Median Filtering by Threshold Decomposition: Induction Proof

by
Connor Bramham, Brandon Mayle, Maura Gallagher, Douglas Magruder, and Cooper Reep
Math Club
Socrates Preparatory School
www.socprep.org
connor.bramham@socprep.org
Sponsors: Saul Rubin and Neal C. Gallagher, Ph.D,
Socrates Preparatory School, 3955 Red Bug Lake Rd., Casselberry, FL 32707
saul.rubin@socprep.org
neal.gallagher@socprep.com

Abstract

In building a robot for the FTC competition, our team needed to remove motor noise from our sensor signals. So we settled on using a median filter because of the medians superior removal of impulsive noise. For us, however, the foundational publications that describe these filters were challenging to understand. Having learned the concept of proof by induction from the MIT OpenCourseWare course, "Mathematics for Computer Science" (MIT Course Number 6.042J / 18.062J), we developed an original proof for the principle of median filter threshold decomposition in order to better understand their operation. The induction is over the number of quantized threshold levels for the sequence of input values as applied to both the standard and recursive median filter.

I Introduction

Many of this paper's authors participated in the 2017 First Tech Challenge robotics competition. Our completed robot contained many motors and sensors, including ultrasound sensors for distance measurements, gyroscopic sensors for direction measurements, color sensors to sense the brightness and color of beacons, and more. The robot's electric motors generated a considerable amount of noise that found its way onto the signals from our sensors to our robot controller. One method used to address the noise problem is to use an aluminum sheet to shield robot electronics from the drive motors and servos, but more was needed. It was decided to add the software solution of using median filters to remove the impulsive motor noise.

Simultaneously with building the robot, some Math Club students were taking the MIT OpenCourseWare course on Mathematics for Computer Science. Among other things, the course covered proof by induction and closely related recursive algorithms. The median filters worked very well on our robot, but the reason for this was not easy to understand upon reading the references. It was suggested to us that it might be possible to use proof by induction to develop easier to understand proofs for the key median filter results. Hence we have this paper.

Median filtering of sequences and images is a common method of reducing the effects of noise. Software such as Adobe Photoshop®, Microsoft Excel®, and MathWorks Matlab® have had a median filtering option for many years. The median filter of a one dimensional sequence is produced by sliding a window across the sequence and computing the filter's output as the median of the sequence samples found in the window. If the sample values in a size five window are (*1,5,2,4,2)*, the computed median output is (*2*). If the window contains *2N+1* samples for integer $N > 0$, the median output is associated with the time index located at the window center. For the median, *N+1* of the values in the window are greater than or equal to the median and *N+1* of the values are less than or equal to the median. If the window contains an odd number of samples, the median is unique. This property is better illustrated in Figure 1, in which a sample input signal is median filtered with an *N=1* size window. Note that N values are appended to the beginning and end of the signal in order to apply the median filter to the first and last sample values without changing them. The values in the highlighted window are *(3,7,1)*, and their computed median is 3. In this case, *N+1* values (3 and 7) are greater than or equal to the median, and *N+1* values (1 and 3) are less than or equal to the median.

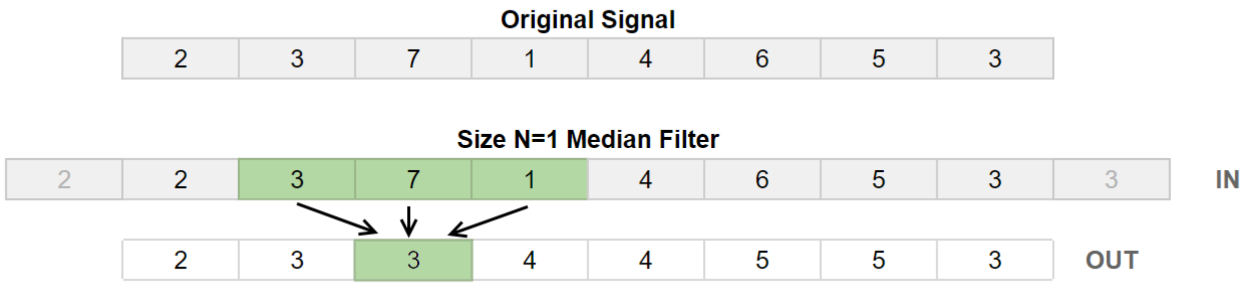


Figure 1: Demonstrating the process of computing the median of a sequence, using a randomly generated sample signal as the input. The window being operated on by the median filter is highlighted in green, and the bottom row depicts the output of the filter, which is the median of the values in the window as it slides across the original signal from left to right. The values appended at the beginning and end of the input signal are grayed out.

The most important property of median filters is that they remove impulsive noise while preserving edge information. Median filters can be found in proprietary medical imaging systems as well as many photo-copying machines to mention a few applications [5]. The operation of a median filter with a window containing *2N+1* samples is as shown:

$$y(m) = median[x(m-N),\dots,x(m-1),x(m),\dots x(m+N)] \qquad (1)$$

The median filter was popularized by Tukey's book Exploratory Data Analysis [8], where Tukey discussed properties of the median that set it apart from traditional linear filtering. The theoretical underpinnings of the median filter began with the cumbersome analysis of Gallagher and Wise [3] where it was proven, among other things, that a repeated application of a median filter to the output of a prior median filter will in succession produce a root sequence that is invariant to additional median filter passes. A following paper by Nodes and Gallagher introduced the concept of a recursive median filter and proved that a root sequence is obtained by only a single pass of the recursive filter [6]. After these early publications, a large community of researchers began analyzing and extending the median concept, notably at Purdue University as well as Tampere University of Technology in Finland. Much of this work is summarized in Arce's text [1]. This paper is about a fundamental property of median filters called threshold decomposition.

II Example of median filtering using a sample Cauchy sequence

The original data from the FTC robot is no longer available, so in order to show the median filter's power to remove spiky noise from the sensor signal sequences, an impulsive noise sequence is modeled as a set of independent, identically distributed samples with the Cauchy probability distribution:

$$F(x) = \frac{1}{\pi}arctan(x) + \frac{1}{2}.$$

Figure 2 shows this Cauchy noise sequence filtered by a window size 5 median filter. Figure 3 shows the same Cauchy sequence filtered by a recursive median filter. To clearly see the difference between the two results, they are plotted together in Figure 4. To increase the level of smoothing, the window size can be increased. Some sensor's output can be sampled as high as 2000 samples/sec while other sensors can be sampled no higher than 20 samples/sec. The larger the sampling rate the larger can be the filter's window size. For our color sensors, window sizes on the order of 21 samples where used. This paper is not about how to design the window size, but to provide a new proof for the important property of threshold decomposition.

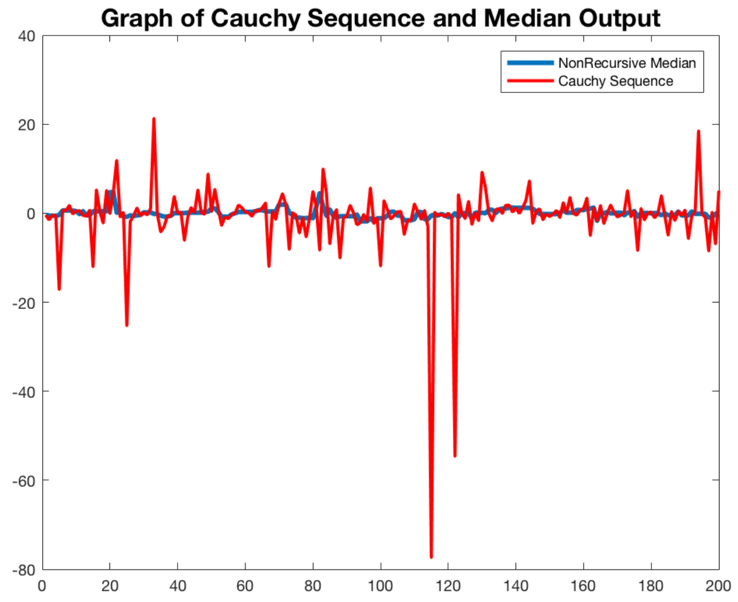


Figure 2: A sequence of 200 Cauchy random values generated in Matlab. Using a window size 5 the sequence is median filtered to produce the sequence in blue.

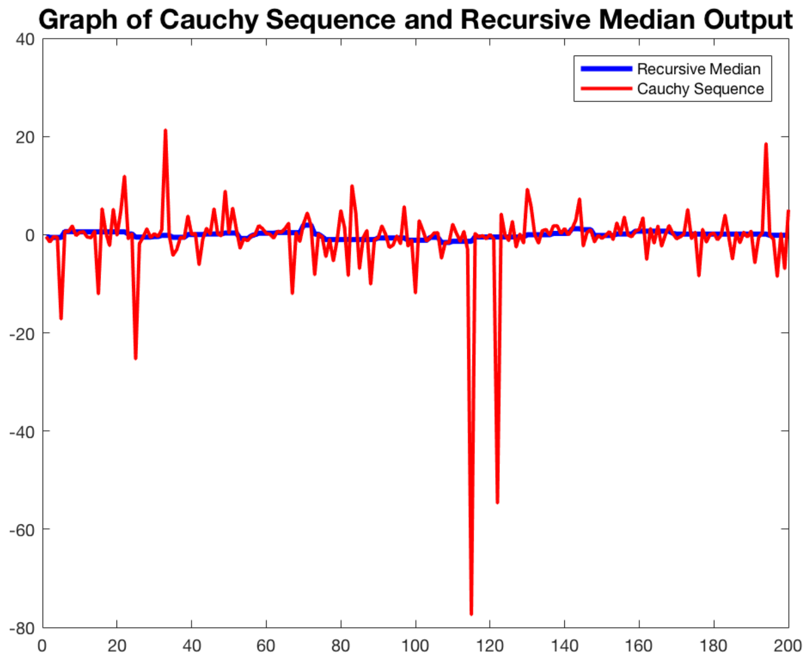


Figure 3: In red we have the same sequence of 200 Cauchy random values as shown in Figure 2. Using a window size 5 recursive median filter the sequence in blue is produced.

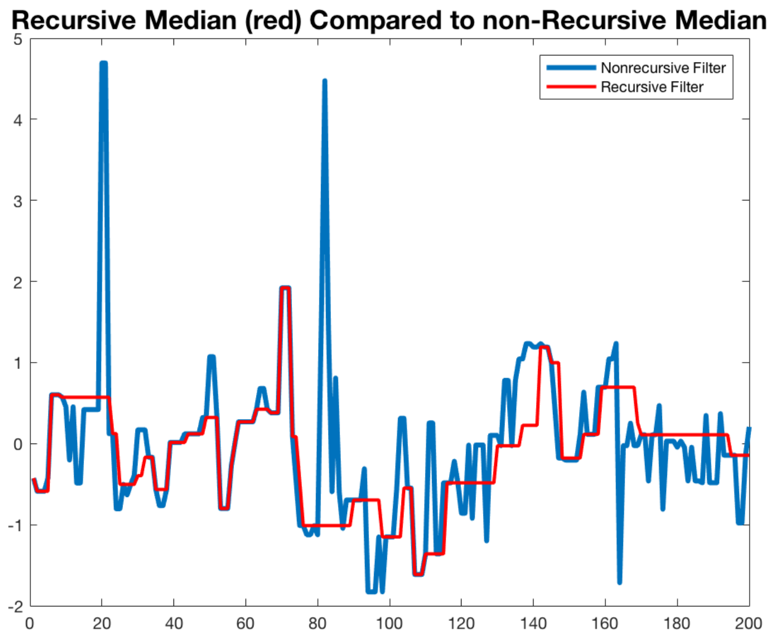


Figure 4: The same median filter sequence generated in Figure 2 is shown in blue compared to the recursive median filtered sequence of Figure 3 in red.

As discussed in section III, linear filters obey the superposition property that gives rise to the impulse response. Median filters do not. Median filters do not have any property analogous to that of the impulse response for a linear filter. However, medians do obey a weak form of superposition called threshold decomposition as was proven by Fitch, Coyle, and Gallagher [2].

This paper contains an alternative, original inductive proof of threshold decomposition for median filters in section IV. The threshold decomposition applied to the recursive median is discussed in section V.

III Linear filters vs. median filters

One of the median filter's potential difficulties is that the basic property of superposition does not hold. For example, consider $\{y_1(k), k= 0,1,2,\ldots\}$ as the output of a median filter with input $\{x_1(k), k =0,1,2,\ldots\}$ and $\{y_2(k), k= 0,1,2,\ldots\}$ as the output of the same filter for input $\{x_2(k), k =0,1,2,\ldots\}$. Unlike linear filters, for the median, it is not true that the input $\{\alpha x_1(k) + \beta x_2(k)\}$ produces output $\{\alpha y_1(k) + \beta y_2(k)\}$ for any constants α and β. This property, known as the linearity property, is what gives rise to the linear filter's impulse response, $\{h(k)\}$ which is the filter's response to a pulse of value "1" embedded in zeros such as $\{\ldots 0,0,1,0,0\ldots\}$. If the input is $\{x(k), k= \ldots,-1,0,1,\ldots\}$, each time shifted input sample value $x(k-p)$ adds a time shifted and scaled version of the filter's impulse response $\{h(k-p)\}$ at the output [7]. Figure 5 illustrates the input and output for a linear filter using the impulse response. The unit pulse (impulse) response for the linear filter is $h(k)$. When the input sequence $x(k)$ is divided into the sum of impulses shifted in time and scaled by constants, the output can be written as a similarly shifted and scaled sum of the resultant impulse responses. This time-shifted superposition gives rise to the very important concept of the linear filter's impulse response as the basis for analyzing these filters. This then leads to the concept of a filter's transfer function that represents the filter's frequency response. It is not an understatement to say that almost all analysis of linear filters is based on this pair of concepts. However, time shifting is not the only way to decompose the input of a linear filter by use of superposition. Another method, rarely used or even discussed, is to decompose a sequence by thresholding the input and representing the original input by summing the threshold values. Thresholding could be thought of as mapping the values of an input function to an output of 2 possible values. The input value is then compared to a value called the threshold level. For us, the output values can be 0 or 1, where the value is 0 if the input value is below the threshold level, and 1 if the input value is above the threshold level. Examine Figure 6.

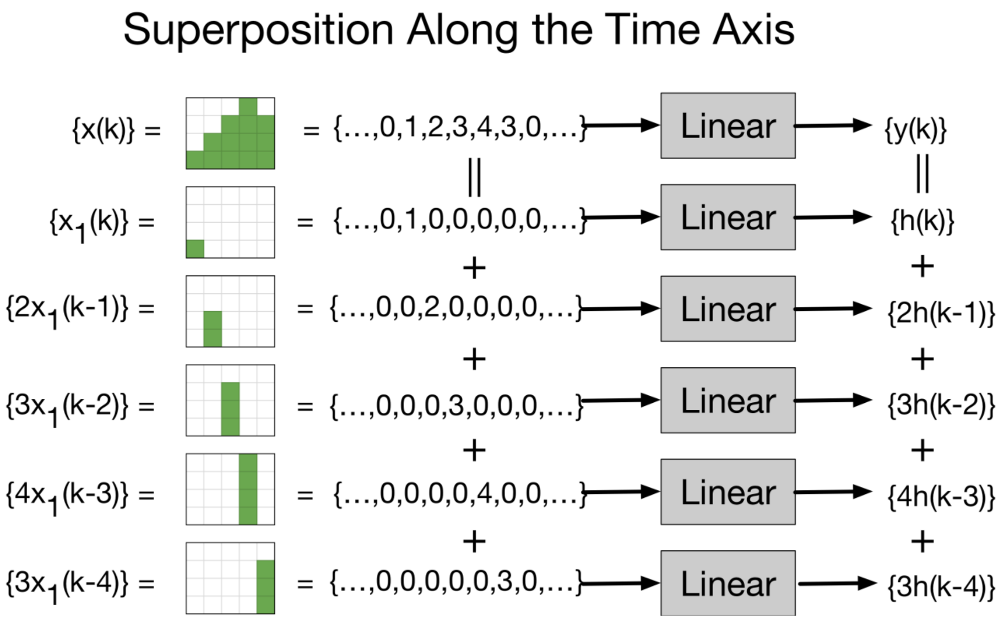


Figure 5: Use of superposition to represent the output of a linear filter as a scaled and shifted sum of copies of the filter's impulse response.

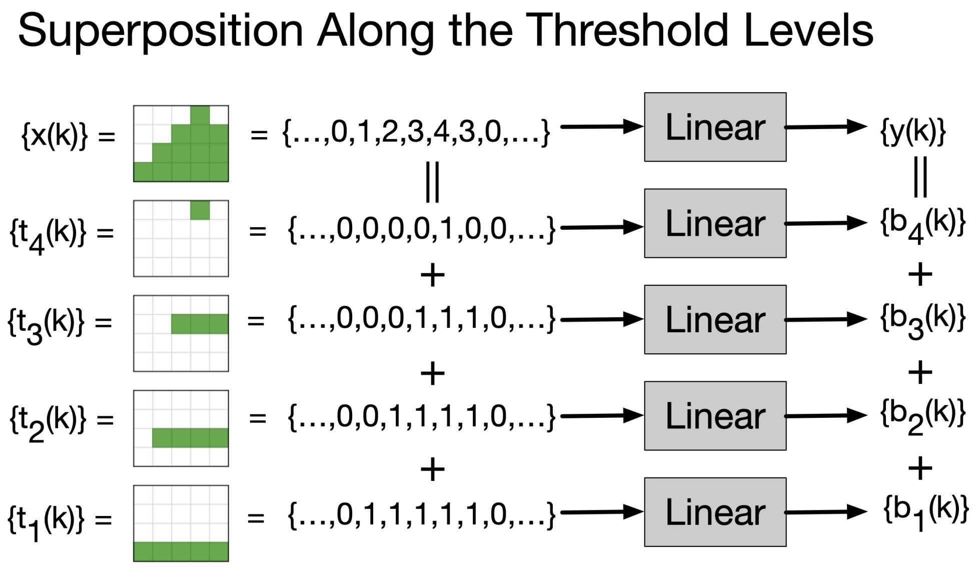


Figure 6: The input sequence {…,0,1,2,3,4,3,0,…} is thresholded to produce {$t_1(k)$}, through {$t_4(k)$}, where $t_m(k) = 0$ if $x(k) < m$ and $t_m(k) = 1$ if $x(k) \geq m$.

As an example of how superposition works, consider the following difference equation that relates a linear filter's output $\{y(k)\}$ to the input $\{x(k)\}$:

$$y(k) = -(0.25)y(k-1) + (0.375)y(k-2) + (0.4)x(k) + (0.3)x(k-1) + (0.3)x(k-2). \qquad (2)$$

The filter's impulse response can be determined by using the input $\{\ldots,0,1,0,\ldots\}$. If the input is a single sample value of one surrounded on either side by zeros; it is called an impulse. The impulse response for equation (2) is graphed in Figure 7.

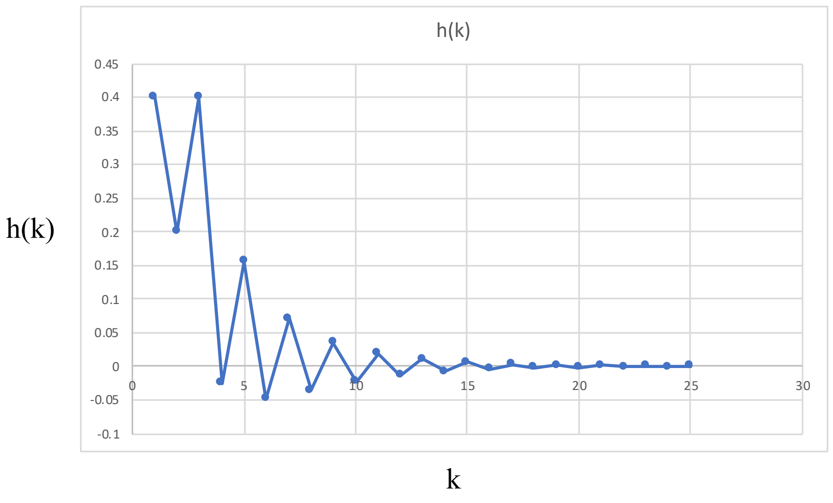


Figure 7: Impulse response, h(k) for the digital filter with difference equation found in Eq. (2).

The input sequence below is the same as the sequence $\{x(k)\}$ in Figure 5 and Figure 6:

$$\{x(1) = 1, x(2) = 2, x(3) = 3, x(4) = 4, x(5) = 3\}, \textit{ all other values } x(k) = 0.$$

This integer sequence is thresholded at four levels to create the thresholded sequences found in Figure 6 of $\{t_1(k)\}$ through $\{t_4(k)\}$ as follows:

$$t_i(x) = \begin{cases} 1, & \text{if } x(k) \geq i,\ 1 < i \leq k \\ 0, & otherwise. \end{cases}$$

Examination of Figure 6 reveals a basic property of thresholding we refer to as the stacking property:

$$(t_m(k) = 1) \implies (t_n(k) = 1),\ n < m\,.$$

To demonstrate this property, the input sequence is decomposed into a sum of binary thresholded sequences each with levels (0, 1). If a threshold value is a '1' at any time index, all the lower threshold values at the same time index are also equal to a '1'. Additionally, if a thresholded value is '0', all the above values at the same time index are also '0'. The threshold sequences may be viewed as a stack of zeros atop a stack of ones, hence the term stacking property. The sum of the thresholded sequences equals the original sequence as illustrated in Figure 6. If we linear filter each threshold sequence using the same filter as expressed in Eq. (2), as shown in Figure 8, the sum of the resulting outputs equals the original output with the multivalued input.

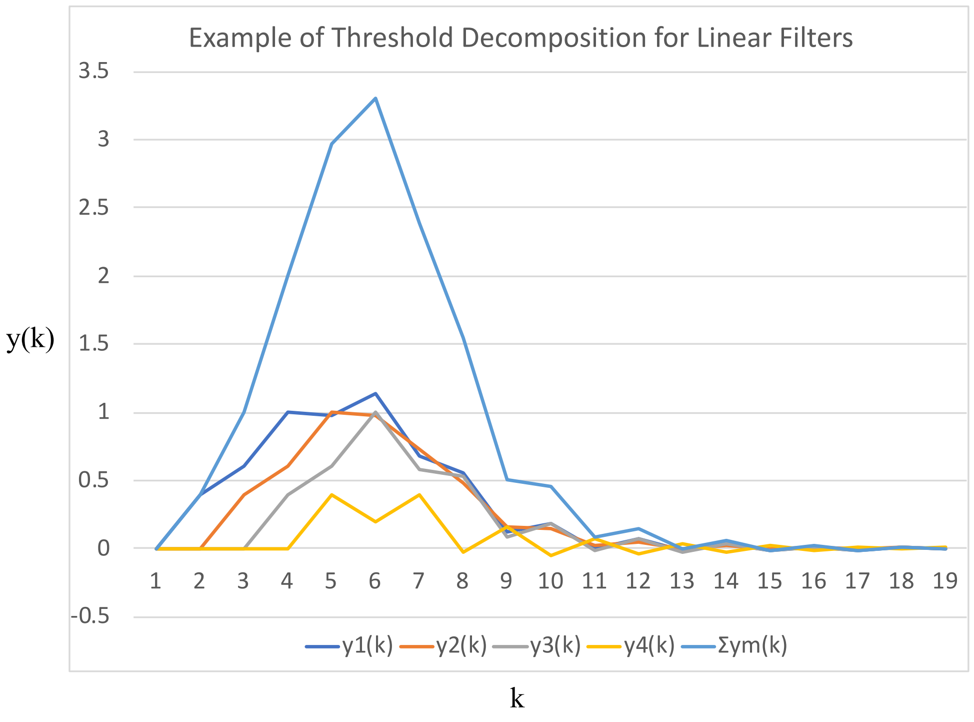


Figure 8: Each threshold level in Figure 6 has an output denoted as $y_1(k)$, …, $y_4(k)$. The sum of these outputs is denoted by $\Sigma y_m(k)$, the top blue plot, which is the same as $y(k)$, in Figure 6.

Before describing in detail how median filters operate, it is informative to use examples to show how they differ from linear filters. Two examples show how median filters respond differently than linear filters. The first example illustrates how a linear filter and a median filter respond differently to a noise spike. This can be seen in Figure 9. The blue curve shows a linear filter response to a random input with an added noise impulse at a single input sample value near the center of the graph. The impulse, smeared by the filter's impulse response, oscillates over the filter output. This oscillation response is sometimes called "ringing" of the filter. Similarly, the red line shows the output of a window size 5 median filter to the same sequence. The medan filter appears to completely remove the noise spike with no ringing.

The second example contains a random sequence with a step function of height one in the middle of the graph. See Figure 10. The red curve is the median and the blue curve is the linear filter response. The median preserves the sharp edge while the linear filter smears the edge due to the impulse response.

The fact that superposition applies to linear filters makes their analysis rich and varied. The Fourier transform and z-transform [7] are very powerful tools that facilitate the analysis of linear filters in either the time domain or frequency domain. For linear filters, the most common way to decompose a sequence is along the time axis as shown in Figure 5.

The discovery of threshold decomposition for median and related filters opened a new tool box for analyzing these filters [2]. As stated in section I, threshold decomposition was proposed by Fitch, Coyle, and Gallagher. Threshold decomposition is a weak form of superposition. Unlike the theory of linear filters, the theory of median filters is not usually treated to any degree in text books. Median theory is based on an application of symbolic logic tools rather than the orthogonal expansions and Hilbert space methods used for linear digital filters.

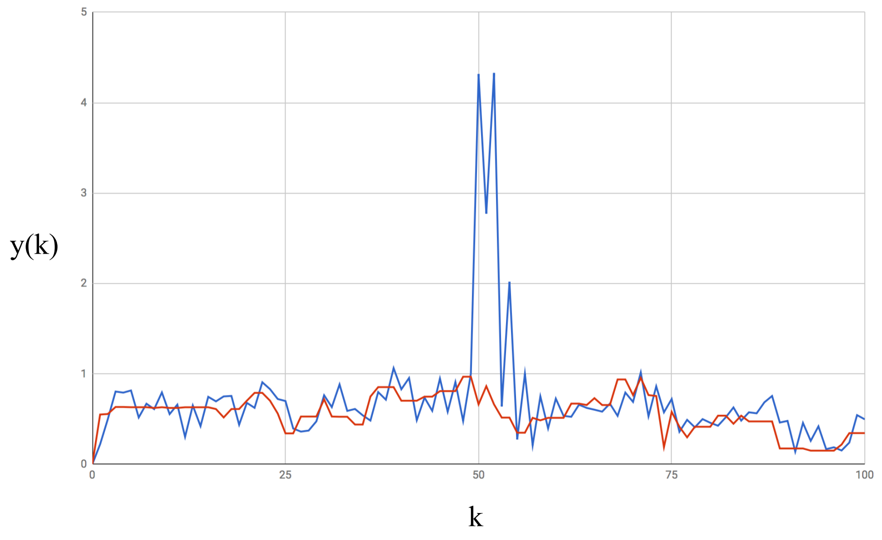


Figure 9: Typical linear filter response to a noise spike (blue) compared to the response of a median filter (red). The linear filter produces a spike in the output while the median removes the impulse.

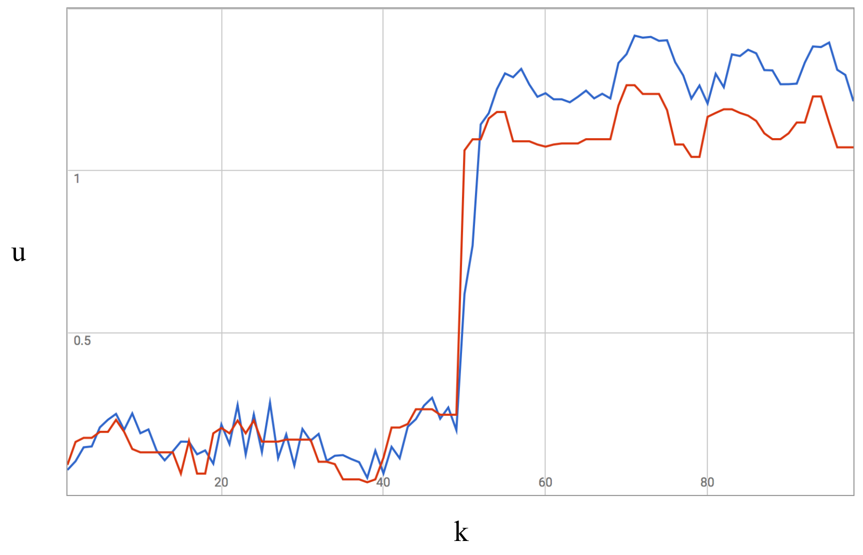


Figure 10: Response of a linear filter (blue) compared to the response of a median filter (red) for a step change in the input sequence.

While median filters do not obey the superposition property of linear filters, they do satisfy this lemma not satisfied by linear filters.

Lemma 1: *Median filters commute with any non-increasing or non-decreasing operator* g(). *Consider the real-valued function* g(), *where*

$$g(a) \leq g(b), \text{ when } a < b, \text{ and } a, b \in \mathbb{R}$$
$$\text{or}$$
$$g(a) \geq g(b), \text{ when } a < b, \text{ and } a, b \in \mathbb{R}.$$

Proof: *Now consider a median window that contains a set of 2N+1 samples, an odd number of sample values. The median is the value m such that m ≥ the value of N+1 samples, and m ≤ the value of N+1* *samples. Form the image of this set by evaluating g() for each element of the set. Now we have the image set consisting of 2N+1 values of g() for each value in the median window. It must be true that g(m) ≥ N+1 of these values and g(m) ≤ N+1 of these values. Thus g(m) is the median of the image set. This means that we can interchange the order of the median and the operator* g(), *or again the median of the image set is the image of the median m. So the median operation commutes with function g().* ■

Consider the stacking property. Using the transitive property of inequalities, we have proven the stacking property for the set of thresholded sequences repeated below for convenience:

$$(t_i(m) = 1)) \Rightarrow (t_j(m) = 1), \forall j < i, \text{and}$$
$$(t_i(m) = 0)) \Rightarrow (t_j(m) = 0), \forall j > i\,.$$

The stack sequences are unique. For each signal value $x(m)$ there is one and only one set of stack values $\{t_i(m),\ i = 1,2\ldots,\ k\}$. Proof by contradiction is straightforward and found in the Appendix.

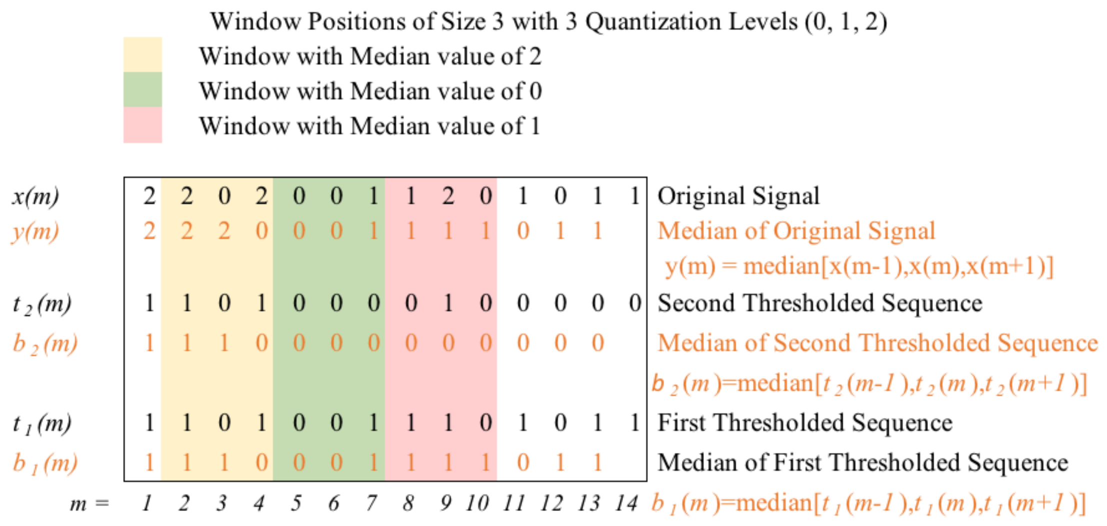
Define the median filter of the thresholded sequence $\{t_i(m)\}$ using window size $(2N+1)$ to be $b_i(m)$ such that

$$b_i(m) = \text{median}\{t_i(m\text{-}N),\ldots,\ t_i(m),\ldots,\ t_i(m\text{+}N)\}.$$

Lemma 2: *The stacking property holds for the computed median sequences {b$_i$(m)}. This means*
(a) *If $b_i(m) = 1$, then $b_j(m) = 1$, if $j < i$, and*
(b) *If $b_i(m) = 0$, then $b_j(m) = 0$, if $j > i$.*

Proof: *Proof of (a): Consider the median filtered sequence with integer values m and i such that $b_i(m) = 1$. We know that in order for the median value $b_i(m)$ to be 1, there must be at least (N+1) ones in the window. Because we know that the stacking property holds for $t_i(m)$, as we move the same sample window to lower threshold levels down the stack, it can only gain more ones inside the window and thus the median values $b_j(m)$ in all related time windows at lower thresholds ($j < i$) must remain a '1'. Proof of (b): follows in a similar fashion.* ■

The relatively simple case for two threshold levels is illustrated in Figure 11. This will be the base case for the proof by induction discussed in section I.



Window Positions of Size 3 with 3 Quantization Levels (0, 1, 2)

🟨 Window with Median value of 2
🟩 Window with Median value of 0
🟥 Window with Median value of 1

x(m)	2	2	0	2	0	0	1	1	2	0	1	0	1	1	Original Signal
y(m)	2	2	2	0	0	0	1	1	1	1	0	1	1		Median of Original Signal
															y(m) = median[x(m-1),x(m),x(m+1)]
$t_2(m)$	1	1	0	1	0	0	0	0	1	0	0	0	0	0	Second Thresholded Sequence
$b_2(m)$	1	1	1	0	0	0	0	0	0	0	0	0	0		Median of Second Thresholded Sequence
															$b_2(m)$=median[$t_2(m\text{-}1)$,$t_2(m)$,$t_2(m\text{+}1)$]
$t_1(m)$	1	1	0	1	0	0	1	1	1	0	1	0	1	1	First Thresholded Sequence
$b_1(m)$	1	1	1	0	0	0	1	1	1	1	0	1	1		Median of First Thresholded Sequence
m =	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	$b_1(m)$=median[$t_1(m\text{-}1)$,$t_1(m)$,$t_1(m\text{+}1)$]

Figure 11: Two threshold levels at Δy and $2\Delta y$, with $\Delta y=1$. The stacking property applies to both $t_i(m)$ and $b_i(m)$. So ($t_2(m) = 1 \Rightarrow t_1(m) = 1$) and ($b_2(m) =1 \Rightarrow b_1(m) = 1$) as well as ($t_1(m) = 0 \Rightarrow t_2(m) = 0$) and ($b_1(m) =0 \Rightarrow b_2(m) = 0$) . The median output is associated with the time index at window's center.

IV Inductive Proof of Median Filter Threshold Decomposition

Theorem 1: *Consider a sequence of samples $\{x(m)\}$, each quantized to k equally spaced amplitude values, where the spacing is equal to Δy, from 0 to $(k\text{-}1)\Delta y$, representing the range of possible amplitude values of $\{x(m)\}$. This sequence is median filtered using a window containing an odd number, 2N+1, of elements to produce the sequence $\{y(m)\}$. Additionally, the values of $x(m)$ each produce (k-1) binary (0,1) thresholded sequences, $\{t_i(m)\}$, where*

$$t_i(m) = \begin{cases} 1, & if\, x(m) \geq i\Delta y,\ 0 < i \leq k-1 \\ 0, & otherwise. \end{cases}$$

The median filtered sequence $\{y(m)\}$ satisfies threshold superposition (decomposition) where there are (k-1) thresholds each producing binary sequence $\{b_i(m)\}$ where these also are the median filtered $\{t_i(m)\}$, as illustrated in Figure 6 for k = 5, and where the sum of the (k-1) binary thresholded medians $\{b_i(m)\}$ equals the median of the original k-1 level sequence $\{y(m)\}$.

Proof by induction: *The base case includes a k-valued input sequence $\{x(m)\}$ with two threshold levels at Δy and $2\Delta y$ producing three thresholded output values $(0, \Delta y, 2\Delta y\}$, as well as two binary-valued threshold sequences $t_1(m)$ and $t_2(m)$. The inductive step is to prove that if the L-level sequence satisfies threshold decomposition, adding an additional level above the top level to create an (L+1)-level sequence must also satisfy threshold decomposition.*

Begin with a sequence $\{x(m), m =1,2,\ldots\}$ with Δy spaced quantized amplitude values . The L level thresholded version of that sequence taken at intervals of constant amplitude Δy is $t_i(m)$ such that,

$$\sum_{i=1}^{L-1} t_i(m)\Delta y = x_L(m) \tag{3}$$

and $t_i(m) = 1$ if $x(m) \geq (i\,\Delta y)$ and 0 otherwise. Because we have not placed a limit on the amplitude range of $\{x(m)\}$ the largest values of each $x(m)$ may be above the highest threshold level $(L\text{-}1)\,\Delta y$ which is the largest value that can be represented by the sum (3). Hence we use $x_L(m)$ rather than $x(m)$ to represent the sum.

Base case: *Take a thresholded, L=3 level sequence, such that the values of the pre-thresholded sequence are 0, Δy, and $2\Delta y$. By showing that we can extend threshold decomposition from a 2 level signal to a 3 level signal, we can prove the inductive step. Performing threshold decomposition on a signal with only 1 threshold level does nothing to the signal, so we opt to start with a 2 level signal, even though we could do a proof extending threshold decomposition from a 1 level signal to a 2 level signal. Also, pick a window and consider the median filtered version of each thresholded sequence as illustrated in Figure 11. If the median of the top level threshold value is a 1, the median of the original sequence must be $2\Delta y$. If the bottom level's median value is a 0, the median of the original sequence must be 0. The only remaining case, where the bottom threshold level is 1 and the top is 0 will indicate the median is Δy.*

Thus,

$$\sum_{i=1}^{2} b_i(m)\Delta y = y(m),$$

where $y(m)$ = the median filtered version of $x(m)$. The median of the top thresholded sequence $\{b_2(m)\}$ added to the median of the bottom threshold sequence $\{b_1(m)\}$ is the same as the median of the original sequence $\{y(m)\}$, thus proving the base case.

Inductive Step: *For the general case, we assume that threshold decomposition works for an L level sequence in order to prove it must also work for an L+1 level sequence. If threshold decomposition is true for an L level sequence then there are L-1 binary threshold sequences such that*

$$\sum_{i=1}^{L-1} t_i(m)\Delta y = x_L(m)$$

with the highest threshold level at $\Delta y(L\text{-}1)$. Add the new threshold level at $L\Delta y$. Proceeding similarly to the base case, consider these top two levels and a median filter spanning 2N+1 sample locations. The binary threshold sequences are $t_{L-1}(m)$ and $t_L(m)$. At the $\Delta y(L\text{-}1)$ level the binary sequence, $t_{L-1}(m)$ either has a median, $b_{L-1}(m)$ = 0 or 1. There are three cases to consider:

(1) *If the median of $t_{L-1}(m)$ is $b_{L-1}(m) = 0$, then because the original sequence satisfies the stacking property, the median at the topmost threshold level $b_L(m)$ must also be zero.*

(2) *If instead the median at the top level, $b_L(m) = 1$, then the median at the level below $b_{L-1}(m)$ must also be one by the stacking property. In this case the new computed sum of median stacks is equal to the old median plus Δy.*

(3) *The third case is when the median at $t_{L-1}(m)$, $b_{L-1}(m) = 1$, and at $t_L(m)$, $b_L(m) = 0$. This implies, by the stacking property, that the median over the prior $\Delta y(L\text{-}1)$ thresholds has value $\Delta y(L\text{-}1)$ and the new median is unchanged. In all cases, the new median, is computed as*

$$\sum_{i=1}^{L} b_i(m)\Delta y = y_{L+1}(m)\,. \blacksquare \qquad (3)$$

V Root Sequences and Recursive Median Filters

As noted in **section I** the repeated filtering of an output sequence $\{y(m)\}$, using the same window size, eventually produces a signal invariant to further median filtering called a root signal. The threshold sequences for this root are also themselves invariant roots. The proof is almost trivial. Each binary median filter pass produces threshold signals that add up to the unthresholded output as shown in eq. (3). If we repeatedly filter the thresholded sequences to obtain $\{b_i(m)\}$ so they no longer change, i.e., the $\{b_i(m)\}$ are themselves roots. They must also sum to an unchanging root sequence $y(m)$.

The recursive median filter proposed by Nodes and Gallagher [6], was the version we employed on the sensor signals for the First Tech Challenge robot we built for the 2017 season. The recursive filter does reduce the input sequence to a root signal in only a single filter pass. No repeated filtering is needed. It nicely cleaned up the noise produced by the many electric motors and servos on the robot. The equation for a recursive median is

$$y(m) = median[y(m-N), \ldots, y(m-1), x(m), \ldots x(m+N)]\,. \qquad (4)$$

The difference between equation (1) and equation (4) is that in the latter the samples in the lefthand side of the window are replaced by the previously computed median values. However when computing the median of the values in the window, the source of the numbers is not important. Every term inside the window satisfies the stacking property which is the key element for satisfying threshold decomposition.

Mathematically it is the nearly identical computation as that for equation (1). The only difference occurs when the window slides one time sample to the right and the value $x(m)$ is replaced by output value $y(m)$. The stacking property and threshold decomposition still apply. Slide down the stack and examine the binary median computation at level i.

$$b_i(m) = median[b_i(m-N), \ldots, b_i(m-1), t_i(m), \ldots t_i(m+N)] .$$

Figure 12 contains the recursive median computation for the same sequence $x(m)$ found in Figure 11. The stacking property and threshold decomposition still apply.

Window Positions of Size 3 with 3 Quantization Levels (0, 1, 2)

🟦	Window with Median value of 2
⬜	Window with Median value of 1
🟩	Window with Median value of 0

$x(m)$	2	2	0	2	0	0	1	1	2	0	1	0	1	1	Original Signal
$y(m)$	2	2	2	2	0	0	1	1	1	1	1	1	1		Recursive Median of Original Signal $y(m)=median[y(m-1),x(m),x(m+1)]$
$t_2(m)$	1	1	0	1	0	0	0	0	1	0	0	0	0	0	Second Thresholded Sequence
$b_2(m)$	1	1	1	1	0	0	0	0	0	0	0	0	0		Recursive Median of Second Thresholded Sequence $b_2(m)=median[b_2(m-1),t_2(m),t_2(m+1)]$
$t_1(m)$	1	1	0	1	0	0	1	1	1	0	1	0	1	1	First Thresholded Sequence
$b_1(m)$	1	1	1	1	0	0	1	1	1	1	1	1	1		Recursive Median of First Thresholded Sequence
$m =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	$b_1(m)=median[b_1(m-1),t_1(m),t_1(m+1)]$

Figure 12: Two threshold levels at Δy and $2\Delta y$, with $\Delta y=1$. The stacking property also applies in the recursive case to both $t_i(m)$ and $b_i(m)$. ($t_2(m) = 1 \Rightarrow t_1(m) = 1$) and ($b_2(m) =1 \Rightarrow b_1(m) = 1$) as well as ($t_1(m) = 0 \Rightarrow t_2(m) = 0$) and ($b_1(m) =0 \Rightarrow b_2(m) = 0$) . The median output is associated with the time index at window's center.

The values b_i(m) are the stack values for the median y(m). By lemma 3 in the Appendix, this stack is unique and the values $t_i(n)$ are the unique stack values for $x(n)$. The result is that the conditions of Theorem 1 all still apply where some of the values for $t_i(n)$ and $x(n)$ are respectively replaced by $b_i(n)$ and $y(n)$. So, this is the completion of the proof of Theorem 2:

Theorem 2: *Consider a sequence of samples* $\{x(m)\}$*, each quantized to k equally spaced amplitude values from 0 to* (k-1)Δy*. This sequence is recursively median filtered using a window containing an odd number, 2N+1, of elements to produce the sequence* $\{y(m)\}$ *as follows:*

$$y(m) = median[y(m-N), \ldots, y(m-1), x(m), \ldots x(m+N)]$$

Additionally, the values of $x(m)$ *each produce* (k-1) *binary* (0,1) *thresholded sequences,* $\{t_i(m)\}$*, where*

$$t_i(m) = \begin{cases} 1, & if\, x(m) \geq i\Delta y,\ 0 < i \leq k-1 \\ 0, & otherwise . \end{cases}$$

The median filtered sequence $\{y(m)\}$ *satisfies threshold decomposition where there are* (k-1) *thresholds each producing binary* $\{b_i(m)\}$ *where these also are the median filtered* $\{t_i(m)\}$ *as follows:*

$$b_i(m) = median[b_i(m-N), \ldots, b_i(m-1), t_i(m), \ldots t_i(m+N)],$$

where the sum of the (*k-1*) *binary recursive medians* $\{b_i(m)\}$ *equals the recursive median of the original k-1 level sequence* $\{y(m)\}$. ■

VI Discussion of results and Conclusion

Typically, computing the median of any signal is performed by sliding a window across values of the signal, and sorting the values within that window. However, threshold decomposition allows a multi-level signal to be median filtered without sorting, which saves a significant amount of time in the computation. In this paper, an original alternate inductive proof of threshold decomposition of median filters is provided. The induction was performed over the threshold levels, and we were able to prove that the regular median filter and the recursive median filter both satisfy threshold decomposition. Every signal can be represented as a sum of binary value threshold signals. Median filtering these binary signals, an operation which requires no sorting, and then adding them together produces the same output that median filtering the original signal would. Prof. Arce suggested to us that this inductive proof could be applied to center weighted median filters, but we are keeping this topic for a later discussion. This new perspective on median filtering is a tool which can be used to analyze and better understand median filters.

Appendix A: Additional Proof

Lemma 3: *There is one and only one threshold decomposition for any positive real number x.*
Proof by contradiction: *We are given a positive integers x, where* $0 \leq x < k$. *We generate a threshold stack* $\{s_i(x)\}$ *for the integer x,*

$$s_i(x) = \begin{cases} 1, & \text{if } x \geq i,\ 0 < i < k \\ 0, & otherwise, \end{cases}$$

This gives us a stack of zeros on top of a stack of ones. The topmost one in the stack occurs at i = x, where

$$\sum_{i=1}^{k-1} s_i(x) = x\,.$$

If there are two different stacks, call them $s_i(x)$ *and* $u_i(x)$*, they must contain the same number of ones in each stack. Therefore the transition of where the values of* $s_i(x)$ *change from values of one to values of zero must occur at the same level for the two stacks to have identical sums = x. The only other way the two stacks can add up to x is for a value of at least one* $s_i(x) = 1$ *to change to a zero and a different* $s_j(x) = 0$ *to change to a one to keep the sum the same. This, however, violates the stacking property where all the zeros are above all the ones. So we have a contradiction. The stack is unique.* ■

Appendix B: MATLAB Programs

These MATLAB programs demonstrate the property described in lemma 1. Recursive and non-recursive median filters commute with non-increasing or non-decreasing functions. They do this in order to generate a random variable *Y* with the Cauchy distribution.

$$F_Y(y) = \frac{1}{\pi} arctan(y) + \frac{1}{2}\,.$$

First, generate a uniformly distributed random variable *X* over the interval (0, 1) and then generate the random variable *Y* by using it as the input to the equation below: [4].

$$Y = tan(\pi(X - \frac{1}{2}))\,.$$

Variable *Y* will then have the desired Cauchy distribution. The Matlab programs that do this are listed below. Program A takes the sequence of random variables *Y* and recursively median filters them. Finally, it displays a graph containing the recursively median filtered and non-median filtered numbers. Program B recursively median filters the sequence of uniform random variables *X* and then median filters these values. These values are used to generate the *Y* variables. As proven in lemma 1 and given the same seed *rng(N)*, the two programs will output the same graphs.

Program A

```
% Environment setup
format compact
clc
clear

% Seed for the random number generator
rng(0);

% Constants
N = 2;        % Window size W=2N+1
W = 2*N+1;
Nrand = 104;   % Total number of random variables generated

% Pre-allocate array sizes
cauchy = zeros(1,Nrand);           % Cauchy - Original random input
B = zeros(1,W);                    % B - Window recursive median window samples
BNR = zeros(1,W);                  % BNR - Nonrecursive window samples
M2 = zeros(1,(Nrand-2*N));         % M2 - Contains recursive median
M2Old = zeros(1,(Nrand-2*N));      % Nonrecursive median
cauchy2 = zeros(1,(Nrand-2*N));    % Cauchy with N samples removed at each end
cauchy2Old = zeros(1,(Nrand-2*N)); % Cauchy2Old - Nonrecursive

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%

% Filled with Cauchy RV's at beginning
for i = 1:Nrand
  cauchy(i) = tan(pi()*(rand()-0.5));
end

% cauchyOld is a copy of the original noise sequence
cauchyOld = cauchy;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%

% The value i is the starting index for the window
for i = 1:(Nrand-2*N)

  % index k counts up to the Window width
  k = 0;

  % j goes through all the window index values
  for j = i:(i+W-1)

    % Counts up the window index values starting with one
```

```
        k = k+1;

        % for each window position k, B(k)
        % contains all the window's samples
        B(k) = cauchy(j);
        BNR(k) = cauchyOld(j);

    end

    % M2 contains all the computed window medians for each
    M2(i) = median(B);

    % Set of window samples B(k) using
    M2Old(i) = median(BNR);

    % The line below prints the median for window B(k)
    fprintf('The computed median for the above values %f6\n\n', M2(i))

    % Index of each window's middle value
    jmid = i+N;

    % This statement makes it a recursive median
    cauchy(jmid) = M2(i);

    % Cauchy now holds the recursive median output
    fprintf('The index of the Window center sample %f\n\n', jmid)

    % For the recursive median we replace each windows original middle
    % sample value with the computed median value for that window

end

% Count the index to collect the reduced length cauchy array
kc = 0;

% This is done for the plots below
for i = (N+1):(Nrand-N) % look here to clip CaucyOld
    kc=kc+1;
    cauchy2(kc)=cauchy(i);
    cauchy2Old(kc)=cauchyOld(i);
end

fprintf('cauchy is now the computed and not clipped recursive median array: \n\n')
cauchy

fprintf('cauchy2 is the computed and clipped recursive median array: \n\n')
cauchy2

plot(cauchy2)
hold on

fprintf('cauchy2Old is clipped original noise array: \n\n')
cauchy2Old

plot(cauchy2Old)
hold off
```

Program B

```
% Environment setup
format compact
clc
clear

% Seed for the random number generator
rng(0);

% Constants
N = 2;         % Window size W=2N+1
W = 2*N+1;
Nrand = 104;   % Total number of random variables generated

% Pre-allocate array sizes
cauchy = zeros(1,Nrand);            % Cauchy - Original random input
B = zeros(1,W);                     % B - Window recursive median window samples
BNR = zeros(1,W);                   % BNR - Nonrecursive window samples
M2 = zeros(1,(Nrand-2*N));          % M2 - Contains recursive median
M2Old = zeros(1,(Nrand-2*N));       % Nonrecursive median
cauchy2 = zeros(1,(Nrand-2*N));     % Cauchy with N samples removed at each end
cauchy2Old = zeros(1,(Nrand-2*N));  % Cauchy2Old - Nonrecursive

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%

% Filled with random values at beginning
for i = 1:Nrand
   cauchy(i) = rand();
end

% cauchyOld is a copy of the original noise sequence
cauchyOld = cauchy;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%

% The value i is the starting index for the window
for i = 1:(Nrand-2*N)

   % index k counts up to the Window width
   k = 0;

   % j goes through all the window index values
   for j = i:(i+W-1)

      % Counts up the window index values starting with one
      k = k+1;

      % for each window position k, B(k)
      % contains all the window's samples
      B(k) = cauchy(j);
      BNR(k) = cauchyOld(j);

   end

   % M2 contains all the computed window medians for each
   M2(i) = median(B);
```

```
    % Set of window samples B(k) using
    M2Old(i) = median(BNR);

    % The line below prints the median for window B(k)
    fprintf('The computed median for the above values %f6\n\n', M2(i))

    % Index of each window's middle value
    jmid = i+N;

    % This statement makes it a recursive median
    cauchy(jmid) = M2(i);

    % Cauchy now holds the recursive median output
    fprintf('The index of the Window center sample %f\n\n', jmid)

    % For the recursive median we replace each windows original middle
    % sample value with the computed median value for that window

end

% Count the index to collect the reduced length cauchy array
kc = 0;

% Compute the cauchy values
for i = 1:Nrand
    cauchy(i) = tan(pi()*(cauchy(i)-0.5));
    cauchyOld(i) = tan(pi()*(cauchyOld(i)-0.5));
end

% This is done for the plots below
for i = (N+1):(Nrand-N) % look here to clip CaucyOld
    kc=kc+1;
    cauchy2(kc)=cauchy(i);
    cauchy2Old(kc)=cauchyOld(i);
end

fprintf('cauchy is now the computed and not clipped recursive median array: \n\n')
cauchy

fprintf('cauchy2 is the computed and clipped recursive median array: \n\n')
cauchy2

plot(cauchy2)
hold on

fprintf('cauchy2Old is clipped original noise array: \n\n')
cauchy2Old

plot(cauchy2Old)
hold off
```

Acknowledgements
We appreciate the encouragement and suggestions of Prof. Edward Coyle, the John B. Peatman Distinguished Professor at Georgia Tech. We also appreciate Prof. Gonzalo Arce, the Charles Black Evans Professor at the University of Delaware for reading and making suggestions on an early draft of our manuscript.

References
[1] G. R. Arce, *Nonlinear Signal Processing*, John Wiley & Sons, 2005.
[2] J. P. Fitch, E. J. Coyle and N. C. Gallagher, Jr., “Median Filter by Threshold Decomposition,“ *IEEE Trans. Acoustics, Speech, and Signal Processing*, Vol. ASSP-32, pp. 1183-1188, Dec. 1984.
[3] N. C. Gallagher and G. L. Wise, “A Theoretical Analysis of the Properties of M edian Filters,” *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. ASSP-29, pp. 1136-1141, Dec. 1981.
[4] P. G. Hoel, S. C. Port and C. J. Stone, *Introduction to Probability Theory, Houghton Mifflin Company,* 1971.
[5] C.-Y. Ning, S.-F. Liu and M. Qu, “Research on Removing Noise in Medical Image Based on Median Filter Method,” *IEEE International Symposium on IT in Medicine & Education,* Jinan, 14-16 August 2009.
[6] T. Nodes and N. C. Gallagher, “Median Filters: Some Modifications and Their Properties,” *IEEE Trans. Acoustics, Speech, Signal Processing*, Vol. ASSP-30, No. 5, Oct. 1982.
[7] A. V. Oppenheim and A. S. Wilsky, *Signals and Systems*, Prentice Hall, 1983.
[8] J. W. Tukey, *Exploratory Data Analysis*, CA, Menlo Park:Addison-Wesley, 1971.