In 1994 Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers

ALGORITHMS FOR SYMBOLIC LINEAR CONVOLUTION

Brian L. Evans

Dept. of Electrical Eng. and Comp. Sciences
211-105 Cory Hall
University of California
Berkeley, CA 94720-1772 USA
E-mail: `ble@eecs.berkeley.edu`

James H. McClellan

School of Electrical and Comp. Eng.
Van Leer Building
Georgia Institute of Technology
Atlanta, GA 30332-0250 USA
E-mail: `mcclella@eedsp.gatech.edu`

Abstract

This paper describes algorithms to perform linear convolution of both piecewise continuous functions and discrete functions symbolically. The algorithms manipulate the piecewise representation of the functions to produce a piecewise representation of the result. In our piecewise representation, the end points of and the function defined over each interval may be symbolic expressions. We have encoded the symbolic linear convolution algorithms in both Mathematica and Maple.

1. Introduction

Linear convolution is a linear time-invariant operator that takes two functions as arguments and produces a new function as its output. In the continuous domain, the linear convolution of two functions, $f(t)$ and $g(t)$, is defined as [1]

$$f(t) \star g(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau \qquad (1)$$

For discrete linear convolution, the integral is replaced by summation [2]

$$f[n] \star g[n] = \sum_{m=-\infty}^{\infty} f[m]\, g[n-m] \qquad (2)$$

The output is computed by accumulating the product of one of the functions with a flipped and shifted version of the other function. In almost all cases, the convolution integral (or summation) has to be performed in pieces. The work is not so much in performing the definite integrations (or definite summations), but in the setting up of the limits of integration (or summation). Once the limits have been determined, a symbolic mathematics environment can perform the integrations (or summations).

This paper extends the work on piecewise convolution algorithms reported by West and McClellan [3]. West and McClellan developed algorithms that perform continuous and discrete linear convolution for finite-extent functions whose piecewise representations are defined over numeric end points. We extend their work to handle infinite-extent functions and functions defined over symbolic end points. We have added our modifications to the Mathematica [4] code written by West and McClellan [3], and we have created an equivalent version for Maple V [5].

In Section 2, we highlight three methods to compute linear convolution. Section 3 discusses our modifications to the algorithm by West and McClellan to support piecewise functions with infinite and symbolic end points. Section 4 discusses the Mathematica and Maple implementations and give examples of convolution problems. Section 5 concludes the paper.

2. Previous Methods

Many methods exist to compute linear convolution. Perhaps the one most often taught is one based on a

B. L. Evans was supported by the Ptolemy project. The Ptolemy project is supported by the Advanced Research Projects Agency and the U.S. Air Force (under the RASSP program, contract F33615-93-C-1317), Semiconductor Research Corporation (project 94-DC-008), National Science Foundation (MIP-9201605), Office of Naval Research (via Naval Research Laboratories), the State of California MICRO program, and the following companies: Bell Northern Research, Dolby, Hitachi, Mentor Graphics, Mitsubishi, NEC, Pacific Bell, Philips, Rockwell, Sony, and Synopsys. World Wide Web information about the Ptolemy project is located at URL `http://ptolemy.eecs.berkeley.edu`. B. L. Evans can be reached by phone at +1 510 643-6686 and by fax at +1 510 642-2739.

J. H. McClellan was supported in part by the Joint Services Electronics Program under contract DAAH-04-93-G-0027. J. H. McClellan can be reached by phone at +1 404 894-8325 and by fax +1 404 894-8363.

graphical approach in which the one function is flipped and slid across the other (Section 2.1). Two other methods, the Square Matrix Rule (Section 2.2) and the Symbolic Convolution Rule (Section 2.3), are algorithmic in nature and require that the functions to be convolved are finite in extent. The rest of this section reviews these three methods.

2.1. Computing Convolutions Graphically

The graphical approach to computing one-dimensional convolutions flips g in time and slides it across f. The answer is obtained by performing the integration or summation over those intervals in which the functions overlap. The functions f and g are most generally described by a piecewise representation that is a union of *function intervals* where a function interval is a function plus its interval of support. The difficulty with the graphical approach is that it requires that the overlapping intervals of the sliding and stationary functions be identified. When the end points are symbolic, setting up the necessary limits on the definite integrations (or summations) requires reasoning about symbolic inequalities.

2.2. The Square Matrix Rule

Zuhao reported the Square Matrix Rule [6] for continuous convolution of two finite-extent functions. The rule automates the setting up of the limits of integration and states that convolving two finite-extent intervals produces four finite-extent intervals. The drawback to this overlap-and-add approach is that it gives four overlapping intervals instead of three non-overlapping intervals, so that its results are harder to interpret and simplify. The Square Matrix Rule, however, is important because it can be used in an algorithm to compute linear convolution.

2.3. The Symbolic Convolution Rule

West and McClellan [3] developed symbolic overlap-and-save convolution algorithms. The key data structure in their algorithms is the representation of piecewise functions as unions of function intervals. Their algorithms perform the following steps:

1. convert the input functions into piecewise representations of unions of function intervals,
2. convolve each function interval in one function with each function interval in the other,
3. implement convolution by adding the results together, and
4. simplify the result so as to minimize the number of intervals.

Interval	Value
$t \in (-\infty, l_f + l_g)$	0
$t \in [l_f + l_g, l_f + u_g)$	$\int_{l_f}^{t-l_g} f(\tau)\, g(t-\tau) d\tau$
$t \in [l_f + u_g, l_g + u_f]$	$\int_{t-u_g}^{t-l_g} f(\tau)\, g(t-\tau) d\tau$
$t \in (u_f + l_g, u_f + u_g]$	$\int_{t-u_g}^{u_f} f(\tau)\, g(t-\tau) d\tau$
$t \in (u_f + u_g, \infty)$	0

(a) continuous-time convolution

Interval	Value
$n \in (-\infty, l_f + l_g - 1]$	0
$n \in [l_f + l_g, l_f + u_g - 1]$	$\sum_{m=l_f}^{n-l_g} f[m]\, g[n-m]$
$n \in [l_f + u_g, l_g + u_f]$	$\sum_{m=n-u_g}^{n-l_g} f[m]\, g[n-m]$
$n \in [u_f + l_g + 1, u_f + u_g]$	$\sum_{m=n-u_g}^{u_f} f[m]\, g[n-m]$
$n \in [u_f + u_g + 1, \infty)$	0

(b) discrete-time convolution

Convolving one function interval $\{f,\ l_f,\ u_f\}$ with another function interval $\{g,\ l_g,\ u_g\}$ generates three intervals with non-zero support if both function intervals are finite. This table assume that the first function interval is longer than the second.

Table 1: Symbolic Convolution Rule for Convolution

For step 2, they developed a rule for convolving two piecewise continuous functions defined over finite-extent function intervals. The rule produces three finite-extent non-overlapping function intervals. They also derive the rule for the discrete case. Their Symbolic Convolution Rules, shown in Table 1, assume that both function intervals have finite extent and that the second interval is shorter than the first function interval. For step 4, they developed strategies to simplify the results of their convolution rules so as to produce a minimum number of non-overlapping intervals. The next section generalizes their work to lift the assumptions on the function intervals.

3. New Symbolic Convolution Rule

Our key modification to the symbolic convolution algorithms reported by West and McClellan [3], which were given in Section 2.3, is the generalization of their Symbolic Convolution Rule. The Symbolic Convolution Rule, given in Table , convolves two function intervals. The functions in the two function intervals may be symbolic formulas but (1) the intervals must be finite in extent, and (2) the second interval must be shorter than the first interval. If the first interval were shorter than the first, then the two function intervals would be swapped. West and McClellan enforce both assumptions by requiring that the end points of the function intervals be numeric, which makes the function intervals finite in extent and gives a convenient way to measure the lengths of the function intervals. Because the end points are finite, convolving two function intervals always produces three non-overlapping intervals having non-zero extent.

3.1. Effect of Infinite End Points

When one or more of the end points is infinite, some or all of the non-zero output function intervals do not appear in the result of the Symbolic Convolution Rule. The "invalid" intervals would either have the same left and right infinite end points or would produce indeterminant results since the underlying arithmetic operations would attempt to add ∞ and $-\infty$. For example, the first non-zero function interval in Table 1a does not exist if $l_f = -\infty$. With $l_f = -\infty$, if $l_g = -\infty$ or $u_g = \infty$, then the calculation of the end points would be indeterminant; otherwise, the interval would have the same left and right infinite end points. Also, with $l_f = -\infty$ already ruled out, the same interval would not exist if $l_g = \infty$ because the computation of the upper limit of the integral $t - l_g$ would be indeterminant since $t \in (-\infty, l_f + u_g]$.

The following conditions on the three output intervals generated by Symbolic Convolution Rule prevent indeterminant arithmetic results from occurring:

- non-zero function interval #1 is valid if $(l_f \neq -\infty)$ and $(l_g \neq -\infty)$
- non-zero function interval #2 is valid if † $(l_g \neq -\infty)$ and $(u_g \neq \infty)$
- non-zero function interval #3 is valid if $(u_f \neq \infty)$ and $(u_g \neq \infty)$

† Interval #2 in the continuous case must also satisfy $l_f + u_g \neq u_f + l_g$ in order to prevent a zero-length interval. This condition is computable if the condition listed above for this interval holds.

3.2. Handling Infinite End Points

The Symbolic Convolution Rule chooses one of two ways to divide the output into three non-overlapping intervals and for each non-overlapping interval, it selects one of two ways to compute the convolution integral (summation). By interchanging the role of f and g in the Symbolic Convolution Rule, we can find the dual rule for choosing the end points of the non-overlapping intervals. To compute the function in each functional interval, we have the freedom to integrate (sum) either $f(\tau)g(t-\tau)$ or $f(t-\tau)g(\tau)$. Therefore, the Symbolic Convolution Rule only represents one of twelve ways to compute overlap-and-add convolution symbolically.

Table 2 shows the two different ways to divide the computation of the output intervals. The Symbolic Convolution Rule uses the order of non-overlapping interval end points $\{l_f + l_g,\ l_f + u_g,\ l_g + u_f,\ u_f + u_g\}$, which is valid if

$$l_f + u_g < l_g + u_f \iff u_g - l_g < u_f - l_f$$

The Dual of the Symbolic Convolution Rule uses the same four end points but swaps the order of the middle two end points: $\{l_f + l_g,\ l_g + u_f,\ l_f + u_g,\ u_f + u_g\}$. An interval is valid if the indeterminant operation $-\infty + \infty$ would not occur when computing its end points and the left end point is less than the right end point.

Table 3 enumerates all twelve possible choices to compute the convolution symbolically and shows what limits of integration (summation) to use in each case. For each interval in Table 3, we generate the alternate limits of integration (summation) by performing the change of variables, $\tau' = t - \tau$. The conditions on the alternate form are less restrictive in four of the six cases and identical in the other two. Therefore, we will use the alternate form in our implementation.

Non-Zero Interval	Valid if
$[l_f + l_g, l_f + u_g)$	$(l_f \neq -\infty)$
$[l_f + u_g, l_g + u_f]$	$(l_g \neq -\infty$ and $u_g \neq \infty)$
$(l_g + u_f, u_f + u_g]$	$(u_f \neq \infty)$

(a) Symbolic Convolution Rule

Non-Zero Interval	Valid if
$[l_f + l_g, l_g + u_f)$	$(l_g \neq -\infty)$
$[l_g + u_f, l_f + u_g]$	$(l_f \neq -\infty)$ and $(u_g \neq -\infty)$
$(l_f + u_g, u_g + u_f]$	$(u_g \neq \infty)$

(b) Dual Symbolic Convolution Rule

Table 2: Avoiding Indeterminant End Point Calculations

Interval	Limits	Formula
1. $[l_f + l_g,$ $l_f + u_g)$	$l_f \ \ldots\ t - l_g$ if $(l_f \neq -\infty)$ and $(l_g \neq -\infty)$	$f(\tau)\,g(t-\tau)$
	OR	
	$l_g \ \ldots\ t - l_f$ if $(l_f \neq -\infty)$	$f(t-\tau)\,g(\tau)$
2. $[l_f + u_g,$ $l_g + u_f]$	$t - u_g \ \ldots\ t - l_g$ if $(l_g \neq -\infty)$ and $(u_g \neq \infty)\dagger$	$f(\tau)\,g(t-\tau)$
	OR	
	$l_g \ \ldots\ u_g$ if $(l_g \neq -\infty)$ and $(u_g \neq \infty)\dagger$	$f(t-\tau)\,g(\tau)$
3. $(l_g + u_f,$ $u_f + u_g]$	$t - u_g \ \ldots\ u_f$ if $(u_f \neq \infty)$ and $(u_g \neq \infty)$	$f(\tau)\,g(t-\tau)$
	OR	
	$t - u_f \ \ldots\ u_g$ if $(u_f \neq \infty)$	$f(t-\tau)\,g(\tau)$

$\dagger$ and $l_f + u_g < l_g + u_f \Longleftrightarrow u_g - l_g < u_f - l_f$

(a) Extended Symbolic Convolution Rule

Interval	Limits	Formula
1. $[l_f + l_g,$ $l_g + u_f)$	$l_g \ \ldots\ t - l_f$ if $(l_f \neq -\infty)$ and $(l_g \neq -\infty)$	$f(t-\tau)\,g(\tau)$
	OR	
	$l_f \ \ldots\ t - l_g$ if $(l_g \neq -\infty)$	$f(\tau)\,g(t-\tau)$
2. $[l_g + u_f,$ $l_f + u_g]$	$t - u_f \ \ldots\ t - l_f$ if $(l_f \neq -\infty)$ and $(u_f \neq \infty)\ddagger$	$f(t-\tau)\,g(\tau)$
	OR	
	$l_f \ \ldots\ u_f$ if $(l_f \neq -\infty)$ and $(u_f \neq \infty)\ddagger$	$f(\tau)\,g(t-\tau)$
3. $(l_f + u_g,$ $u_f + u_g]$	$t - u_f \ \ldots\ u_g$ if $(u_f \neq \infty)$ and $(u_g \neq \infty)$	$f(t-\tau)\,g(\tau)$
	OR	
	$t - u_g \ \ldots\ u_f$ if $(u_g \neq \infty)$	$f(\tau)\,g(t-\tau)$

$\ddagger$ and $l_f + u_g > l_g + u_f \Longleftrightarrow u_g - l_g > u_f - l_f$

(b) Extended Dual Symbolic Convolution Rule

Table 3: The Generalized Symbolic Convolution Rule

For the Rule and its dual, we enumerate all possible cases of end points in Table 4. End points to be either finite or infinite in value. Left end points can be equal to $-\infty$ or a finite value, and right end points can be equal to ∞ or a finite value. Since two function intervals contain four end points and each end point can take on one of two possible types of values, there are $2 \cdot 2 \cdot 2 \cdot 2 = 16$ possible permutations of the Symbolic Convolution Rule. In the general case, then, convolving two function intervals yields one, two, or three non-overlapping function intervals of non-zero extent, whereas the Symbolic Convolution Rule always produces exactly three.

	EndPoints				Extended S.C. Rule Intervals			Extended Dual Rule Intervals		
No.	l_f	u_f	l_g	u_g	1	2	3	1	2	3
0	$-\infty$	F	$-\infty$	F			V			V
1	$-\infty$	F	$-\infty$	∞			V			
2	$-\infty$	F	F	F		V	V	$*$		$*$
3	$-\infty$	F	F	∞			V	V		
4	$-\infty$	∞	$-\infty$	F						V
5	$-\infty$	∞	$-\infty$	∞						
6	$-\infty$	∞	F	F		V		$*$		$*$
7	$-\infty$	∞	F	∞				V		
8	F	F	$-\infty$	F	$*$		$*$		V	V
9	F	F	$-\infty$	∞	$*$		$*$		V	
10	F	F	F	F	V	V	V	V	V	V
11	F	F	F	∞	$*$		$*$	V	V	
12	F	∞	$-\infty$	F	V					V
13	F	∞	$-\infty$	∞	V					
14	F	∞	F	F	V	V		$*$		$*$
15	F	∞	F	∞	V			V		

Notation follows Table 3 and

- F represents a finite end point
- V means a valid functional interval
- $*$ means the interval has valid end points but it overlaps with the other one produced by the Rule

Table 4: Analysis of the Alternate Form of the Generalized Symbolic Convolution Rule

3.3. Algorithm

We next develop an algorithm to perform the convolution of two functional intervals. The algorithm must be deterministic in that it returns the same result for $f_1 \star f_2$ as it does for $f_2 \star f_1$, given that f_1 and f_2 are any two valid function intervals. The problem is that we have two different rules for handling the 16 different cases given in Table 4. For most cases, only one of the two rules works (we will use the alternate form of the Rules as given in Table 3). The Extended Symbolic Convolution Rule would be used in cases 1, 2, 6, 13, and 14. The Extended Dual Symbolic Convolution Rule would be used in cases 4, 7, 8, 9, and 11. Case 5, in which both function intervals are two-sided infinite in extent, would be handled by the definition given by equation (1) or (2). For case 0 and case 15, either rule would give the correct output function interval. That leaves cases 3, 10, and 12.

In cases 3 and 12, each rule gives a different piece of the answer. The correct answer is obtained by combining the results of both rules. For example, case 3 becomes

$$\text{for } t \in (-\infty, l_g + u_f), \int_{l_f}^{t-l_g} f(\tau) g(t-\tau) d\tau$$
$$\text{(Dual Rule)}$$
$$\text{for } t \in (l_g + u_f, \infty) \int_{t-u_f}^{u_g} f(t-\tau) g(\tau) d\tau$$
$$\text{(S.C. Rule)}$$

and case 12 becomes

$$\text{for } t \in (-\infty, l_f + u_g), \int_{l_g}^{t-l_f} f(t-\tau)\, g(\tau)\, d\tau$$
$$\text{(S.C. Rule)}$$
$$\text{for } t \in (l_f + u_g, \infty), \int_{t-u_g}^{u_f} f(\tau)\, g(t-\tau)\, d\tau$$
$$\text{(Dual Rule)}$$

All of the cases except 10 have been handled. In case 10, both input function intervals are finite in extent, and the choice of which rule to apply depends on the relationship between $l_f + u_g$ and $l_g + u_f$ or equivalently between $u_g - l_g$ and $u_f - l_f$. That is, if the interval (l_f, u_f) is longer than (l_g, u_g), then use the Extended Symbolic Convolution Rule; otherwise, use the Extended Dual Symbolic Convolution Rule. If any of the end points are symbolic, then the result of case 10 may be indeterminant. However, we can handle indeterminancy in an implementation by returning the output function intervals of both rules. To enforce which set of function intervals is correct, we scale all of the *functions* generated by the Extended Symbolic Convolution Rule by the factor $u_{-1}([u_f - l_f] - [u_g - l_g])$, which gives 1 if $u_f - l_f > u_g - l_g$, and scale all of the *functions* generated by the Dual Symbolic Convolution Rule by the factor $1 - u_{-1}([u_f - l_f] - [u_g - l_g])$.

4. Implementation in Mathematica and Maple

West and McClellan implemented their piecewise convolution algorithms in Mathematica. Their implementation manipulated signals represented as unions of functions intervals in which the functions were symbolic formulas and the end points were numeric. We have generalized their implementation to work with infinite end points and symbolic end points, according to the strategy given in the previous section.

The Mathematica implementation is part of the Signal Processing Packages (SPPs) [7, 8]. The convolution routines in Version 2.9.5 of the SPPs (see the end of the paper for FTP instructions) support infinite end points, but not symbolic end points. The complete implementation will be available in version 3.0 of the SPPs due for commercial release in January of 1995. The release of version 2 of the SPPs contains several tutorial notebooks, including one on convolution [7, 8].

The rest of this section will show examples of the convolution implementation in version 2.9.5 of the SPPs and an equivalent Maple V implementation. We use the same function names in both implementations. To support convolution, we have defined common piecewise functions such as the Kronecker impulse function `Impulse`, the discrete pulse function `Pulse` and its continuous counterpart `CPulse`, and the discrete step function `Step`. For historical reasons, the SPPs define their own continuous step function and Dirac delta function called `CStep` and `Delta`.

A function interval is defined as a list of three elements: a function, a left end point, and a right end point. A piecewise function is represented as a union of function intervals, i.e., a list of function intervals. The primary continuous/discrete piecewise convolution routines are called `CT/DTPiecewiseConvolution`, respectively. Functions whose names do not imply a domain such as `PiecewiseToFormula` support a `Domain` option.

4.1. Mathematica Version

The linear convolution of a finite sequence and a discrete step function is computed below. The finite sequence has a value of a for $n \in [-3, -1]$, a value of b for $n \in [0, 3]$, and a value of 0 otherwise. Note that the functions in the finite sequence are symbolic quantities and that the discrete step function extends for $n \in [0, \infty]$. (In the context of linear systems theory, we are symbolically computing the step response of a non-causal finite impulse response filter.) The Mathematica dialogue follows, but the initialization of the SPPs is not shown.

```
In[2]:= twoPulses = {{a, -3, -1}, {b, 0, 3}};

In[3]:= DTPiecewiseConvolution[
          twoPulses, Step[n], n ]

Out[3]= {{a (4 + n), -3, -1},
>        {3 a + b + b n, 0, 2},
>        {3 a + 4 b, 3, Infinity}}

In[4]:= PiecewiseToFormula[
             conv, n, Domain -> Discrete ]

Out[4]= a (4 + n) Pulse [3 + n] +
                           3
>       (3 a + b + b n) Pulse [n] +
                                 3
>       (3 a + 4 b) Step[-3 + n]
```

In the Mathematica implementation of the convolution routines, we have added a `Dialogue` option. When enabled, the option will trigger an animation of the "flip-and-slide" approach to computing the convolution. Before the animation frames are generated, all function intervals containing symbolic quantities are removed, and each infinite end point is replaced by a number that is large enough in magnitude to be considered "off the screen".

4.2. Maple Version

Our Maple V implementation is a set of standalone packages. In the following Maple dialogue, the first command computes the linear convolution of two continuous step functions of different symbolic lengths. The first step function extends for $t \in [0, t_1]$ and the second extends for $t \in [0, t_2]$. The second Maple command symbolically computes the autocorrelation of a discrete pulse function centered at the origin with length $2\,a$ (the result is a trapezoid).

```
> CTPiecewiseConvolution(
      CPulse(t1, t), CPulse(t2, t), t );

  [[t, 0, t2],
   [t2, t2, t1],
   [-t + t1 + t2, t1, t1 + t2]]

> AutoCorrelation(
      [[1, -a, a]], n, Domain = Discrete );

  [[n + 2 a + 1, -2 a, -1],
   [2 a + 1, 0, 0],
   [2 a + 1 - n, 1, 2 a]]
```

5. Conclusion

This paper describes algorithms to compute linear discrete and continuous convolution symbolically. The algorithms work on the piecewise representations of the input functions and return a piecewise representation of the result. With our new rules, the piecewise representations may contain symbolic and infinite end points. We have implemented the algorithms in two symbolic mathematics environments, Mathematica and Maple.

Convolution is a fundamental operation in the study of linear operators and linear systems. In the context of linear systems, one of the input functions to convolution (called the convolution kernel) is typically the response of a linear operator an impulse function. Certain classes of convolution kernels (defined symbolically) model certain kinds of physical phenomena. Symbolic convolution algorithms are important in studying the effects that a system (represented by a class of convolution kernels) has on a family of input functions.

Our convolution algorithms are part of the Signal Processing Packages (SPPs). Version 2.9.5 of the SPPs for Mathematica is available from the FTP sites `gauss.eedsp.gatech.edu` (IP #130.207.226.24) in the directory `pub/Mathematica`, and `mathsource.wri.com` (IP #140.177.10.6) in the directory `0202-240`.

6. References

[1] A. V. Oppenheim and A. Willsky, *Signals and Systems*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

[2] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[3] K. A. West and J. H. McClellan, "Symbolic convolution," *IEEE Trans. on Education*, vol. 36, pp. 386–393, Nov. 1993.

[4] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer*. Redwood City, CA: Addison-Wesley, second ed., 1991.

[5] B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt, *First Leaves: A Tutorial Introduction to Maple V*. New York: Springer-Verlag, 1992.

[6] Z. Zuhao, "The square matrix rule of the convolution integral," *IEEE Trans. on Education*, vol. 33, pp. 369–372, Nov. 1990.

[7] B. L. Evans, L. J. Karam, K. A. West, and J. H. McClellan, "Learning signals and systems with Mathematica," *IEEE Trans. on Education*, vol. 36, pp. 72–78, Feb. 1993.

[8] B. L. Evans, J. H. McClellan, and H. J. Trussell, "Investigating signal processing theory with Mathematica," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. I, (Minneapolis, MN), pp. 12–15, Apr. 1993.