution for N=10000 as a function of secondary input vector size

Note that linear convolution outperforms FFT convolution for $m \leq 450$.

In this example, n is held constant; however one can also imagine inputs where both n and m could be significantly smaller than the corresponding vectors that would result in the same computational demand on the FFT approach. For example, consider $x_1 = [1,2,8,10000]$ and $x_2 = [1,2,3,7,20]$. In such a case, $n = 4$ and $m = 5$, while N remains 10000. Such a case would correspond (roughly) to a point in Fig. 2 at 0.002 ($= 4 \cdot 5/10^4$) on the horizontal axis[3].

VI. Error Analysis

Theoretically, the standard FFT-based approach has the advantage of representing functions very precisely (up to Nyquist limitations). Consequently, it does a very good job of estimating convolutions. The discretization of using line segments to represent continuous, smooth functions would be expected to introduce greater errors in such calculations. However, it is important to recognize that the advantage for this approach is in cases where a full sampling of the distribution is not available, in which case any approximation of the shape of the function to be convolved would introduce the same errors in the FFT approach.

Some evaluation of the residual errors resulting from the algorithm were assessed using representations of two Gaussians (mean of 4, standard deviation of 1), the convolution of which is known analytically to be a third Gaussian (mean of 8, standard deviation of $\sqrt{2}$). The inputs were normalized to unit area under the curves, as were the outputs[4]. For well-sampled distributions (each input

[3] One can consider the horizontal axis scale in Fig. 2 as corresponding to 10^{-4} of the quantity $n \cdot m$, since n is always 10^4 in this example.

[4] Note that only the FFT-based result of the convolution required an adjustment. The output of this algorithm is automatically normalized such that the area under the curve is the product of the areas under the input curves.

consisting of 100 points in the analysis that was performed), a plot of the outputs is not informative, as all three outputs (the analytic solution, the FFT-based result, and the line segment convolution result) are indistinguishable by eye at a scale where the entire distribution is represented. The root mean square relative difference for this algorithm was ~0.9%. In comparison, the FFT-based approach resulted in a value of ~0.5%.

For poorly sampled distributions, the line segment approach tended to broaden the convolution just slightly more than did the FFT-based approach, as shown in Fig. 3. Note that what appears to be a noticeable difference in peak values is only an artifact of the sample spacing.

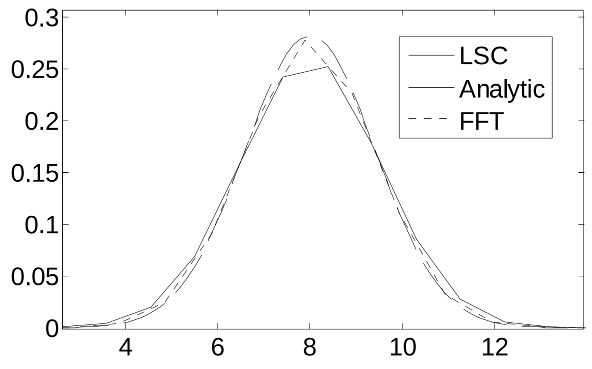

Fig. 3. Comparison of the convolution of poorly sampled inputs to analytic solution

Lastly, it is reasonable to expect that one could reduce any residual errors from this algorithm to be even closer to those realized by the FFT-based approach by using a more robust curve representation strategy than that of straight lines, such as an extension of the basic concept described in this paper that uses higher-order polynomials instead of line segments for estimating the curve between the points.

VII. SUMMARY OBSERVATIONS

It has been shown that for certain applications, the algorithm described in this paper can realize computational savings over traditional convolution approaches. Specifically, these savings occur when traditional approaches require significant upsampling of sparsely known functions, such as might occur from measurement-based probability distributions.

Two key questions that are central in assessing any new algorithm or numerical approach are those of accuracy and computational efficiency. For the former, it has been observed that the linearization of a function does produce some variation from exact solutions, however, those variations are very small for well-sampled functions. Since the primary value of the algorithm is for cases where the functions are not finely sampled, however, future development might look at applying a more sophisticated interpolation scheme, such as cubic (or perhaps even a more generalized polynomial) interpolation, that will still result in a closed-form solution for convolution.

The computational efficiency that has been demonstrated here can also be improved in a variety of ways. In the course of assessing the computational comparison against the more typical FFT-based convolution approach, several ideas for increased efficiency have suggested themselves, such as more complex logic for ordering calculations, or even dynamic generation of the output vector spacing. It seems likely that significant improvement in the algorithm's efficiency could be realized in future iterations.

REFERENCES

[1] W. Hackbusch, "Fast projected convolution of piecewise linear functions on non-equidistant grids," in *From nano to space*, Berlin: Springer, 2008, pp. 145–160.
[2] FFT Java Implementation from the Princeton University CS Department Programming in Java Course, http://www.cs.princeton.edu/introcs/97data/FFT.java.html
[3] General Colley-Tukey FFT Algorithm information from Wikipedia, http://en.wikipedia.org/wiki/Cooley–Tukey_FFT_algorithm