Dependence structures in multidimensional arrays

by
Kwok-Ho Lau

Submitted in partial fulfillment of the requirements for the degree

Magister Scientiae

In the Department of Statistics
In the Faculty of Natural and Agricultural Sciences
University of Pretoria

July 2016

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

I, *Kwok-Ho Lau,* declare that this mini-dissertation, which I hereby submit for the degree Magister Scientiae in Mathematical Statistics at the University of Pretoria, is my own work and has not previously been submitted by me for a degree at this or any other tertiary institution.

Signature:

Date:

Summary

In the process of data acquisition the information obtained are more than often contaminated by noise. To purify the data smoothers are designed to remove the noise. The $LULU$ operators are such smoothers, more specifically, they are designed to remove impulsive noise. Carl Rohwer and his collaborators developed the $LULU$ operators in one dimension in the last four decades and, more recently, the operators have been extended to higher dimensions by Roumen Anguelov and Inger Fabris-Rotelli [2]. The properties in shape preservation and total variation preservation are extended from one-dimensional $LULU$ operators. This allows for smoothing with the operators in images. However, because their definition uses a morphological concept of a connection, the question of how complex the connectivity should be therefore arises. Using the results from correlation analysis, we explore the extent at which the pixels of an image depend on its neighbours and establish the complexity of the connectivity for $LULU$ operators in two-dimensions. In addition, as a measure of how effective the $LULU$ smoothers remove noise, we examine the noise extractions by the operators for images.

Contents

Chapter 1

Introduction

In the process of data acquisition the information obtained are more than often contaminated by noise. To purify the data smoothers are designed to remove the noise. The $LULU$ operators are such smoothers, more specifically, they are designed to remove impulsive noise. Carl Rohwer and his collaborators developed the $LULU$ operators in one dimension in the last four decades and, more recently, the operators have been extended to higher dimensions by Roumen Anguelov and Inger Fabris-Rotelli [2]. The properties in shape preservation and total variation preservation are extended from one-dimensional $LULU$ operators. This allows for smoothing with the operators in images. However, because their definition uses a morphological concept of a connection, the question of how complex the connectivity should be therefore arises. Using the results from correlation analysis, we explore the extent at which the pixels of an image depend on its neighbours and establish the complexity of the connectivity for $LULU$ operators in two-dimensions. In addition, as a measure of how effective the $LULU$ smoothers remove noise, we examine the noise extractions by the operators for images.

We first lay the foundation of $LULU$ operators in Chapter 2. Therein we also discuss the properties and the Discrete Pulse Transform of the operators in one- and two- dimensions. In Chapter 3 we study the dependence of image pixels with respect to its neighbours by use of a dependence structure. In Chapter 4 we investigate the effectiveness of the $LULU$ operators in removing noise.

Chapter 2

Smoothers

2.1 Introduction

In data analysis smoothing is required to capture patterns inherent in the dataset or to purify the noisy data so that inferences regarding the data are not polluted by unwanted variation. Linear filters are traditionally used for such purposes, however, their efficiency is apparent only when the data contains well-behaved noise - such as noise that have a Gaussian distribution.

Alternatively non-linear filters have shown to be more robust than linear filters. For example, Tukey [20] proposed the median filter which selects the central order statistic in a running window of observations. Rohwer [28] bases the $LULU$ operators on the extreme order statistics.

In this chapter we cover the $LULU$ operators in one and two dimensions. More specifically, Section 2.2 covers the foundation of smoothers, Section 2.3 introduces the $LULU$ operators in one dimension, of which their compositions and properties are discussed in Sections 2.4 and 2.5 respectively, and the one-dimensional Discrete Pulse Transform in Section 2.6. The rest of the chapter covers $LULU$ operators in higher dimensions in Section 2.7.

2.2 Operators on sequences

To study sequences it is convenient to use the vector space framework. Let X be the set of bi-infinite sequences of real numbers $x = \langle x_i \rangle_{i=-\infty}^{\infty}$.

Some criteria for the designing and comparing a smoother P functioning as a separator of "signal" from a sequence where it's contaminated by "noise" are [28]:

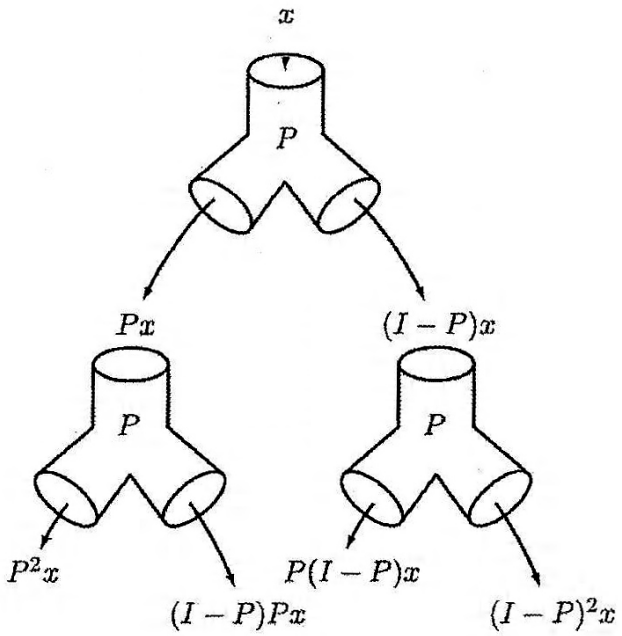


Figure 2.1: Two-stage separator cascade (graphic from [28]).

Effectiveness	For each x, Px should be signal and $(I-P)x$ noise.
Consistency	Signals should be preserved and noise mapped onto 0.
Stability	The smoother should be robust to small input perturbations.
Efficiency	The computations should not require excessive running time.

We thus have the following smoother axioms [28]:

Smoother Axioms An operator P on X is a smoother if:

1. $PE = EP$, where E is the shift operator.
2. $P(x + c) = P(x) + c$, for each $x, c \in X$ such that c is a constant sequence.
3. $P(\alpha x) = \alpha P(x)$, for each $x \in X$ and scalar $\alpha \geq 0$.

The consistency criterion then describes the following separator axioms [28]:

Separator Axioms A smoother P is a separator if it also satisfies the additional axioms:

4. $P^2 = P$ (Idempotence).
5. $(I-P)^2 = I - P$ (Co-idempotence).

An argument for the axioms above can be presented with a simple analogy. Suppose the operator P separates milk x into two components: Px - the curd, and $(I-P)x$ - the whey. The performance of the separator P could be measured by passing Px into P again or to pass $(I-P)x$ through P and compare them to Px and $(I-P)x$. This is illustrated in Figure 2.1.

Suppose the curd is the signal and the whey the noise. If $P^2x = Px$, then P is considered to be a consistent signal-extractor. On the other hand, if $(I-P)^2x = (I-P)x$, the separator is considered noise-consistent. In the above cases, P is called idempotent and co-idempotent [23] respectively.

2.3 *LULU* operators in one dimension

We introduce the *LULU* operators in one dimension, developed by Rohwer [28, 33] and his collaborators Wild and Laurie. The *LULU* operators are first presented in [31], their properties in [5, 6, 7, 23, 26, 25, 24, 27, 28, 33], and [38], the comparison with the median filter in [22, 32], the Discrete Pulse Transform in [8, 16, 17] and [30], the *LULU* distributions and statistical properties in [4, 14], and [29], and the *LULU* operators on a continuous domain in [1, 3] and [18]. Unless stated otherwise, the definitions and results provided in the rest of Section 2.3 are obtained from [28].

Definition 2.1. Let the operators $\bigvee^n$ and $\bigwedge^n$, for $n \geq 1$, that map the sequence x onto y be defined by

$$\begin{aligned} \textstyle\bigvee^n: &\quad y = \textstyle\bigvee^n x = \langle y_i = \max\{x_i, ..., x_{i+n-1}, x_{i+n}\}\rangle\,, \\ \textstyle\bigwedge^n: &\quad y = \textstyle\bigwedge^n x = \langle y_i = \min\{x_{i-n}, x_{i-n+1}, ..., x_i\}\rangle\,. \end{aligned}$$

The *LULU* operators are then compositions of these min and max selectors; defined as

$$L_n = \bigvee^n \bigwedge^n, \quad U_n = \bigwedge^n \bigvee^n.$$

It can be seen that the *LULU* operators are defined in such a way to temper the initial use of extreme selectors $\bigvee$ and $\bigwedge$ by the immediate application of its opposite. That is the L_n operator selects the maximum of the set of minimums within a sequence whereas the U_n operator selects the minimum of the set of maximums.

Note further that the original sequence x is always contained within the interval $[L_nx, U_nx]$, and, for any $n > 0$, it is true that $L_{n+1} \leq L_n$ and $U_n \leq U_{n+1}$. Thus the following relationship holds:

$$L_{n+1} \leq L_n \leq I \leq U_n \leq U_{n+1},$$

where I is the identity operator. In relation to the median operator $M_n = \text{median}\{x_{i-n}, ..., x_i, ..., x_{i+n}\}$, we have that the following inequality holds true:

$$U_nL_n \leq M_n \leq L_nU_n.$$

Definition 2.2. Let N be the negative operator on any sequence, then an operator B is the dual of A if $AN = NB$.

The L_n and U_n operators are duals of each other. Furthermore, any composition of L_n and U_n is also dual to the operator by interchanging L_n and U_n for each n.

Definition 2.3.

1. A subsequence $\langle x_i, x_{i+1}, ..., x_{i+n}, x_{i+n+1}\rangle$ is an upward n-pulse if

$$\min\{x_{i+1}, x_{i+2}, ..., x_{i+n}\} > \max\{x_i, x_{i+n+1}\}.$$

2. A subsequence $\langle x_i, x_{i+1}, ..., x_{i+n}, x_{i+n+1}\rangle$ is a downward n-pulse if

$$\max\{x_{i+1}, x_{i+2}, ..., x_{i+n}\} < \min\{x_i, x_{i+n+1}\}.$$

The subsequence is a constant upward pulse (constant downward pulse) if $x_j = c\ \forall j = i+1, i+2, ..., i+n$ for some constant c in the above the definition. Note that L_n removes all upwards pulses of size n or smaller and U_n removes all downward pulses of size n or smaller. Note that by 'remove' we imply the sequence has been smoothed by L_nU_n or U_nL_n and the n-pulse sequence positions have been replaced by either the value at position i or $i+n+1$ - see the example that follows for the details as well as the Roadmaker's algorithm in Section 2.6. This then gives us the First Idempotence Theorem, since there will be no n-pulses to remove after the application of L_m and U_m:

$$L_nL_m = L_m \text{ and } U_nU_m = U_m \text{ for } m \geq n.$$

Result 2.1. An n-pulse is removed from any constant sequence by L_mU_m and U_mL_m, if $m \geq n$.

From Result 2.1 we obtain the Second Idempotence Theorem, since reapplying the operators will have no effect:

$$(L_nU_n)^2 = L_nU_n \text{ and } (U_nL_n)^2 = U_nL_n.$$

Example 2.1. Consider the sequence $x = \{1, 1, 5, 4, 7, 1, 2\}$. We smooth x sequentially with U_nL_n as follows[1].

At $n = 1$,

$$\begin{aligned}(L_1x)_i &= \bigvee\left(\bigwedge x\right)_i \\ &= \max\left\{\left(\bigwedge x\right)_i, \left(\bigwedge x\right)_{i+1}\right\} \\ &= \max\{\min\{x_{i-1}, x_i\}, \min\{x_i, x_{i+1}\}\}\end{aligned} \qquad \begin{aligned}(U_1x)_i &= \bigwedge\left(\bigvee x\right)_i \\ &= \min\left\{\left(\bigvee x\right)_{i-1}, \left(\bigvee x\right)_i\right\} \\ &= \min\{\max\{x_{i-1}, x_i\}, \max\{x_i, x_{i+1}\}\}\end{aligned}$$

It follows that

$$(L_1x)_1 = \max\{\min\{0, 1\}, \min\{1, 1\}\} = \max\{0, 1\} = 1$$
$$(L_1x)_2 = \max\{\min\{1, 1\}, \min\{1, 5\}\} = \max\{1, 1\} = 1$$
$$(L_1x)_3 = \max\{\min\{1, 5\}, \min\{5, 4\}\} = \max\{1, 4\} = 4$$
$$(L_1x)_4 = \max\{\min\{5, 4\}, \min\{4, 7\}\} = \max\{4, 4\} = 4$$

[1] Recall that x is a bi-infinite sequence of real numbers. Thus $\{1, 1, 5, 4, 7, 1, 2\}$ represents $\{...0, 0, 1, 1, 5, 4, 7, 1, 2, 0, 0, ...\}$

$(L_1x)_5 = \max\{\min\{4,7\},\min\{7,1\}\} = \max\{4,1\} = 4$

$(L_1x)_6 = \max\{\min\{7,1\},\min\{1,2\}\} = \max\{1,1\} = 1$

$(L_1x)_7 = \max\{\min\{1,2\},\min\{2,0\}\} = \max\{1,0\} = 1$

At $n = 2$,

$$\begin{aligned}
(L_2x)_i &= \bigvee^{2}\left(\bigwedge^{2}x\right)_i \\
&= \max\left\{\left(\bigwedge^{2}x\right)_i, \left(\bigwedge^{2}x\right)_{i+1}, \left(\bigwedge^{2}x\right)_{i+2}\right\} \\
&= \max\{\min\{x_{i-2},x_{i-1},x_i\},\min\{x_{i-1},x_i,x_{i+1}\},\min\{x_i,x_{i+1},x_{i+2}\}\} \\
(U_2x)_i &= \bigwedge^{2}\left(\bigvee^{2}x\right)_i \\
&= \min\left\{\left(\bigvee^{2}x\right)_{i-2}, \left(\bigvee^{2}x\right)_{i-1}, \left(\bigvee^{2}x\right)_{i}\right\} \\
&= \min\{\max\{x_{i-2},x_{i-1},x_i\},\max\{x_{i-1},x_i,x_{i+1}\},\max\{x_i,x_{i+1},x_{i+2}\}\}
\end{aligned}$$

And so on. Applying L_1 on x results in a new sequence $L_1x = \{1,1,4,4,4,1,1\}$. Note that L_1 has removed all upward pulses of size 1 (see Figure 2.2). Applying U_1 on L_1x would result in L_1x since there are no downward pulses of size 1. Similarly, $U_2L_2U_1L_1x = L_1x$ because there are no upward nor downward pulses of size 2. A new sequence would be obtained when L_3 is applied since L_1x contains an upward 3-pulse: $L_3U_2L_2U_1L_1x = \{1,1,1,1,1,1,1\}$.

This process is continued until the zero sequence (in this case) is obtained. Note that by the same reasoning as above, $U_6L_6...U_1L_1x = U_5L_5...U_1L_1x = ... = U_3L_3U_2L_2U_1L_1x = \{1,1,1,1,1,1,1\}$. Finally, L_7 removes the last upward 7-pulse and the zero sequence remains. This is depicted Figure 2.2.

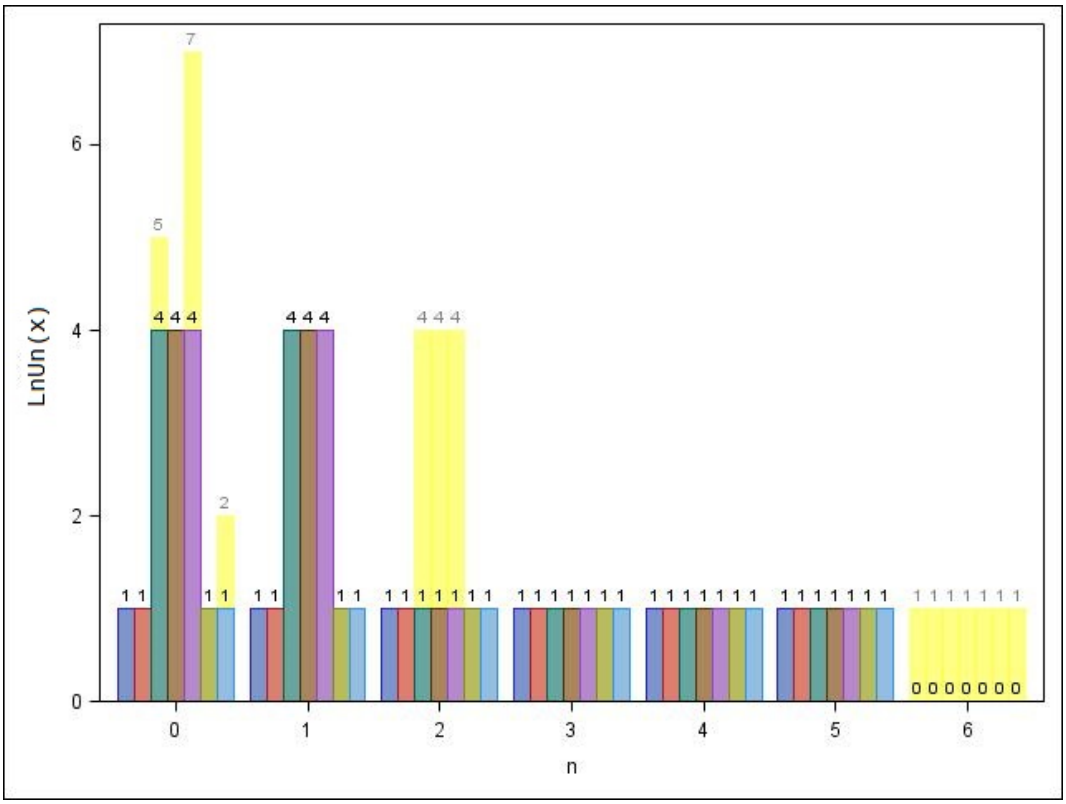


Figure 2.2: Decomposition of a sequence x using $LULU$ operators. Note that $n = 0$ is the original sequence.

Note that in Figure 2.2, the transparent bars indicate the pulses that will be removed at the next application of U_nL_n - that is, they are still part of the sequence at the n^{th} step. For instance, at $n = 0$, we see three transparent bars at $i = 3, 5, 7$ this is because, at the next step $n = 1$, U_1L_1 removes all pulses of size 1. At $n = 2$, we see one transparent bar of width 3. This indicates that it will be removed by U_3L_3 at the next step.

■

2.4 Compositions of L_n and U_n operators

The successive application of L_nU_n and U_nL_n yields better smoothing results as opposed to just an application for a single specified n. Here, we define the 'ceiling' and 'flooring' operators C_n and F_n respectively. The definitions and results in this section are also obtained from [28] unless otherwise stated.

Definition 2.4.

$C_1 = L_1U_1$ and $C_{n+1} = L_{n+1}U_{n+1}C_n$,

$F_1 = U_1L_1$ and $F_{n+1} = U_{n+1}L_{n+1}F_n$.

Result 2.2. For each n, $U_nL_n \leq F_n \leq C_n \leq L_nU_n$.

Example 2.2. A sequence $y = \sin^3(4\pi x^3)$ with added noise from $N(0, 0.03^2)$ is simulated. The original graph line of y as well as noisy y is shown in Figure 2.3.

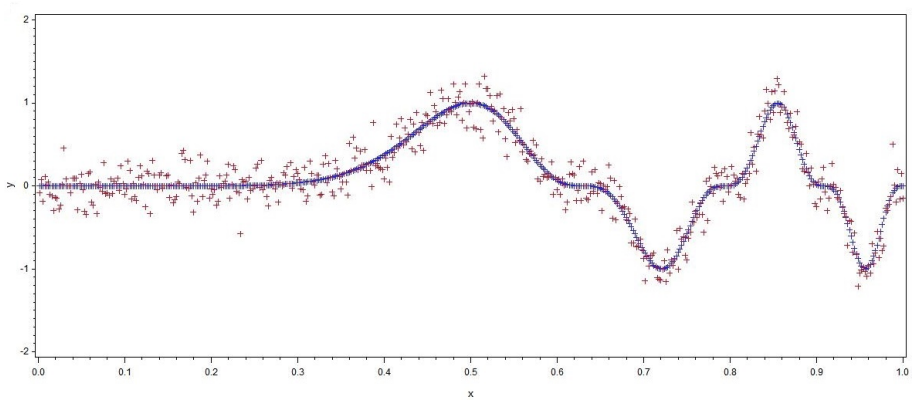


Figure 2.3: Series plot of y (original) in blue and noisy y in red.

The effects of L_nU_n, U_nL_n, F_n, and C_n are shown in Figure 2.4 to 2.10 for different values of n. Notice that, for each n, $U_nL_n \leq F_n \leq C_n \leq L_nU_n$, so $[F_n, C_n]$ concentrates the signal interval $[L_n, U_n]$.

Furthermore, since the $LULU$ operators remove noise with respect to *pulse length*, it can be seen that the performance of these operators deteriorates as the value of n increases. There are several reasons for this. The simulated noise are iid Gaussian distributed, thus the only pulses believed to be noise within the contaminated line are of length one. Already after the application of L_1U_1 or U_1L_1, the amount of unwanted variation has been removed by these operators. The extent to which we remove noise will be discussed in Chapter 4.

As the value of n increases, the operator removes all pulses that are of length n or less even when these pulses are considered to be natural according to the definition of the line - this is most evident at $n = 50$ in Figure 2.10 where the operators completely ignore the natural pulses occurring within the line. We thus introduce a formal definition of variation in order to measure the amount of smoothing by the $LULU$ operators.

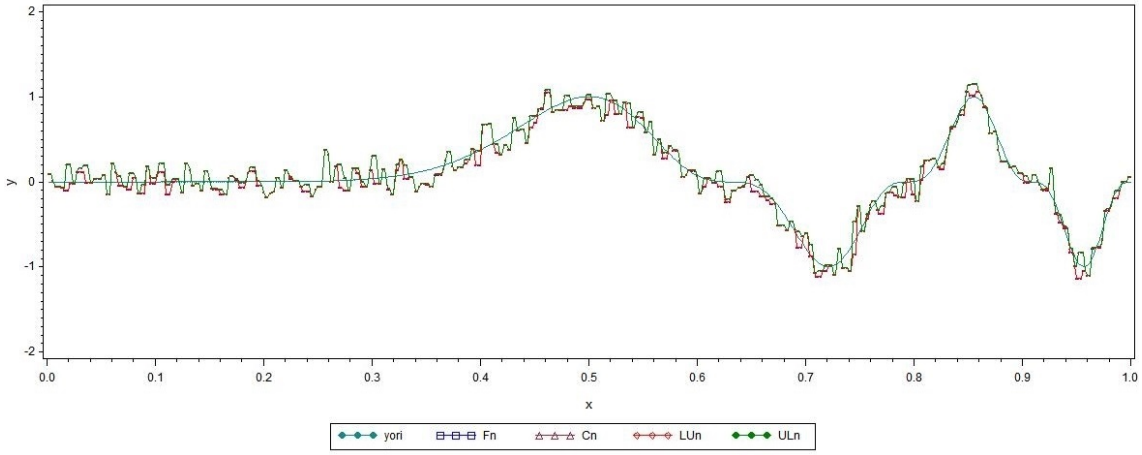


Figure 2.4: F_1, C_1, L_1U_1, and U_1L_1 applied to noisy signal y.

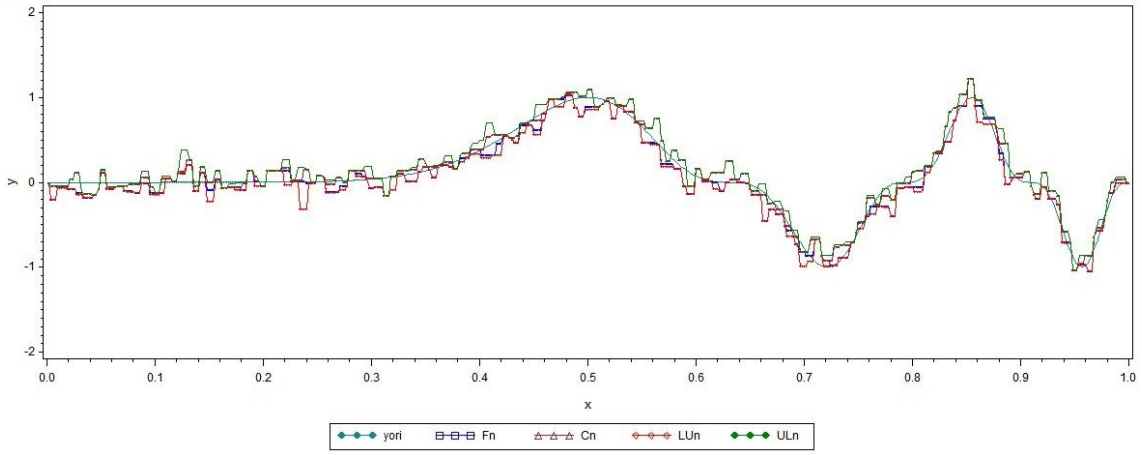


Figure 2.5: F_2, C_2, L_2U_2, and U_2L_2 applied to noisy signal y.

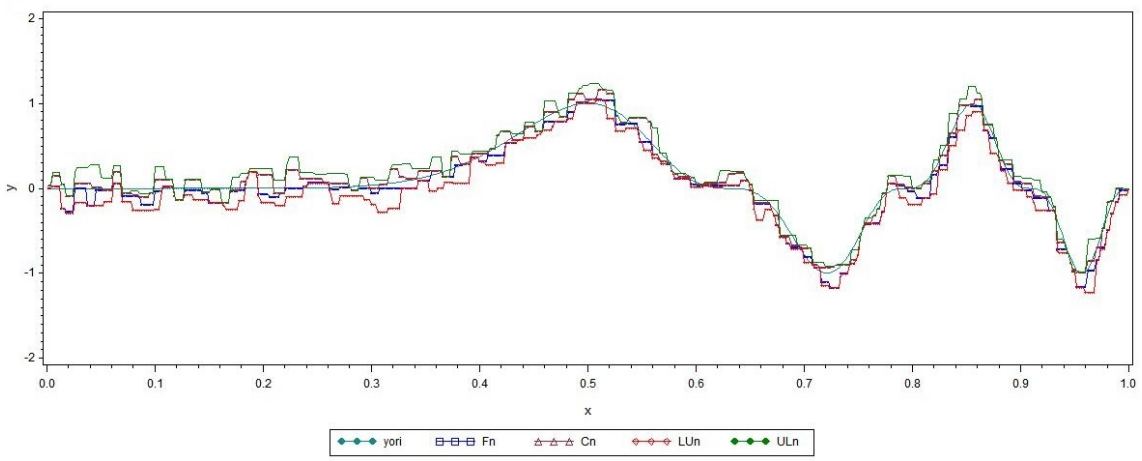


Figure 2.6: F_3, C_3, L_3U_3, and U_3L_3 applied to noisy signal y.

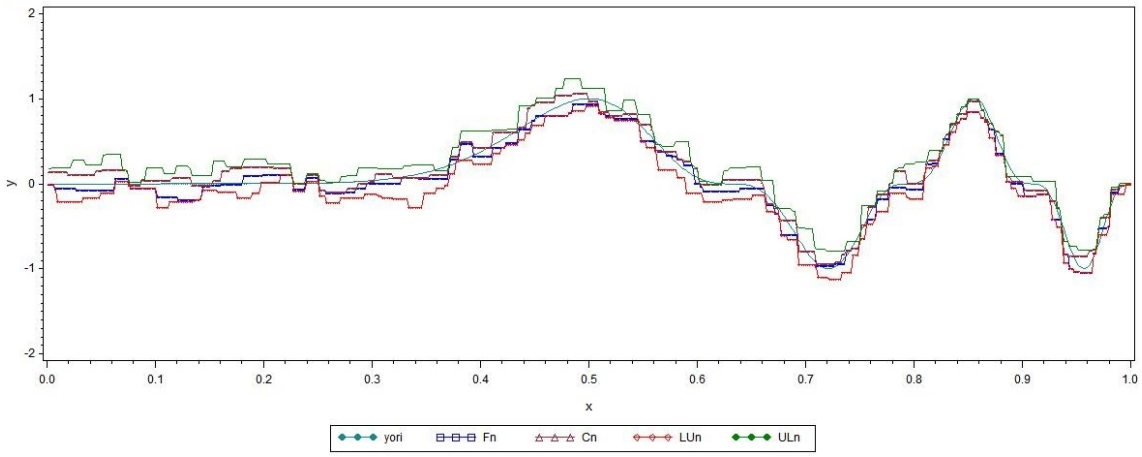


Figure 2.7: F_5, C_5, L_5U_5, and U_5L_5 applied to noisy signal y.

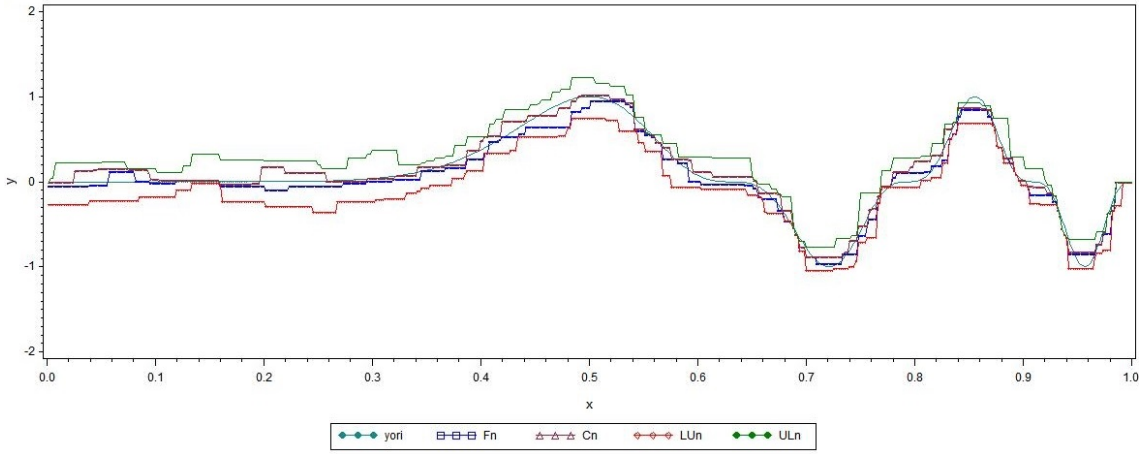


Figure 2.8: F_{10}, C_{10}, $L_{10}U_{10}$, and $U_{10}L_{10}$ applied to noisy signal y.

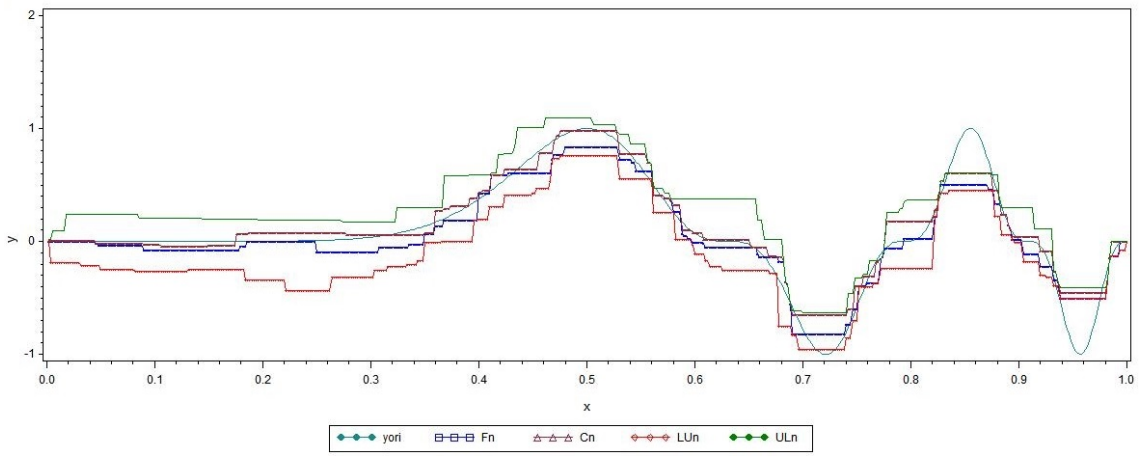


Figure 2.9: F_{20}, C_{20}, $L_{20}U_{20}$, and $U_{20}L_{20}$ applied to noisy signal y.

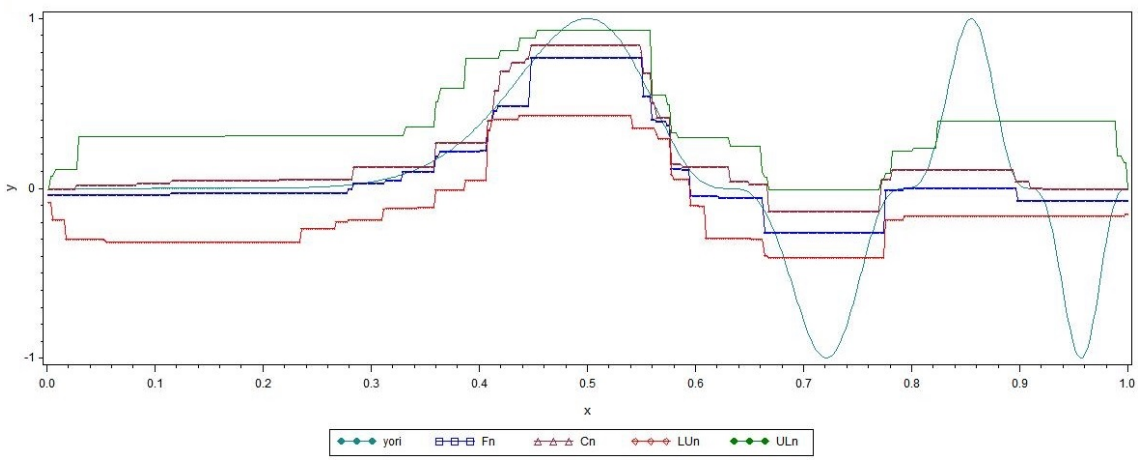


Figure 2.10: F_{50}, C_{50}, $L_{50}U_{50}$, and $U_{50}L_{50}$ applied to noisy signal y.

■

2.5 Variation reduction and shape preservation

To measure the degree of smoothing, Rohwer [28] makes use of *total variation* to measure the level of continuity present within the original sequence and its $LULU$ derivatives.

Definition 2.5. The total variation of a sequence x is defined as $T(x) = \sum_{i=-N}^{N} |x_{i+1} - x_i|$.

It is clear that if $x \in l_1$, then $T(x) \in {l_1}^2$, since

$$\sum_{i=-N}^{N} |x_{i+1} - x_i| \leq \sum_{i=-N}^{N} (|x_i| + |x_{i+1}|) \leq 2\|x\|_1.$$

where $\|x\|_1 = \sum_{i=-N}^{N} |x_i|$.
The total variation has the following properties:

Result 2.3.

i. $T(Ex) = T(x)$, where E is the shift operator.

[2]The l_1 norm [15] of a vector $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ is defined by

$$\|\mathbf{x}\|_1 = |x_1| + |x_2| + ... + |x_n|,$$

and satisfies the following conditions:

1. $\|\mathbf{x}\|_1 = 0$ only if $\mathbf{x}$ is the zero vector.
2. $\|c\mathbf{x}\|_1 = |c|\|\mathbf{x}\|_1$.
3. $\|\mathbf{x} + \mathbf{y}\|_1 \leq \|\mathbf{x}\|_1 + \|\mathbf{y}\|_1$.

ii. $T(x+y) \leq T(x) + T(y)$ (subadditivity).

iii. $T(\alpha x) = |\alpha| T(x)$ and $T(x) = 0$ only if $x = 0$.

From the above Result 2.3 we can see that the operator T is a semi-norm [15].

Result 2.4. $T(\bigwedge\bigvee x) = T(\bigvee x)$ and $T(\bigvee\bigwedge x) = T(\bigwedge x)$ for each $x \in l_1$.

Result 2.5 below shows that the total variation for any sequence x can be separated into two parts, namely, the total variation of the signal extracted by the operator (U_n or L_n) and its noise (($I - U_n$) or ($I - L_n$) respectively). Therefore, no variation is added or lost during the application of the $LULU$ operators or their compositions and $T(\cdot)$ thus provides a mechanism to track the information (variation) in the sequence as it is smoothed by the $LULU$ operators.

Result 2.5. Variation preservation

a) $T(x) = T(U_n x) + T((I - U_n)x)$.

b) $T(x) = T(L_n x) + T((I - L_n)x)$.

c) $T(x) = T(L_n U_n x) + T((I - L_n U_n)x)$.

d) $T(x) = T(U_n L_n x) + T((I - U_n L_n)x)$.

Example 2.3. Continuing from Example 2.1 the total variation for the sequence $x = \{1, 1, 5, 4, 7, 1, 2\}$ is calculated as $T(x) = 18$. If we find the total variation for each new sequence obtained by $F_n = U_n L_n U_{n-1} L_{n-1} ... U_1 L_1$ for different values of n in Table 2.1. We can see that total variation is preserved.

n	0	1, 2	3, 4, 6,	7
$T(F_n x)$	18	8	2	0
$T[(I - F_n)x]$	18	10	16	18
$T(F_n x)/T(x)$	1	0.4444	0.1111	0
$T[(I - F_n)x]/T(x)$	1	0.5556	0.8889	1

Table 2.1: Preservation of total variation in sequence x

■

Example 2.4. From Example 2.2, we obtain the variation for each $F_n, (I - F_n), C_n, (I - C_n)$ on x for $n = 1, 2, ..., 20$ and plot them together. Figure 2.11 depicts the results. It can be noted that beyond $n = 3$ the variation tends slower to 0. As noted before, the added noise are iid Gaussian, thus the expected pulse length is 1 at all points of the line (see Figure 2.3). Even so, it is not unlikely that these noise should appear consecutively to create pseudo -upward or -downward pulses of length longer than 1. Hence the

rate of total variation decreases fast for small n. However, because is it unlikely that long pulses exist on the line in the presence of iid Gaussian noise, we have that the change in total variation is less obvious for larger values of n.

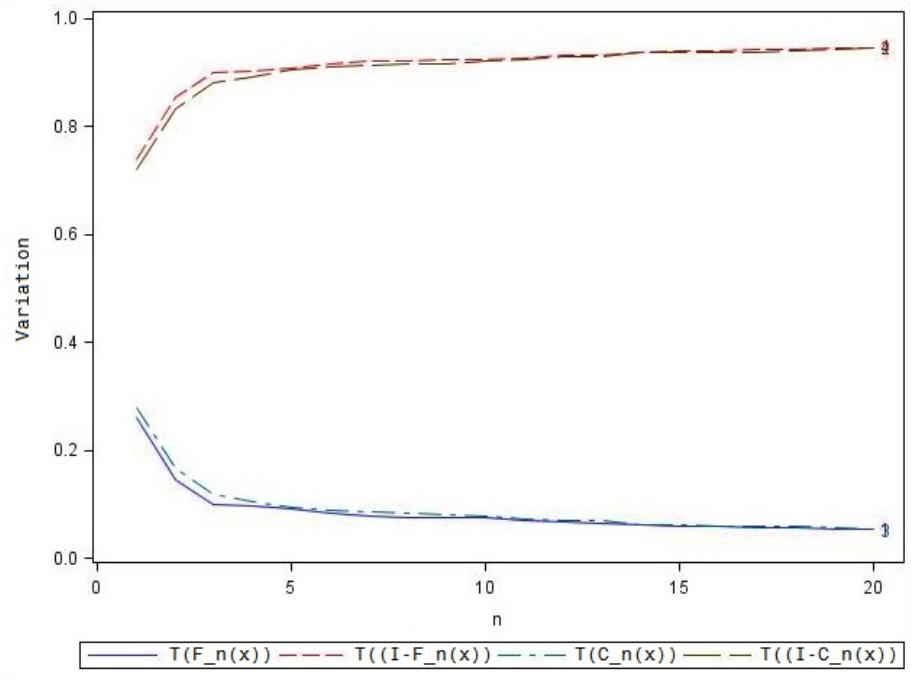


Figure 2.11: Graphical depiction of total variation in F_n and C_n.

■

2.6 The Discrete Pulse Transform

The Discrete Pulse Transform (DPT) [8, 17, 30] is a mapping of a sequence $x = \langle x_i \rangle_{i=-\infty}^{\infty}$ onto a vector of sequences $r^{(n)} = D_n(x)$, at different 'resolution levels', such that

$$DPT(x) = [D_1(x), D_2(x), ..., D_N(x), D_0(x)],$$

where $D_0(x) = C_N x$ or $F_N x$ and $D_n(x) = (I - C_n)x$ or $D_n(x) = (I - F_n)x$ with $N > 0$.

There are two equivalent natural primary choices for such a decomposition procedure, based on the smoothers C_n, or F_n. Considering the first choice, the decomposition proceeds recursively as follows: the sequence x is initially divided by C_1 into a 'smoother' sequence $C_1 x$ and the highest resolution sequence $(I - C_1)x = D_1(x) = r_{(1)}$. The smoother part $C_1 x$ is then separated by C_2 to yield the second 'resolution component' $r^{(2)} = D_2(x) = (I - C_2)C_1 x$ and the smoother part $C_2 C_1 x$. This is continued until after the N^{th} separation only a constant sequence $D_0(x)$ remains. A similar method exists for the decomposition of a sequence x by the flooring operator F_n.

Let ϕ_{ns} be the sequences each containing a removed pulse from x of size s at level n, then any sequence x can be written as the sum of all its pulses removed by $LULU$ operators, providing a decomposition of the sequence x,

$$x = \sum_{n=1}^{N} \sum_{s=1}^{\gamma_n} \phi_{ns}. \tag{2.1}$$

where γ_n is the number of pulses of size n. This decomposition obtained via the DPT provides a breakdown of a sequence into scales $n = 1, 2, ..., N$, thus a multiscale representation.

2.6.1 The Roadmaker's Algorithm

The Roadmaker's algorithm [16, 17] is an alternative algorithm to decompose a sequence x into constant pulses as the direct method (i.e. using the theoretical definition as illustrated in Example 2.1) described above is computationally intensive as n becomes large. Define the features n-bump as a constant upward pulse of size n and an n-pit as a constant downward pulse of size n. Then the Roadmaker's algorithm sequentially removes all n-bumps and n-pits for increasing values of n:

1. Suppose x is a constant upward n-pulse, then we define B_n to level x by setting x equal to the largest of its neighbours.

2. Suppose x is a constant downward n-pulse, then we define P_n to fill x by setting x equal to the smallest of its neighbours.

Whenever we extract from x, we also store the pieces that were subtracted or added. If x is a sequence of length N, then the following operations sequentially decompose x:

B_1 Level all 1-bumps.

P_1 Fill all 1-pits.

B_2 Level all 2-bumps.

P_2 Fill all 2-pits.

... B_3 Level all N-bumps.

P_3 Fill all N-pits.

The total sum of the pieces obtained from the extractions then results in the original sequence, namely the DPT decomposition as in equation 2.1.

Note that the application of the Roadmaker's algorithm is directly equivalent to applying DPT to any sequence x. However, because application of the DPT directly is computionally taxing, the Roadmakers algorithm is preferred. In particular, the Roadmaker's algorithm has $O(n)$ complexity. Note that B_n is equivalent to L_n, and P_n to U_n.

Example 2.5. Consider the same sequence $x = \{1, 1, 5, 4, 7, 1, 2\}$ in Example 2.1 and 2.3.

For $n = 1$, the subsequences at $i = 1, 2$ are $\{0, 1, 1\}$ and $\{1, 1, 5\}$ do not form bumps of size 1 - so B_1 ignores them. However, at $i = 3$, $\{1, 5, 4\}$ is a bumps since 5 exceeds both 1 and 4, thus B_1 brings 5 to the maximum of its neighbours by subtracting 1. The same arguments holds for $i = 4, 5, 6$, and 7, in particular, $\{4, 7, 1\}$ and $\{1, 2, 0\}$ are bumps and B_1 subtracts 3 and 1 respectively to level them out. The resultant sequence after applying B_1 is then $\{1, 1, 4, 4, 4, 1, 1\}$ and, since it does not contain any pits, we store the extractions in a new sequence as $\{0, 0, 1, 0, 3, 0, 1\}$.

The decompositions at different values for n are given in Table 2.2 below:

n	Extractions						
1	0	0	1	0	3	0	1
2	0	0	0	0	0	0	0
3	0	0	3	3	3	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1
sum	1	1	5	4	7	1	2

Table 2.2: The DPT decomposition of x into constant pulses using the Roadmaker's algorithm.

■

2.7 *LULU* operators and DPT for multidimensional arrays

The *LULU* operators have been extended from one dimension to multidimensions in [2]. Their extension preserves all essential properties such as consistent separation, total variation and shape preservation (see [28]). However, since $\mathbb{Z}^d$ is only partially ordered[3], the concept of connectivity is used to assist in the definition of *LULU* operators for multidimensional arrays. The morphological concept of a set connection [35] is given below:

Definition 2.6. Let B be an arbitrary nonempty set. A family $\mathcal{C}$ of subsets of B is called a connected class or a connection on B if:

[3]The definition of a partially ordered space [28] is as follows: Let A be a set and R a relation in A. R is a partial order (*partial order relation*) if the following are satisfied:

1. R is reflexive: $(a, a) \in R$, $\forall a \in A$.
2. R is anti-symmetric: $(a, b) \in R$ and $(b, a) \in R$ implies $a = b$.
3. R is transitive: $(a, b) \in R$ and $(b, c) \in R$ imply $(a, c) \in R$.

i. $\emptyset \in \mathcal{C}$,

ii. $\{x\} \in \mathcal{C}$ for all $x \in B$,

iii. $\{C_i : i \in I\} \subseteq \mathcal{C}, \bigcap_{i \in I} C_i \neq \emptyset \Rightarrow \bigcup_{i \in I} C_i \in \mathcal{C}$.

If C belongs to a connection $\mathcal{C}$ then C is called connected.

In addition to the definition of a connected class $\mathcal{C}$, we assume that the set $\mathbb{Z}^d$ equipped with connection also satisfies the following conditions [2]:

- $\mathbb{Z}^d \in \mathcal{C}$.
- For any $a \in \mathbb{Z}^d, E_a(C) \in \mathcal{C}$ whenever $C \in \mathcal{C}$ so that C is translation invariant, where E_a is the shift operator for a shift of a.
- If $V, W \in \mathcal{C}$ and $V \subsetneq W$, then there exists $x \in W \setminus V$ such that $V \bigcup \{x\} \in \mathcal{C}$.

Definition 2.7. Given a point $x \in \mathbb{Z}^d$ and $n \in \mathbb{N}$ we denote by $\mathcal{N}_n(x)$ the set of all connected sets of size $n+1$ that contain point x [2], that is

$$\mathcal{N}_n(x) = \{V \in \mathcal{C} : x \in V, \text{card}(V) = n+1\}.$$

2.7.1 Intricasy of $\mathcal{N}_n(x)$

Some illustrations are given below for $d = 2$ dimensions using 4-connectivity[4]. Figure 2.12 shows all the possible connected sets of size $n+1 = 1+1$ containing x and Figure 2.13 shows all connections of size $n+1 = 2+1$ containing x.

Figure 2.12: $\mathcal{N}_1(x)$ for $\mathbb{Z}^2$.

[4] 4-connectivity refers to the immediate adjacent elements of $x \in \mathbb{Z}^2$.

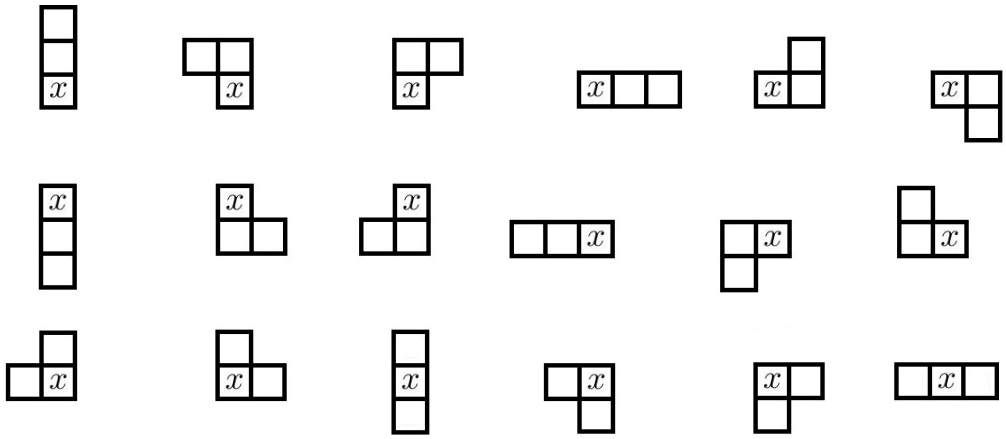


Figure 2.13: $\mathcal{N}_2(x)$ for $\mathbb{Z}^2$.

As is deducible from above, if n in $\mathcal{N}_n(x)$ increases, then so will the complexity in listing the unique connected sets. We provide programs in SAS[5] that determine the number of elements in the set $\mathcal{N}_n(x)$ for any $n > 1$ (see the Appendix). Table 2.3 contains the computed numbers.

n	card $(N_n(x))$	card $(N_n(x))$ /card $(N_{n-1}(x))$
1	4	
2	18	4.5
3	72	4
4	213	$2.958\dot{3}$
5	596	2.7981
6	1628	2.7315
7	4484	2.7543
8	12200	2.7208
9	33316	2.7308
10	90252	2.7090

Table 2.3: The number of elements in the set $N_n(x)$ for $n = 1, 2, ..., 8$ in $\mathbb{Z}^2$.

To get the general idea of how connected sets are structured, we display the sum of the connected sets in matrices (since for $n = 8$ we have 12200 outputs to show). Figures 2.16 to 2.23 illustrate the overlapping of connected sets. That is, each number in the respective matrices represents the number of times connected sets overlap for a particular element when we sum all the possible sets aligned at x. This is illustrated in Figure 2.14. Note how all the sets are aligned at the centre element x (shown with bold lines) and the elements which uniquely defines each set in $\mathcal{N}_2(x)$ protude from x at each level.

[5]SAS 9.4 (SAS Institute)

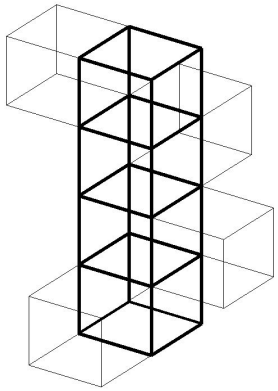


Figure 2.14: Stacking of all possible connected sets in $\mathcal{N}_1(x)$ for $\mathbb{Z}^2$.

For each of the summed matrices below from Figures 2.15 to 2.23, let $S_n = \left[s_{ij}^{(n)}(x)\right]$ denote the matrix of the sum of all possible connected sets of $\mathcal{N}_n(x)$ in $\mathbb{Z}^2$. For example, the sum of connected sets in $\mathcal{N}_1(x)$ is given by

$$S_1 = \begin{bmatrix} s_{11}^{(1)}(x) & s_{12}^{(1)}(x) & s_{13}^{(1)}(x) \\ s_{21}^{(1)}(x) & s_{22}^{(1)}(x) & s_{23}^{(1)}(x) \\ s_{31}^{(1)}(x) & s_{32}^{(1)}(x) & s_{33}^{(1)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

```
0  1  0
1  4  1
0  1  0
```

Figure 2.15: S_1 in $\mathbb{Z}^2$.

```
0  0  1  0  0
0  2  6  2  0
1  6 18  6  1
0  2  6  2  0
0  0  1  0  0
```

Figure 2.16: S_2 in $\mathbb{Z}^2$.

0	0	0	1	0	0	0
0	0	3	6	3	0	0
0	3	11	30	11	3	0
1	6	30	72	30	6	1
0	3	11	30	11	3	0
0	0	3	6	3	0	0
0	0	0	1	0	0	0

Figure 2.17: S_3 in $\mathbb{Z}^2$.

0	0	0	0	1	0	0	0	0
0	0	0	4	6	4	0	0	0
0	0	6	17	27	17	6	0	0
0	4	17	36	95	36	17	4	0
1	6	27	95	213	95	27	6	1
0	4	17	36	95	36	17	4	0
0	0	6	17	27	17	6	0	0
0	0	0	4	6	4	0	0	0
0	0	0	0	1	0	0	0	0

Figure 2.18: S_4 in $\mathbb{Z}^2$.

0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	5	6	5	0	0	0	0
0	0	0	10	23	33	23	10	0	0	0
0	0	10	34	64	86	64	34	10	0	0
0	5	23	64	106	275	106	64	23	5	0
1	6	33	86	275	596	275	86	33	6	1
0	5	23	64	106	275	106	64	23	5	0
0	0	10	34	64	86	64	34	10	0	0
0	0	0	10	23	33	23	10	0	0	0
0	0	0	0	5	6	5	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0

Figure 2.19: S_5 in $\mathbb{Z}^2$.

0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	6	6	6	0	0	0	0
0	0	0	0	15	29	41	29	15	0	0	0
0	0	0	20	57	100	120	100	57	20	0	0
0	0	15	57	131	198	260	198	131	57	15	0
0	6	29	100	198	309	744	309	198	100	29	6
1	6	41	120	260	744	1628	744	260	120	41	6
0	6	29	100	198	309	744	309	198	100	29	6
0	0	15	57	131	198	260	198	131	57	15	0
0	0	0	20	57	100	120	100	57	20	0	0
0	0	0	0	15	29	41	29	15	0	0	0
0	0	0	0	0	6	6	6	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0

Figure 2.20: S_6 in $\mathbb{Z}^2$.

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	7	6	7	0	0	0	0	0	0
0	0	0	0	0	21	35	51	35	21	0	0	0	0	0
0	0	0	0	35	86	147	166	147	86	35	0	0	0	0
0	0	0	35	114	236	335	400	335	236	114	35	0	0	0
0	0	21	86	236	409	601	742	601	409	236	86	21	0	0
0	7	35	147	335	601	868	2084	868	601	335	147	35	7	0
1	6	51	166	400	742	2084	4484	2084	742	400	166	51	6	1
0	7	35	147	335	601	868	2084	868	601	335	147	35	7	0
0	0	21	86	236	409	601	742	601	409	236	86	21	0	0
0	0	0	35	114	236	335	400	335	236	114	35	0	0	0
0	0	0	0	35	86	147	166	147	86	35	0	0	0	0
0	0	0	0	0	21	35	51	35	21	0	0	0	0	0
0	0	0	0	0	0	7	6	7	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 2.21: S_7 in $\mathbb{Z}^2$.

0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	8	6	8	0	0	0	0	0	0	0
0	0	0	0	0	0	28	41	63	41	28	0	0	0	0	0	0
0	0	0	0	0	56	121	208	224	208	121	56	0	0	0	0	0
0	0	0	0	70	200	394	535	620	535	394	200	70	0	0	0	0
0	0	0	56	200	482	775	1077	1209	1077	775	482	200	56	0	0	0
0	0	28	121	394	775	1267	1696	2115	1696	1267	775	394	121	28	0	0
0	8	41	208	535	1077	1696	2433	5632	2433	1696	1077	535	208	41	8	0
1	6	63	224	620	1209	2115	5632	12200	5632	2115	1209	620	224	63	6	1
0	8	41	208	535	1077	1696	2433	5632	2433	1696	1077	535	208	41	8	0
0	0	28	121	394	775	1267	1696	2115	1696	1267	775	394	121	28	0	0
0	0	0	56	200	482	775	1077	1209	1077	775	482	200	56	0	0	0
0	0	0	0	70	200	394	535	620	535	394	200	70	0	0	0	0
0	0	0	0	0	56	121	208	224	208	121	56	0	0	0	0	0
0	0	0	0	0	0	28	41	63	41	28	0	0	0	0	0	0
0	0	0	0	0	0	0	8	6	8	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 2.22: S_8 in $\mathbb{Z}^2$.

0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	9	6	9	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	36	47	77	47	36	0	0	0	0	0	0	0	0
0	0	0	0	0	0	84	162	286	294	286	162	84	0	0	0	0	0	0	0
0	0	0	0	0	126	321	623	816	944	816	623	321	126	0	0	0	0	0	0
0	0	0	0	126	400	897	1373	1846	1993	1846	1373	897	400	126	0	0	0	0	0
0	0	0	84	321	897	1608	2489	3154	3614	3154	2489	1608	897	321	84	0	0	0	0
0	0	36	162	623	1373	2489	3624	4874	5866	4874	3624	2489	1373	623	162	36	0	0	0
0	9	47	286	816	1846	3154	4874	6708	15540	6708	4874	3154	1846	816	286	47	9	0	0
1	6	77	294	944	1993	3614	5866	15540	33316	15540	5866	3614	1993	944	294	77	6	1	0
0	9	47	286	816	1846	3154	4874	6708	15540	6708	4874	3154	1846	816	286	47	9	0	0
0	0	36	162	623	1373	2489	3624	4874	5866	4874	3624	2489	1373	623	162	36	0	0	0
0	0	0	84	321	897	1608	2489	3154	3614	3154	2489	1608	897	321	84	0	0	0	0
0	0	0	0	126	400	897	1373	1846	1993	1846	1373	897	400	126	0	0	0	0	0
0	0	0	0	0	126	321	623	816	944	816	623	321	126	0	0	0	0	0	0
0	0	0	0	0	0	84	162	286	294	286	162	84	0	0	0	0	0	0	0
0	0	0	0	0	0	0	36	47	77	47	36	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	9	6	9	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 2.23: S_9 in $\mathbb{Z}^2$.

From plain view, these matrices show little mathematical insight since the only thing we are certain of from these matrices is the fact that the number of overlaps decrease as we move away from the centre. However, if we start summarising the matrices above, we arrive at some intriguing results.

Define the difference operator Δ on a ordered sequence $x = \{x_1, x_2, x_3, ...\}$ as,

$$\Delta x_i = x_i - x_{i-1}$$

for $i = 2, 3, 4, ...$ Applying Δ twice to a sequence yields:

$$\begin{aligned}\Delta^2 x_i &= \Delta(\Delta x_i) \\ &= \Delta(x_i - x_{i-1}) \\ &= \Delta x_i - \Delta x_{i-1} \\ &= (x_i - x_{i-1}) - (x_{i-1} - x_{i-2}) \\ &= x_i - 2x_{i-1} + x_{i-2}.\end{aligned}$$

Note that for any sequence x, it is true that if $\Delta^k x$ (for some $k \geq 0$) yields a nonzero constant sequence, then the sequence x can be written as a polynomial of order k.

Example 2.6. Consider a quadratic sequence given by $x = \{x_n\}_{n=1}^{\infty} = \{27, 33, 41, 51, 63, 77, ...\}$. This is a polynomial of order 2 since we can capture x_n with a second order polynomial function

$$x_n = an^2 + bn + c,$$

where coefficients a, b and c are real numbers. To solve the coefficients, we substitute the values of the sequence in the polynomial with corresponding values of n:

$$\begin{aligned}n = 1: &\quad 27 = a + b + c \\ n = 2: &\quad 33 = 4a + 2b + c \\ n = 3: &\quad 41 = 9a + 3b + c\end{aligned}$$

This can be rewritten as an augmented matrix, in which the coefficients can be obtained by solving the system of equations. Thus,

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 27 \\ 4 & 2 & 1 & 33 \\ 9 & 3 & 1 & 41 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 23 \end{array}\right],$$

which implies that the sequence may be predicted with the following equation:

$$x_n = n^2 + 3n + 23.$$

To establish that the order of the polynomial may be determined by the number of the difference needed to be taken until a nonzero sequence occurs, we note that,

- $\Delta x = \{6, 8, 10, 12, 14, ...\}$, and
- $\Delta^2 x = \Delta(\Delta x) = \{2, 2, 2, 2, ...\}$.

Therefore $\Delta^k x$ is a constant sequence of two's, and the order of the polynomial is $k = 2$.

■

Consider the first row of each matrix S_n for $n = 1, 2, 3, \ldots$. The sequence given by the nonzero element in the first row of S_n is

$$\left\{s_{1,n+1}^{(n)}\right\}_{n=1}^{\infty} = \left\{s_{12}^{(1)}, s_{13}^{(2)}, s_{14}^{(3)}, \ldots\right\} = \{1, 1, 1, \ldots\},$$

which implies $\left\{s_{1,n+1}^{(n)}\right\}_{n=1}^{\infty}$ is a polynomial sequence of *order* 0 since the sequence itself is *constant* containing only 1's. That is, since $\Delta^k s_{1,n+1}^{(n)}$ is constant for $k = 0$, the sequence is a polynomial of order 0.

The second row of each matrix S_n yields three sequences from the three nonzero elements. These are given by (from the left sequence to right):

- $\left\{s_{2,n}^{(n)}\right\}_{n=1}^{\infty} = \left\{s_{21}^{(1)}, s_{22}^{(2)}, s_{23}^{(3)}, \ldots\right\} = \{1, 2, 3, \ldots\}$ which is a polynomial of order 1 since $\Delta s_{2,n}^{(n)}$ yields a constant sequence of 1's.
- $\left\{s_{2,n+2}^{(n+1)}\right\}_{n=1}^{\infty} = \left\{s_{23}^{(2)}, s_{24}^{(3)}, s_{25}^{(4)}, \ldots\right\} = \{6, 6, 6, \ldots\}$ which is a polynomial of order 0 since the sequence is a constant of 6's.
- $\left\{s_{2,n+2}^{(n)}\right\}_{n=1}^{\infty} = \left\{s_{2,n}^{(n)}\right\}_{n=1}^{\infty}$ by symmetry of S_n.

If we summarise the sequences so far, we have that the sequences in pyramid form:

$$\begin{array}{ccccccc}
 & & & \left\{s_{1,n+1}^{(n)}\right\} & & & \\
 & & \left\{s_{2,n}^{(n)}\right\} & \left\{s_{2,n+2}^{(n+1)}\right\} & \left\{s_{2,n+2}^{(n)}\right\} & & \\
\iddots & \iddots & & \vdots & & \ddots & \ddots
\end{array}$$

have polynomials of *order*,

$$\begin{array}{ccccc}
 & & 0 & & \\
 & 1 & 0 & 1 & \\
\iddots & \iddots & \vdots & \ddots & \ddots
\end{array}$$

and their *constant sequences*,

$$\begin{array}{ccccc}
 & & c_1 & & \\
 & c_1 & c_6 & c_1 & \\
\iddots & \iddots & \vdots & \ddots & \ddots
\end{array}$$

where c_i defines the constant sequence with the recurring number i.

Consider now the third row of each S_n for $n \geq 2$. Starting on the left of the nonzero elements:

- $\left\{s_{3,n}^{(n+1)}\right\}_{n=1}^{\infty} = \left\{s_{31}^{(2)}, s_{32}^{(3)}, s_{33}^{(4)}, \ldots\right\} = \{1, 3, 6, 10, \ldots\}$ which is a polynomial sequence of order 2 since $\Delta^2 s_{3,n}^{(n+1)}$ yields a constant sequence of 1's.

- $\left\{s_{3,n+2}^{(n+2)}\right\}_{n=1}^{\infty} = \left\{s_{33}^{(3)}, s_{34}^{(4)}, s_{35}^{(5)}, ...\right\} = \{11, 17, 23, ...\}$ which is a polynomial sequence of order 1 since $\Delta s_{3,n+2}^{(n+2)}$ yields a constant sequence of 6's.
- $\left\{s_{3,n+4}^{(n+3)}\right\}_{n=1}^{\infty} = \left\{s_{35}^{(4)}, s_{36}^{(5)}, s_{37}^{(6)}, ...\right\} = \{27, 33, 41, 51, ...\}$ which is a polynomial sequence of order 2 since $\Delta^2 s_{3,n+4}^{(n+3)}$ yields a constant sequence of 2's.
- $\left\{s_{3,n+4}^{(n+2)}\right\}_{n=1}^{\infty} = \left\{s_{3,n+2}^{(n+2)}\right\}_{n=1}^{\infty}$ by symmetry of S_n.
- $\left\{s_{3,n+4}^{(n+1)}\right\}_{n=1}^{\infty} = \left\{s_{3,n}^{(n+1)}\right\}_{n=1}^{\infty}$ by symmetry of S_n.

Continuing in this fashion, we obtain the following pyramids through the evidence from the sums of the connected sets $S_n(x)$. The order of the polynomials inherent in the sequence of summed connected sets is found as:

$$\begin{array}{ccccccccccc}
 & & & & & 0 & & & & & \\
 & & & & 1 & 0 & 1 & & & & \\
 & & & 2 & 1 & 2 & 1 & 2 & & & \\
 & & 3 & 2 & 3 & 2 & 3 & 2 & 3 & & \\
 & 4 & 3 & \circ & 3 & \circ & 3 & \circ & 3 & 4 & \\
\iddots & \iddots & \vdots & \iddots & \iddots & \vdots & \ddots & \ddots & \vdots & \ddots & \ddots
\end{array}$$

and their *constant sequences*:

$$\begin{array}{ccccccccccc}
 & & & & & c_1 & & & & & \\
 & & & & c_1 & c_6 & c_1 & & & & \\
 & & & c_1 & c_6 & c_2 & c_6 & c_1 & & & \\
 & & c_1 & c_6 & c_3 & c_{12} & c_3 & c_6 & c_1 & & \\
 & c_1 & c_6 & \circ & c_{18} & \circ & c_{18} & \circ & c_6 & c_1 & \\
\iddots & \iddots & \vdots & \iddots & \iddots & \vdots & \ddots & \ddots & \vdots & \ddots & \ddots
\end{array}$$

We can see a cascading effect exists in the structure of these sequences since each constant sequence is exactly the sum of the two directly above it[6] (excluding the middle). For example, $c_3 = c_1 + c_2$ and $c_{12} = c_6 + c_6$.

For the order of the polynomials, we see a tendency for vertical pairs and row-alternating numbers in the triangle and that a linear relationship exists along the diagonals. This means that the order complexity of the polynomial sequences obtained from S_n increases linearly as n increases.

Since we can only obtain S_n for $n = 1, 2, ..., 9$ because of the intensive computing power required to run any number n that is greater than 9, we lack the information required to complete the last row of both the order and constant sequence pyramids (indicated by $\circ$). If we postulate the following pattern for row 5 of constant sequences as,

$$c_1 \quad c_6 \quad c_4 \quad c_{18} \quad c_6 \quad c_{18} \quad c_4 \quad c_6 \quad c_1,$$

[6] Much like Pascal's triangle [21].

then it is possible to deduce that the order of the polynomial for sequences number 3, 5, and 7 of row 5 is 4. From this, the complete row for row 5 is,

$$4\ 3\ 4\ 3\ 4\ 3\ 4\ 3\ 4.$$

It is interesting to see how intricately the connections are designed by a simple definition such as $\mathcal{N}_n(x)$. How the rows, columns, diagonals, and elements satisfy strict mathematical rules (when we add up connections of specific order and form sequences from them) and how they ultimately depend on each other.

2.7.2 *LULU* operators on $\mathbb{Z}^2$

The definitions for L_n and U_n in general 2-dimensions are given below [2]:

Definition 2.8. Let $f \in \mathcal{A}(\mathbb{Z})$ and $n \in \mathbb{N}$. Then for $x \in \mathbb{Z}^2$,

$$L_n(f)(x) = \max_{V \in \mathcal{N}_n(x)} \min_{y \in V} f(y),$$

$$U_n(f)(x) = \min_{V \in \mathcal{N}_n(x)} \max_{y \in V} f(y).$$

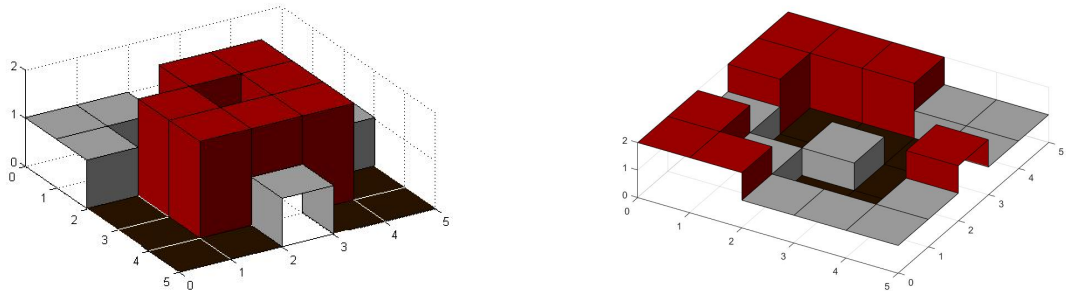
For example, refering to Figure 2.13, $L_2(x)$ first determines the minimum of each individual 12 connections. From there, $L_2(x)$ then returns the maximum of these 12 minimums in $\mathbb{Z}^2$.

Analogous to their case in 1-dimension, L_n and U_n operating in $\mathbb{Z}^2$ also removes upward and downward pulses. Here, we refer to them also as local maximum and minimum sets respectively [2]:

Definition 2.9. A connected subset V of $\mathbb{Z}^d$ is a

1. Local maximum set of $f \in \mathcal{A}(\mathbb{Z}^2)$ if $\sup_{y \in \text{adj}(V)} f(y) < \inf_{x \in V} f(x)$.
2. Local minimum set of $f \in \mathcal{A}(\mathbb{Z}^2)$ if $\inf_{y \in \text{adj}(V)} f(y) < \sup_{x \in V} f(x)$.

In Figure 2.24 we graphically present an example of local maximum and minimums sets of size 7 in $\mathbb{Z}^2$. We see that the adjacent elements of the local maximum set are strictly less than the set itself. A similar observation holds for the local minimum set.



a) b)

Figure 2.24: Local maximum (a) and minimum (b) sets of size 7 in $\mathbb{Z}^2$.

The characterisations of L_n and U_n operators are preserved from their extension to multidimensions [2]. The operator L_n only removes local maximum sets of size n or smaller in f. That is to say that it will not create any new local minimum sets except the case when, as a consequence of the removal of the local maximum set[7], it enlarges an existing minimum or joins two or more of them to become one local minimum set. Furthermore, $L_n(f) = f$ if and only if f does not possess any local maximum sets of size n or less. A similar case holds for U_n [2]. We focus in this text on the case $d = 2$, that is $\mathbb{Z}^2$, for an image domain.

2.7.3 The Discrete Pulse Transform

The Discrete Pulse Transform (DPT) of $f \in \mathbb{Z}^2$ [2] is obtained iteratively by applying the L_n and U_n operators with n increasing from 1 to N:

$$DPT(f) = (D_1(f), D_2(f), ..., D_N(f)),$$

where

$$\begin{aligned} D_1(f) &= (I - P_1)(f), \\ D_n(f) &= (I - P_n) \circ Q_{n-1}(f) \quad \text{for } n = 2, ..., N, \end{aligned}$$

and $P_n = L_n \circ U_n$ or $P_n = U_n \circ L_n$ and $Q_n = P_n \circ ... \circ P_1, n \in \mathbb{N}$.

Let ϕ_{ns} be the zero matrices each containing a removed pulse from f of size s at position n, then any multiscale sequence f can be written as the sum of all its pulses removed by $LULU$ operators, providing a multiscale decomposition of the image f,

$$f = \sum_{n=1}^{N} \sum_{s=1}^{\gamma_n} \phi_{ns}.$$

Example 2.7. Consider an image f of grey-scale values given by:

$$f = \begin{bmatrix} 1 & 3 & 4 & 4 \\ 3 & 3 & 5 & 4 \\ 6 & 2 & 8 & 4 \\ 9 & 5 & 5 & 1 \end{bmatrix}$$

The decomposition of f using the DPT (with 4-connectivity) with $P_n = U_n L_n$ is displayed below. Recall that in the one-dimensional case, zeros are appended to the finite sequence so that it becomes a bi-infinite sequence. The same methodology is applied here. The finite matrix f is appended with zeroes surrounding

[7] Recall that L_n removes a local maximum set V by subtracting the appropriate amount such that V is a constant set with values equal to the supremum of adj(V).

f itself. Thus, in actuality, the decomposition of f using the DPT is performed on f^* given as

$$f^* = \begin{bmatrix} & \ddots & \vdots & \vdots & \vdots & \vdots & \iddots & \\ \ddots & \ddots & 0 & 0 & 0 & 0 & \iddots & \iddots \\ \cdots & 0 & 1 & 3 & 4 & 4 & 0 & \cdots \\ \cdots & 0 & 3 & 3 & 5 & 4 & 0 & \cdots \\ \cdots & 0 & 6 & 2 & 8 & 4 & 0 & \cdots \\ \cdots & 0 & 9 & 5 & 5 & 1 & 0 & \cdots \\ \iddots & \iddots & 0 & 0 & 0 & 0 & \ddots & \ddots \\ & \iddots & \vdots & \vdots & \vdots & \vdots & \ddots & \end{bmatrix}$$

Since $D_1(f) = (I - P_1)(f) = f - P_1 f = f - U_1 L_1(f)$, and

$$U_1(L_1(f)) = U_1 \begin{bmatrix} 1 & 3 & 4 & 4 \\ 3 & 3 & 5 & 4 \\ 6 & 2 & 5 & 4 \\ 6 & 5 & 5 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 4 & 4 \\ 3 & 3 & 5 & 4 \\ 6 & \mathbf{3} & \mathbf{5} & 4 \\ \mathbf{6} & 5 & 5 & 1 \end{bmatrix},$$

where the changes of $Q_{n-1}f$ after an application of the Q_n operator are given in bold. Thus, we have that,

$$D_1(f) = f - U_1 L_1(f) = \begin{bmatrix} 1 & 3 & 4 & 4 \\ 3 & 3 & 5 & 4 \\ 6 & 2 & 8 & 4 \\ 9 & 5 & 5 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 3 & 4 & 4 \\ 3 & 3 & 5 & 4 \\ 6 & 3 & 5 & 4 \\ 6 & 5 & 5 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 3 & 0 \\ 3 & 0 & 0 & 0 \end{bmatrix}$$

Similarly,

$$Q_2(f) = \begin{bmatrix} 1 & 3 & 4 & 4 \\ 3 & 3 & 5 & 4 \\ \mathbf{5} & 3 & 5 & 4 \\ \mathbf{5} & 5 & 5 & 1 \end{bmatrix} \text{ and } D_2(f) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Since there are no pulses of size 3, 4 or 5, $Q_5(f) = Q_4(f) = Q_3(f) = Q_2(f)$, and $D_5(f) = D_4(f) = D_3(f) = \mathbf{0}$, where $\mathbf{0} : 4 \times 4$ is the zero matrix.

$$Q_6(f) = \begin{bmatrix} 1 & 3 & 4 & 4 \\ 3 & 3 & \mathbf{4} & 4 \\ \mathbf{4} & 3 & \mathbf{4} & 4 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} & 1 \end{bmatrix} \text{ and } D_6(f) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Similarly, $Q_7(f) = Q_8(f) = Q_9(f) = Q_6(f)$ since there are no pulses of size 7, 8, or 9, with corresponding

zero matrices $D_7(f), D_8(f)$, and $D_9(f)$. To end this:

$$Q_{10}(f) = \begin{bmatrix} 1 & 3 & \mathbf{3} & \mathbf{3} \\ 3 & 3 & \mathbf{3} & \mathbf{3} \\ \mathbf{3} & 3 & \mathbf{3} & \mathbf{3} \\ \mathbf{3} & \mathbf{3} & \mathbf{3} & 1 \end{bmatrix} \text{ and } D_{10}(f) = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$Q_{14}(f) = \begin{bmatrix} 1 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 \end{bmatrix} \text{ and } D_{14}(f) = \begin{bmatrix} 0 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 0 \end{bmatrix}$$

and

$$Q_{16}(f) = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \text{ and } D_{16}(f) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

■

2.8 Conclusion

In Section 2.7.1 4-connectivity was used to define the sets in $\mathcal{N}_n(x)$. In Definition 2.8 $\mathcal{N}_n(x)$ allows for any connected set definition satisfying Definition 2.7, of which 8-connectivity is also valid. The question of what connection to use therefore arises. We investigate this in the next chapter by empirically investigating the dependence structure of local pixel neighbours.

Chapter 3

An empirical study of dependence structures in images

3.1 Introduction

From the definition of the $LULU$ operators in two dimensions, the operators act on a pixel x using information from the neighbours of x as illustrated in Figures 2.12 and 2.13. Thus, if independence between pixels is not assumed (and should not be because not all images are white noise as they contain structure) then we must incorporate the dependence between pixels with, say, some matrix-covariance structure. Using the property of global independence and local dependence shown herein, we motivate the connectivity choice for the $LULU$ operators.

We investigate the property of *global independence* and *local dependence*. The idea is that for any one pixel, the dependence it exhibits in relation to neighbouring pixels decreases as the distance[1] between that pixel and the neighbouring pixel increases. The assumption is not unsupported. For a group of pixels in an image that dictate a particular object should be in close proximity to one another as well as have high correlation with each other in that group. For a pixel x in an image, the following property is then postulated:

The correlations of x to its neighbours $\{y \in \mathcal{N}(x)\}$ decreases as the distance between x and y in the image increases, where $\mathcal{N}(x)$ is the set of neighbouring pixels of x. That is, at some distance, the correlations are statistically insignificant.

The rest of the chapter is presented as follows. Section 3.2 discusses the theory required to perform tests of independence between any two pixels in a video stream, Section 3.3 performs the tests of indepen-

[1] Discrete 4-connected distance on $\mathbb{Z}^2$, defined as $D\left(x_{(i_1,j_1)}, x_{(i_2,j_2)}\right) = |i_1 - i_2| + |j_1 - j_2|$ for image pixel locations (i_1, j_1) and (i_2, j_2).

dence between pixels in the dependence structure with arbitrary videos, and we end in Section 3.5 with concluding remarks.

3.2 Testing dependence of neighbouring pixels

For this study our data set will consist of a number of videos which provides a sample of images. Since a video stream is an ordered sequence of image frames, each frame has dependence on previous realisations. In other words, each successive pixel can be explained (to some degree) by its previous values. This allows us to model each pixel with an autoregressive (AR) time series model. Furthermore, our video database consists of recordings of stationary scenes (i.e. the camera does not pan or zoom in the process of recording) since most surveillance systems are of this type, thus all our models are covariance-stationary. Note that we discuss the case of non-stationary scenes in the conclusion. Therefore, we can implement the test of independence between two univariate covariance stationary time series from Haugh [12]. We begin with some definitions and notations:

For each pixel $x(i,j) \equiv x$ for position $(i,j) \in \mathbb{Z}^2$, in a video frame we fit an autoregressive time series model of order p, with p chosen such that the residuals of the model are uncorrelated. This then allows us to write a pixel x_t at time t as

$$x_t = c + \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + u_t,$$

where c is the intercept of the model, $\{\phi_1, \phi_2, ..., \phi_p\}$ are the autoregressive parameters, and u_t is white noise innovations with variance equal to σ^2.

To begin the correlation (dependence) analysis we impose, again for each pixel x_t at time t, a dependence structure $M_t^{(K)}$ with a frame containing pixel x_t and all neighbours with discrete distances less than or equal to K - this is portrayed in Figure 3.1. The notation, $x_t^{(D,d)}$, used for each neighbour of x_t reveals the discrete distance on $\mathbb{Z}^2$ from the neighbour to x_t as D, and the number (labeling) of the neighbour $(d = 1, 2, ..., 4D)$ with distance D from x_t.

For example in a single frame:

- $M_t^{(1)}$ will have 5 elements,
- $M_t^{(2)}$ will have 13 elements,
- ...
- $M_t^{(K)}$ will have $K^2 + (K+1)^2$ elements.

According to Haugh [12], if two covariance-stationary univariate time series are dependent, then their white noise innovations will also be dependent. That is, suppose $\{x_t\}_{t=-\infty}^{\infty}$ and $\{y_t\}_{t=-\infty}^{\infty}$ are two time

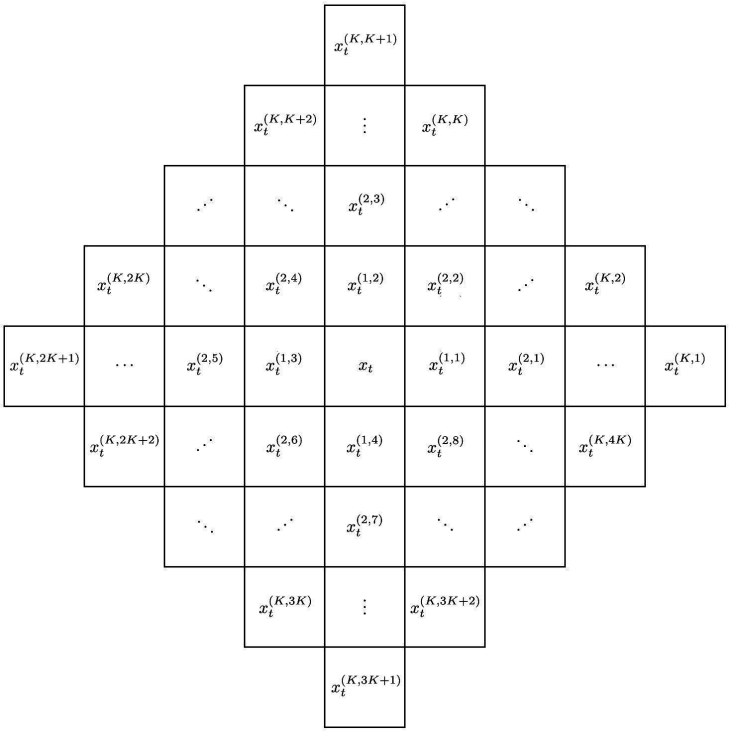


Figure 3.1: An illustration of the dependence structure $M_t^{(K)}$.

series with white innovations $\{u_t\}_{t=-\infty}^{\infty}$ and $\{v_t\}_{t=-\infty}^{\infty}$ respectively. Then if u_{t-j} and v_t are independent at all lags $j \in \mathbb{Z}$, then $\{x_t\}_{t=-\infty}^{\infty}$ and $\{y_t\}_{t=-\infty}^{\infty}$ are independent.

To measure the relationship between the two innovations at lag j, Haugh [12] uses the crosscorrelation function,

$$\rho_{uv}(j) = \frac{\gamma_{uv}(j)}{\sigma_u \sigma_v}, \quad j \in \mathbb{Z},$$

where $\gamma_{uv}(j)$ is the covariance between the innovations at lag j, with σ_u^2 and σ_v^2 the variances of u_t and v_t respectively. For a sample size of n observations from each time series, we can obtain the white noise residuals[2] $\hat{u}_t$ and $\hat{v}_t$, and estimate the cross correlations with,

$$\hat{\rho}_{uv}(j) = r_{uv}(j) = \left[\sum_{t=1}^{n} \hat{u}_t^2 \sum_{t=1}^{n} \hat{v}_t^2\right]^{-1/2} \hat{\gamma}_{uv}(j),$$

with

$$\hat{\gamma}_{uv}(j) = \begin{cases} \sum_{t=j+1}^{n} \hat{u}_{t-j}\hat{v}_t & \text{if } j = 0, 1, 2, \ldots \\ \sum_{t=|j|+1}^{n} \hat{u}_t \hat{v}_{t-|j|} & \text{if } j = -1, -2, -3, \ldots . \end{cases}$$

[2] From OLS, if the model is $E[\mathbf{y}] = \mathbf{Xb}$, where $\mathbf{y}$ contains the dependent variables, $\mathbf{X}$ the design matrix, and $\mathbf{b}$ the vector of parameters to be estimated, then $\hat{\boldsymbol{\epsilon}} = \mathbf{y} - \mathbf{Xb}$ is the vector of residuals from the model. Here, for our AR(p) model, $\mathbf{y} = \mathbf{x}_n = [x_n \;\; x_{n-1} \;\; \ldots \;\; x_{p+1}]'$, $\mathbf{X} = [\mathbf{1} \;\; \mathbf{x}_{n-1} \;\; \mathbf{x}_{n-2} \;\; \ldots \;\; \mathbf{x}_{n-p}]$ and $\mathbf{b} = [c \;\; \phi_1 \;\; \phi_2 \;\; \ldots \;\; \phi_p]'$.

Under the null hypothesis that x_t and y_t are independent, we have from Haugh [12] that,

$$s = n \sum_{j=-M}^{M} r_{uv}^2(j),$$

is asymptotically χ^2 distributed with $2M + 1$ degrees of freedom.

In light of the above, we perform the test of independence of time series (our pixel sample data) from Haugh [12] with x_t as our centre pixel and y_t as each one of its neighbours in the considered neighbourhood $M_t^{(K)}$.

3.3 Application

In this section, we perform the empirical analysis of the image pixels' dependence structure. Denote an image sequence (video) as f - with f_t the t^{th} frame of the video. Then for a video f, we use MATLAB[3], for the following:

1. For each pixel $x_t \in f_t$, we fit an AR(p) model with 500 observations from f_1 to f_{500} (i.e. 500 frames). The AR parameters are found by ordinary least squares estimation and the degree p is such that the residuals of the model are uncorrelated.

2. Impose for each pixel x_t the dependence structure $M_t^{(K)}$ (see Figure 3.1). Then from each structure at each pixel,

   (a) Find the correlation between the centre pixel time series x_t and its neighbours at lag 0 (i.e. in the same frame), $r_{x_t,y_t}(0)$, $y_t \in M_t^{(K)}$

   (b) Perform the test of independence from Haugh [12] for $M = 6$ with x_t as the centre pixel and y_t as each one of its neighbours, $y_t \in M_t^{(K)}$, and find the p-value of each test.

3. Average the results from step 2 over the whole pixel domain.

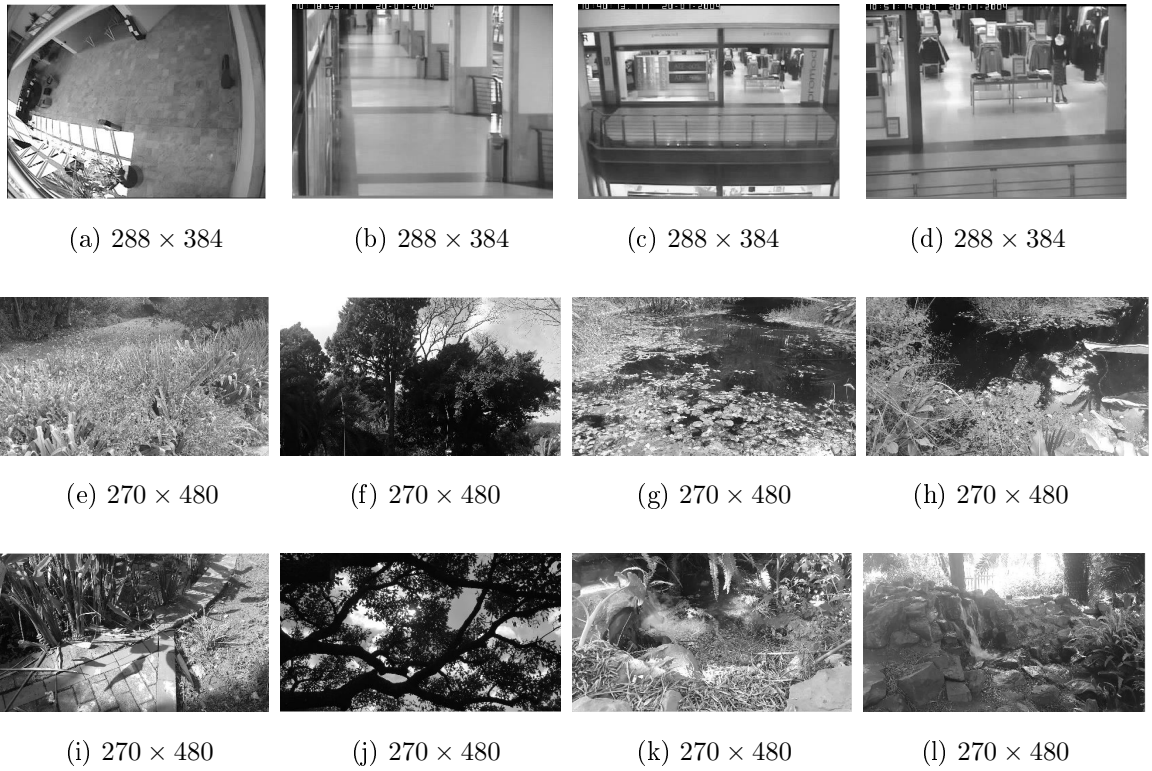
In our application, we implemented the test for different values of K and only will output $K = 3$ results. The reason for this is that information in terms of rejecting of the null hypothesis is already contained in $K = 3$ structures and by having larger values of K we will only include redundant information.

Table 3.1 contains the correlation and p-values. The results obtained from MATLAB are doubly symmetric (to 2 decimal places) about the horizontal and vertical axis. Therefore, we display one full quadrant of the average correlation and p-value structures of f. Here, the lower triangular numeric values represent the average correlations and the upper triangle the average p-values of the independence test.

Figure 3.2 contains the single frames of each video. The first four videos are indoor videos taken inside a mall[4], all of which is security footage. The others are recorded by the author[5], all of outdoor scenes.

[3] MATLAB and Statistics Toolbox Release 2016a, The MathWorks, Inc., Natick, Massachusetts, United States.
[4] Videos can be obtained from `http://homepages.inf.ed.ac.uk/rbf/CAVIAR/`.
[5] Link to videos: `https://github.com/AlexUP/MMD-Videos`.

For every video, 500 frames were considered.



(a) 288 × 384 (b) 288 × 384 (c) 288 × 384 (d) 288 × 384

(e) 270 × 480 (f) 270 × 480 (g) 270 × 480 (h) 270 × 480

(i) 270 × 480 (j) 270 × 480 (k) 270 × 480 (l) 270 × 480

Figure 3.2: A single frame of all videos with their dimensions.

From Table 3.1, we can see that the average correlation decreases and the p-value of the test increases as we venture away from the centre. This relationship is consistent throughout all the videos.

Furthermore, all videos, except for (b),(c),(d), and (j) in Table 3.1, show the characteristics of global independence and local dependence since we can reject the null hypothesis of independence between the centre pixel and neighbours at $\alpha = 10\%$ for neighbours with discrete distances less than or equal to 1. The same may not be said for videos (b),(c),(d), and (j) in Table 3.1 where we can reject the null hypothesis for the whole region of the dependence structure at $\alpha = 5\%$. Note that the 'NaN' values in Table 3.1 indicate that the pixel considered in that position is outside the limits of the dependence structure.

Table 3.2 contains the average of the autoregressive parameters of each pixel from each video. Note that zeros entries imply that the AR parameter is less than 0.001. We can see that the videos that are consistent with the assumption have significant AR parameter only at lag 1. However for videos (b),(c),(d) in Table 3.1, where the assumptions does not hold for $K = 3$, they have significant AR parameters at lags 1 and 2. This could possibly describe the need of a bigger dependence structure to incorporate the dependence in time series with AR(p) processes where $p > 1$. But in general the dependence does not extend for larger values of K.

	NaN	NaN	0.1467	NaN	NaN	NaN
NaN		0.1729	0.1290	0.1722	NaN	NaN
NaN	0.0844		0.0729	0.1422	0.1680	NaN
0.1295	0.0657	0.2767	1	0.0807	0.1251	0.1521
NaN	0.0816	0.1502	0.3643		0.1678	NaN
NaN	NaN	0.0985	0.1722	0.0990		NaN
NaN	NaN	NaN	0.1557	NaN	NaN	

(a)

	NaN	NaN	0.0147	NaN	NaN	NaN
NaN		0.0250	0.0144	0.0249	NaN	NaN
NaN	0.3949		0.0049	0.0139	0.0226	NaN
0.4745	0.5237	0.6457	1	0.0057	0.0136	0.0158
NaN	0.3941	0.4697	0.6639		0.0230	NaN
NaN	NaN	0.3862	0.5073	0.3855		NaN
NaN	NaN	NaN	0.4769	NaN	NaN	

(b)

	NaN	NaN	0.0448	NaN	NaN	NaN
NaN		0.0690	0.0531	0.0691	NaN	NaN
NaN	0.2167		0.0212	0.0464	0.0598	NaN
0.3815	0.4467	0.5873	1	0.0148	0.0321	0.0369
NaN	0.2175	0.2687	0.4225		0.0601	NaN
NaN	NaN	0.1850	0.2565	0.1850		NaN
NaN	NaN	NaN	0.2724	NaN	NaN	

(c)

	NaN	NaN	0.0306	NaN	NaN	NaN
NaN		0.0730	0.0566	0.0735	NaN	NaN
NaN	0.1976		0.0122	0.0359	0.0499	NaN
0.3777	0.4262	0.5680	1	0.0064	0.0142	0.0158
NaN	0.1992	0.2557	0.4405		0.0512	NaN
NaN	NaN	0.1344	0.2159	0.1350		NaN
NaN	NaN	NaN	0.2750	NaN	NaN	

(d)

	NaN	NaN	0.2648	NaN	NaN	NaN
NaN		0.2382	0.2108	0.2454	NaN	NaN
NaN	0.0938		0.0556	0.1592	0.2666	NaN
0.0512	0.1020	0.2475	1	0.1023	0.2524	0.2980
NaN	0.0866	0.1723	0.3006		0.2624	NaN
NaN	NaN	0.1103	0.1429	0.2382		NaN
NaN	NaN	NaN	0.0895	NaN	NaN	

(e)

	NaN	NaN	0.2609	NaN	NaN	NaN
NaN		0.1982	0.1591	0.1935	NaN	NaN
NaN	0.1870		0.0287	0.0899	0.2163	NaN
0.1402	0.2078	0.3452	1	0.0317	0.1885	0.3015
NaN	0.1987	0.2887	0.3819		0.2147	NaN
NaN	NaN	0.2094	0.2445	0.1940		NaN
NaN	NaN	NaN	0.1803	NaN	NaN	

(f)

	NaN	NaN	0.3110	NaN	NaN	NaN
NaN		0.2703	0.2591	0.2812	NaN	NaN
NaN	0.1611		0.0716	0.1567	0.2455	NaN
0.1755	0.2749	0.4708	1	0.0390	0.1647	0.2431
NaN	0.1474	0.1951	0.2555		0.2255	NaN
NaN	NaN	0.0793	0.1045	0.0854		NaN
NaN	NaN	NaN	0.0498	NaN	NaN	

(g)

	NaN	NaN	0.2918	NaN	NaN	NaN
NaN		0.2425	0.2289	0.2489	NaN	NaN
NaN	0.2307		0.0661	0.1462	0.2346	NaN
0.1877	0.2604	0.4322	1	0.0473	0.1870	0.2614
NaN	0.2014	0.2698	0.3303		0.2126	NaN
NaN	NaN	0.1861	0.1990	0.1864		NaN
NaN	NaN	NaN	0.1454	NaN	NaN	

(h)

	NaN	NaN	0.2605	NaN	NaN	NaN
NaN		0.2027	0.1732	0.2094	NaN	NaN
NaN	0.2157		0.0230	0.0889	0.1842	NaN
0.2203	0.2917	0.4509	1	0.0482	0.1605	0.2237
NaN	0.2699	0.3345	0.4529		0.1998	NaN
NaN	NaN	0.2204	0.2788	0.2293		NaN
NaN	NaN	NaN	0.1866	NaN	NaN	

(i)

	NaN	NaN	0.0351	NaN	NaN	NaN
NaN		0.0205	0.0123	0.0161	NaN	NaN
NaN	0.5441		2.6013e-...	0.0047	0.0176	NaN
0.5012	0.5707	0.7069	1	7.1962e-...	0.0122	0.0283
NaN	0.5490	0.6330	0.7025		0.0159	NaN
NaN	NaN	0.5401	0.5551	0.5317		NaN
NaN	NaN	NaN	0.4764	NaN	NaN	

(j)

	NaN	NaN	0.1454	NaN	NaN	NaN
NaN		0.1009	0.0835	0.0875	NaN	NaN
NaN	0.3306		0.0135	0.0207	0.0529	NaN
0.2828	0.3777	0.5711	1	0.0028	0.0440	0.1102
NaN	0.3536	0.4526	0.5169		0.0849	NaN
NaN	NaN	0.3390	0.3493	0.3196		NaN
NaN	NaN	NaN	0.2623	NaN	NaN	

(k)

	NaN	NaN	0.2754	NaN	NaN	NaN
NaN		0.1941	0.1771	0.2163	NaN	NaN
NaN	0.3157		0.0265	0.0943	0.1840	NaN
0.2836	0.3533	0.4956	1	0.0390	0.1474	0.2124
NaN	0.2990	0.3582	0.4431		0.1719	NaN
NaN	NaN	0.2558	0.2985	0.2904		NaN
NaN	NaN	NaN	0.2213	NaN	NaN	

(l)

Table 3.1: The correlation results from videos.

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)
ϕ_1	0.6574	0.5691	0.5224	0.5577	0.8762	0.8337	0.9849	0.9203	0.8763	0.9800	0.8602	0.9564
ϕ_2	0.0106	0.1039	0.1591	0.0981	0	0	0	0	0	0	0	0
ϕ_3	0.0011	0.0048	0.0101	0	0	0.0020	0.0054	0.0064	0	0.0042	0.0021	0.0045

Table 3.2: Average of the AR parameters of videos from Figure 3.2.

3.4 Local dependence in the presence of noise

In order to study how added noise affects the dependence structure, we add Gaussian white noise to the videos in section 3.3. Using the Box-Muller algorithm [11] with uniform variates simulated from the Linear Congruential Generator (LCG) [10], we create three additional videos from each of the originals such that a specified signal to noise (SNR) is obtained. The SNR, as used by Parrish et. al [19], is a measure of the strength of a signal, defined as the ratio of the mean of the signal and the standard deviation of the noise, that is,

$$\text{SNR} = \frac{\mu_{\text{signal}}}{\sigma_{\text{noise}}}.$$

For each video we define μ_{signal} as the grand average of all the pixel values within the video sequence, and σ_{noise} as the variance of the noise content in the image. The algorithm for creating a new video with additive noise for a specified SNR is given as follows:

1. Specify SNR.
2. For a video $f = \{f_1, f_2, ..., f_N\}$, where $f_i : m \times n$ is the i^{th} frame of the video and n is the total number of frames, populate a vector $\mathbf{u} : (mnN) \times 1$ with uniform variates using the LCG. This is done as follows:
   (a) Specify three positive integers:
   - a - the multiplier,
   - c - the increment, and
   - m - the modulus.

   (b) With an initial value x_0 (seed), obtain a sequence of integers $x_1, x_2, ..., x_{mnN}$ with the recursive formula given by
   $$x_n = (ax_{n-1} + c)(\text{mod } m).$$
   Thus x_n is the remainder of $ax_{n-1} + c$ after dividing by m.

   (c) Multiply the sequence $x_1, x_2, ..., x_{mnN}$ by $1/m$ to obtain mnN random numbers on the interval $[0, 1)$.

3. Transform the uniform variates into normal variates using the Box-Muller algorithm. This is done as follows:

   (a) For each pair of uniform variates (u_i, u_{i+1}) in $\mathbf{u}$, $i = 1, 3, 5, ..., mnN - 1$, compute,

   $$z_i = \sqrt{-2 \ln u_1} \cos(2\pi u_2), \text{ and}$$
   $$z_{i+1} = \sqrt{-2 \ln u_1} \sin(2\pi u_2).$$

   Note that z_i and z_{i+1} are independent standard normal variates. Next, store all random standard normal variates in a vector $\mathbf{z} : mnN \times 1$.

4. Transform the standard normal random numbers to a normal distribution with variance for a specified SNR. This is done by multiplying the vector $\mathbf{z}$ by σ_{noise}, where $\sigma_{\text{noise}} = \text{SNR}\mu_{\text{signal}}$

5. Add the noise to f. Note that because the simulated noise are from continuous distributions with (bi) infinite support, f needs to be discretized and set such that the maximum of any element within f is 255 and minimum 0.

3.4.1 Correlation analysis of noisy videos

Using the methods discussed in section 3.3, we analyse the dependence structure for videos with added noise. The intensity of the noise is chosen such that three levels of SNR are achieved (according to the Rose criterion [37]), namely,

- Strong signal: SNR = 9,
- Medium signal: SNR = 5, and
- Weak signal: SNR = 1.

The tables below contain the results from the correlation analysis. For ease of comparison we included the original tables from Table 3.1 as a sub-table for videos without added noise (indicated by .1). The labelling of sub-tables .2, .3, and .4 correspond to the results from videos with strong, medium and weak signals respectively. Furthermore, the lettering of (a) through (l) is kept consistent with video description in Figure 3.2.

It can be seen that the same patterns hold in both correlation and p-values in the dependence structures. That is, the correlations between the centre pixel and its neighbour decreases and the p-values of the test of independence increases as we venture away from the centre even for noisy videos.

It can be seen also from the results that the addition of noise annihilates the dependence structure in general. Most notably in videos (a) to (d) since the hypothesis that the centre pixel is independent of its neighbours cannot be rejected at p-values higher than 0.40 for all dependence structures $M_t^{(k)}$ where $k \geq 1$.

This effect is less prominent for videos (e) to (l). Although the addition of noise does remove the dependence between neighbouring pixels, there are some cases where exceptions are required, namely:

- Video (i) - The analysis on strong signal in (i2) contains a partial $M_t^{(1)}$.
- Video (k) - The analysis on strong (k2) and medium (k3) signal preserves the $M_t^{(1)}$ depdendence structure.
- Video (j) - The analysis on all videos in (j) reveals dependence structures previously unseen from the original video. Here, (j2) has $M_t^{(2)}$, (j3) has $M_t^{(2)}$, and (j4) has $M_t^{(1)}$ at 5% signifiance level.

	NaN	NaN	0.1467	NaN	NaN	NaN
NaN		0.1729	0.1290	0.1722	NaN	NaN
NaN	0.0844		0.0729	0.1422	0.1680	NaN
0.1295	0.0657	0.2767	1	0.0807	0.1251	0.1521
NaN	0.0816	0.1502	0.3643		0.1678	NaN
NaN	NaN	0.0985	0.1722	0.0990		NaN
NaN	NaN	NaN	0.1557	NaN	NaN	

(a1)

	NaN	NaN	0.4612	NaN	NaN	NaN
NaN		0.4608	0.4578	0.4636	NaN	NaN
NaN	0.0174		0.4461	0.4558	0.4624	NaN
0.0176	0.0136	0.0261	1	0.4501	0.4597	0.4630
NaN	0.0145	0.0177	0.0297		0.4601	NaN
NaN	NaN	0.0142	0.0178	0.0162		NaN
NaN	NaN	NaN	0.0166	NaN	NaN	

(a2)

	NaN	NaN	0.4809	NaN	NaN	NaN
NaN		0.4795	0.4794	0.4821	NaN	NaN
NaN	0.0104		0.4745	0.4763	0.4802	NaN
0.0098	0.0090	0.0147	1	0.4743	0.4781	0.4812
NaN	0.0086	0.0103	0.0153		0.4784	NaN
NaN	NaN	0.0081	0.0096	0.0092		NaN
NaN	NaN	NaN	0.0088	NaN	NaN	

(a3)

	NaN	NaN	0.4983	NaN	NaN	NaN
NaN		0.4987	0.4991	0.5001	NaN	NaN
NaN	9.1495e-...		0.4982	0.4971	0.4988	NaN
<0.001	<0.001	0.0011	1	0.4969	0.4988	0.4990
NaN	<0.001	<0.001	0.0011		0.4981	NaN
NaN	NaN	<0.001	<0.001	<0.001		NaN
NaN	NaN	NaN	<0.001	NaN	NaN	

(a4)

Table 3.3: Correlation results from video (a)

	NaN	NaN	0.0147	NaN	NaN	NaN
NaN		0.0250	0.0144	0.0249	NaN	NaN
NaN	0.3949		0.0049	0.0139	0.0226	NaN
0.4745	0.5237	0.6457	1	0.0057	0.0136	0.0158
NaN	0.3941	0.4697	0.6639		0.0230	NaN
NaN	NaN	0.3862	0.5073	0.3855		NaN
NaN	NaN	NaN	0.4769	NaN	NaN	

(b1)

	NaN	NaN	0.4219	NaN	NaN	NaN
NaN		0.4245	0.4201	0.4252	NaN	NaN
NaN	0.0509		0.4130	0.4223	0.4272	NaN
0.0469	0.0534	0.0612	1	0.4171	0.4251	0.4301
NaN	0.0506	0.0570	0.0636		0.4272	NaN
NaN	NaN	0.0541	0.0587	0.0544		NaN
NaN	NaN	NaN	0.0565	NaN	NaN	

(b2)

	NaN	NaN	0.4510	NaN	NaN	NaN
NaN		0.4515	0.4508	0.4526	NaN	NaN
NaN	0.0278		0.4477	0.4501	0.4529	NaN
0.0250	0.0288	0.0327	1	0.4487	0.4524	0.4559
NaN	0.0276	0.0309	0.0337		0.4532	NaN
NaN	NaN	0.0295	0.0316	0.0298		NaN
NaN	NaN	NaN	0.0306	NaN	NaN	

(b3)

	NaN	NaN	0.4934	NaN	NaN	NaN
NaN		0.4940	0.4942	0.4958	NaN	NaN
NaN	0.0021		0.4931	0.4937	0.4952	NaN
0.0017	0.0021	0.0024	1	0.4935	0.4956	0.4966
NaN	0.0021	0.0023	0.0026		0.4952	NaN
NaN	NaN	0.0022	0.0023	0.0021		NaN
NaN	NaN	NaN	0.0023	NaN	NaN	

(b4)

Table 3.4: Correlation results from video (b)

	NaN	NaN	0.0448	NaN	NaN	NaN
NaN		0.0690	0.0531	0.0691	NaN	NaN
NaN	0.2167		0.0212	0.0464	0.0598	NaN
0.3815	0.4467	0.5873	1	0.0148	0.0321	0.0369
NaN	0.2175	0.2687	0.4225		0.0601	NaN
NaN	NaN	0.1850	0.2565	0.1850		NaN
NaN	NaN	NaN	0.2724	NaN	NaN	

(c1)

	NaN	NaN	0.4415	NaN	NaN	NaN
NaN		0.4443	0.4431	0.4454	NaN	NaN
NaN	0.0391		0.4354	0.4430	0.4445	NaN
0.0396	0.0446	0.0520	1	0.4307	0.4377	0.4431
NaN	0.0393	0.0434	0.0497		0.4444	NaN
NaN	NaN	0.0405	0.0434	0.0402		NaN
NaN	NaN	NaN	0.0427	NaN	NaN	

(c2)

	NaN	NaN	0.4587	NaN	NaN	NaN
NaN		0.4600	0.4605	0.4612	NaN	NaN
NaN	0.0213		0.4580	0.4594	0.4603	NaN
0.0205	0.0234	0.0269	1	0.4568	0.4590	0.4618
NaN	0.0215	0.0237	0.0263		0.4599	NaN
NaN	NaN	0.0223	0.0236	0.0221		NaN
NaN	NaN	NaN	0.0230	NaN	NaN	

(c3)

	NaN	NaN	0.4945	NaN	NaN	NaN
NaN		0.4948	0.4953	0.4963	NaN	NaN
NaN	0.0016		0.4952	0.4944	0.4957	NaN
0.0012	0.0016	0.0019	1	0.4944	0.4962	0.4965
NaN	0.0015	0.0016	0.0018		0.4959	NaN
NaN	NaN	0.0015	0.0016	0.0014		NaN
NaN	NaN	NaN	0.0016	NaN	NaN	

(c4)

Table 3.5: Correlation results from video (c)

	NaN	NaN	0.0306	NaN	NaN	NaN
NaN		0.0730	0.0566	0.0735	NaN	NaN
NaN	0.1976		0.0122	0.0359	0.0499	NaN
0.3777	0.4262	0.5680	1	0.0064	0.0142	0.0158
NaN	0.1992	0.2557	0.4405		0.0512	NaN
NaN	NaN	0.1344	0.2159	0.1350		NaN
NaN	NaN	NaN	0.2750	NaN	NaN	

(d1)

	NaN	NaN	0.4991	NaN	NaN	NaN
NaN		0.5000	0.5002	0.5015	NaN	NaN
NaN	0.0022		0.4956	0.4987	0.5004	NaN
0.0048	0.0057	0.0083	1	0.4929	0.4970	0.4996
NaN	0.0022	0.0030	0.0065		0.5007	NaN
NaN	NaN	0.0012	0.0024	0.0013		NaN
NaN	NaN	NaN	0.0037	NaN	NaN	

(d2)

	NaN	NaN	0.5017	NaN	NaN	NaN
NaN		0.5019	0.5031	0.5031	NaN	NaN
NaN	<0.001		0.5019	0.5010	0.5022	NaN
<0.001	0.0020	0.0029	1	0.5003	0.5014	0.5029
NaN	<0.001	<0.001	0.0021		0.5023	NaN
NaN	NaN	<0.001	<0.001	<0.001		NaN
NaN	NaN	NaN	0.0013	NaN	NaN	

(d3)

	NaN	NaN	0.5014	NaN	NaN	NaN
NaN		0.5027	0.5028	0.5034	NaN	NaN
NaN	<0.001		0.5028	0.5015	0.5025	NaN
<0.001	<0.001	<0.001	1	0.5017	0.5023	0.5033
NaN	<0.001	<0.001	<0.001		0.5024	NaN
NaN	NaN	<0.001	<0.001	<0.001		NaN
NaN	NaN	NaN	<0.001	NaN	NaN	

(d4)

Table 3.6: Correlation results from video (d)

	NaN	NaN	0.2648	NaN	NaN	NaN
NaN		0.2382	0.2108	0.2454	NaN	NaN
NaN	0.0938		0.0556	0.1592	0.2666	NaN
0.0512	0.1020	0.2475	1	0.1023	0.2524	0.2980
NaN	0.0866	0.1723	0.3006		0.2624	NaN
NaN	NaN	0.1103	0.1429	0.2382		NaN
NaN	NaN	NaN	0.0895	NaN	NaN	

(e1)

	NaN	NaN	0.3330	NaN	NaN	NaN
NaN		0.3113	0.2860	0.3193	NaN	NaN
NaN	0.0744		0.1056	0.2365	0.3398	NaN
0.0390	0.0805	0.1995	1	0.1692	0.3266	0.3599
NaN	0.0683	0.1381	0.2406		0.3319	NaN
NaN	NaN	0.0872	0.1137	0.0982		NaN
NaN	NaN	NaN	0.0707	NaN	NaN	

(e2)

	NaN	NaN	0.3441	NaN	NaN	NaN
NaN		0.3249	0.3010	0.3298	NaN	NaN
NaN	0.0552		0.1310	0.2547	0.3509	NaN
0.0281	0.0594	0.1485	1	0.1926	0.3397	0.3699
NaN	0.0503	0.1023	0.1775		0.3442	NaN
NaN	NaN	0.0641	0.0840	0.0729		NaN
NaN	NaN	NaN	0.0523	NaN	NaN	

(e3)

	NaN	NaN	0.4669	NaN	NaN	NaN
NaN		0.4605	0.4545	0.4638	NaN	NaN
NaN	0.0060		0.4098	0.4450	0.4681	NaN
0.0029	0.0063	0.0157	1	0.4253	0.4665	0.4754
NaN	0.0052	0.0108	0.0181		0.4663	NaN
NaN	NaN	0.0065	0.0089	0.0081		NaN
NaN	NaN	NaN	0.0057	NaN	NaN	

(e4)

Table 3.7: Correlation results from video (e)

	NaN	NaN	0.2609	NaN	NaN	NaN
NaN		0.1982	0.1591	0.1935	NaN	NaN
NaN	0.1870		0.0287	0.0899	0.2163	NaN
0.1402	0.2078	0.3452	1	0.0317	0.1885	0.3015
NaN	0.1987	0.2887	0.3819		0.2147	NaN
NaN	NaN	0.2094	0.2445	0.1940		NaN
NaN	NaN	NaN	0.1803	NaN	NaN	

(f1)

	NaN	NaN	0.2946	NaN	NaN	NaN
NaN		0.2549	0.2219	0.2474	NaN	NaN
NaN	0.1196		0.0960	0.1628	0.2641	NaN
0.0909	0.1374	0.2337	1	0.1002	0.2489	0.3242
NaN	0.1353	0.1962	0.2624		0.2678	NaN
NaN	NaN	0.1421	0.1648	0.1240		NaN
NaN	NaN	NaN	0.1207	NaN	NaN	

(f2)

	NaN	NaN	0.3285	NaN	NaN	NaN
NaN		0.3004	0.2670	0.2885	NaN	NaN
NaN	0.0992		0.1467	0.2112	0.3036	NaN
0.0756	0.1153	0.1969	1	0.1513	0.2932	0.3539
NaN	0.1140	0.1653	0.2211		0.3106	NaN
NaN	NaN	0.1202	0.1388	0.1033		NaN
NaN	NaN	NaN	0.1015	NaN	NaN	

(f3)

	NaN	NaN	0.4047	NaN	NaN	NaN
NaN		0.3994	0.3750	0.3899	NaN	NaN
NaN	0.0286		0.3140	0.3506	0.3948	NaN
0.0218	0.0341	0.0577	1	0.3198	0.3910	0.4204
NaN	0.0339	0.0493	0.0635		0.4022	NaN
NaN	NaN	0.0364	0.0404	0.0306		NaN
NaN	NaN	NaN	0.0295	NaN	NaN	

(f4)

Table 3.8: Correlation results from video (f)

	NaN	NaN	0.3110	NaN	NaN	NaN
NaN		0.2703	0.2591	0.2812	NaN	NaN
NaN	0.1611		0.0716	0.1567	0.2455	NaN
0.1755	0.2749	0.4708	1	0.0390	0.1647	0.2431
NaN	0.1474	0.1951	0.2555		0.2255	NaN
NaN	NaN	0.0793	0.1045	0.0854		NaN
NaN	NaN	NaN	0.0498	NaN	NaN	

(g1)

	NaN	NaN	0.3581	NaN	NaN	NaN
NaN		0.3355	0.3290	0.3467	NaN	NaN
NaN	0.1216		0.1525	0.2449	0.3133	NaN
0.1356	0.2155	0.3704	1	0.0849	0.2229	0.2965
NaN	0.1127	0.1460	0.1842		0.2919	NaN
NaN	NaN	0.0538	0.0699	0.0589		NaN
NaN	NaN	NaN	0.0291	NaN	NaN	

(g2)

	NaN	NaN	0.3821	NaN	NaN	NaN
NaN		0.3650	0.3593	0.3748	NaN	NaN
NaN	0.0985		0.2052	0.2841	0.3422	NaN
0.1095	0.1746	0.2997	1	0.1159	0.2525	0.3249
NaN	0.0913	0.1181	0.1467		0.3241	NaN
NaN	NaN	0.0420	0.0541	0.0462		NaN
NaN	NaN	NaN	0.0213	NaN	NaN	

(g3)

	NaN	NaN	0.4510	NaN	NaN	NaN
NaN		0.4432	0.4402	0.4456	NaN	NaN
NaN	0.0157		0.3760	0.3995	0.4228	NaN
0.0171	0.0273	0.0460	1	0.3048	0.3794	0.4198
NaN	0.0141	0.0185	0.0222		0.4201	NaN
NaN	NaN	0.0058	0.0074	0.0068		NaN
NaN	NaN	NaN	0.0027	NaN	NaN	

(g4)

Table 3.9: Correlation results from video (g)

	NaN	NaN	0.2918	NaN	NaN	NaN
NaN		0.2425	0.2289	0.2489	NaN	NaN
NaN	0.2307		0.0661	0.1462	0.2346	NaN
0.1877	0.2604	0.4322	1	0.0473	0.1870	0.2614
NaN	0.2014	0.2698	0.3303		0.2126	NaN
NaN	NaN	0.1861	0.1990	0.1864		NaN
NaN	NaN	NaN	0.1454	NaN	NaN	

(h1)

	NaN	NaN	0.3467	NaN	NaN	NaN
NaN		0.3160	0.3061	0.3180	NaN	NaN
NaN	0.1458		0.1441	0.2323	0.3037	NaN
0.1119	0.1695	0.3072	1	0.1120	0.2596	0.3152
NaN	0.1187	0.1711	0.2177		0.2820	NaN
NaN	NaN	0.1031	0.1123	0.1056		NaN
NaN	NaN	NaN	0.0718	NaN	NaN	

(h2)

	NaN	NaN	0.3753	NaN	NaN	NaN
NaN		0.3485	0.3389	0.3488	NaN	NaN
NaN	0.1131		0.1894	0.2701	0.3360	NaN
0.0861	0.1325	0.2420	1	0.1458	0.2930	0.3448
NaN	0.0905	0.1316	0.1678		0.3138	NaN
NaN	NaN	0.0765	0.0836	0.0796		NaN
NaN	NaN	NaN	0.0517	NaN	NaN	

(h3)

	NaN	NaN	0.4593	NaN	NaN	NaN
NaN		0.4449	0.4424	0.4473	NaN	NaN
NaN	0.0160		0.3817	0.4121	0.4396	NaN
0.0121	0.0189	0.0337	1	0.3444	0.4158	0.4413
NaN	0.0121	0.0178	0.0225		0.4255	NaN
NaN	NaN	0.0095	0.0107	0.0109		NaN
NaN	NaN	NaN	0.0062	NaN	NaN	

(h4)

Table 3.10: Correlation results from video (h)

	NaN	NaN	0.2605	NaN	NaN	NaN
NaN		0.2027	0.1732	0.2094	NaN	NaN
NaN	0.2157		0.0230	0.0889	0.1842	NaN
0.2203	0.2917	0.4509	1	0.0482	0.1605	0.2237
NaN	0.2699	0.3345	0.4529		0.1998	NaN
NaN	NaN	0.2204	0.2788	0.2293		NaN
NaN	NaN	NaN	0.1866	NaN	NaN	

(i1)

	NaN	NaN	0.2803	NaN	NaN	NaN
NaN		0.2396	0.2063	0.2458	NaN	NaN
NaN	0.1735		0.0454	0.1290	0.2158	NaN
0.1799	0.2397	0.3723	1	0.0783	0.1963	0.2519
NaN	0.2206	0.2748	0.3736		0.2417	NaN
NaN	NaN	0.1777	0.2268	0.1840		NaN
NaN	NaN	NaN	0.1496	NaN	NaN	

(i2)

	NaN	NaN	0.2889	NaN	NaN	NaN
NaN		0.2556	0.2197	0.2616	NaN	NaN
NaN	0.1364		0.0639	0.1516	0.2282	NaN
0.1433	0.1912	0.2972	1	0.0992	0.2127	0.2628
NaN	0.1750	0.2190	0.2984		0.2588	NaN
NaN	NaN	0.1402	0.1799	0.1444		NaN
NaN	NaN	NaN	0.1177	NaN	NaN	

(i3)

	NaN	NaN	0.3978	NaN	NaN	NaN
NaN		0.3854	0.3639	0.3874	NaN	NaN
NaN	0.0218		0.2819	0.3331	0.3632	NaN
0.0237	0.0315	0.0487	1	0.2985	0.3562	0.3808
NaN	0.0285	0.0362	0.0492		0.3848	NaN
NaN	NaN	0.0229	0.0295	0.0229		NaN
NaN	NaN	NaN	0.0192	NaN	NaN	

(i4)

Table 3.11: Correlation results from video (i)

	NaN	NaN	0.0351	NaN	NaN	NaN
NaN		0.0205	0.0123	0.0161	NaN	NaN
NaN	0.5441		2.6013e-...	0.0047	0.0176	NaN
0.5012	0.5707	0.7069	1	7.1962e-...	0.0122	0.0283
NaN	0.5490	0.6330	0.7025		0.0159	NaN
NaN	NaN	0.5401	0.5551	0.5317		NaN
NaN	NaN	NaN	0.4764	NaN	NaN	

(j1)

	NaN	NaN	0.0569	NaN	NaN	NaN
NaN		0.0366	0.0234	0.0279	NaN	NaN
NaN	0.5352		8.6357e-...	0.0095	0.0290	NaN
0.4930	0.5614	0.6952	1	0.0019	0.0216	0.0438
NaN	0.5401	0.6227	0.6909		0.0271	NaN
NaN	NaN	0.5315	0.5463	0.5232		NaN
NaN	NaN	NaN	0.4689	NaN	NaN	

(j2)

	NaN	NaN	0.0884	NaN	NaN	NaN
NaN		0.0597	0.0401	0.0457	NaN	NaN
NaN	0.5181		0.0022	0.0168	0.0453	NaN
0.4773	0.5435	0.6731	1	0.0039	0.0354	0.0664
NaN	0.5229	0.6030	0.6691		0.0442	NaN
NaN	NaN	0.5147	0.5290	0.5066		NaN
NaN	NaN	NaN	0.4542	NaN	NaN	

(j3)

	NaN	NaN	0.1004	NaN	NaN	NaN
NaN		0.0624	0.0484	0.0630	NaN	NaN
NaN	0.2821		0.0104	0.0292	0.0624	NaN
0.2607	0.2951	0.3618	1	0.0138	0.0507	0.0861
NaN	0.2846	0.3264	0.3602		0.0594	NaN
NaN	NaN	0.2809	0.2884	0.2767		NaN
NaN	NaN	NaN	0.2493	NaN	NaN	

(j4)

Table 3.12: Correlation results from video (j)

	NaN	NaN	0.1454	NaN	NaN	NaN
NaN		0.1009	0.0835	0.0875	NaN	NaN
NaN	0.3306		0.0135	0.0207	0.0529	NaN
0.2828	0.3777	0.5711	1	0.0028	0.0440	0.1102
NaN	0.3536	0.4526	0.5169		0.0849	NaN
NaN	NaN	0.3390	0.3493	0.3196		NaN
NaN	NaN	NaN	0.2623	NaN	NaN	

(k1)

	NaN	NaN	0.1945	NaN	NaN	NaN
NaN		0.1494	0.1272	0.1285	NaN	NaN
NaN	0.2702		0.0291	0.0413	0.0890	NaN
0.2297	0.3090	0.4721	1	0.0095	0.0797	0.1608
NaN	0.2895	0.3731	0.4265		0.1332	NaN
NaN	NaN	0.2769	0.2860	0.2614		NaN
NaN	NaN	NaN	0.2132	NaN	NaN	

(k2)

	NaN	NaN	0.2185	NaN	NaN	NaN
NaN		0.1735	0.1497	0.1509	NaN	NaN
NaN	0.2133		0.0430	0.0580	0.1136	NaN
0.1806	0.2444	0.3766	1	0.0213	0.1022	0.1861
NaN	0.2283	0.2962	0.3397		0.1569	NaN
NaN	NaN	0.2177	0.2261	0.2066		NaN
NaN	NaN	NaN	0.1676	NaN	NaN	

(k3)

	NaN	NaN	0.3844	NaN	NaN	NaN
NaN		0.3615	0.3464	0.3484	NaN	NaN
NaN	0.0300		0.2713	0.2885	0.3309	NaN
0.0250	0.0344	0.0536	1	0.2455	0.3241	0.3689
NaN	0.0316	0.0418	0.0480		0.3530	NaN
NaN	NaN	0.0301	0.0317	0.0293		NaN
NaN	NaN	NaN	0.0232	NaN	NaN	

(k4)

Table 3.13: Correlation results from video (k)

	NaN	NaN	0.2754	NaN	NaN	NaN
NaN		0.1941	0.1771	0.2163	NaN	NaN
NaN	0.3157		0.0265	0.0943	0.1840	NaN
0.2836	0.3533	0.4956	1	0.0390	0.1474	0.2124
NaN	0.2990	0.3582	0.4431		0.1719	NaN
NaN	NaN	0.2558	0.2985	0.2904		NaN
NaN	NaN	NaN	0.2213	NaN	NaN	

(l1)

	NaN	NaN	0.2641	NaN	NaN	NaN
NaN		0.2110	0.2032	0.2322	NaN	NaN
NaN	0.1720		0.0617	0.1372	0.2013	NaN
0.1525	0.1974	0.2888	1	0.0713	0.1683	0.2157
NaN	0.1600	0.1965	0.2493		0.1910	NaN
NaN	NaN	0.1293	0.1577	0.1530		NaN
NaN	NaN	NaN	0.1076	NaN	NaN	

(l2)

	NaN	NaN	0.2939	NaN	NaN	NaN
NaN		0.2449	0.2377	0.2655	NaN	NaN
NaN	0.1106		0.1017	0.1751	0.2346	NaN
0.0973	0.1279	0.1895	1	0.1060	0.2013	0.2476
NaN	0.1017	0.1264	0.1613		0.2245	NaN
NaN	NaN	0.0802	0.0996	0.0966		NaN
NaN	NaN	NaN	0.0652	NaN	NaN	

(l3)

	NaN	NaN	0.4711	NaN	NaN	NaN
NaN		0.4595	0.4572	0.4654	NaN	NaN
NaN	0.0116		0.4215	0.4408	0.4549	NaN
0.0101	0.0133	0.0197	1	0.4143	0.4421	0.4559
NaN	0.0102	0.0129	0.0160		0.4495	NaN
NaN	NaN	0.0075	0.0096	0.0097		NaN
NaN	NaN	NaN	0.0059	NaN	NaN	

(l4)

Table 3.14: Correlation results from video (l)

3.5 Conclusion

We have motivated the use of a univariate AR process for a sample of stationary videos to model each pixel in a video stream in Section 3.2, as well as discussed its covariance-stationarity. This allowed us to make use of the test of independence between two univariate covariance-stationary times series [12], and hence aids in the investigation of the extent to which the assumption of global independence and local dependence holds in images.

In Section 3.3 we used the dependence structure $M_t^{(K)}$ (defined in Section 3.2) for $K = 3$ and found the correlation and the p-value of the independence test of each pixel x_t with its neighbour $y_t \in M_t^{(3)}$ for 12 different videos and averaged the results over the whole pixel domain. Furthermore, we also found the average of the AR parameters of each time series x_t.

From the results, we discovered a relationship between the assumption of global independence and local dependence, and the average of the AR parameters of each video. Here, if the majority of the pixel time series can be modeled with an AR(1), then the assumption holds (with exception to video (j) from Table 3.1); and if the pixels in a video stream can be modeled with an AR(2), then the assumption will not hold in that video.

In regards to video (j) from Table 3.1, we note that the contents of the image consisted only of the silhouette of a tree and the sky background with clouds - these areas are also relatively large considering the dimensions of the video as well as the size of the dependence structure ($K = 3$). Meaning video (j) is comparatively less complicated than the rest of the videos, and $M_t^{(3)}$ captures the full dependence of silhouette to silhouette pixels and sky to sky pixels. This implies, on average, we rejected the null hypothesis of independence for the whole region of $M_t^{(3)}$. Thus in an image with lower content/texture level a larger local dependence structure could be motivated. The case of dependence in non-stationary images will be discussed in the conclusion.

In Section 3.4 we added Gaussian noise to the sample video data and performed the correlation analysis on the noisy videos in Section 3.4.1. The addition of noise decreased the size of the dependence structures since the noise themselves are iid Gaussian white noise. This renders the relationship between pixels ambiguous when the variation present within the video increases. However, the motivation of local dependence and global independence is still clearly justified by these experiments.

In the chapter that follows, with the justification of using local dependence, we investigate the ability of the $LULU$ smoothers to remove various noise types in contaminated images.

Chapter 4

Noise removal in images

4.1 Introduction

In this chapter we demonstrate the efficiency of the $LULU$ operators in removing noise added to an image. This research follows on from work done in [9] on one-dimensional noise removal by the $LULU$ smoothers. The objectives of this study are to see the extent at which the $LULU$ operators restore an image that has been contaminated by noise, as well as the level of retrieval of the noise particles added to the image. We use the structural similarity index (SSIM) to measure the similarity of the original image to the purified noisy image at level n of the $LULU$ operators as an indication of how well the image is restored [13], and probability plots to see how the extracted noise fits its original distribution.

4.2 Analysis of noisy images

To investigate the efficiency of the $LULU$ operators in removing noise, we first add noise distributions with different shape properties (see Table 4.1) to the image. For each noise distribution, three noisy images are created from f such that the signal to noise ratio (SNR) of each noisy image is either 1, 5, or 9 since these values represent the weak, medium and strong signals (according to the Rose criterion [19]). Using the DPT with $Q_n = L_n U_n$[1] the noise is extracted, and both the purified image and noise at level n of the DPT is analysed using measures SSIM (structural similarity index) and SSE (sum of squared errors) respectively. The SSIM [13] measures the similarity between two images, x and y, where one is the reference image and is considered to be distortion-free, is given as

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)},$$

where μ_x is the mean of image x, μ_y the mean of image y, σ_x^2 and σ_y^2 are the variances of images x and y respectively, σ_{xy} the covariance between image x and y, $c_1 = (0.01L)^2$ and $c_2 = (0.03L)^2$ are used to stabilise

[1] The results obtained using the DPT with $Q_n = L_n U_n$ and $Q_n = U_n L_n$ are similar. Thus we focus only on the former.

the division with a weak denominator, and L is the dynamic range (typically $2^{\text{number of bits per pixel}} - 1$).

For the noise analysis, denote the noise extracted at position (i, j) at level n of the DPT as $e_{ij}^{(n)} = [(I - L_n U_n)f]_{ij}$. Since its distribution is believed to be completely specified by its CDF $F(x)$, we have that $\widehat{F\left(e_{ij}^{(n)}\right)}$ is uniformly distributed over the interval $[0, 1]$. Thus a measure to see how well the extracted noise fits its orginal distribution, we find the SSE defined as

$$\text{SSE}^{(n)} = \sum_{k=1}^{N} \left(\frac{k}{N} - \widehat{F\left(e_{k:N}^{(n)}\right)} \right)^2$$

where N is the total number of pixels, and $e_{k:N}^{(n)}$ are the ordered noise observations.

The properties of distributions of the different noise used are displayed in Table 4.1. As we can see, except for the Rayleigh and exponential distribution, we can set the mean of each distribution to zero. The parameters for these distributions are then obtained by setting the variance equal to σ^2_{noise} and solving them accordingly in the SNR formula:

$$\text{SNR} = \frac{\mu_{\text{signal}}}{\sigma_{\text{noise}}}.$$

For the Rayleigh and exponential distribution, the parameter σ (Rayleigh) and θ (exponential) are also obtained in the same way. However, because the support of these distributions are positive, we shift the Rayleigh and exponential variates by subtracting their respective medians to simulated noise, in order for positive and negative noise to be simulated. This is required since the $LULU$ operators smooth from below and above.

	PDF	z	Support	$E[X]$	$var(X)$
1. Uniform	$f(x) = \frac{1}{b-a}$	.	$a \leq x \leq b$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
2. Normal	$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-z^2/2}$	$z = \frac{x-\mu}{\sigma}$	$-\infty < x < \infty$	μ	σ^2
3. Logistic	$f(x) = \frac{e^{-z}}{s(1+e^{-z})^2}$	$z = \frac{x-\mu}{s}$	$-\infty < x < \infty$	μ	$\frac{s^2\pi^2}{3}$
4. Rayleigh	$f(x) = \frac{x}{\sigma^2} e^{-x^2/2\sigma^2}$	.	$x \geq 0$	$\sigma\sqrt{\frac{\pi}{2}}$	$\frac{4-\pi}{2}\sigma^2$
5. Gumbel	$f(x) = \frac{1}{\beta} e^{-(z+e^{-z})}$	$z = \frac{x-\mu}{\beta}$	$-\infty < x < \infty$	$\mu + \beta\gamma$ [2]	$\frac{\pi^2}{6}\beta^2$
6. Exponential	$f(x) = \frac{1}{\theta} e^{-x/\theta}$	.	$x \geq 0$	θ	θ^2

Table 4.1: The distributions of the noise added to images.

A graph for the pdfs of the different noise types is given in Figure 4.1. We see that the uniform, normal, and logistic distributions are symmetric about the mean, and the rest are skewed to the right.

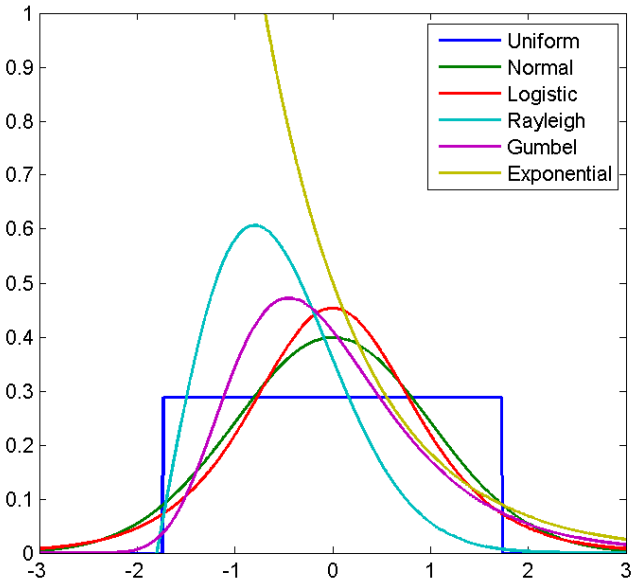


Figure 4.1: Probability density functions of noise distributions with standard deviations all equal to 1.

The algorithm for this study is given as follows:

1. Specify SNR as 1, 5 or 9.
2. For an image $f : m \times n$, populate a vector $\mathbf{u} : mn \times 1$ with independent uniform variates using the LCG (see step 2 of the algorithm in Section 3.4).
3. For each noise distribution, transform the uniform variates into random numbers from the respective noise distribution:
   - If the distribution is Gaussian, use the Box-Muller algorithm (see step 3 of the algorithm in Section 3.4).
   - For other noise types in Tables 4.1, use the inverse transform method [34] to obtain variates from the required distribution. This is done as follows:
     (a) Set $E[X]$ equal to zero and solve for the unknown parameters by setting $var(X)$ equal to the required variance of the noise content in the image. That is, set $var(X) = \sigma^2_{\text{noise}} = SNR\mu_f$ and solve for the unknown parameter. Note that because the $E[X]$ and $var(x)$ of the Rayleigh and exponential distribution depends on the same parameters, their variates are obtained by solving for the required parameter through $var(X)$ (without setting $E[X] = 0$).
     (b) For each element u_i in $\mathbf{u}$, compute $x_i = F^{-1}(u_i)$. If the distribution is Rayleigh or exponential, shift the variate x to the left by the median of the distribution such that 50% of the variates lie above and below 0.

4. Add the noise to the original image f. Note that because the simulated noise are from continuous distributions with (bi) infinite support, the noise contaminated f needs to be discretized and set such that the maximum of any element within f is 255 and minimum 0.

5. Smooth the noisy images of f with $Q_n = L_n U_n$.

4.3 Application

The study is performed on the same video dataset as in Figure 3.2 of Chapter 3. However, instead of using the whole video stream, only a single frame is selected for the analysis. Each image is resized to contain 270×384 pixels for comparison of results. First we look at the total variation as each image is smoothed by the $LULU$ operators.

4.3.1 Total variation plots

The total variation of an image f is given by [2]:

$$TV(f) = \sum_{(i,j)\in\mathbb{Z}^2} (|f(x_{i,j+1}) - f(x_{ij})| + |f(x_{i+1,j}) - f(x_{ij})|).$$

The properties of total variation stated in Results 2.3 - 2.5 in Chapter 2 are preserved in two dimensions.

In Figures 4.2 to 4.13 the proportion of total variation (y-axis) retained at level n (x-axis) of the DPT is shown. Since the required number of levels to fully decompose the noisy images is $n = 103680$, the linear x-axis has been changed to the natural logarithm of the original n values. Furthermore, reference lines are included in each figure to indicate the proportion of the total variation of the original image to the total variation of the noisy image for different SNR values. For each line the point where the horizontal section meets the y-axis indicates the proportion of the total variation of the original image to the total variation of the noisy image and, for that same line, the point where the vertical section meets the x-axis indicates the required level $\ln(n)$ of the DPT to reach the original total variation from the noisy image.

For all total variation plots we can see a disproportional relationship exists between the total variation of a noisy image and the n_{th} level of the decomposition. This is expected since the $LULU$ operators do not increase variation in the process of smoothing [2] and is consistent in the decomposition.

Furthermore, there is a tendency for the total variation of images with strong signals to be the largest, and the total variation of the images with weak signals to be the smallest. Since these plots show the *proportion* of total variation retained with their respective noisy images, it does not necessarily show that strong signals have larger total variation than weak signals, however, it does imply that strong signals *behave* the best and weak signal the worst in terms of smoothing possibility. The reason is that images with strong signals retain most of the information in the original image and so contains much more information than the rest. That is, it can withstand the decomposition and retain more variability because it is less

sparse. Images with high noisy content are subjected to pixel value limits (minimum 0 and maximum 255) and so appear to have less 'true' variation than that of the images with strong signals.

Lastly, the decaying rate of the total variation is fast for the first few n levels and then slows down for the remainder of the process (recall the x-axis is the natural logarithm of the n^{th} level of the decomposition). This implies that most of the variation is contained within small pulses. However, this may also be due to image size and content.

The proportion of the original total variation to the total variation of their noisy counterparts may be explained in two parts, their values may be found in Table 4.2:

- Consider the total variation plots of images (a) to (d). We can see that for these images the level n required to achieve the original total variation is spread out amongst the signal strength. Most notably, the noisy images with strong signal require less decomposition than the others to reach original total variation, while the weak signals require the most.
- For images (e) to (l), we see that the total variation of the original image can be reached after one application of $LULU$ operators (L_1U_1) for medium and strong signals.

It should be noted that images (a) to (d) are obtained from an internet source, while images (e) to (l) are otained by the author. Therefore, a difference in results with respect to image sources imply that the way in which the images are obtained influence results. For example, images (a) to (d) are downloaded from the internet, and so have most likely gone through processes of cleaning, compressing, and editing.

	SNR	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)
Uniform	1	140	3096	663	1657	12	7	11	11	11	5	13	27
	5	11	239	64	66	1	1	1	1	1	1	1	2
	9	5	95	26	27	1	1	1	1	1	1	1	1
Normal	1	81	2122	429	683	8	5	8	7	8	4	8	17
	5	8	167	49	49	1	1	1	1	1	1	1	1
	9	3	70	20	21	1	1	1	1	1	1	1	1
Logistic	1	72	1852	408	549	7	4	7	6	6	3	7	15
	5	7	137	43	42	1	1	1	1	1	1	1	1
	9	3	61	19	19	1	1	1	1	1	1	1	1
Rayleigh	1	87	1751	424	832	8	5	8	7	8	4	9	18
	5	8	162	50	54	1	1	1	1	1	1	1	1
	9	3	69	20	21	1	1	1	1	1	1	1	1
Gumbel	1	72	1524	345	583	7	4	7	6	7	3	7	16
	5	7	137	42	42	1	1	1	1	1	1	1	1
	9	3	59	19	18	1	1	1	1	1	1	1	1
Exponential	1	59	1392	287	464	5	4	5	5	6	4	6	12
	5	6	102	34	34	1	1	1	1	1	1	1	1
	9	2	46	14	14	1	1	1	1	1	1	1	1

Table 4.2: The required level of DPT to reach original total variation from noisy image.

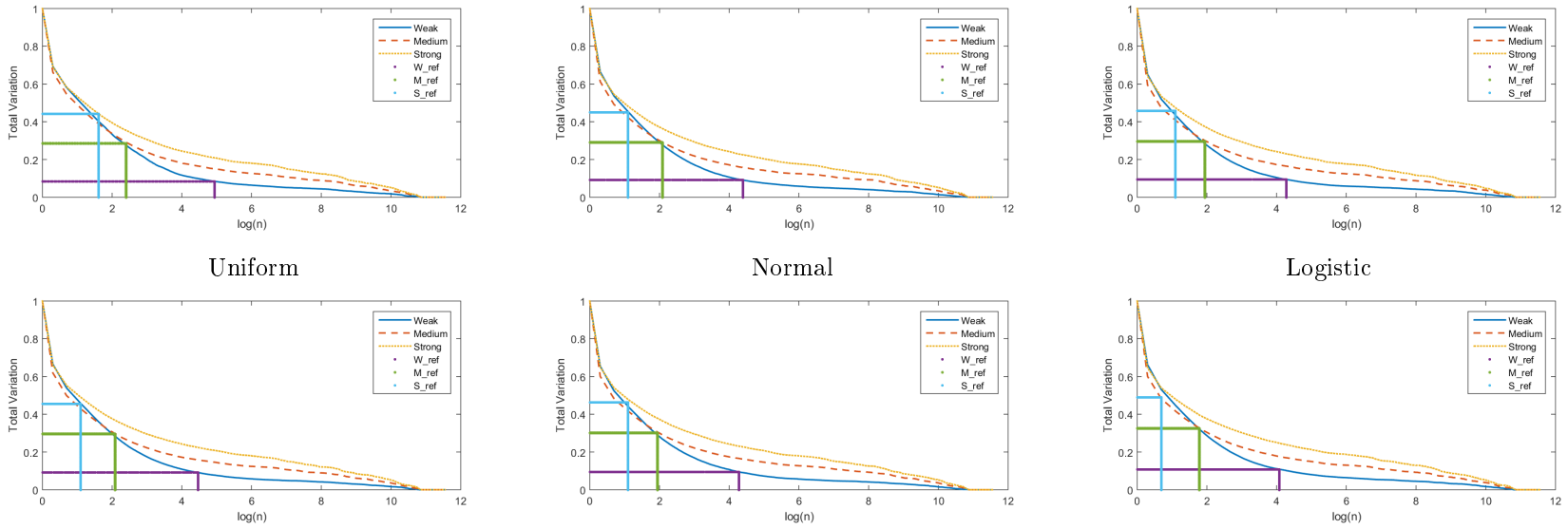


Figure 4.2: Total variation removed at each level of DPT for video (a) using $LULU$.

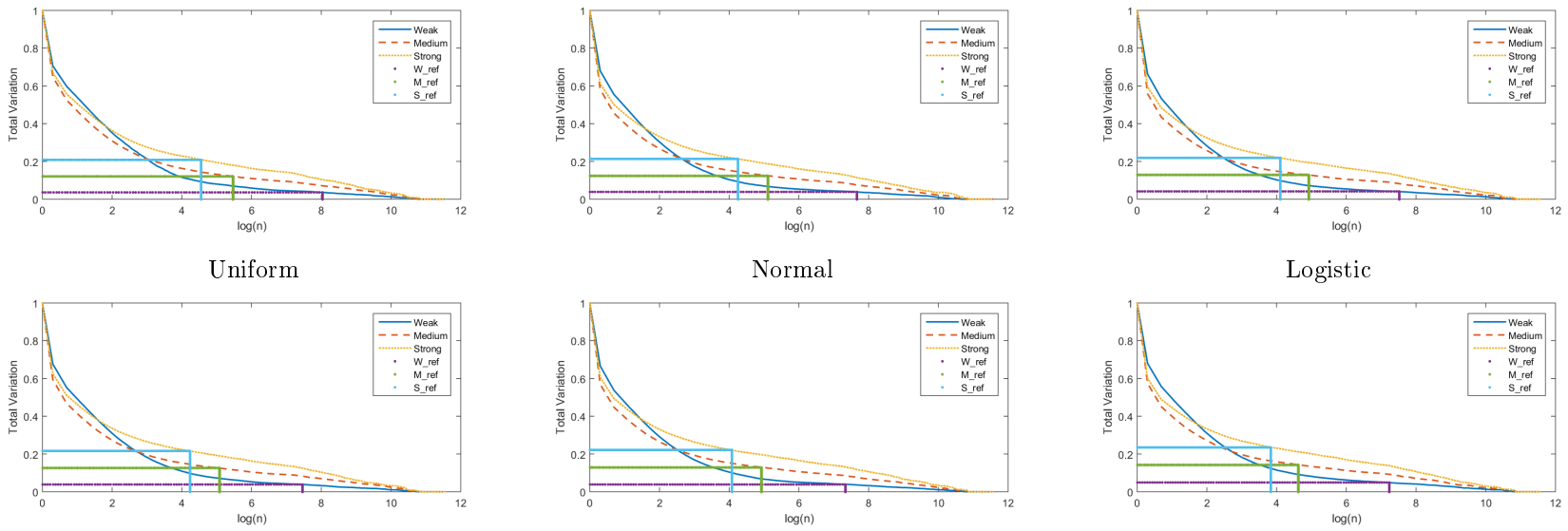


Figure 4.3: Total variation removed at each level of DPT for video (b) using $LULU$.

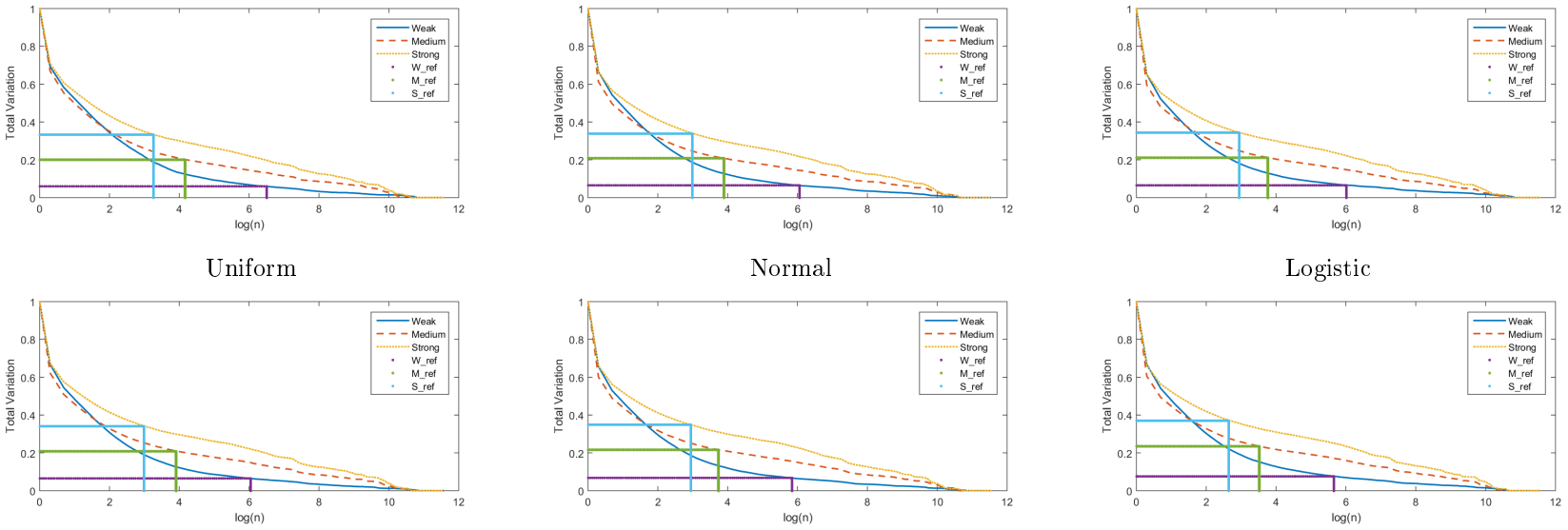


Figure 4.4: Total variation removed at each level of DPT for video (c) using $LULU$.

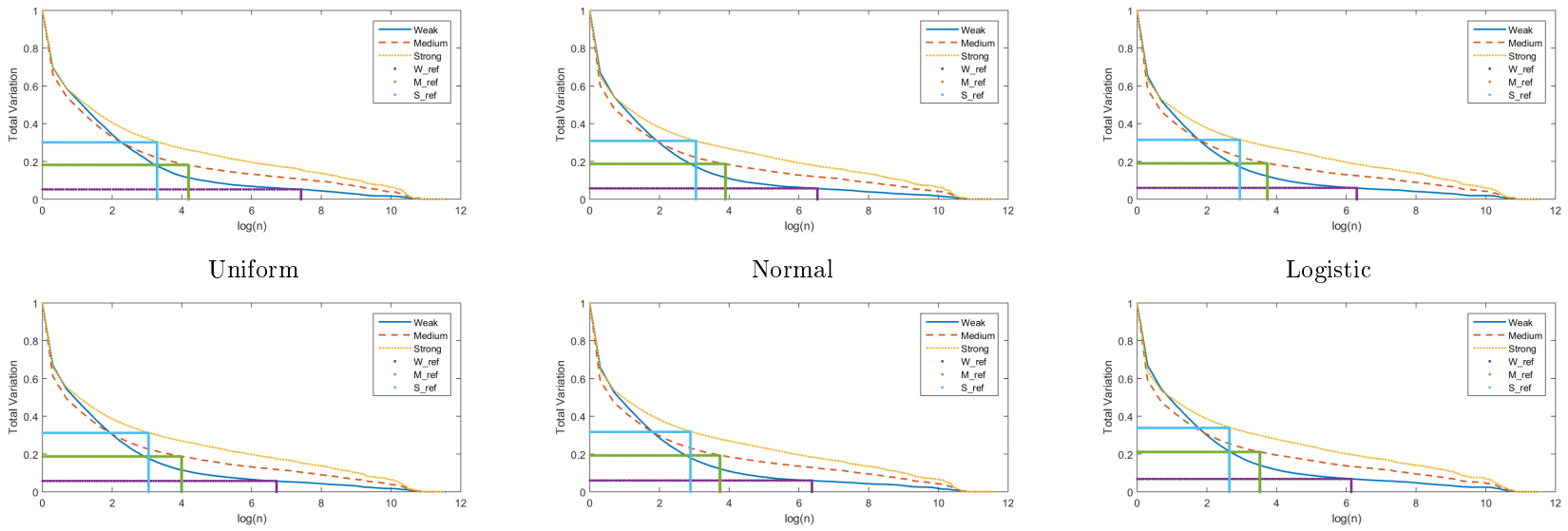


Figure 4.5: Total variation removed at each level of DPT for video (d) using $LULU$.

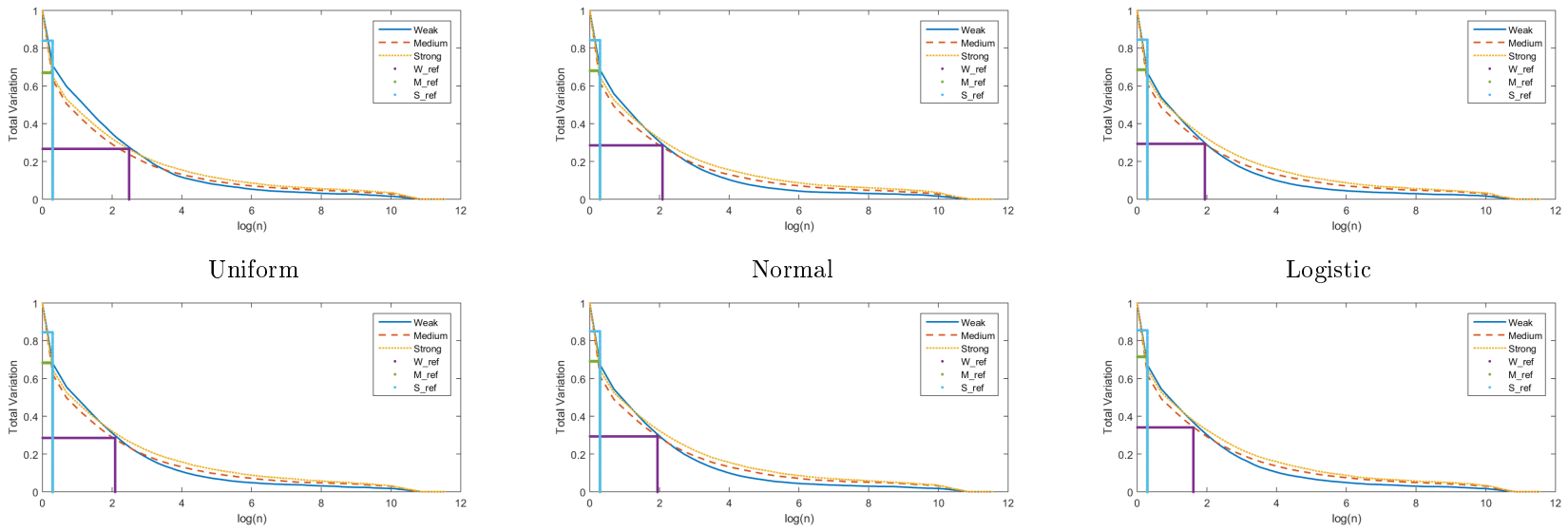
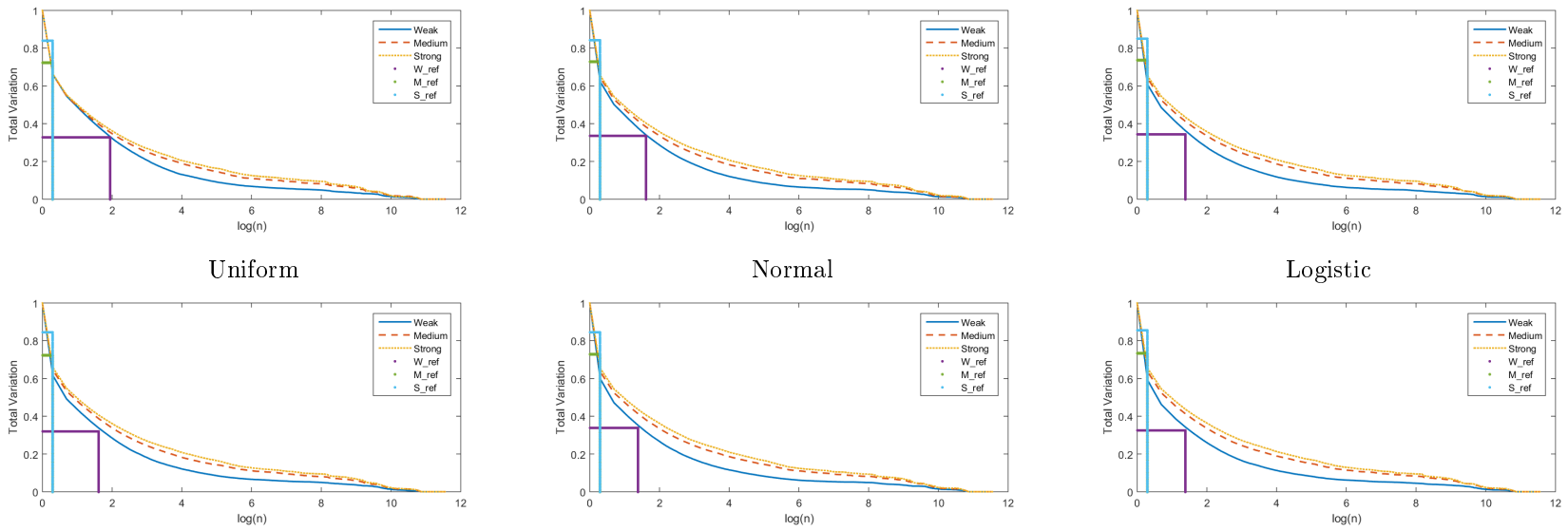


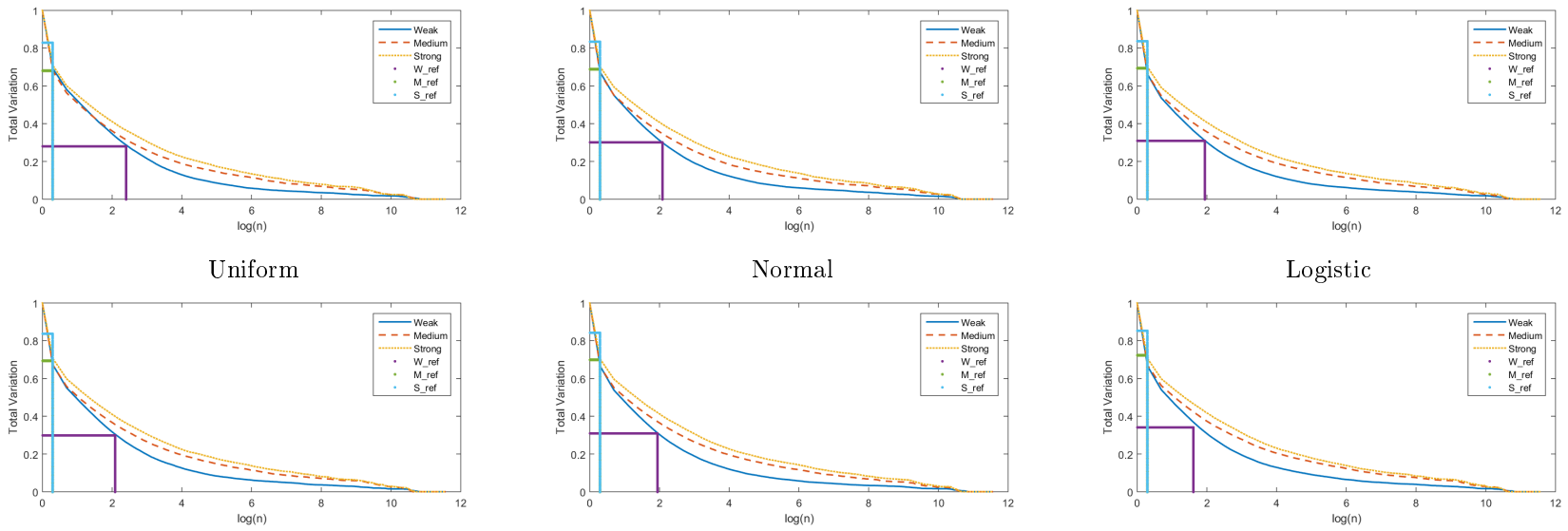
Figure 4.6: Total variation removed at each level of DPT for video (e) using $LULU$.



Uniform | Normal | Logistic

Rayleigh | Gumbel | Exponential

Figure 4.7: Total variation removed at each level of DPT for video (f) using *LULU*.



Uniform | Normal | Logistic

Rayleigh | Gumbel | Exponential

Figure 4.8: Total variation removed at each level of DPT for video (g) using $LULU$.

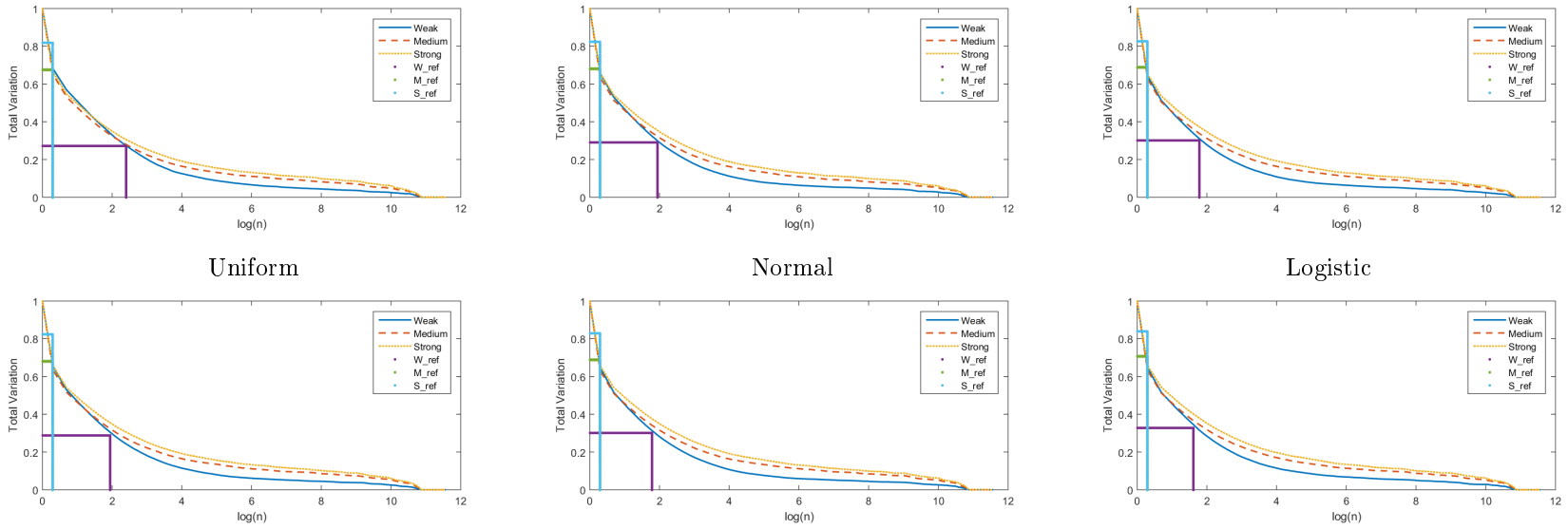


Figure 4.9: Total variation removed at each level of DPT for video (h) using $LULU$.

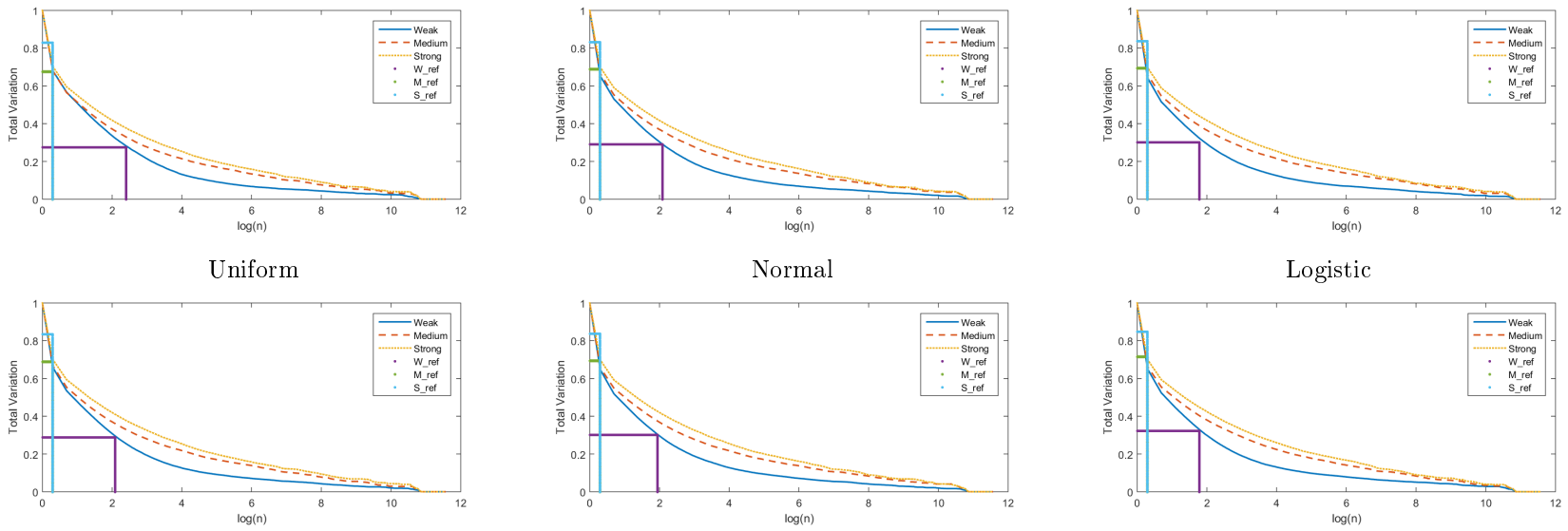


Figure 4.10: Total variation removed at each level of DPT for video (i) using *LULU*.

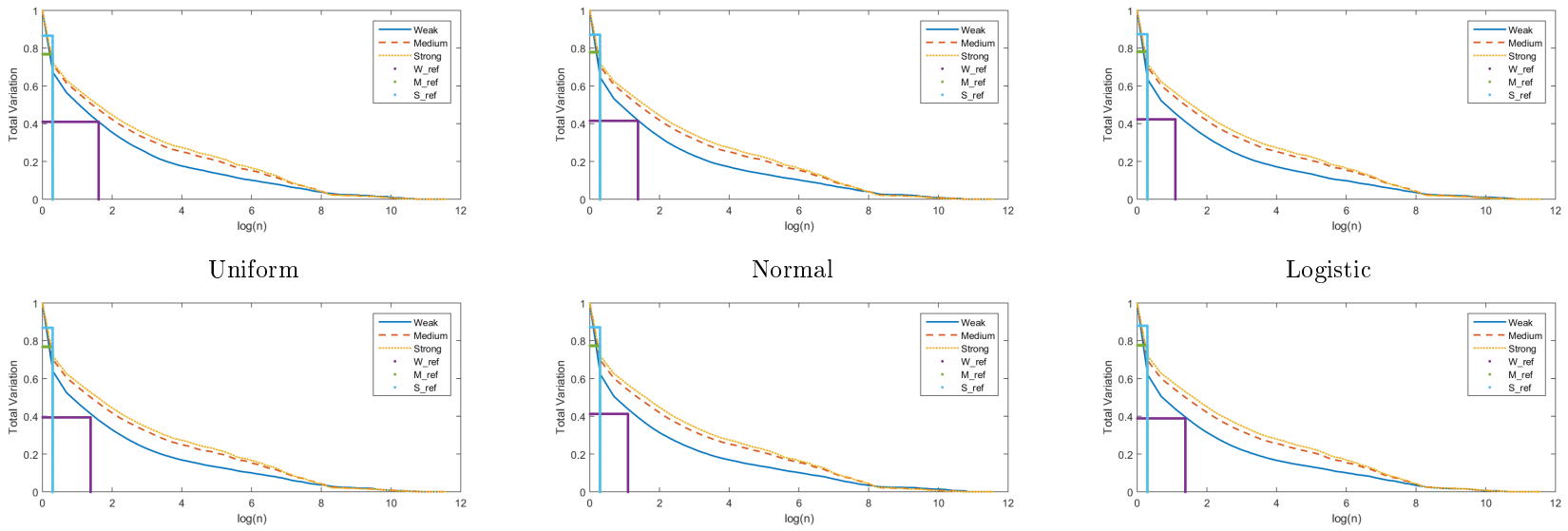


Figure 4.11: Total variation removed at each level of DPT for video (j) using *LULU*.

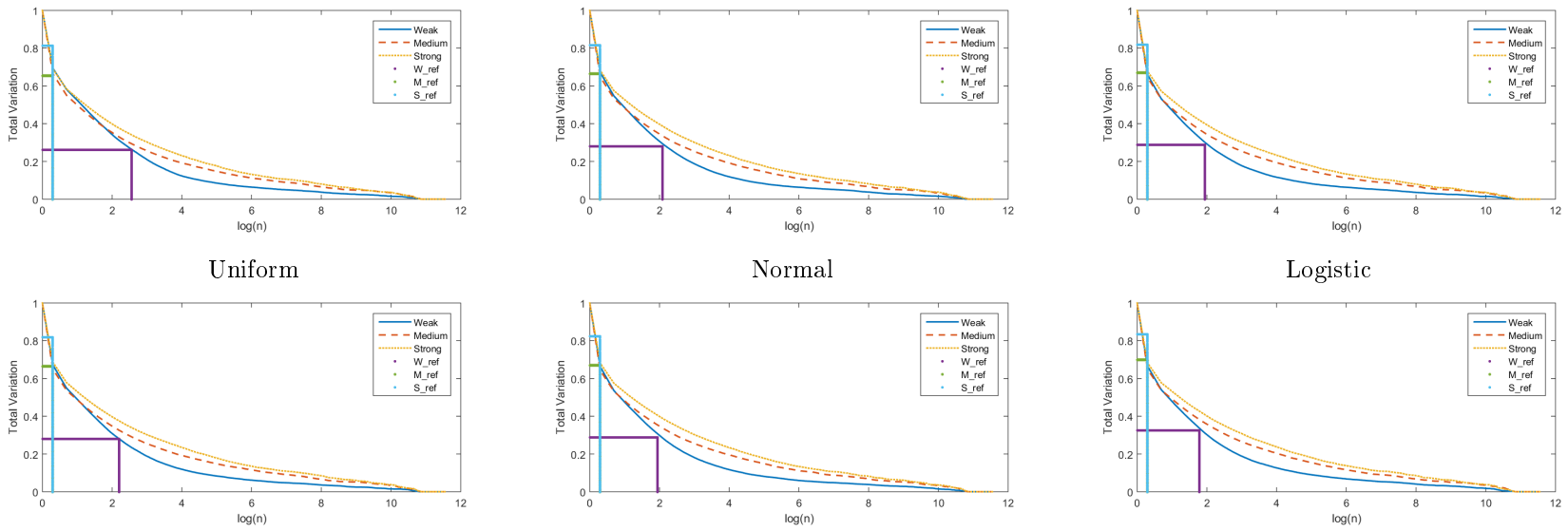


Figure 4.12: Total variation removed at each level of DPT for video (k) using $LULU$.

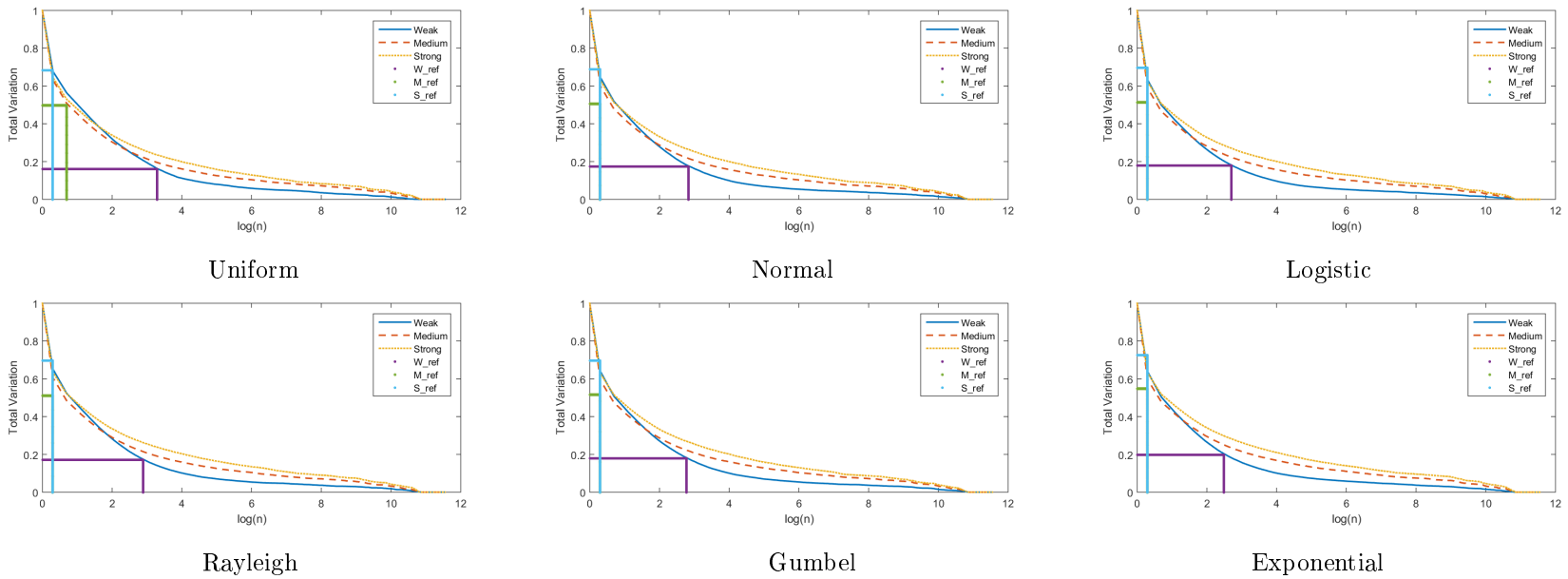


Figure 4.13: Total variation removed at each level of DPT for video (l) using *LULU*.

The noise removed in each case is determined as the pulses of the DPT up until the n^{th} as indicated in Table 4.2. We now look in detail at this removed noise and the smoothed images for each case.

4.3.2 Smoothed vs original images

In Figures 4.14 to 4.25 the SSIM index (y-axis) of the decomposed image at level n (x-axis) of the DPT with the original image is shown. The SSIM is a similarity measure, thus it is preferable to have SSIM close to 1. Note that, for all graphs, the maximum n for the SSIM calculated is such that the total variation of decomposed image at level n is smaller than or equal to the total variation of the original image. Thus the stop criterion is to decompose the noisy image until its total variation is as large as the total variation of the original image. The level of DPT required to achieve maximum SSIM is displayed in Table 4.3

For all SSIM plots, it can be seen that the set of noise distributions follows a certain order when considering the SSIM at each level of the DPT. In descending order: exponential, logistic, Gumbel, normal, Rayleigh, and uniform. That is, the SSIM of the purified image at level n of the DPT will always be the largest for the exponential distribution regardless of the signal strength for any image, and similarly for the other noise distributions.

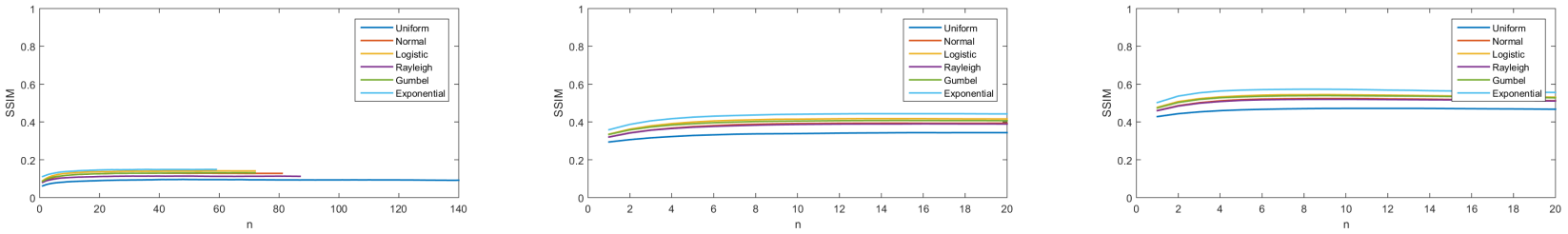
Furthermore, there are two pairs of noise distributions that behave similarly. These pairs tend to stay close to one another at each level of the DPT: logistic and Gumbel distribution, and normal and Rayleigh distribution. This is most prominent in the medium and strong signaled SSIM plots of images (a) to (d) and all the weak signaled SSIM plot of images (e) to (l).

For images (a) to (d), the SSIM for weak signals tend to increase quickly and then deteriorate slowly in the process of decomposition. In medium and strong signals, the SSIM increases quickly and stabilises. Furthermore, the SSIM plots of each noise distribution is well-separated from each other.

For image (e) to (l), the SSIM for weak signals increase during the initial phase of the DPT then decreases slowly. However, the SSIM for the other signals only decreases as the decomposition furthers on. Moreover, as the strength of the signal increases, the width of the band of distributions decreases. This is because, as a result of high variance in noise distributions, the images with weak signals have a lower similarity than those with medium and strong signals. That is, the variability of the noise causes variability of image similarity.

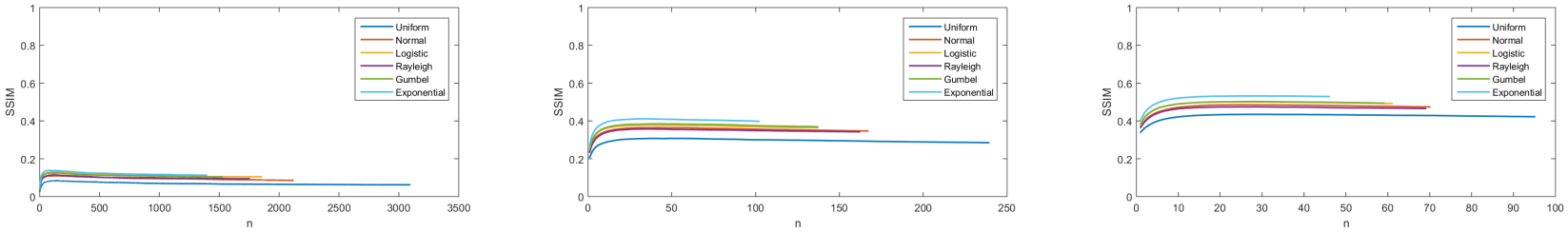
	SNR	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)
Uniform	1	48	147	62	165	15	1	8	6	9	1	9	27
	5	18	50	20	23	1	1	1	1	1	1	1	1
	9	12	27	12	21	1	1	1	1	1	1	1	1
Normal	1	31	166	81	87	9	2	5	5	7	1	7	20
	5	15	43	18	25	1	1	1	1	1	1	1	1
	9	10	26	10	16	1	1	1	1	1	1	1	1
Logistic	1	59	180	87	91	6	2	5	6	7	1	7	18
	5	14	35	18	20	1	1	1	1	1	1	1	2
	9	9	21	9	15	1	1	1	1	1	1	1	1
Rayleigh	1	50	118	79	93	9	2	6	5	7	2	10	17
	5	16	38	16	28	1	1	1	1	1	1	1	1
	9	10	29	12	18	1	1	1	1	1	1	1	1
Gumbel	1	60	100	63	98	7	2	5	4	6	2	7	20
	5	15	43	18	21	1	1	1	1	1	1	1	2
	9	9	28	10	16	1	1	1	1	1	1	1	1
Exponential	1	57	85	75	66	4	3	5	4	7	3	6	14
	5	17	33	14	26	1	1	1	1	1	1	1	2
	9	8	29	9	12	1	1	1	1	1	1	1	1

Table 4.3: The level of DPT required to achieve maximum SSIM.



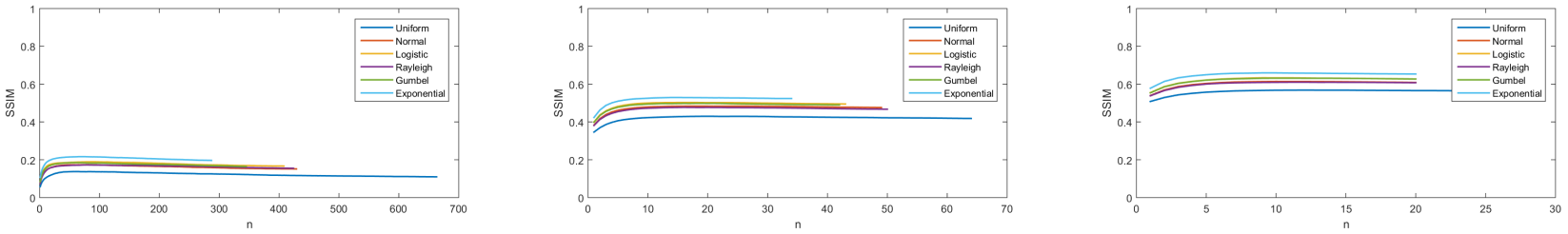
Weak Medium Strong

Figure 4.14: SSIM of noisy image at each level of DPT for video (a) using *LULU*.



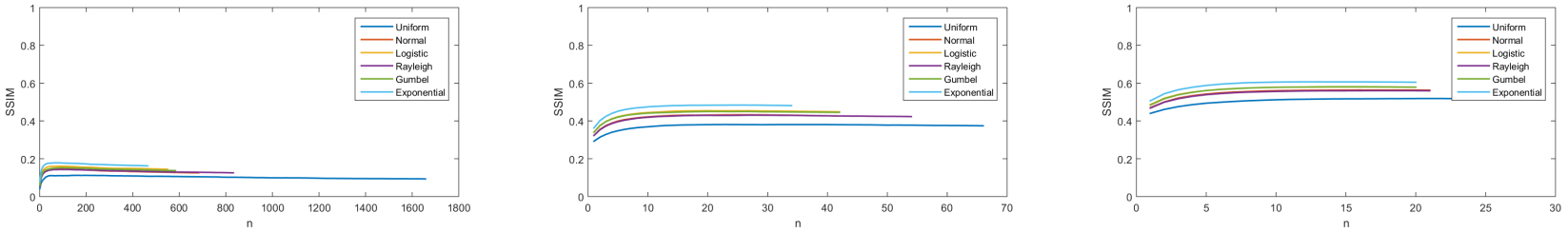
Weak Medium Strong

Figure 4.15: SSIM of noisy image at each level of DPT for video (b) using *LULU*.



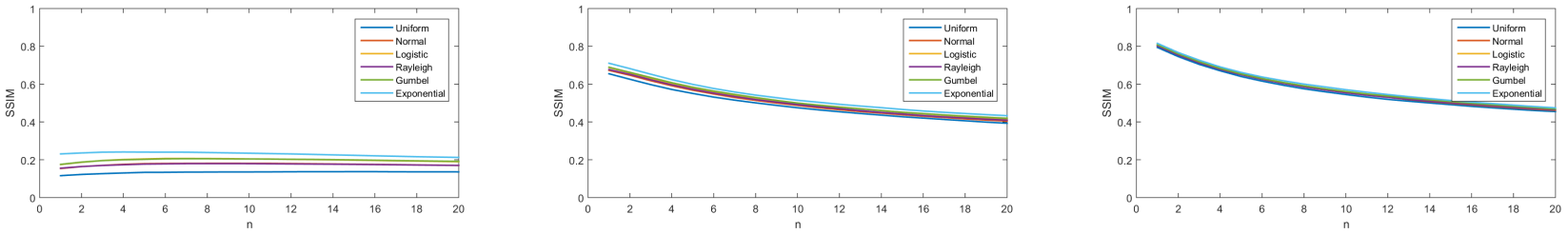
Weak Medium Strong

Figure 4.16: SSIM of noisy image at each level of DPT for video (c) using $LULU$.



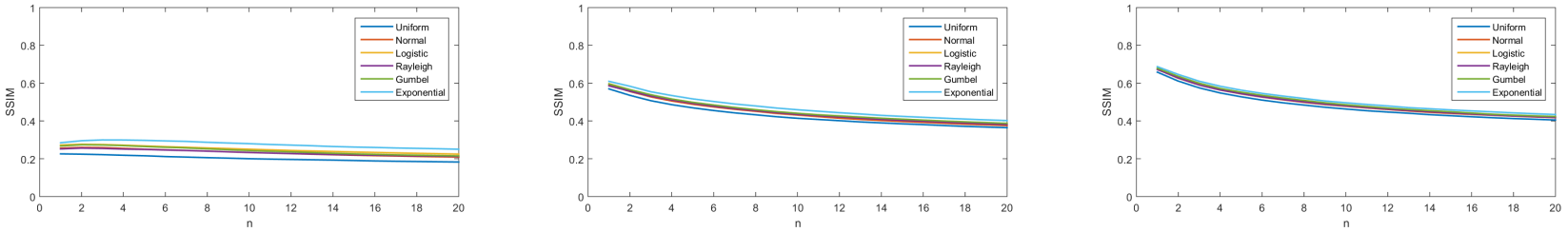
Weak Medium Strong

Figure 4.17: SSIM of noisy image at each level of DPT for video (d) using $LULU$.



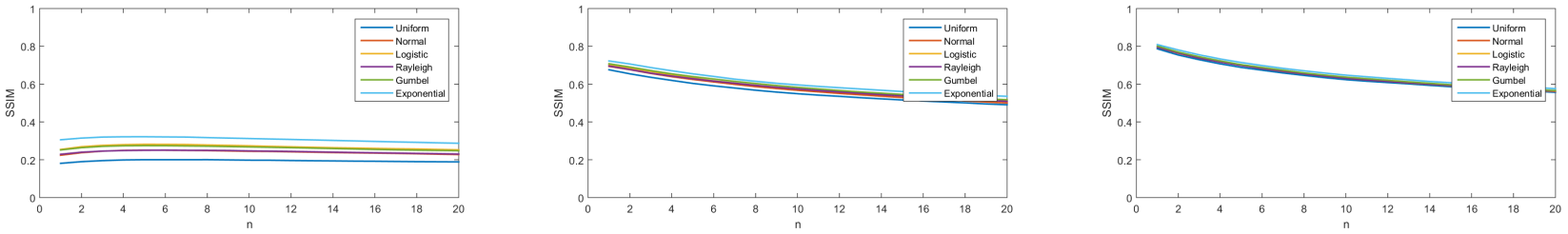
Weak Medium Strong

Figure 4.18: SSIM of noisy image at each level of DPT for video (e) using $LULU$.



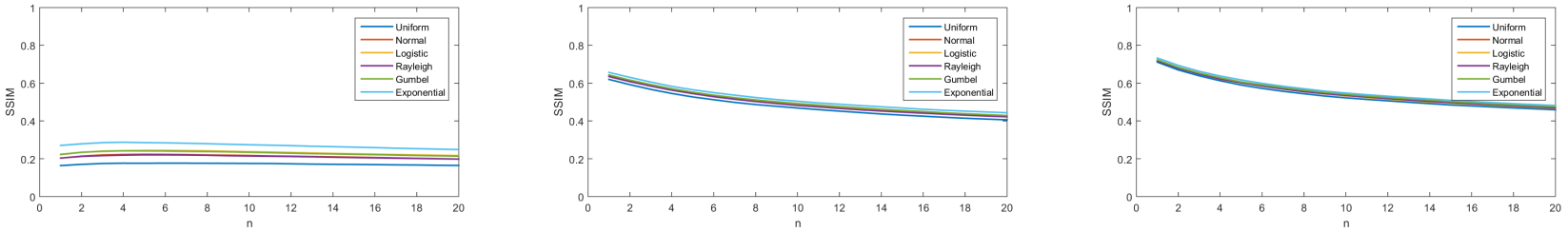
Weak Medium Strong

Figure 4.19: SSIM of noisy image at each level of DPT for video (f) using $LULU$.



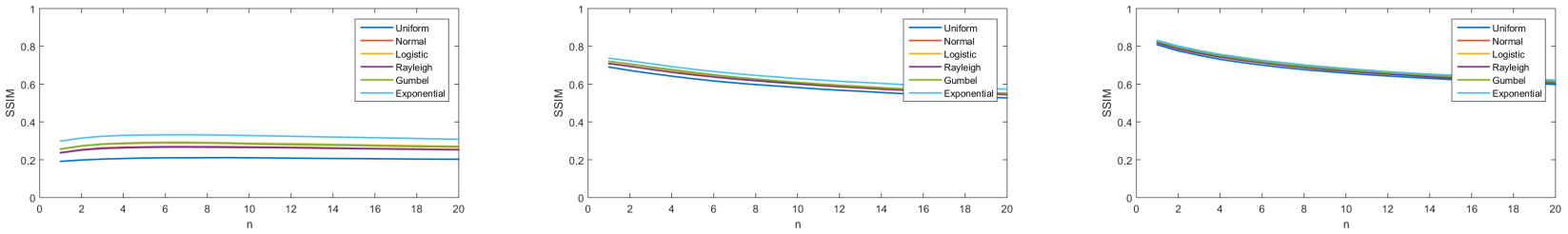
Weak Medium Strong

Figure 4.20: SSIM of noisy image at each level of DPT for video (g) using $LULU$.



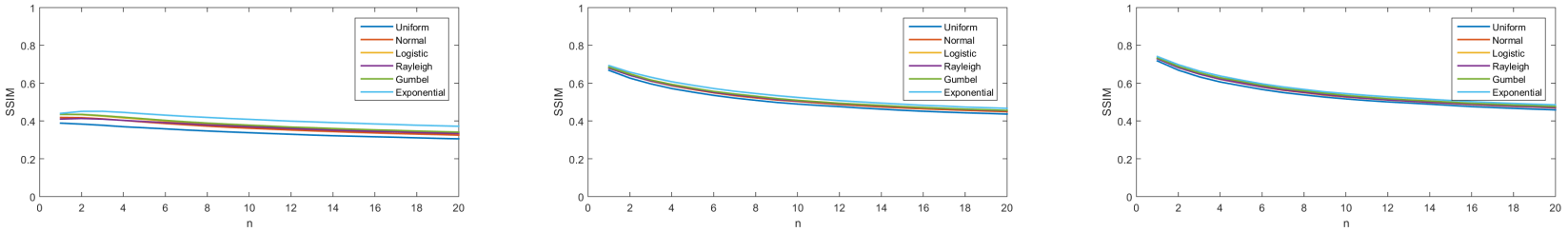
Weak Medium Strong

Figure 4.21: SSIM of noisy image at each level of DPT for video (h) using $LULU$.



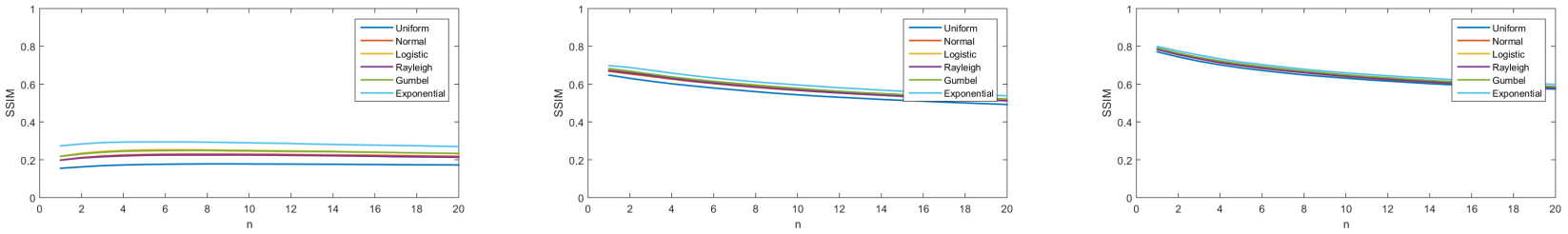
Weak Medium Strong

Figure 4.22: SSIM of noisy image at each level of DPT for video (i) using $LULU$.



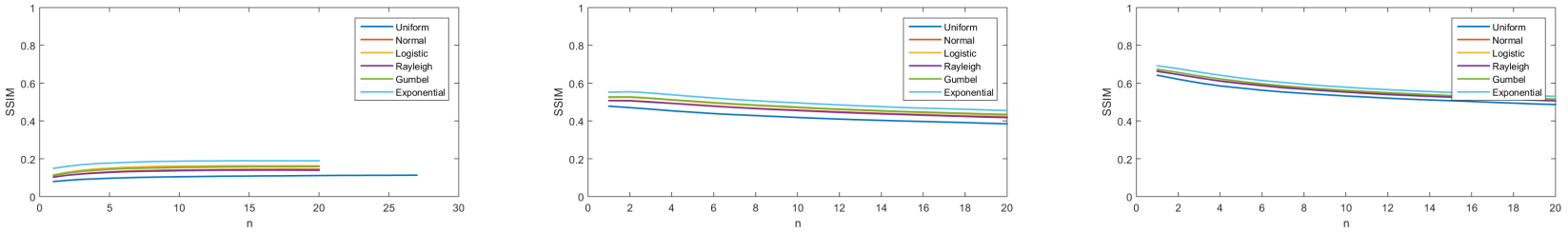
Weak Medium Strong

Figure 4.23: SSIM of noisy image at each level of DPT for video (j) using $LULU$.



Weak Medium Strong

Figure 4.24: SSIM of noisy image at each level of DPT for video (k) using $LULU$.



Weak Medium Strong

Figure 4.25: SSIM of noisy image at each level of DPT for video (l) using $LULU$.

4.3.3 Removed vs original noise

The PP plot evaluates the goodness-of-fit of a sample that is believed to follow a certain distribution by comparing the empirical CDF of the sample to the theoretical CDF. Thus, if the sample is from the specified distribution, then the graph with co-ordinates $(F(x_{k:n}), \frac{k}{n})$, where $x_{k:n}$ is the k^{th} ordered observation from a sample size of n and $F(x)$ is the theoretical distribution function [39]. Here our sample is the noise extracted at position (i, j) at level n of the DPT as $e_{ij}^{(n)} = [(I - L_n U_n)f]_{ij}$. The level of DPT required to achieve minimum SSE is displayed in Table 4.4[3].

In Figures 4.26 to 4.37 the plots of the SSE of the PP plot (y-axis) to the n^{th} level (x-axis) of the DPT is shown. Note that the stop criterion is the same as that for the SSIM plots in Section 4.3.2. Since SSE is a measure for the amount of error, it is preferable for SSE to be small.

For images (a) to (d), an ordering in SSE can be observed for five of the six noise distribution at each level of the DPT (in ascending order): logistic, normal, Rayleigh, uniform, and Gumbel. For these distributions, the SSE is strictly increasing and is concave. This implies that the larger the pulses $LULU$ extracts from the image the less the noise resembles its anticipated distribution, however, this is true since the noise added to the images were all independent pulses of size 1.

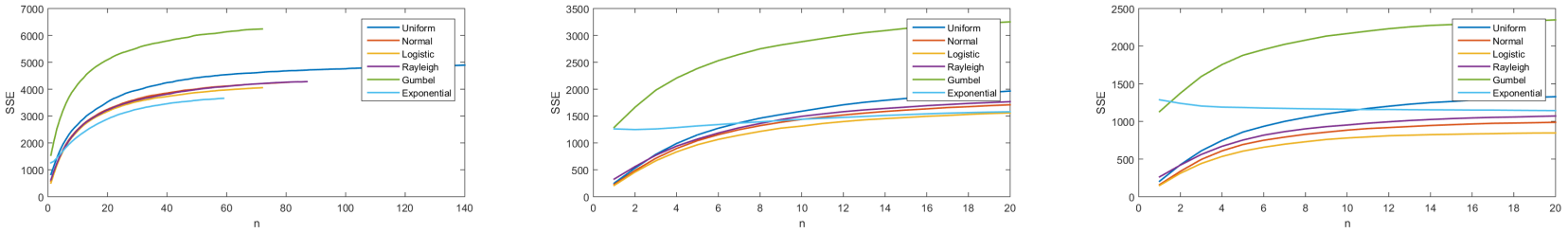
In contrast to its peers, the exponential distribution does not behave consistently in terms of signal strength. For weak and medium signal strength, the shape of the SSE is concave, however, for strong signals it is convex. Notice also that the starting SSE for each signal strength (with respect to an image) is approximately the same. This is because the exponential pdf contains a sharp upward tail on the left side and the noise sample is relatively symmetric (i.e. equally weighted above and below zero). This renders the ability to mimic the cdf of the exponential distribution with only one noise sample from the images difficult and so results in the recurring starting SSE.

For images (e) to (l), the same order exists in the weak signal SSE plots of images (a) to (d). In the plots of medium and strong signals, it is clear that the exponential and Gumbel distributions perform worst.

[3]The code used to determine the minimum SSE is written such that at least the first levels of the DPT will be run to capture a range of values of the DPT and the maximum level is equal when the total variation of the noisy image is close to the original image. That is, if the level of DPT required to reach minimum SSE is 20, it means the PP plot improves as the decomposition furthers on. However, this speculation is spurious as the noise simulated is of size 1.

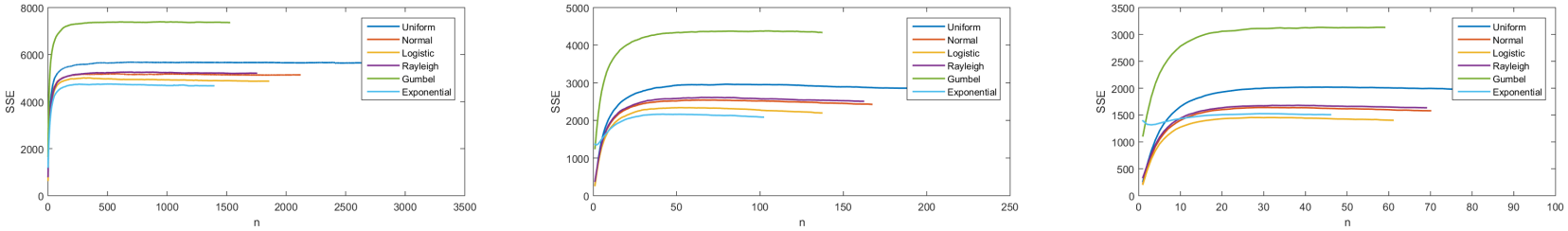
	SNR	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)
Uniform	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1
Normal	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	20	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1
Logistic	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	20	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1
Rayleigh	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	20	1	20	20	20	1	20	1
	9	1	1	1	1	5	2	7	5	9	2	10	1
Gumbel	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	20	1	20	20	20	1	20	1
	9	1	1	1	1	4	7	6	3	8	1	10	20
Exponential	1	1	1	1	1	1	3	1	1	1	18	1	1
	5	2	2	2	2	5	20	20	20	20	20	19	20
	9	20	3	19	20	1	13	8	14	6	20	7	20

Table 4.4: The level of DPT require to acquire minimum SSE of noise sample.



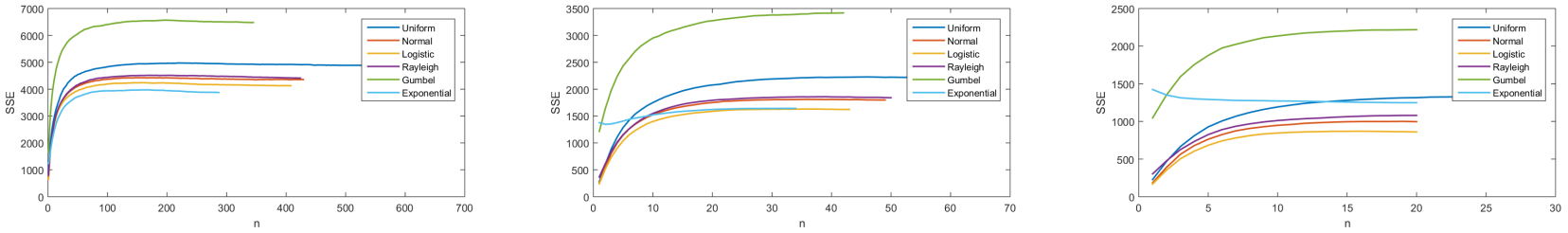
Weak Medium Strong

Figure 4.26: SSE of noise at each level of DPT for video (a) using $LULU$.



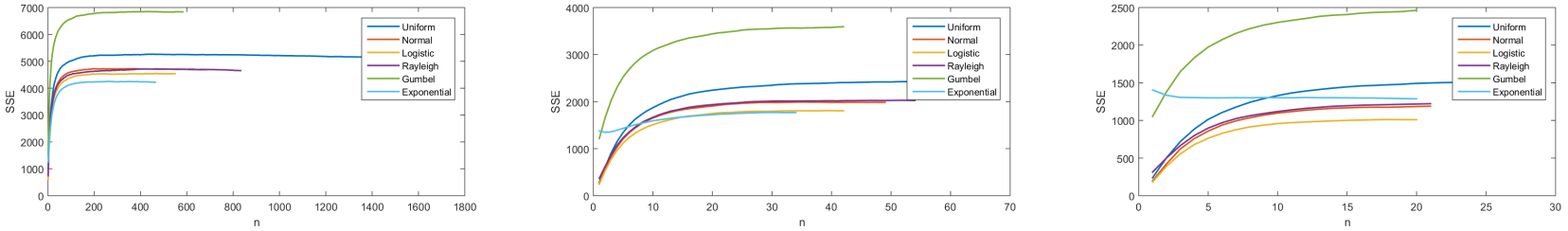
Weak Medium Strong

Figure 4.27: SSE of noise at each level of DPT for video (b) using $LULU$.



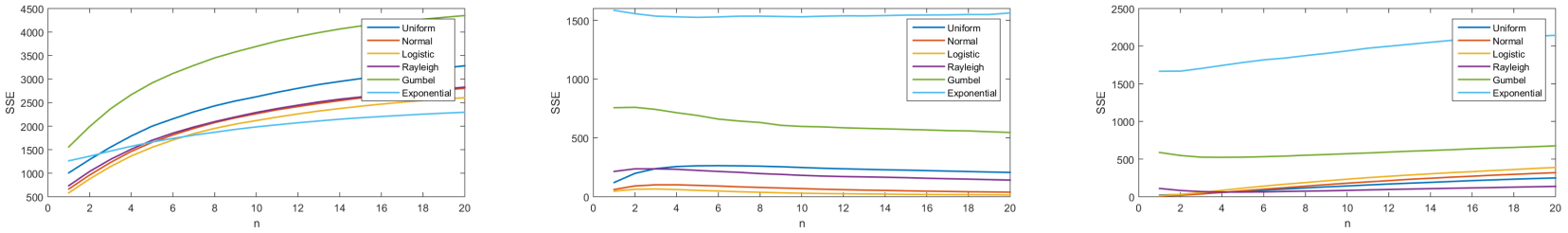
Weak Medium Strong

Figure 4.28: SSE of noise at each level of DPT for video (c) using $LULU$.



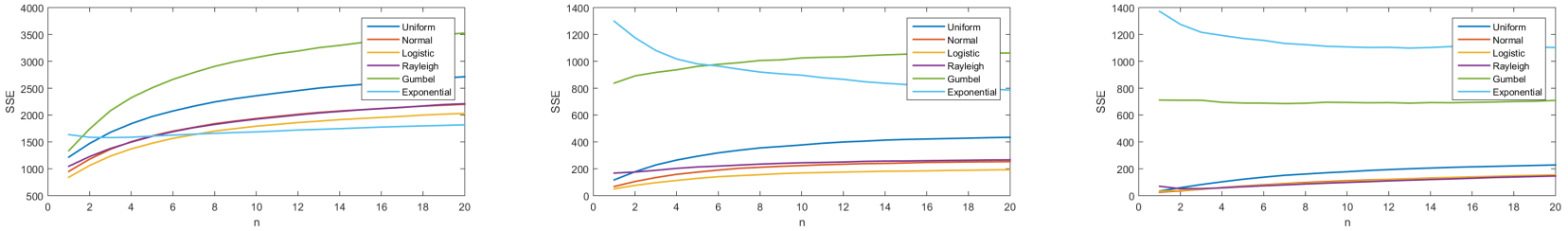
Weak Medium Strong

Figure 4.29: SSE of noise at each level of DPT for video (d) using $LULU$.



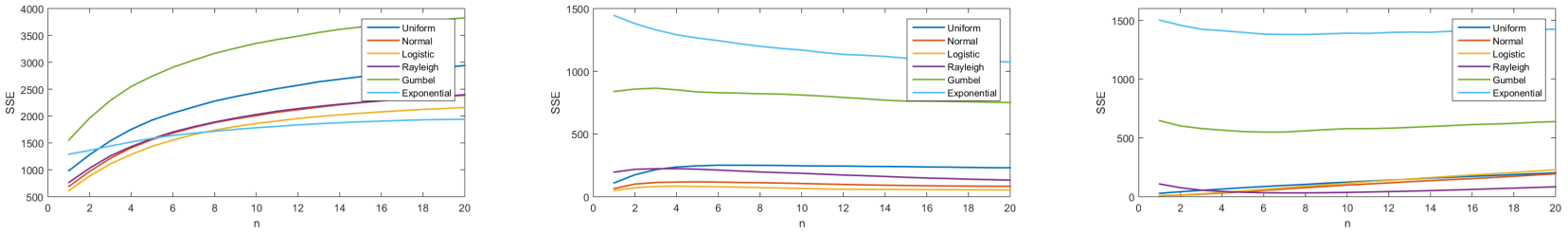
Weak Medium Strong

Figure 4.30: SSE of noise at each level of DPT for video (e) using $LULU$.



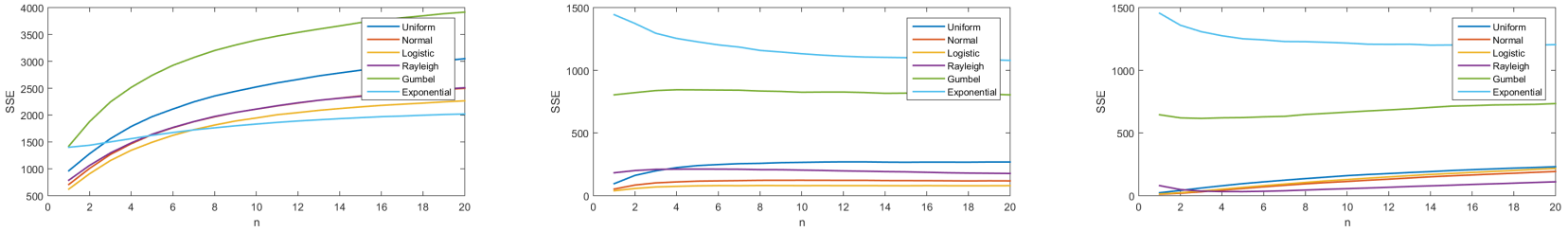
Weak Medium Strong

Figure 4.31: SSE of noise at each level of DPT for video (f) using $LULU$.



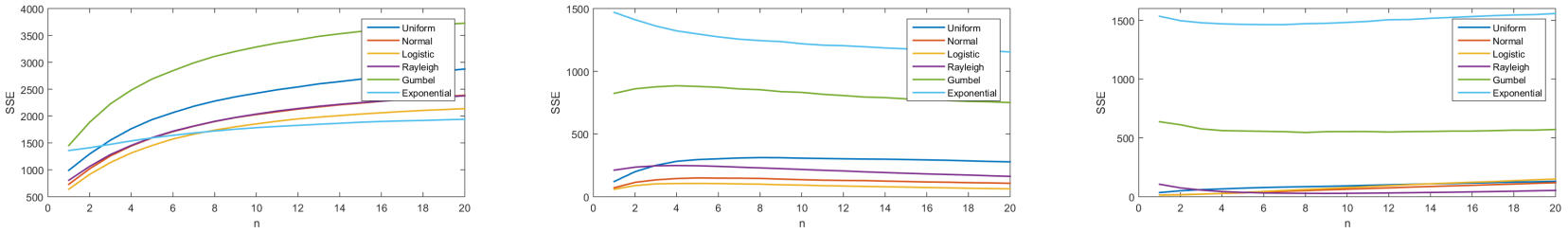
Weak Medium Strong

Figure 4.32: SSE of noise at each level of DPT for video (g) using $LULU$.



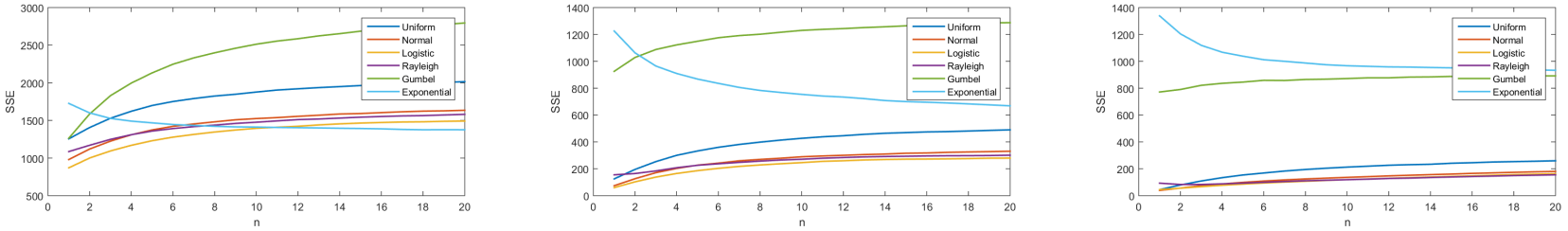
Weak Medium Strong

Figure 4.33: SSE of noise at each level of DPT for video (h) using $LULU$.



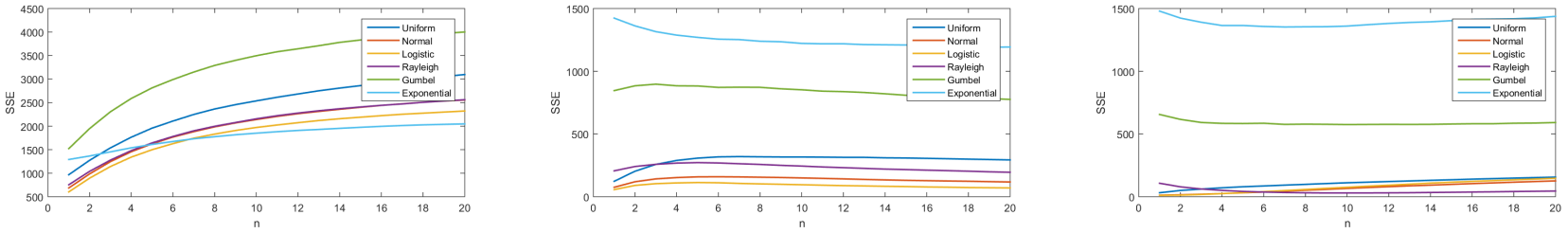
Weak Medium Strong

Figure 4.34: SSE of noise at each level of DPT for video (i) using *LULU*.



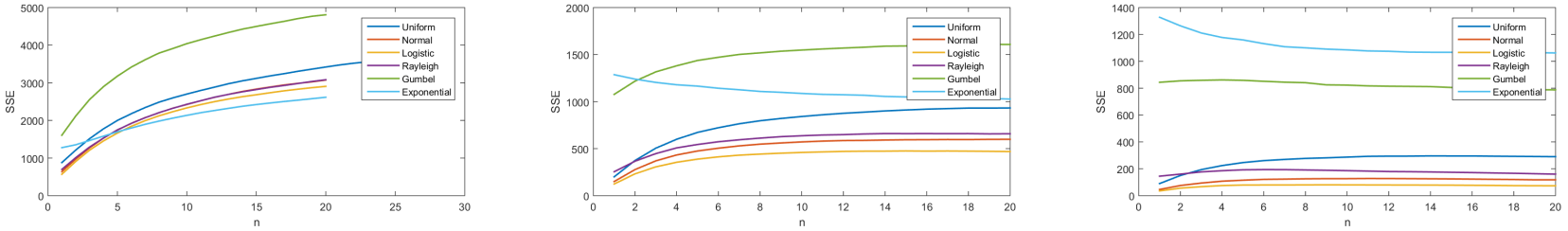
Weak Medium Strong

Figure 4.35: SSE of noise at each level of DPT for video (j) using *LULU*.



Weak Medium Strong

Figure 4.36: SSE of noise at each level of DPT for video (k) using $LULU$.



Weak Medium Strong

Figure 4.37: SSE of noise at each level of DPT for video (l) using $LULU$.

4.3.4 PP plots for extracted noise samples with minimum SSE

In Figures 4.38 to 4.49 we plot the PP plots for n^{th} level of the DPT which yields the lowest SSE. That is, the graphs show the best possible PP plots attainable throughout the decomposition of the noisy images.

In general it can be seen that the PP plots with weak signals struggles to match its original distribution and that the performance of noise from images with medium and strong signals are the same. Furthermore, the PP plots of the noise from weak signals are smooth, this is followed by the noise from medium signals, and lastly the noise from strong signals possess a 'block' quality about its PP plots. This is because when we consider the range of values the weak, medium, and strong signals can possess and the fact the noise is discretized, the noise from weak signals have a larger range (because of a larger variance) than the noise from strong signals, thus its 'block' feature is masked when the PP plot normalises percentile values to [0,1].

In terms of distributions, the PP plots of the noise from symmetric distributions are more linear than those of asymmetric distributions. Which implies that the $LULU$ operators extract noise more effectively for distributions which have equal weights in values below and above zero. There are however alternative compositions of L_n and U_n available, see [14]. Their ability for smoothing in such cases should be investigated.

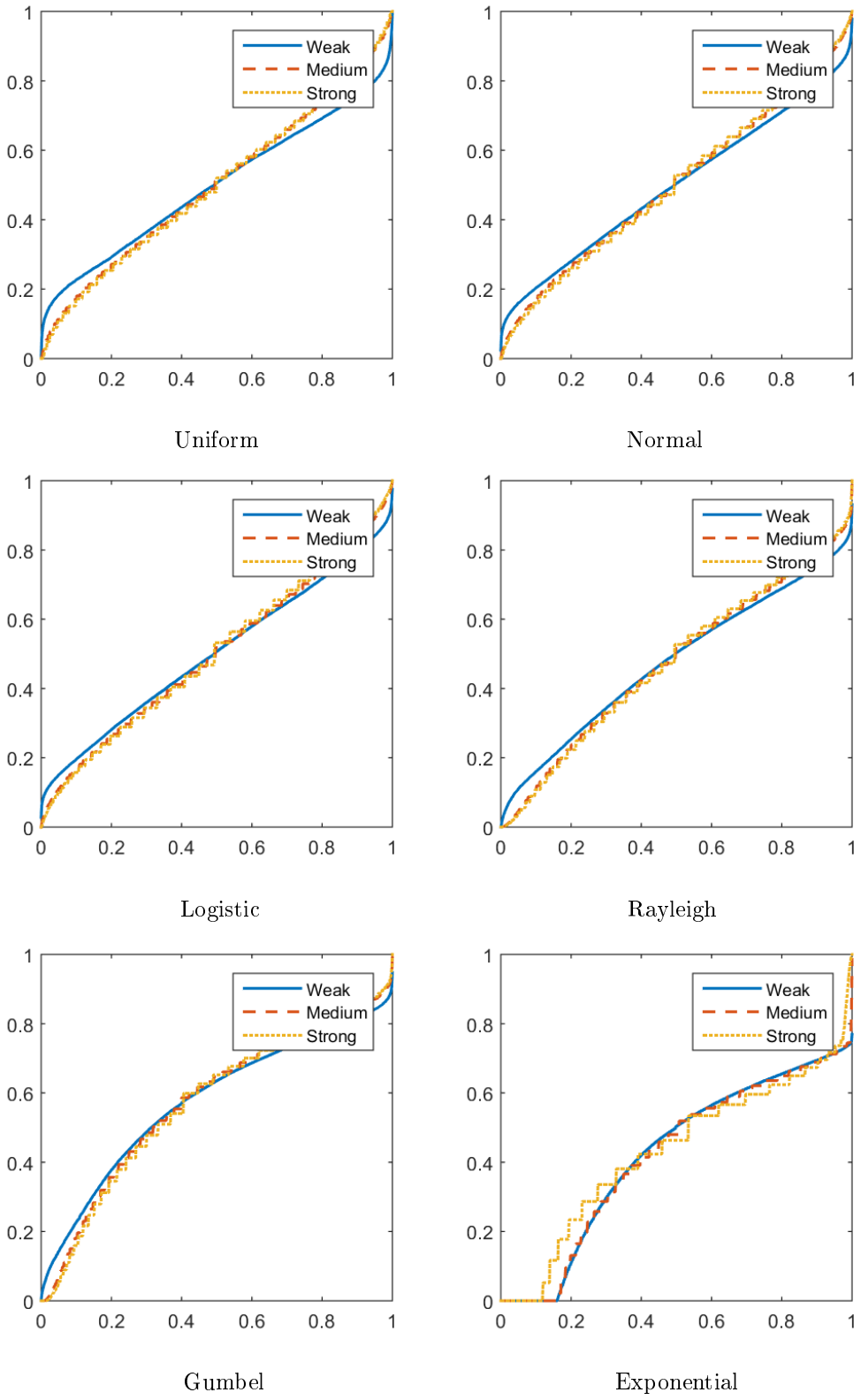


Figure 4.38: PP plot with minimum SSE for video (a) using $LULU$.

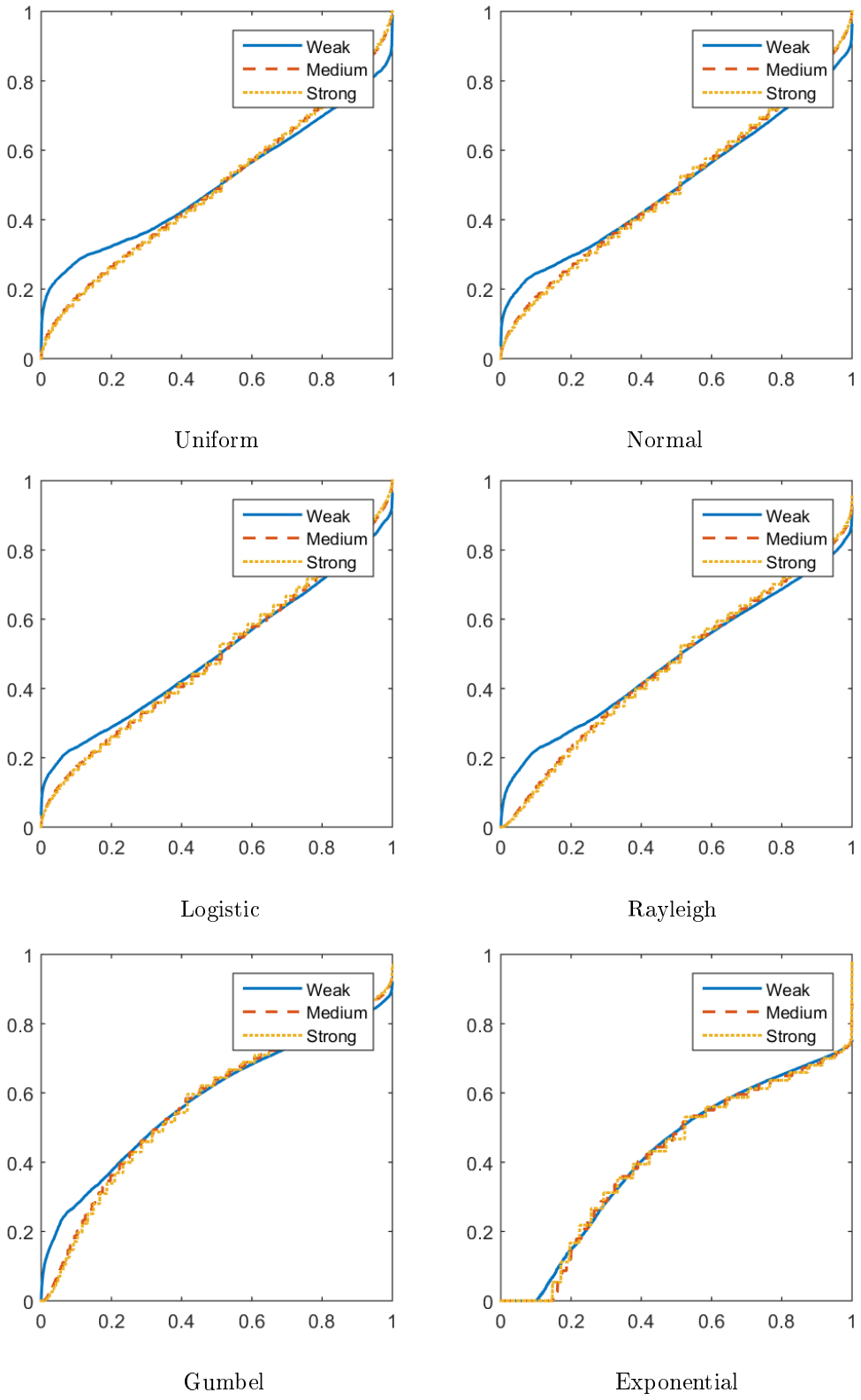


Figure 4.39: PP plot with minimum SSE for video (b) using $LULU$.

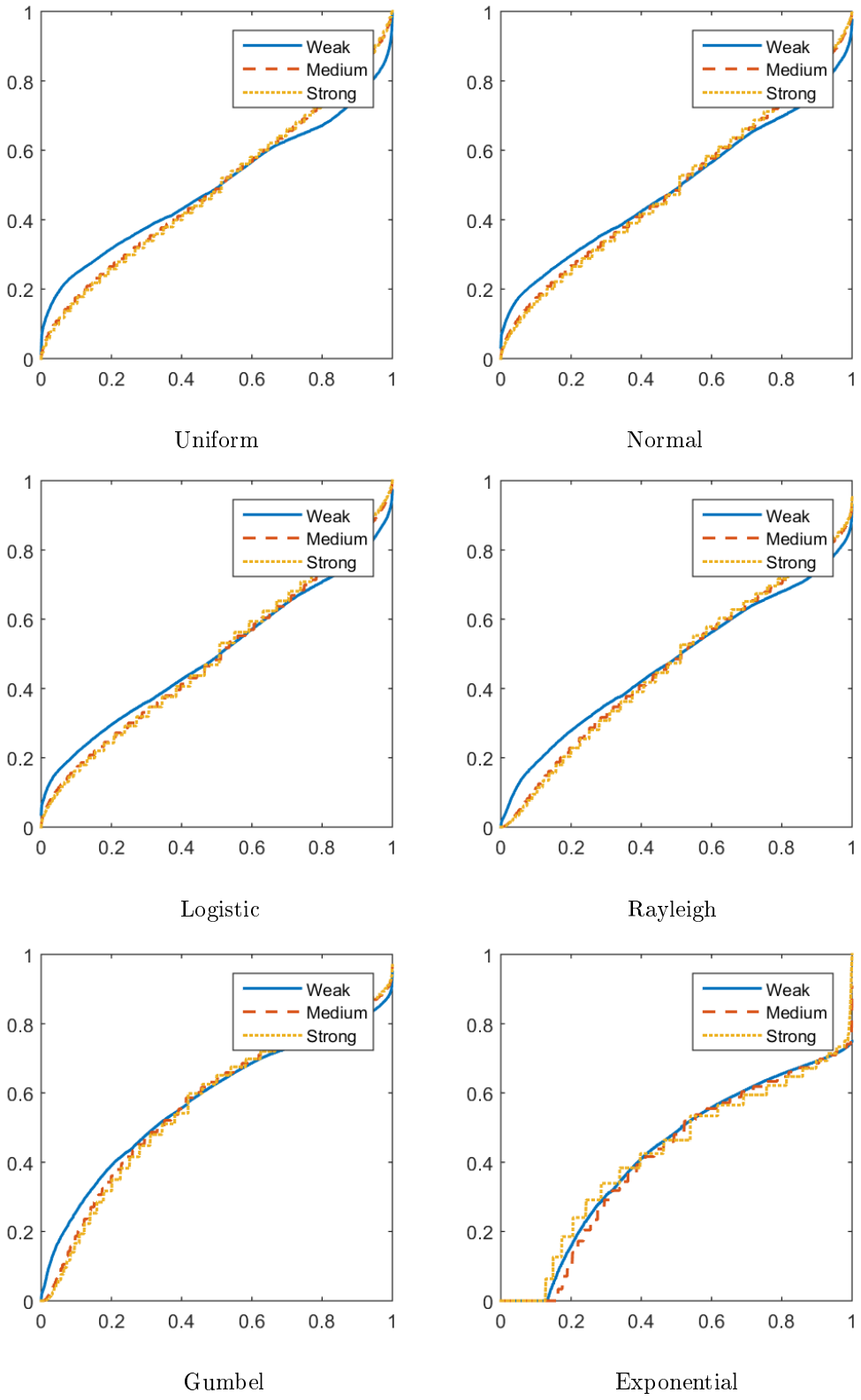


Figure 4.40: PP plot with minimum SSE for video (c) using *LULU*.

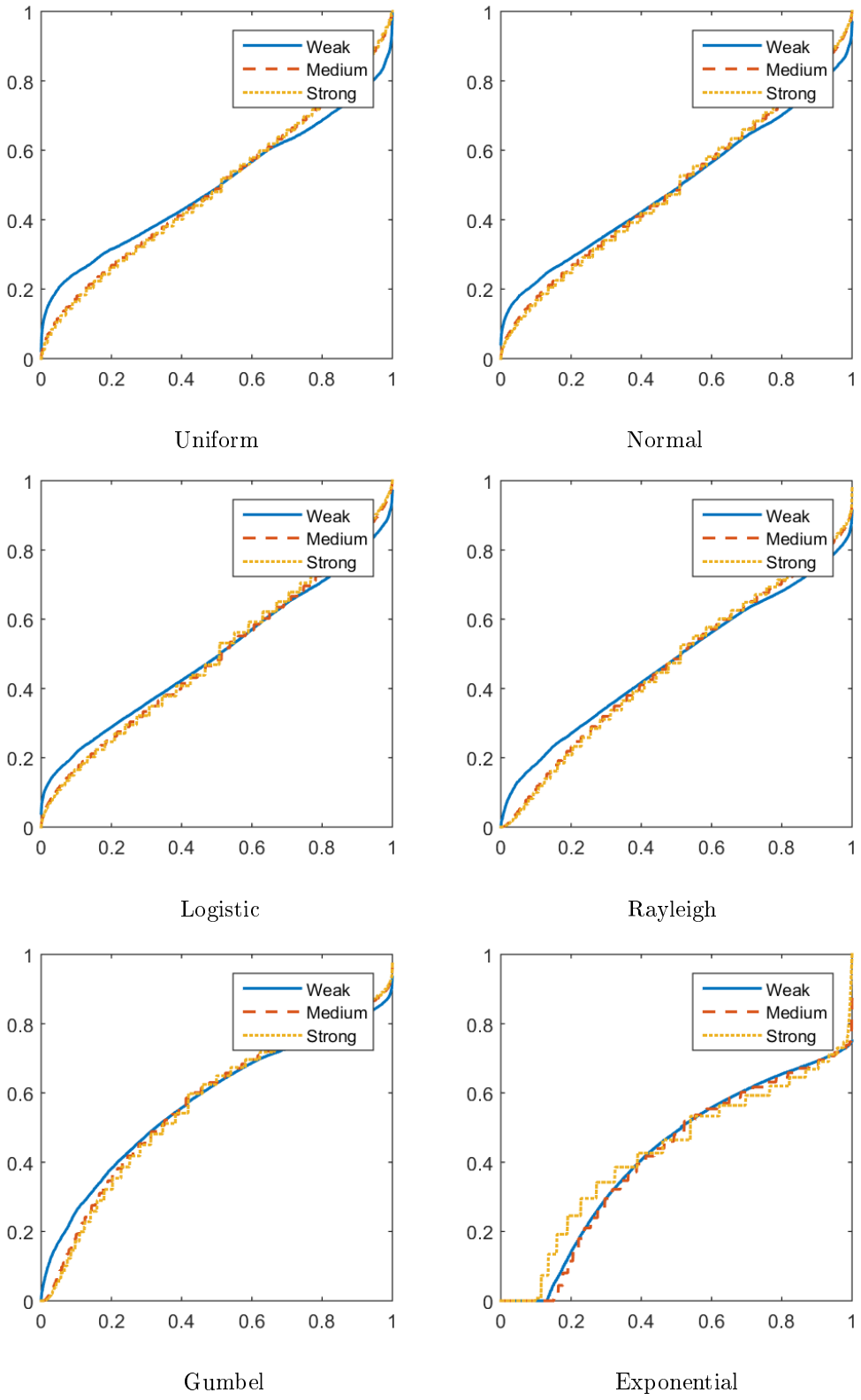


Figure 4.41: PP plot with minimum SSE for video (d) using *LULU*.

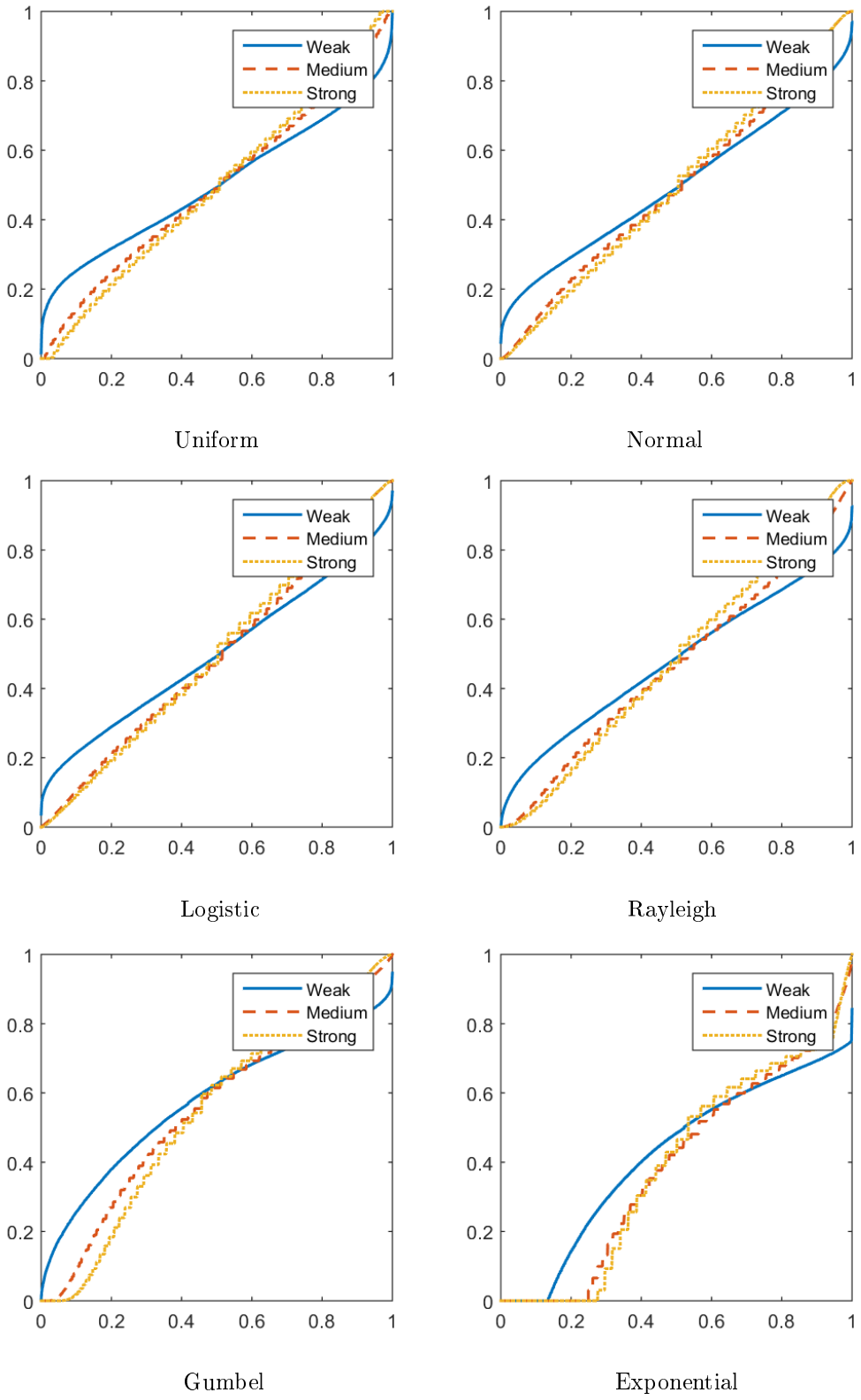


Figure 4.42: PP plot with minimum SSE for video (e) using *LULU*.

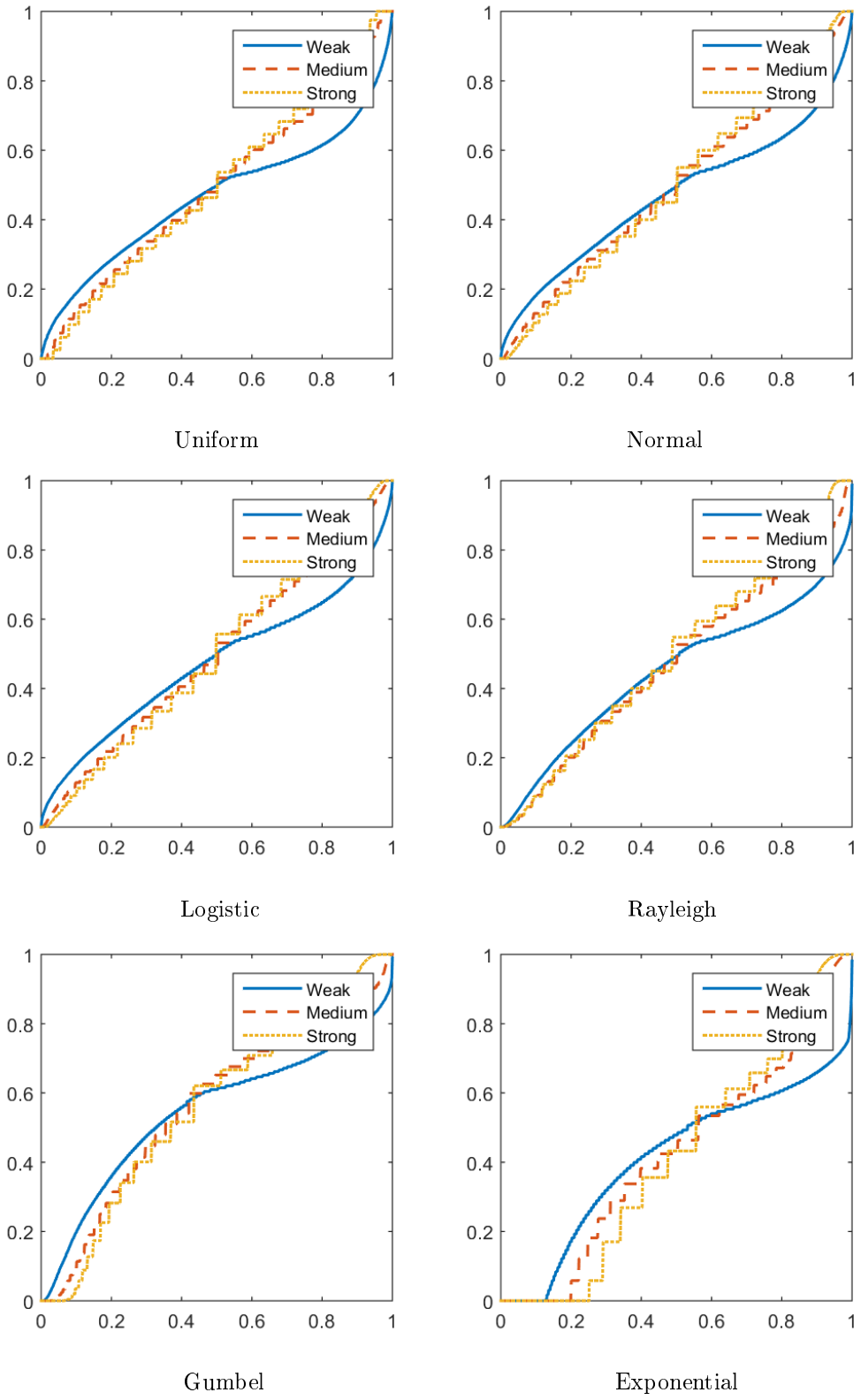


Figure 4.43: PP plot with minimum SSE for video (f) using *LULU*.

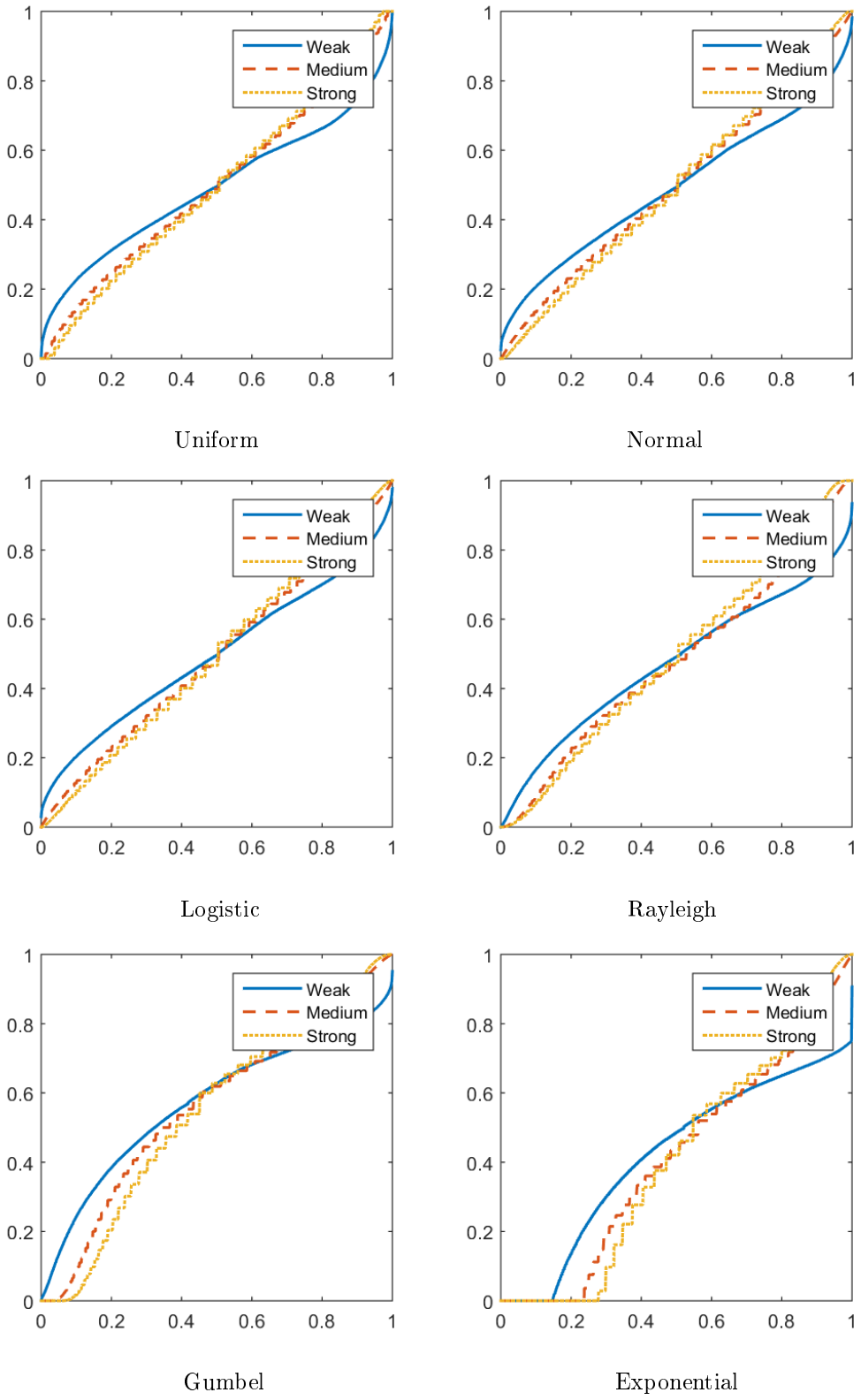


Figure 4.44: PP plot with minimum SSE for video (g) using $LULU$.

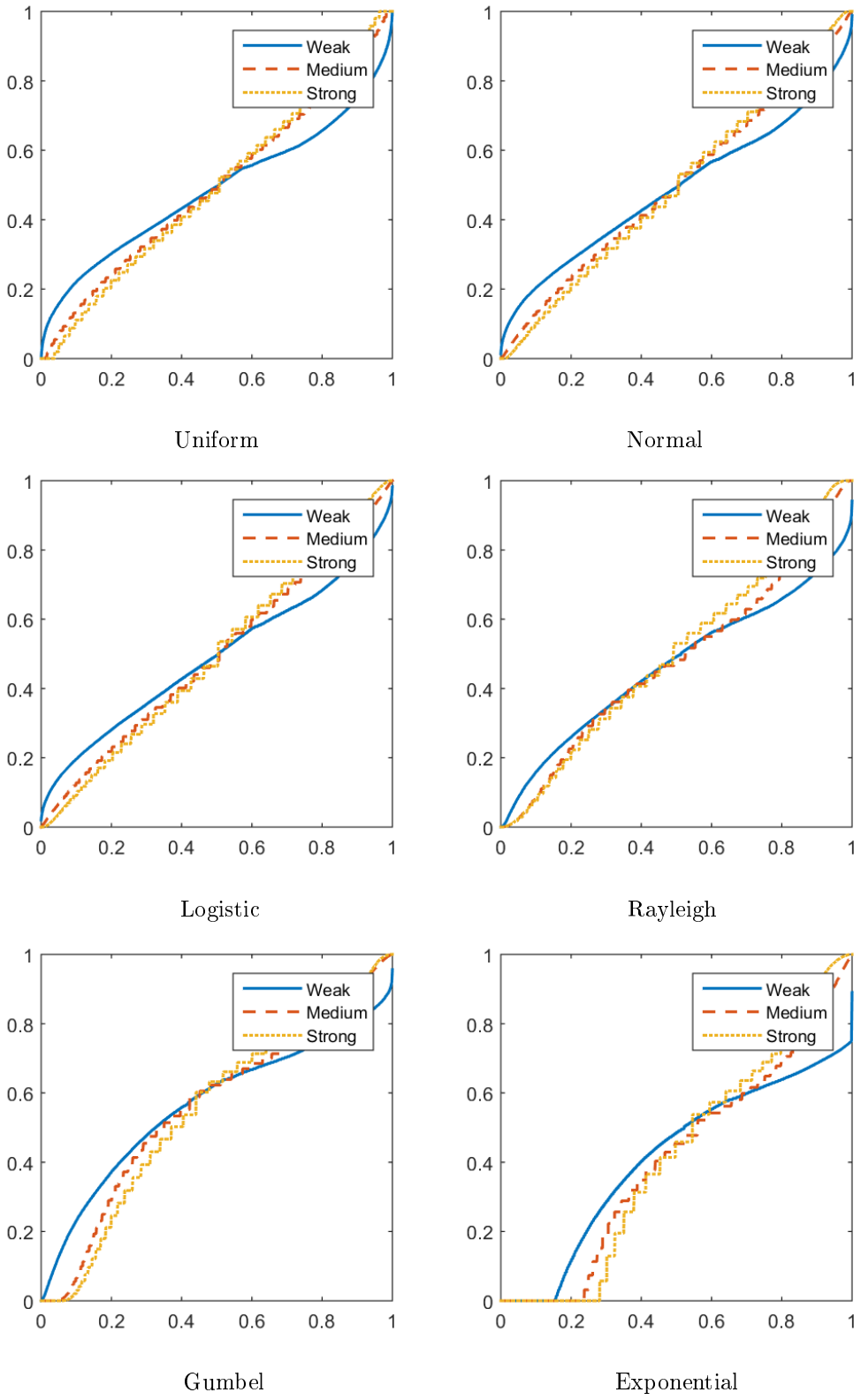


Figure 4.45: PP plot with minimum SSE for video (h) using *LULU*.

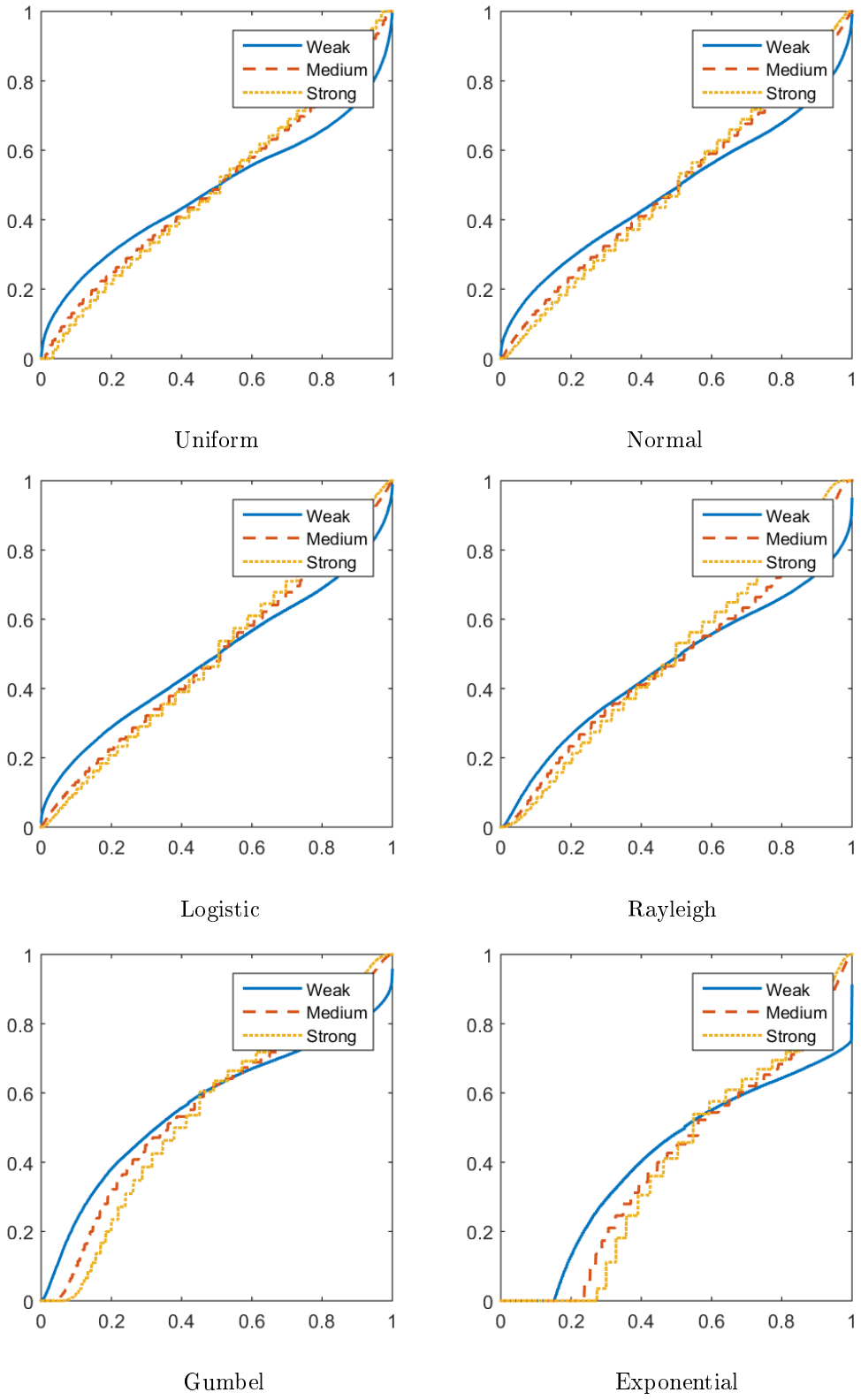


Figure 4.46: PP plot with minimum SSE for video (i) using $LULU$.

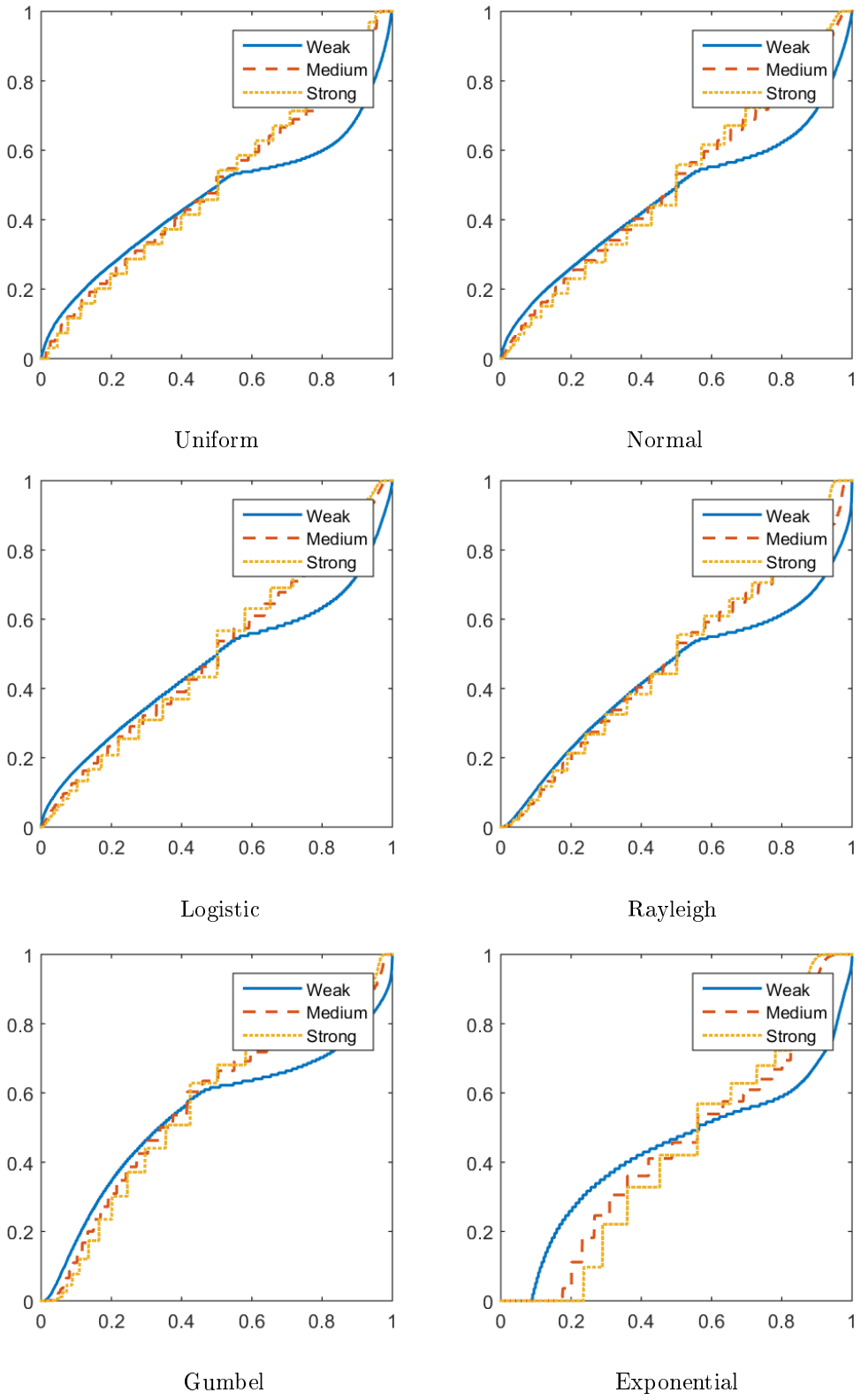


Figure 4.47: PP plot with minimum SSE for video (j) using $LULU$.

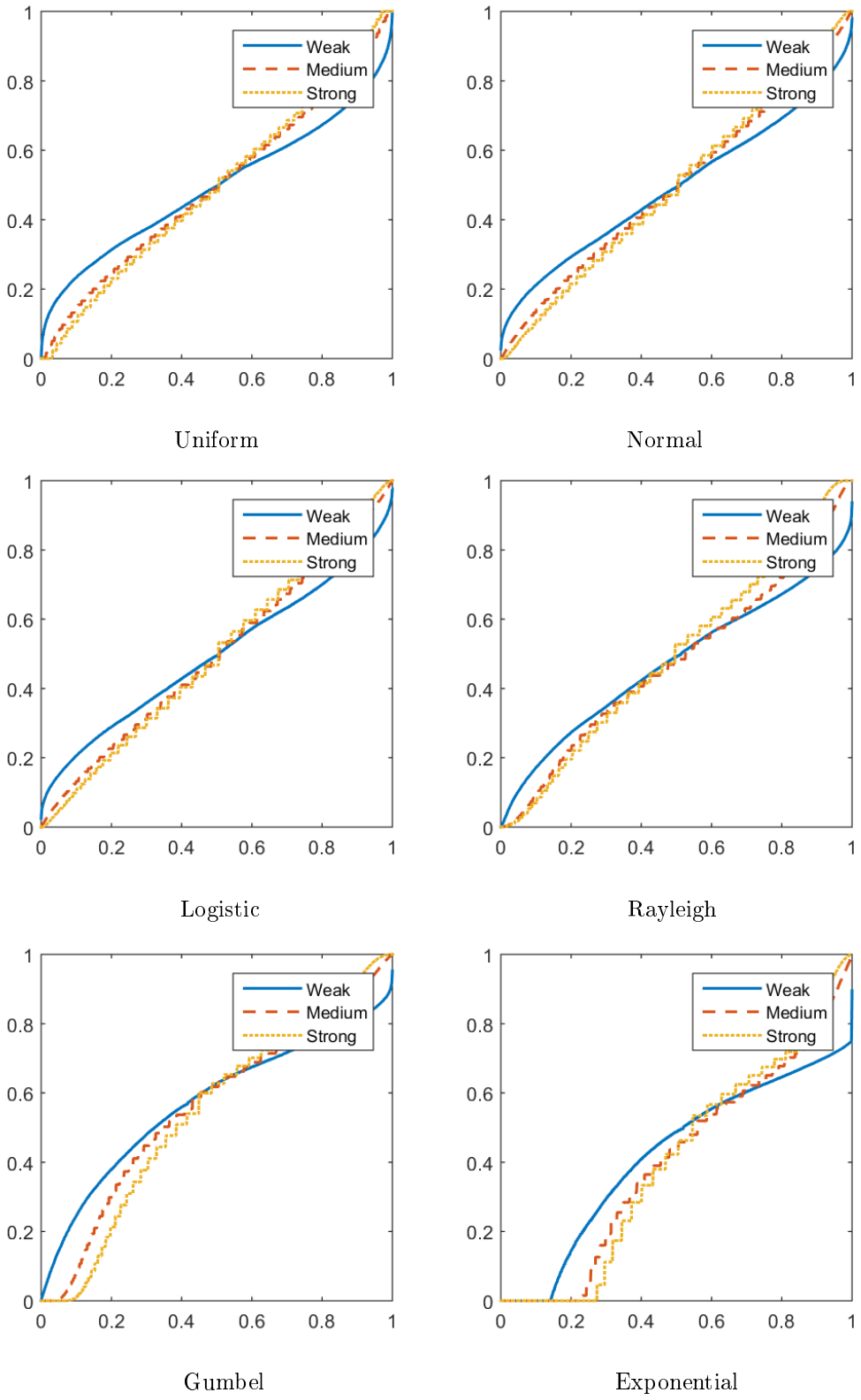


Figure 4.48: PP plot with minimum SSE for video (k) using *LULU*.

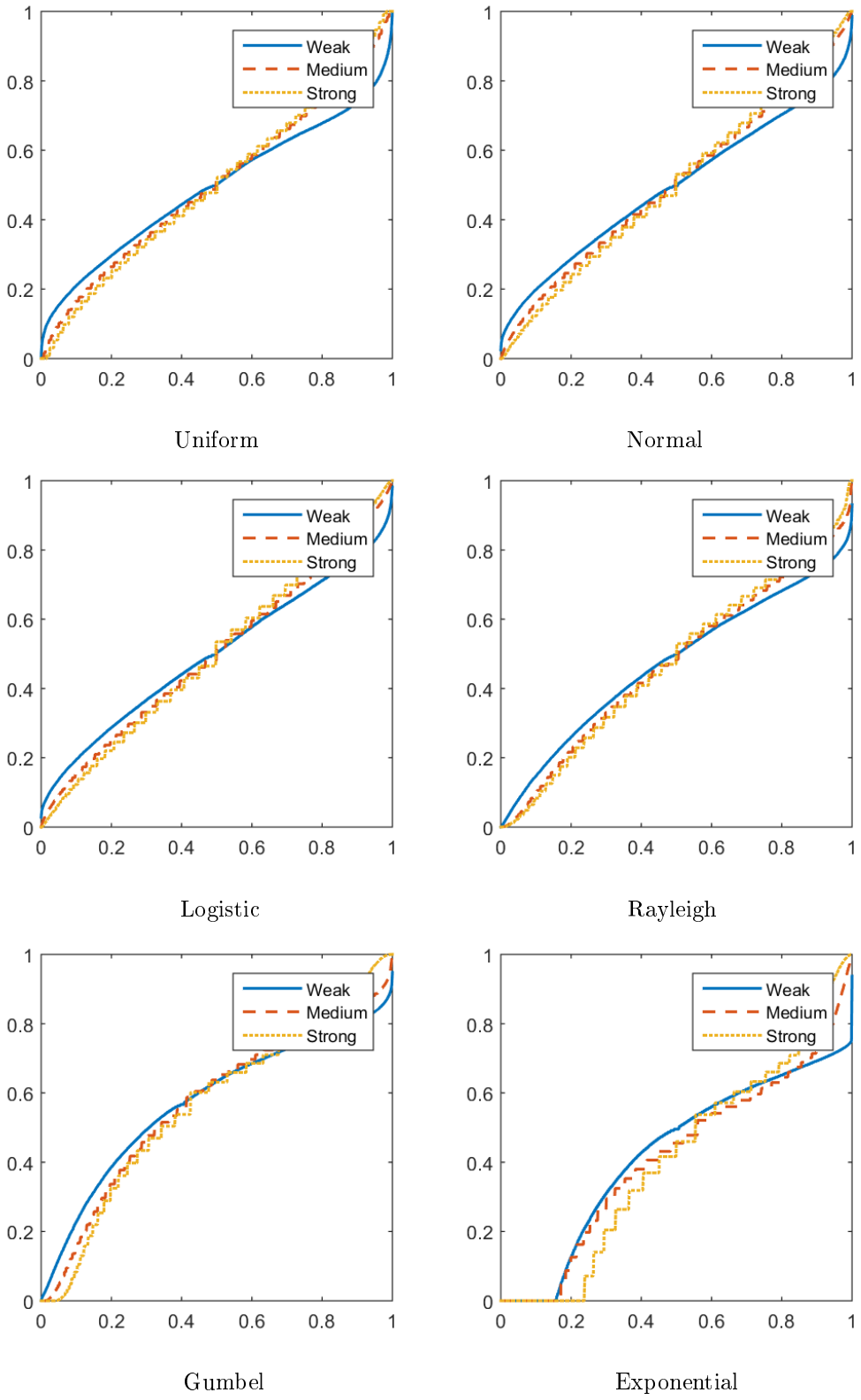


Figure 4.49: PP plot with minimum SSE for video (l) using $LULU$.

4.3.5 Final observations

In Tables 4.5 to 4.16 we provide a summary of the results for n which highlight the study. For each noise distribution and at each SNR level, we consider three criteria:

1: The n^{th} level of the DPT which yields the total variation closest to the total variation of the original image.

2: The n^{th} level of the DPT which yields the smallest SSE for the PP plots.

3: The n^{th} level of the DPT which yields the highest SSIM.

In addition we include the SSE and SSIM for the corresponding n level. Ideally, the preferable outcome is that the n which satisfies all three criteria is the same since that indicates, for some n:

- The total variation at level n is approximately the same as that of the original image.
- For that same n, the noise that is extracted matches best to the simulated noise distribution.
- Finally, for the same n, the image retrieved is most similar to the original image.

Thus, having satisified all three criterias simultaneously with the same n implies that no extra or deficit variation resulted from the decomposition, and the noise was neither over- or under- extracted.

In general, we see that the value of SSIM increases as the signal strength increases, as well as that the SSE of the PP plots decreases as the signal strength decreases. This is expected since the images with weak signals contain noise with high variance which greatly distorts the original image and the retrieval is not performed on the full noise distribution since the pixel values are limited to 0 and 255. Similarly for the other signal strengths.

For images (a) to (d) we make the following comments:

- For weak signals, a large n is usually required to reach the original variation. For strong signals it is the opposite.
- The level of DPT required to achieve best PP plot is 1 (except for exponential).

The implication is that the noise distributions with high variance (weak signal) inflate the total variation more prominently than the noise distributions with small variance (strong signal). Furthermore, the level $n = 1$ of DPT required to obtain the best PP plot indicates that the $LULU$ operators can retrieve noise effectively since all simulated noise are of size 1.

For images (e) to (l) we make the following comments:

- For weak signals, a small n is required to reach the original variation. For the other signal strengths, a $n = 1$ is sufficient.

- Triplets of 1's appear consistently for:
  - Uniform, SNR = 5 and 9
  - Normal, SNR = 9

  This implies that the $LULU$ operators are able to effectively retrieve images with noise distributed as uniform or normal.

The difference in the level of DPT required to reach original total variation between images (a) to (d) and (e) to (l) shows that the way in which the data is obtained affects results as the same noise (the LCG use for each noisy image set uses the same parameters) is used for the study.

4.4 Conclusion

In this chapter we have studied the effectiveness at which $LULU$ removes noise. In Section 4.2 we described the process of adding noise to images as well as introduced measures to analyse results. For Section 4.3, we applied the $LULU$ operators on a set images obtained by taking a single frame out of the video dataset used in Chapter 3. We found that results differ with respect to the source of the image in terms of total variation plots, SSIM and PP plots.

For total variation, it was observed that the level of DPT required to reach the total variation of the original image from the noisy image is less for images with strong and medium signals than images with weak signals.

For SSIM plots, depending on the source of the image, the SSIM either stabilises or deteriorates, both at a fast rate. The variability of SSIM indices decreases as the strength of signal increases.

For PP plots, the noise obtained from images with strong and medium signal strengths matches better than those from weak signals. Furthermore, the $LULU$ operators are able to retrieve noise from symmetric distributions better than those from asymmetric distributions. Most importantly, the $LULU$ operators are able to extract the noise from multiple distribution types with similar effectiveness, a strong property for a smoother. For example, linear smoothers are not able to remove noise with long tailed distributions [36].

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	140	4894.06	0.09	Normal	1	1	81	4258.51	0.13	Logistic	1	1	72	4054.92	0.14
		2	1	828.51	0.06			2	1	577.39	0.08			2	1	510.31	0.09
		3	48	4389.84	0.10			3	31	3665.17	0.13			3	59	3965.20	0.14
	5	1	11	1651.64	0.34		5	1	8	1321.77	0.39		5	1	7	1144.23	0.41
		2	1	252.02	0.29			2	1	227.45	0.32			2	1	209.37	0.34
		3	18	1917.17	0.34			3	15	1611.09	0.39			3	14	1452.18	0.42
	9	1	5	858.97	0.46		9	1	3	494.58	0.50		9	1	3	441.00	0.52
		2	1	205.07	0.43			2	1	160.39	0.46			2	1	149.21	0.48
		3	12	1201.94	0.47			3	10	884.11	0.52			3	9	760.00	0.54
Rayleigh	1	1	87	4280.71	0.11	Gumbel	1	1	72	6238.97	0.13	Exponential	1	1	59	3659.22	0.15
		2	1	629.28	0.08			2	1	1543.29	0.09			2	1	1260.19	0.11
		3	50	3989.05	0.11			3	60	6134.15	0.13			3	57	3646.47	0.15
	5	1	8	1360.87	0.38		5	1	7	2646.96	0.40		5	1	6	1341.69	0.43
		2	1	327.62	0.32			2	1	1289.93	0.33			2	2	1248.97	0.39
		3	16	1693.30	0.39			3	15	3130.31	0.41			3	17	1553.67	0.44
	9	1	3	561.34	0.50		9	1	3	1596.60	0.52		9	1	2	1239.86	0.54
		2	1	263.23	0.46			2	1	1130.70	0.47			2	20	1143.89	0.56
		3	10	952.50	0.52			3	9	2133.74	0.54			3	8	1168.34	0.57

Table 4.5: Summary of video (a) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	3096	5685.10	0.06	Normal	1	1	2122	5137.02	0.09	Logistic	1	1	1852	4872.42	0.10
		2	1	1121.35	0.02			2	1	745.44	0.03			2	1	654.67	0.04
		3	147	5453.14	0.08			3	166	5079.86	0.11			3	180	4953.02	0.13
	5	1	239	2777.60	0.29		5	1	167	2427.20	0.35		5	1	137	2198.52	0.36
		2	1	294.35	0.21			2	1	282.62	0.24			2	1	269.71	0.25
		3	50	2934.37	0.31			3	43	2513.89	0.36			3	35	2306.28	0.38
	9	1	95	1957.64	0.42		9	1	70	1579.94	0.47		9	1	61	1402.83	0.49
		2	1	264.99	0.34			2	1	226.47	0.37			2	1	212.30	0.39
		3	27	1986.04	0.43			3	26	1634.10	0.49			3	21	1435.46	0.50
Rayleigh	1	1	1751	5206.13	0.09	Gumbel	1	1	1524	7361.00	0.10	Exponential	1	1	1392	4672.65	0.11
		2	1	813.18	0.03			2	1	1673.26	0.04			2	1	1215.62	0.05
		3	118	4990.02	0.11			3	100	6978.76	0.12			3	85	4468.77	0.14
	5	1	162	2509.03	0.34		5	1	137	4338.17	0.37		5	1	102	2088.51	0.40
		2	1	386.74	0.23			2	1	1255.11	0.25			2	2	1347.51	0.31
		3	38	2556.77	0.36			3	43	4314.23	0.38			3	33	2149.56	0.41
	9	1	69	1635.42	0.47		9	1	59	3132.08	0.49		9	1	46	1507.26	0.53
		2	1	336.96	0.37			2	1	1113.30	0.38			2	3	1318.08	0.47
		3	29	1673.19	0.47			3	28	3108.69	0.50			3	29	1526.23	0.53

Table 4.6: Summary of video (b) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	663	4851.88	0.11	Normal	1	1	429	4358.19	0.15	Logistic	1	1	408	4126.68	0.17
		2	1	1027.78	0.06			2	1	735.21	0.08			2	1	647.68	0.08
		3	62	4650.88	0.14			3	81	4319.30	0.17			3	87	4161.13	0.19
	5	1	64	2205.50	0.42		5	1	49	1798.72	0.48		5	1	43	1623.08	0.49
		2	1	279.35	0.35			2	1	256.51	0.38			2	1	239.79	0.40
		3	20	2077.74	0.43			3	18	1727.13	0.48			3	18	1567.55	0.50
	9	1	26	1319.25	0.56		9	1	20	996.82	0.61		9	1	19	863.62	0.63
		2	1	227.80	0.51			2	1	185.20	0.54			2	1	168.05	0.56
		3	12	1241.52	0.57			3	10	948.38	0.61			3	9	834.27	0.63
Rayleigh	1	1	424	4413.12	0.15	Gumbel	1	1	345	6478.80	0.16	Exponential	1	1	287	3876.05	0.20
		2	1	788.29	0.08			2	1	1658.08	0.08			2	1	1233.33	0.11
		3	79	4384.89	0.17			3	63	6231.50	0.18			3	75	3893.13	0.22
	5	1	50	1840.96	0.47		5	1	42	3418.08	0.49		5	1	34	1642.70	0.52
		2	1	366.61	0.38			2	1	1213.82	0.40			2	2	1346.63	0.46
		3	16	1739.59	0.48			3	18	3235.01	0.50			3	14	1577.54	0.53
	9	1	20	1079.55	0.61		9	1	19	2216.85	0.63		9	1	14	1260.57	0.66
		2	1	303.43	0.54			2	1	1046.28	0.55			2	19	1248.48	0.65
		3	12	1036.94	0.61			3	10	2135.31	0.63			3	9	1274.98	0.66

Table 4.7: Summary of video (c) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	1657	5156.60	0.09	Normal	1	1	683	4686.55	0.13	Logistic	1	1	549	4538.10	0.14
		2	1	987.76	0.04			2	1	688.55	0.06			2	1	609.48	0.06
		3	165	5178.19	0.11			3	87	4568.00	0.15			3	91	4370.73	0.16
	5	1	66	2417.29	0.37		5	1	49	1983.72	0.42		5	1	42	1806.12	0.45
		2	1	283.00	0.29			2	1	260.60	0.32			2	1	245.46	0.34
		3	23	2285.91	0.38			3	25	1963.69	0.43			3	20	1747.48	0.45
	9	1	27	1506.74	0.52		9	1	21	1188.36	0.56		9	1	19	1010.01	0.58
		2	1	239.05	0.44			2	1	198.57	0.47			2	1	182.55	0.49
		3	21	1498.02	0.52			3	16	1173.53	0.56			3	15	1001.08	0.58
Rayleigh	1	1	832	4654.60	0.13	Gumbel	1	1	583	6839.68	0.14	Exponential	1	1	464	4223.48	0.16
		2	1	742.95	0.05			2	1	1595.15	0.06			2	1	1254.20	0.08
		3	93	4497.33	0.14			3	98	6574.68	0.15			3	66	4001.12	0.18
	5	1	54	2024.30	0.42		5	1	42	3591.75	0.45		5	1	34	1766.87	0.48
		2	1	368.30	0.32			2	1	1214.76	0.34			2	2	1346.81	0.40
		3	28	2002.81	0.43			3	21	3460.94	0.45			3	26	1758.00	0.48
	9	1	21	1222.22	0.56		9	1	18	2440.32	0.58		9	1	14	1301.87	0.61
		2	1	314.51	0.47			2	1	1053.21	0.48			2	20	1287.68	0.60
		3	18	1210.26	0.56			3	16	2425.73	0.58			3	12	1304.78	0.61

Table 4.8: Summary of video (d) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	12	2804.1	0.14	Normal	1	1	8	2074.99	0.18	Logistic	1	1	7	1840.8	0.21
		2	1	1007.66	0.12			2	1	663.29	0.15			2	1	582.21	0.17
		3	15	3009.88	0.14			3	9	2176.69	0.18			3	6	1704.87	0.21
	5	1	1	120.88	0.65		5	1	1	61.39	0.67		5	1	1	46.74	0.69
		2	1	120.88	0.65			2	20	38.77	0.4			2	20	15.01	0.42
		3	1	120.88	0.65			3	1	61.39	0.67			3	1	46.74	0.69
	9	1	1	20.4	0.79		9	1	1	9.59	0.8		9	1	1	13.68	0.81
		2	1	20.4	0.79			2	1	9.59	0.8			2	1	13.68	0.81
		3	1	20.4	0.79			3	1	9.59	0.8			3	1	13.68	0.81
Rayleigh	1	1	8	2094.88	0.18	Gumbel	1	1	7	3290.08	0.21	Exponential	1	1	5	1663.55	0.24
		2	1	733.72	0.16			2	1	1559.19	0.18			2	1	1263.37	0.23
		3	9	2195.9	0.18			3	7	3290.08	0.21			3	4	1571.42	0.24
	5	1	1	214.89	0.68		5	1	1	756.7	0.69		5	1	1	1584.33	0.71
		2	20	141.38	0.41			2	20	546.13	0.42			2	5	1524.63	0.6
		3	1	214.89	0.68			3	1	756.7	0.69			3	1	1584.33	0.71
	9	1	1	109.75	0.8		9	1	1	587.57	0.81		9	1	1	1666.48	0.82
		2	5	62.01	0.65			2	4	523.03	0.68			2	1	1666.48	0.82
		3	1	109.75	0.8			3	1	587.57	0.81			3	1	1666.48	0.82

Table 4.9: Summary of video (e) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	7	2164.93	0.21	Normal	1	1	5	1612.39	0.25	Logistic	1	1	4	1370.35	0.27
		2	1	1217.34	0.23			2	1	951.59	0.26			2	1	842.64	0.27
		3	1	1217.34	0.23			3	2	1179.67	0.26			3	2	1061.74	0.28
	5	1	1	117.26	0.57		5	1	1	68.05	0.59		5	1	1	50.26	0.60
		2	1	117.26	0.57			2	1	68.05	0.59			2	1	50.26	0.60
		3	1	117.26	0.57			3	1	68.05	0.59			3	1	50.26	0.60
	9	1	1	34.36	0.66		9	1	1	25.85	0.67		9	1	1	32.01	0.68
		2	1	34.36	0.66			2	1	25.85	0.67			2	1	32.01	0.68
		3	1	34.36	0.66			3	1	25.85	0.67			3	1	32.01	0.68
Rayleigh	1	1	5	1602.90	0.25	Gumbel	1	1	4	2323.73	0.27	Exponential	1	1	4	1587.10	0.30
		2	1	1046.97	0.25			2	1	1336.12	0.27			2	3	1580.19	0.30
		3	2	1227.50	0.26			3	2	1741.70	0.27			3	3	1580.19	0.30
	5	1	1	168.22	0.59		5	1	1	837.34	0.60		5	1	1	1298.42	0.61
		2	1	168.22	0.59			2	1	837.34	0.60			2	20	785.70	0.40
		3	1	168.22	0.59			3	1	837.34	0.60			3	1	1298.42	0.61
	9	1	1	69.83	0.67		9	1	1	711.68	0.68		9	1	1	1371.76	0.69
		2	2	51.92	0.63			2	7	685.27	0.52			2	13	1098.19	0.47
		3	1	69.83	0.67			3	1	711.68	0.68			3	1	1371.76	0.69

Table 4.10: Summary of video (f) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	11	2509.55	0.20	Normal	1	1	8	1876.46	0.25	Logistic	1	1	7	1656.34	0.28
		2	1	982.98	0.18			2	1	691.71	0.22			2	1	612.11	0.25
		3	8	2278.50	0.20			3	5	1561.86	0.25			3	5	1438.05	0.28
	5	1	1	110.35	0.68		5	1	1	65.19	0.69		5	1	1	49.13	0.70
		2	1	110.35	0.68			2	1	65.19	0.69			2	1	49.13	0.70
		3	1	110.35	0.68			3	1	65.19	0.69			3	1	49.13	0.70
	9	1	1	28.96	0.79		9	1	1	9.39	0.80		9	1	1	10.71	0.80
		2	1	28.96	0.79			2	1	9.39	0.80			2	1	10.71	0.80
		3	1	28.96	0.79			3	1	9.39	0.80			3	1	10.71	0.80
Rayleigh	1	1	8	1888.38	0.25	Gumbel	1	1	7	3040.68	0.27	Exponential	1	1	5	1587.22	0.32
		2	1	763.92	0.23			2	1	1554.07	0.25			2	1	1290.00	0.31
		3	6	1704.41	0.25			3	5	2741.01	0.27			3	5	1587.22	0.32
	5	1	1	196.13	0.70		5	1	1	838.74	0.71		5	1	1	1443.93	0.72
		2	20	131.85	0.51			2	20	749.88	0.52			2	20	1072.93	0.54
		3	1	196.13	0.70			3	1	838.74	0.71			3	1	1443.93	0.72
	9	1	1	108.28	0.80		9	1	1	646.63	0.80		9	1	1	1501.01	0.81
		2	7	32.46	0.67			2	6	549.12	0.69			2	8	1378.84	0.67
		3	1	108.28	0.80			3	1	646.63	0.80			3	1	1501.01	0.81

Table 4.11: Summary of video (g) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	11	2598.89	0.17	Normal	1	1	7	1877.15	0.22	Logistic	1	1	6	1623.33	0.24
		2	1	964.12	0.16			2	1	708.53	0.20			2	1	622.67	0.22
		3	6	2112.66	0.18			3	5	1636.55	0.22			3	6	1623.33	0.24
	5	1	1	96.06	0.62		5	1	1	53.10	0.63		5	1	1	39.42	0.64
		2	1	96.06	0.62			2	1	53.10	0.63			2	1	39.42	0.64
		3	1	96.06	0.62			3	1	53.10	0.63			3	1	39.42	0.64
	9	1	1	23.62	0.71		9	1	1	11.54	0.72		9	1	1	14.24	0.72
		2	1	23.62	0.71			2	1	11.54	0.72			2	1	14.24	0.72
		3	1	23.62	0.71			3	1	11.54	0.72			3	1	14.24	0.72
Rayleigh	1	1	7	1878.59	0.22	Gumbel	1	1	6	2925.74	0.24	Exponential	1	1	5	1622.93	0.29
		2	1	787.71	0.20			2	1	1422.79	0.22			2	1	1401.10	0.27
		3	5	1643.12	0.22			3	4	2516.50	0.24			3	4	1560.88	0.29
	5	1	1	182.89	0.64		5	1	1	804.24	0.64		5	1	1	1443.18	0.66
		2	20	177.23	0.42			2	20	804.08	0.43			2	20	1078.47	0.44
		3	1	182.89	0.64			3	1	804.24	0.64			3	1	1443.18	0.66
	9	1	1	80.56	0.72		9	1	1	645.49	0.72		9	1	1	1454.92	0.73
		2	5	32.79	0.60			2	3	616.37	0.65			2	14	1200.28	0.52
		3	1	80.56	0.72			3	1	645.49	0.72			3	1	1454.92	0.73

Table 4.12: Summary of video (h) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	11	2489.08	0.21	Normal	1	1	8	1898.07	0.27	Logistic	1	1	6	1573.87	0.29
		2	1	990.14	0.19			2	1	730.35	0.24			2	1	639.58	0.26
		3	9	2357.80	0.21			3	7	1810.90	0.27			3	7	1662.36	0.29
	5	1	1	120.84	0.69		5	1	1	71.00	0.71		5	1	1	59.27	0.72
		2	1	120.84	0.69			2	1	71.00	0.71			2	1	59.27	0.72
		3	1	120.84	0.69			3	1	71.00	0.71			3	1	59.27	0.72
	9	1	1	35.01	0.81		9	1	1	14.20	0.82		9	1	1	13.37	0.82
		2	1	35.01	0.81			2	1	14.20	0.82			2	1	13.37	0.82
		3	1	35.01	0.81			3	1	14.20	0.82			3	1	13.37	0.82
Rayleigh	1	1	8	1901.74	0.27	Gumbel	1	1	7	2989.41	0.29	Exponential	1	1	6	1641.69	0.33
		2	1	805.56	0.24			2	1	1452.30	0.26			2	1	1359.16	0.30
		3	7	1814.55	0.27			3	6	2843.90	0.29			3	7	1684.53	0.33
	5	1	1	211.69	0.71		5	1	1	823.44	0.72		5	1	1	1469.06	0.74
		2	20	162.04	0.55			2	20	749.98	0.55			2	20	1153.67	0.57
		3	1	211.69	0.71			3	1	823.44	0.72			3	1	1469.06	0.74
	9	1	1	105.92	0.82		9	1	1	637.34	0.82		9	1	1	1534.02	0.83
		2	9	27.92	0.68			2	8	545.71	0.69			2	6	1463.05	0.73
		3	1	105.92	0.82			3	1	637.34	0.82			3	1	1534.02	0.83

Table 4.13: Summary of video (i) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	5	1698.49	0.36	Normal	1	1	4	1309.15	0.40	Logistic	1	1	3	1094.41	0.43
		2	1	1255.07	0.39			2	1	979.54	0.42			2	1	870.00	0.44
		3	1	1255.07	0.39			3	1	979.54	0.42			3	1	870.00	0.44
	5	1	1	124.89	0.67		5	1	1	73.92	0.68		5	1	1	59.81	0.69
		2	1	124.89	0.67			2	1	73.92	0.68			2	1	59.81	0.69
		3	1	124.89	0.67			3	1	73.92	0.68			3	1	59.81	0.69
	9	1	1	44.85	0.72		9	1	1	39.25	0.73		9	1	1	44.98	0.73
		2	1	44.85	0.72			2	1	39.25	0.73			2	1	44.98	0.73
		3	1	44.85	0.72			3	1	39.25	0.73			3	1	44.98	0.73
Rayleigh	1	1	4	1310.48	0.40	Gumbel	1	1	3	1829.15	0.43	Exponential	1	1	4	1490.70	0.45
		2	1	1085.82	0.41			2	1	1263.30	0.43			2	18	1375.77	0.38
		3	2	1169.77	0.41			3	2	1585.16	0.43			3	3	1526.51	0.45
	5	1	1	155.22	0.68		5	1	1	925.17	0.68		5	1	1	1225.75	0.69
		2	1	155.22	0.68			2	1	925.17	0.68			2	20	668.69	0.47
		3	1	155.22	0.68			3	1	925.17	0.68			3	1	1225.75	0.69
	9	1	1	93.07	0.73		9	1	1	771.46	0.73		9	1	1	1338.40	0.74
		2	2	83.23	0.68			2	1	771.46	0.73			2	20	932.98	0.49
		3	1	93.07	0.73			3	1	771.46	0.73			3	1	1338.40	0.74

Table 4.14: Summary of video (j) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	13	2751.62	0.18	Normal	1	1	8	1984.86	0.23	Logistic	1	1	7	1744.45	0.25
		2	1	967.43	0.15			2	1	685.59	0.20			2	1	602.56	0.22
		3	9	2456.52	0.18			3	7	1882.45	0.23			3	7	1744.45	0.25
	5	1	1	122.58	0.65		5	1	1	74.99	0.67		5	1	1	56.30	0.68
		2	1	122.58	0.65			2	1	74.99	0.67			2	1	56.30	0.68
		3	1	122.58	0.65			3	1	74.99	0.67			3	1	56.30	0.68
	9	1	1	31.54	0.77		9	1	1	12.20	0.78		9	1	1	10.51	0.79
		2	1	31.54	0.77			2	1	12.20	0.78			2	1	10.51	0.79
		3	1	31.54	0.77			3	1	12.20	0.78			3	1	10.51	0.79
Rayleigh	1	1	9	2076.77	0.22	Gumbel	1	1	7	3150.37	0.25	Exponential	1	1	6	1673.67	0.29
		2	1	753.14	0.20			2	1	1521.42	0.22			2	1	1290.59	0.27
		3	10	2155.55	0.23			3	7	3150.37	0.25			3	6	1673.67	0.29
	5	1	1	206.19	0.67		5	1	1	845.69	0.68		5	1	1	1423.45	0.70
		2	20	194.76	0.51			2	20	775.54	0.52			2	19	1192.40	0.54
		3	1	206.19	0.67			3	1	845.69	0.68			3	1	1423.45	0.70
	9	1	1	107.37	0.78		9	1	1	655.59	0.79		9	1	1	1479.15	0.80
		2	10	29.85	0.64			2	10	574.96	0.65			2	7	1352.61	0.69
		3	1	107.37	0.78			3	1	655.59	0.79			3	1	1479.15	0.80

Table 4.15: Summary of video (k) results.

Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM	Noise	SNR	Criteria	n	SSE	SSIM
Uniform	1	1	27	3722.40	0.11	Normal	1	1	17	2942.26	0.14	Logistic	1	1	15	2682.53	0.16
		2	1	878.69	0.08			2	1	638.10	0.10			2	1	568.81	0.12
		3	27	3722.40	0.11			3	20	3073.70	0.14			3	18	2830.57	0.16
	5	1	2	373.90	0.47		5	1	1	150.34	0.51		5	1	1	123.93	0.52
		2	1	200.53	0.48			2	1	150.34	0.51			2	1	123.93	0.52
		3	1	200.53	0.48			3	1	150.34	0.51			3	2	231.94	0.52
	9	1	1	90.01	0.64		9	1	1	46.18	0.66		9	1	1	35.33	0.67
		2	1	90.01	0.64			2	1	46.18	0.66			2	1	35.33	0.67
		3	1	90.01	0.64			3	1	46.18	0.66			3	1	35.33	0.67
Rayleigh	1	1	18	2981.86	0.14	Gumbel	1	1	16	4567.82	0.16	Exponential	1	1	12	2263.90	0.19
		2	1	697.91	0.10			2	1	1606.48	0.11			2	1	1276.20	0.15
		3	17	2929.40	0.14			3	20	4808.56	0.16			3	14	2376.56	0.19
	5	1	1	254.68	0.51		5	1	1	1076.89	0.53		5	1	1	1286.27	0.55
		2	1	254.68	0.51			2	1	1076.89	0.53			2	20	1027.09	0.46
		3	1	254.68	0.51			3	2	1217.52	0.53			3	2	1239.27	0.55
	9	1	1	145.04	0.66		9	1	1	843.79	0.67		9	1	1	1327.75	0.69
		2	1	145.04	0.66			2	20	787.67	0.52			2	20	1062.26	0.53
		3	1	145.04	0.66			3	1	843.79	0.67			3	1	1327.75	0.69

Table 4.16: Summary of video (l) results.

Chapter 5

Conclusion

We have provided the foundation of $LULU$ operators in Chapter 2. The $LULU$ operators in one- and two- dimensions were discussed along with their shape and total variation properties, and their respective Discrete Pulse Transforms. We have also included a small study on the intricasy of $\mathcal{N}_n(x)$ in $\mathbb{Z}^2$.

Using results from correlation analysis, we have explored the extent at which the pixels of an image depend on its neighbours and establish the complexity of the connectivity for $LULU$ operators in two-dimensions in Chapter 3. In there, using videos obtained from an internet source and our own recorded ones, we have shown that the property of local dependence and global independence holds for images, as well as 4-connectivity is sufficient for the defined connection in two-dimensional $LULU$ operators. Furthermore, we have also demonstrated how the addition of noise renders the dependence between pixels ambiguous as a result of inflated variance.

In Chapter 4 we investigated how effective the $LULU$ smoothers remove noise by examining the noise extractions by the $LULU$ operators from images and the purified images themselves. We saw that for images with low noise content (well-behaved variation) are able to withstand the DPT in terms of variation preserving than those with medium or high noise content, that is, the rate of deterioration of total variation is faster for images with medium or high noise content. In addition, the optimal n-level DPT required to extract the noise is 1 since the added noise are idd observations from the same distribution with pulse size 1.

Future work includes exploring alternative compositions of the $LULU$ operators by [14] in noise removal for images. Also we will improve the noise removal algorithm, that is, a second step to improve the blocky effect of the purified image.

Bibliography

[1] R. Anguelov. LULU operators and locally δ-monotone approximations. In *Constructive Theory of Functions*. Citeseer, 2005.

[2] R. Anguelov and I. Fabris-Rotelli. LULU operators and Discrete Pulse Transform for multidimensional arrays. *Image Processing, IEEE Transactions on*, 19(11):3012–3023, 2010.

[3] R. Anguelov and C.H. Rohwer. LULU operators for functions of continuous argument. *Quaestiones Mathematicae*, 32(2):187–202, 2009.

[4] WJ Conradie, T De Wet, and M Jankowitz. Exact and asymptotic distributions of LULU smoothers. *Journal of Computational and Applied Mathematics*, 186(1):253–267, 2006.

[5] W.J. Conradie, T. De Wet, and M. D. Jankowitz. Performance of nonlinear smoothers in signal recovery. *Applied Stochastic Models in Business and Industry*, 25(4):425–444, 2009.

[6] W.J. Conradie, T. De Wet, and M.D. Jankowitz. An overview of LULU smoothers with application to financial data. *Journal for Studies in Economics and Econometrics*, 29(1):97–121, 2005.

[7] T. de Wet and W. Conradie. Smoothing sequences of data by extreme selectors. *Proceedings of ICOTS7*, 2006.

[8] J.P. Du Toit. *The Discrete Pulse Transform and applications*. PhD thesis, Stellenbosch: University of Stellenbosch, 2007.

[9] I. Fabris-Rotelli, K. Van Oldenmark, and P. Van Staden. Evaluation of noise removal in signals by LULU operators. *Proceedings of South African Statistical Association*, 2010.

[10] C. Fontaine. Linear Congruential Generator. *Encyclopedia of Cryptography and Security*, pages 721–721, 2011.

[11] E.R. Golder and J.G. Settle. The Box-Muller method for generating pseudo-random normal deviates. *Applied Statistics*, pages 12–20, 1976.

[12] L.D. Haugh. Checking the independence of two covariance-stationary time series: a univariate residual cross-correlation approach. *Journal of the American Statistical Association*, 71(354):378–385, 1976.

[13] A. Hore and D. Ziou. Image quality metrics: PSNR vs. SSIM. In *Pattern recognition (icpr), 2010 20th international conference on*, pages 2366–2369. IEEE, 2010.

[14] M.D. Jankowitz. *Some statistical aspects of LULU smoothers*. PhD thesis, Stellenbosch: University of Stellenbosch, 2007.

[15] E. Kreyszig. *Introductory functional analysis with applications*, volume 81. Wiley New York, 1989.

[16] D.P. Laurie. The roadmaker's algorithm for the Discrete Pulse Transform. *Image Processing, IEEE Transactions on*, 20(2):361–371, 2011.

[17] D.P. Laurie and C.H. Rohwer. Fast implementation of the Discrete Pulse Transform. In *Proceedings International Conference Numerical and Analytical Applied Mathematics*, pages 15–19. Weinheim, Germany, 2006.

[18] E Malkowsky and CH Rohwer. The LULU-semigroup for envelopes of functions. *Quaestiones Mathematicae*, 27(1):89–97, 2004.

[19] T.B. Parrish, D.R. Gitelman, K.S. LaBar, and M.M. Mesulam. Impact of signal-to-noise on functional MRI. *Magnetic Resonance in Medicine*, 44:925–932, 2000.

[20] I. Pitas and A.N. Venetsanopoulos. Median Filters. In *Nonlinear Digital Filters*, pages 63–116. Springer, 1990.

[21] D.G. Rogers. Pascal triangles, Catalan numbers and renewal arrays. *Discrete Mathematics*, 22(3):301–310, 1978.

[22] C.H. Rohwer. Idempotent one-sided approximation of median smoothers. *Journal of Approximation Theory*, 58(2):151–163, 1989.

[23] C.H. Rohwer. Projections and separators. *Quaestiones Mathematicae*, 22(2):219–230, 1999.

[24] C.H. Rohwer. Fast approximation with locally monotone sequences. In *Proceedings 4th FAAT Conference, Maratea. In: Supplemento ai rendiconti del Circolo matimatico di Palermo, Serie II*, volume 68, 2002.

[25] C.H. Rohwer. Multiresolution analysis with pulses. In *Advanced Problems in Constructive Approximation*, pages 165–186. Springer, 2002.

[26] C.H. Rohwer. Variation reduction and LULU-smoothing. *Quaestiones Mathematicae*, 25(2):163–176, 2002.

[27] C.H. Rohwer. Fully trend preserving operators. *Quaestiones Mathematicae*, 27(3):217–229, 2004.

[28] C.H. Rohwer. *Nonlinear Smoothing and Multiresolution Analysis*, volume 150. Springer Science & Business Media, 2006.

[29] C.H. Rohwer. The estimation of moments of an unknown error distribution in the Discrete Pulse Transform. *Numerical Algorithms*, 45(1-4):239–251, 2007.

[30] C.H. Rohwer and D.P. Laurie. The Discrete Pulse Transform. *SIAM Journal on Mathematical Analysis*, 38(3):1012–1034, 2006.

[31] C.H. Rohwer and L.M. Toerien. Locally monotone robust approximation of sequences. *Journal of Computational and Applied Mathematics*, 36(3):399–408, 1991.

[32] C.H. Rohwer and M. Wild. Natural alternatives for one dimensional median filtering. *Quaestiones Mathematicae*, 25(2):135–162, 2002.

[33] C.H. Rohwer and M. Wild. LULU theory, idempotent stack filters, and the mathematics of vision of marr. *Advances in Imaging and Electron Physics*, 146:57–162, 2007.

[34] R.Y. Rubinstein and B. Melamed. *Modern Simulation and Modeling*, volume 7. Wiley New York, 1998.

[35] J. Serra. A lattice approach to image segmentation. *Journal of Mathematical Imaging and Vision*, 24(1):83–130, 2006.

[36] P.F. Velleman. Robust nonlinear data smoothers: Definitions and recommendations. *Proceedings of the National Academy of Sciences*, 74(2):434–436, 1977.

[37] M. Watanabe, D.B. Williams, and Y. Tomokiyo. Comparison of detection limits for elemental mapping by EF-TEM and STEMXEDS. *Microscopy and Microanalysis*, 8:1588–1589, 2002.

[38] M. Wild. Idempotent and co-idempotent stack filters and min–max operators. *Theoretical Computer Science*, 299(1):603–631, 2003.

[39] M.B. Wilk and R. Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55(1):1–17, 1968.

Appendix

Intricasy of $\mathcal{N}_n(x)$ [SAS]

Introduction

In the following pages, we describe the process of obtaining the actual numbers of $\mathcal{N}_n(x)$ in SAS `iml` for any $n > 1$ and $x \in \mathbb{Z}^2$. This is the first step to acquiring a definitive formula for the total number of elements in $\mathcal{N}_n(x)$. The definition of usability in $\mathcal{N}_n(x)$ are

1. Any one connected set in $\mathcal{N}_n(x)$ must have length exactly equal to $n+1$.
2. Any two connections must differ from each other.

Some terminology before starting:

Branch A branch is any single connected set beginning from the centre.

Burst A burst is defined as mapping of 3 points in the direction of a branch as well as the immediate adjacent elements in $\mathbb{Z}^2$. For example, suppose that a branch is moving in the upward direction, then a burst is mapping of the right, top and left elements from the reference element. This is depicted in Figure A.1.

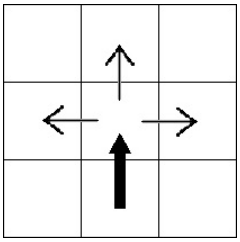


Figure A.1: A depiction of a burst for a branch approaching from the bottom.

Subroutines

This section describes some pieces of code that are used extensively throughout the program. We stored them as subroutines to make the coding more elegant. There are 5 in total which can be grouped into 2

parts:

- `Three_Rotations, Maps` and `One_Branch`

  `One_Branch` finds branches of length $N+1$, that is, it finds connections with the centre at the starting element of any branch. `Three_Rotations` accepts a square matrix and outputs three matrices containing the original matrix but rotated 90°, 180°, and 270°. `Maps` direction vector and outputs the burst co-ordinates with respect to the direction vector.

- `n_Comb` and `N_Connect`

  `N_Connect` is the final subroutine which finds all connected sets of length $N + 1$ using integral subroutine `One_Branch` since the centre x may occur anywhere in the connection. This is done by finding all possible 2, 3 or 4 combinations of connections of length 1 to N with auxiliary subroutine `n_Comb`.

We now describe the subroutines in detail.

```
start Three_Rotations(In, Out1,Out2,Out3);
    n = nrow(In);
    Out1 = J(n,n,0);
    Out2 = Out1;
    Out3 = Out2;
    do i = 1 to n;
        do j = 1 to n;
            k = n-i+1;
            l = n-j+1;
            Out1[j,k] = In[i,j];
            Out2[k,l] = In[i,j];
            Out3[l,i] = In[i,j];
        end;
    end;
finish Three_Rotations;
```

Input variable

`In`: $n \times n$ Any square matrix.

Output variables

`Out1`: $n \times n$ The rotated form of input matrix `In` by 90° clockwise.

`Out2`: $n \times n$ The rotated form of input matrix `In` by 180° clockwise.

`Out3`: $n \times n$ The rotated form of input matrix `In` by 270° clockwise.

Method For any square matrix `In`, `Three_Rotations` rotates `In` by $90°k$ and stores the new matrices in `Out1`, `Out2`, and `Out3`. This subroutine decreases the total number of computations for finding all branches by a factor of 4 since we need only to find one branch and `Three_Rotations` will give the rest.

For example, the first matrix in Figure A.2 is a single branch of $\mathcal{N}_2(x)$ connectivity.

In

0	0	0	0	0
0	0	1	1	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

Figure A.2: A single branch of $\mathcal{N}_2(x)$ connectivity.

Calling `Three_Rotations` with input matrix equal to `In` we get our 3 outputs in Figure A.3.

Out1

0	0	0	0	0
0	0	0	0	0
0	0	1	1	0
0	0	0	1	0
0	0	0	0	0

Out2

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	1	1	0	0
0	0	0	0	0

Out3

0	0	0	0	0
0	1	0	0	0
0	1	1	0	0
0	0	0	0	0
0	0	0	0	0

Figure A.3: The 3 outputs of `Three_Rotations` for a given square matrix `In`.

```
start Maps(Dir);
    Mu = { 0 -1  0,
          -1  0  1};
    Mr = {-1  0  1,
           0  1  0};
    Md = { 0  1  0,
           1  0 -1};
    Ml = { 1  0 -1,
           0 -1  0};

```

```
    if (Dir = (-1 || 0))[+] = 2 then M = Mu;
    if (Dir = ( 0 || 1))[+] = 2 then M = Mr;
    if (Dir = ( 1 || 0))[+] = 2 then M = Md;
    if (Dir = ( 0 ||-1))[+] = 2 then M = Ml;

    return (M);
finish;
```

Input variable

`Dir`: 1×2 A vector indicating the direction a branch is moving from. Here, the first element represents i and the second element j for matrix element displacement $[i, j]$ from one matrix element to another. For example, if a branch grows from matrix elements $(3, 3)$ to $(2, 3)$ the direction vector representing this event is given by $[-1 \;\; 0]^T$.

Return variable

`M`: 2×3 The set of the directions the current branch will be moving to.

Method The subroutine `Maps` creates a burst by defining auxiliary matrices that depend on the direction (`Dir`) a branch is moving to. Its usage shall become clear once we have explained `One_N`. For now, the letters U, R, D and L represent directions up, right, down, and left respectively. And for any direction, three 'map points' are created in the front and on the immediate adjacent elements with respect to the direction the branch is facing.

For example, if the branch is going U, then maps are dictated at elements positioned at right, up, and left from where the branch is sitting given by direction vectors $[0 \;\; -1]^T, [-1 \;\; 0]^T$ and $[0 \;\; 1]^T$ respectively.

```
start One_Branch(N,size);
    *Creating an empty matrix to store the connected sets;
    fix = J(2*size + 1,2*size + 1,0);
    fix[(size || size+1),size + 1] = 1;
    l = (2*size + 1)##2;

    /*WS = Walking Stick - an auxiliary matrix that guides the program on how and where
    to make connections*/
    *1st column contains the number of iterations left before completing a single burst;
    *2nd and 3rd column stores the position of the last burst;
    *4th and 5th column contains the direction of moving from i-1 to i in WS;
    initial = 0 || (size || (size+1)) || (-1 || 0);
    WS = initial // (J(N-1,1,3) || J(N-1,4,0));

    C_temp = fix;
    p = 2; *p (position) indexes WS;
    do while ((WS[,1])[+] ^= 0);
```

```
18
19          /*Adding 1 of N Bursts*/
20          M = Maps(WS[p-1,4:5]);
21          ij = WS[p-1,2:3] + M[,WS[p,1]]‘;
22          C_temp[ij[1], ij[2]] = C_temp[ij[1], ij[2]] + 1;
23          WS[p,2:3] = ij;
24          WS[p,4:5] = M[,WS[p,1]]‘;
25          WS[p,1] = WS[p,1] - 1;
26
27          /*Incrementing p or Updating Parameters*/
28          if p+1 ^= N+1 then p = p+1;
29          else do;
30              call Three_Rotations(C_temp,C1,C2,C3);
31              C = C || shape(C_temp,1,1) || shape(C1,1,1)
32                    || shape(C2,1,1) || shape(C3,1,1);
33
34              if (WS[,1]^=0)[+] ^= 0 then p = max(loc(WS[,1]^=0));
35
36              if N-p ^= 0 then WS[p+1:N,1] = J(N-p,1,3);
37
38              if p ^= 2 then do;
39                  do i = p to N;
40                      C_temp[WS[i,2],WS[i,3]] = C_temp[WS[i,2],WS[i,3]] - 1;
41                  end;
42              end;
43              else C_temp = fix;
44          end;
45      end;
46
47      return (C);
48  finish One_Branch;
```

Input variable

`N`: 1×1 The order (or length) of the connected set.

`size`: 1×1 A value specifying the matrix size in which to store the connections. Note that the final connection will be contained within a $(2 * \texttt{size} + 1) \times (2 * \texttt{size} + 1)$ matrix.

Return variable

`C`: $(2 * \texttt{size} + 1)^2 \times 4 * (3^{\texttt{N}-1})$ Each column of `C` contains a variation of connected set of length $N + 1$ reshaped from a $(2 * \texttt{size} + 1) \times (2 * \texttt{size} + 1)$ matrix to a $(2 * \texttt{size} + 1)^2 \times 1$ vector. Since the total number ways (including replications) we can have a connected set of length $N + 1$ with only 1 branch is $4 * (3^{\texttt{N}-1})$, that number is also our total number of columns.

Method `One_Branch` finds all possible combinations of having a connected set of length $N + 1$ with only 1 branch spawning from the centre. The subroutine uses an auxiliary matrix `WS` (for `Walking Stick`)

that helps guide the program to know how and where to make a single burst by storing the number of bursts left, the historical and direction of previous bursts. We shall explain the subroutine by example for `(N,size)` = (3,3).

The subroutine begins by defining a template matrix called `fix`. The template is created so that when the program has finished all bursts for a single branch we can refresh process by simply assigning the temporary matrix to fix again. It is assumed that the process already has one single burst.

fix

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Next we create the auxiliary matrix `WS`. Some facts regarding `WS`:

- The 1st column indicates the number of bursts left before the process ends, 2nd and 3rd stores the position [matrix index (i, j)] of the last burst, and the last two columns contains the direction of the latest burst. The initial position for this process is the index $(3, 4)$ and the direction is upwards $[-1 \;\; 0]$ since we assume that we moved from the centre (4,4) to (3,4) - this is indicated in row 1.
- Each row of `WS` represents the *level* of the process in creating a branch. For connected sets of length $N + 1$, we will require `N` levels (with centre as level-0). Note that since the first element of `WS` is 0, it implies all possible combinations of level-1 bursts have been exhausted (because we will call `Three_Rotations` for the other branches starting at East, South and West from the centre); and since we have values of 3 for the first element of row 2 and row 3, we still have $3^2 = 9$ more combinations till completion. When the sum of the first column equals 0, it implies all possible branches have been created.
- `WS` operates by iteratively running through consecutive rows until the last row. For each row being operated, the following updates occur: the position and direction of new bursts, as well as decrementing the first element of each row to indicate a completed singular burst.

WS

0	3	4	-1	0
3	0	0	0	0
3	0	0	0	0

We let `C_temp` be the temporary matrix that stores the connected sets. Initially `C_temp` is set to `fix`. Each paired output below is the result of one completed iteration of the `do while` loop. From the first pair we can note the following:

- The length of the branch in `C_temp` has been increased by one unit (element $(3,5)$ has been changed from 0 to 1). That is, the branch has grown from matrix elements $(3,4)$ to $(3,5)$.
- The change in `WS` in the second row (for level-2 changes) are as follows: the *position* of the current single burst from $(0,0)$ to $(3,5)$, and the *direction* in which it was created $(3,4) \rightarrow (3,5) = [0 \;\; 1]$. In addition, we have decremented the level-2 connections from 3 to 2 in the first element of the 2^{nd} row. This indicates 1 of 3 bursts have been completed for level-2 - given the history of the previous levels.

Since we want a `N` $= 3$ connected set, we require the next iteration for a complete branch.

```
C_temp
0     0     0     0     0     0     0
0     0     0     0     0     0     0
0     0     0     1     1     0     0                 WS
0     0     0     1     0     0     0
0     0     0     0     0     0     0     0     3     4    -1     0
0     0     0     0     0     0     0     2     3     5     0     1
0     0     0     0     0     0     0     3     0     0     0     0
```

The output below for `C_temp` indicates the branch has grown from $(3,5)$ to $(4,5)$. This information is recorded similarly as for the first iteration in `WS`. However, because we have one full branch of a `N` $= 3$ connected set, we also call subroutine `Three_Rotations` to replicate this branch for branches spawning from the East, South, and West of the centre. All connected sets are then reshaped into vectors and the results are stored in a matrix `C`.

```
C_temp
0     0     0     0     0     0     0
0     0     0     0     0     0     0
0     0     0     1     1     0     0                 WS
0     0     0     1     1     0     0
0     0     0     0     0     0     0     0     3     4    -1     0
0     0     0     0     0     0     0     2     3     5     0     1
0     0     0     0     0     0     0     2     4     5     1     0
```

We now display significant output for reader to grasp the methodology of creating connected sets.

```
C_temp
0     0     0     0     0     0     0
0     0     0     0     0     0     0
0     0     0     1     1     1     0                 WS
0     0     0     1     0     0     0
0     0     0     0     0     0     0     0     3     4    -1     0
0     0     0     0     0     0     0     2     3     5     0     1
0     0     0     0     0     0     0     1     3     6     0     1
```

```
C_temp
0     0     0     0     0     0     0
0     0     0     0     1     0     0
0     0     0     1     1     0     0                 WS
0     0     0     1     0     0     0
0     0     0     0     0     0     0     0     3     4    -1     0
0     0     0     0     0     0     0     2     3     5     0     1
0     0     0     0     0     0     0     0     2     5    -1     0
```

It is important to note the changes of the first column of `WS` from the output above and below. The

output above gives $[0 \quad 2 \quad 0]^T$ which indicates level-3 bursts have been exhausted (given the history of previous levels).

To renew the process, we assign 3 to level-3 again and subtract 1 from level-2. That is, for a *second* single burst of level-2, we have 3 *new* bursts for level-3. This is given by the first column of `WS` below as $[0 \quad 1 \quad 3]^T$ as well as a change in the level-2 branch element in `C_temp`.

```
C_temp
0  0  0  0  0  0  0
0  0  0  1  0  0  0
0  0  0  1  0  0  0          WS
0  0  0  1  0  0  0
0  0  0  0  0  0  0  0  3   4  -1  0
0  0  0  0  0  0  0  1  2   4  -1  0
0  0  0  0  0  0  0  3  2   5  -1  0
```

```
C_temp
0  0  0  0  0  0  0
0  0  0  1  1  0  0
0  0  0  1  0  0  0          WS
0  0  0  1  0  0  0
0  0  0  0  0  0  0  0  3   4  -1  0
0  0  0  0  0  0  0  1  2   4  -1  0
0  0  0  0  0  0  0  2  2   5   0  1
```

... This process will continue until all bursts have been exhausted (indicated by a column of zeros in the first column of `WS`).

The program may be run for any `N` > 1. The relevance of the `size` parameter (which sets the size of the matrix where we store connected sets) becomes apparent in the next subroutine.

Note that, by the definition of the code, repeats and overlaps may occur. For example, for $n = 5$, a particular branch is created by turning right 4 times (since we exhaust bursts by consistently mapping right, forward, and left of any direction) this implies that the final connection is only of order 4 and not 5 as needed. These problems will be alleviated in subroutine `N_Connect` where we finalised all possible connections.

The need for the following subroutines arises when we concern ourselves with connected sets not only with the centre occuring at the start of any connection, but also anywhere *within* the connecte set. Therefore, we need to find all possible combinations (2, 3 or 4 since these are the possible ways of creating branches from the centre with 4-connecttivity) of 1, 2, ..., $n-1$ lengthed connections that creates a n-connected set in addition to the one combination of n-connected sets.

```
start n_Comb (N,c);
    free key_c keep list;
    do i = 1 to N/2;
        rem = mod(N,i);
        mult = min((N-rem)/i,c-1);
        key_c = key_c || J(1,mult,i);
```

```
    end;

    if round(N/c) - N/c = 0 then do;
        key_c = key_c || N/c;
        key_c = key_c || (round(N/2):N);
    end;
    else key_c = key_c || (round(N/2):N);

    comb_key = allcomb(ncol(key_c),c);
    do i = 1 to nrow(comb_key);
        check_N = key_c[,comb_key[i,]];
        dup = (check_N # ({1 10 100 1000})[,1:c])[+];

        if check_N[+] = N & ncol(xsect(dup,list)) = 0 then do;
            keep = keep // check_N;
            list = list // dup;
        end;
    end;

    return (keep);
finish n_Comb;
```

Input variables

`N`: 1×1 The order (or length) of any one connected set.

`c`: 1×1 The number of combinations to consider.

Return variable

`keep`: $c_N \times c$ The matrix with each row containing a particular `c` permutation of $1, 2, ..., n-1$ items such that its sum is equal to `N`.

Method `n_Comb` begins by determining a vector of `c` or less multiples of numbers $1, 2, ..., N$ such that the sum of all multiples of any one number is less than `N` and stores them in `key_c`. For example, for input parameters `(N,c)` $= (5, 3)$ (possible 3 combinations of 1, 2, 3, 4, 5 lengthed connected sets that satisfies a 5-lengthed connected set), we have that the vector `key_c` is equal to [1 1 2 2 3 4 5]. The reasoning is as follows:

- Two values of 1's occur since three 1's do not make a 5, however, the sum of two 1's and one 3 makes a 5. Similarly for two values of 2's.
- Any number greater than half of 5 only appears once since twice that number is greater 5. That is, inclusion of any multiples of them is redundant.

Thus the order of the multiple of any one number included in the vector `key_c` is such that no multiple is redundant and that possible `c` combinations of the numbers in `key_c` adds up to `N`.

Next we determine all possible `c` combinations of the elements in `key_c` and select those combinations where their sum is equal to `N`. We also eliminate repeats of these combinations in this process with a vector `list` which keeps track of the number of unique combinations. The final variable returned for this subroutine are permutations of numbers $1, 2, ..., n-1$ such that its sum is equal to `N`. The output for input parameters `(5,c)`, `c` $= 2, 3, 4$, is given in the Figures below.

keep

2	3
1	4

Figure A.4: All possible 2 permutations such that their sum is equal to 5.

keep

1	2	2
1	1	3

Figure A.5: All possible 3 permutations such that their sum is equal to 5.

keep

1	1	1	2

Figure A.6: All possible 4 permutations such that their sum is equal to 5.

Note that the maximum number of items to permute is 4 since we can only branch away from the centre in 4 or less branches. Since $N+1 > N$, the only way to branch once from the centre is N-lengthed connections since anymore would violate the definition.

From Figure A.4, we have two possible cases for 5-lengthed connected sets using single branches from 1 - 4 -lengthed connections. Namely we use one single branch from 2-lengthed and 3-lengthed connections, and one single branch from 1-lengthed and 4-lengthed connections. The rest follows similarly.

```
start N_Connect (N);
    size = N;
    l = (2*size + 1)##2;

    /*N_1 Connections*/
    N_1 = J(2*size + 1,2*size + 1,0);
    N_1[(size||size+1),size+1] = 1;
    call Three_Rotations(N_1,O1,O2,O3);
    C = shape(N_1,l,1) || shape(O1,l,1) || shape(O2,l,1) || shape(O3,l,1);
```

```
    key = J(4,1,1);

    /*N_n Connections*/
    do i = 2 to N;
        C = C || One_Branch(i,size);
        key = key // J(4*(3**(i-1)),1,i);
    end;

    /*Determining the 1 - 4 combinations of 1 - N that make n-lengthed connected set*/
    *1 Combination;
    idx_N = loc(key = N);
    All_C = C[,idx_N];
    check = (All_C > 0)[+,];
    keep = loc(check = N+1);
    All_C = All_C[,keep];
    *2 - 4 Combinations;
    mix = 0;
    do cmb = 2 to min(4,N);
        N_sum = n_Comb (N,cmb);

        *Applying the form of the c permutation to our complete one branch set C;
        do i = 1 to nrow(N_sum);
            u_j = unique(N_sum[i,])‘;

            *Determining the number of the multiple of each number in c;
            free idx_i idx_n;
            u_j = u_j || J(nrow(u_j),1,0);
            do j = 1 to nrow(u_j[,1]);
                u_j[j,2] = ncol(loc(N_sum[i,] = u_j[j,1]));
                idx_n = idx_n || ncol(loc(key = u_j[j,1]));
            end;

            *Finding all combinations of c;
            context1 = ncol(loc(key < u_j[1,1]))[+];
            mix = allcomb(idx_n[1],u_j[1,2]) + context1;
            if nrow(u_j) > 1 then do;
                do j = 2 to nrow(u_j);
                    next_mix = allcomb(idx_n[j],u_j[j,2]);

                    free mixmix;
                    do k = 1 to nrow(next_mix);
                        context2 = ncol(loc(key < u_j[j,1]))[+];
                        mixmix = mixmix // (mix
                                || repeat(next_mix[k,],nrow(mix),1) + context2);
                    end;
                    mix = mixmix;
                end;
            end;
        end;

        *Adding the connected set to the full set All_C;
        dup_vec = uniform(J(nrow(C),1,1);
        Add_C = J(1,nrow(mix),0);
        ii = 1;
```

```
64          do i = 1 to nrow(mix);
65              C_temp = (C[,mix[i,]])[,+];
66              dup = (C_temp # dup_vec)[+];
67              if (C_temp > 0)[+] = N+1 & ncol(xsect(dup,list_dup)) = 0 then do;
68                  Add_C[,ii] = (C_temp > 0);
69                  list_dup = list_dup // dup;
70                  ii = ii + 1;
71              end;
72          end;
73          All_C = All_C || Add_C[,1:ii-1];
74      end;
75
76      return (All_C);
77  finish N_Connect;
```

Input variable

`N`: 1×1 The order (or length) of the connected sets.

Return variable

`All_C`: $(2N+1)^2 \times c_N$ The matrix containing all the possible forms of `N`-lengthed connected sets in its columns.

Method `N_Connect` begins by finding all branches of length 1 to `N` by calling `One_Branch` (and manually for 1-lengthed connected sets, since `One_Branch` does not work for `N` = 1). We store all branches in matrix `C` and keep track of their locations with vector `key` where j^{th} element in `key` represents the order of the connecte set and references the j^{th} column in `C`. For example, if the 10^{th} element in `key` is 4, then it implies that the 10^{th} column in `C` contains a 4-lengthed connected sets.

Next we determine the 1 to 4 permutations of items $\{1, 2, ..., \texttt{N}\}$ such that the sum of the permutation is equal to `N`.

- For 1 permutation, only item `N` is possible.
- For 2 to 4 permutations, we require subroutine `n_Comb` (see above for details).

For each of the output from `n_Comb` we then determine the *multiple* of each item in the permutation. This then indicates the number of distinct items to choose from each connected set. For example, from Figure A.6, we require three items from set 1 and one item from set 2 for possible 5-lengthed connected sets. However, because we cannot choose any two items twice within any set (since this would decrease the order of the connection), we need to choose three *distinct* items from the set of 1-lengthed connected sets and one item the set of 2-lengthed connections.

From here, we find all the possible ways of choosing n_1 items from set 1, n_2 items from set 2, ..., $n_{\mathtt{N}}$ items from set `N`, where n_k is the multiple of the number appearing from set k in `n_Comb` and card$\{n_1, n_2, ..., n_{\mathtt{N}}\}$, for $k = 1, 2, ..., \mathtt{N}$. For all these combinations, we:

1. Sum up the combination of the connected sets.
2. Check that the final connected set is of order `N`.
3. Check for uniqueness.

If the resulting connected set is of order `N` and is unique to the set `All_C` (to prevent duplicates), we append it to `All_C`.

We now demonstrate the subroutine by calling `N_Connect` with `N` $= 5$. Since the beginning of the code is straight forward, we start the demonstration at the `cmb` loop - where we find all possible $2, 3, 4$ permutations of the items in the set $\{1, 2, ..., \mathtt{N} - 1\}$ such that the permutation's sum is equal to `N`.

For all 2 permutations of the numbers $\{1, 2, 3, 4, 5\}$, we find that two of them have their sum equal to 5. This is given in the matrix `N_sum` below.

```
cmb      N_sum

 2         2          3
           1          4
```

For each row `N_sum`, we determine the multiple of all the numbers that occur in the permutation and store it in `u_j`. The first row implies that we need to choose 1 item from the set that contains all 2-lengthed connected sets and 1 item from the set containing all 3-lengthed connected sets. From output `idx_N`, we see that there are totals of 12 and 36 items from sets of 2 and 3 -lengthed connected sets respectively. Therefore, for this particular combination, we require $12 \times 36 = 432$ different combinations to consider.

```
u_j

   2          1
   3          1


idx_N

   12         36
```

The matrix `mixmix` contains at each row the column indices where we reference `C` - the matrix that stores all $1, 2, ..., 5$ -lengthed connected sets.

mixmix

5	17
6	17
7	17
8	17
9	17
10	17
11	17
12	17
13	17
14	17
15	17
16	17
5	18
6	18
7	18

... More Output ...

15	51
16	51
5	52
6	52
7	52
8	52
9	52
10	52
11	52
12	52
13	52
14	52
15	52
16	52

For the first row of `mixmix`, we can have a (possible) 5-lengthed connected set if we sum columns 5 and 17 of `C` since it contains a 2 and 3 -lengthed connected set there. Similarly for the other rows.

The connected set obtained from adding the the 5^{th} and 17^{th} column (first row of `mixmix`) is displayed in the output below. The 1-lengthed connected set is displayed in Figure A.7 and the 4-lengthed[1] connected set in Figure A.8. Since this results in a 4-lengthed connected set, we do not add this to our final connected set `All_C`.

[1] As noted before, since one particular 4-lengthed connected set can be created by turning right 3 times, some connections are invalid from the start. Hence any addition of connections to column 17 of `C` will not result in a 5-lengthed connected set.

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure A.7: Matrix form of column 5 from `C`.

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	2	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure A.8: Matrix form of column 17 from `C`.

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	1	0	0	0	0
0	0	0	0	0	3	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure A.9: Resultant connected set from adding columns 5 and 7 of `C`.

For an example of a successful addition of 2 permuted connected sets, we have that adding columns 3 and 57 of the full set `C` results in a 5-lengthed connected set. Note that column 3 contains a 1-lengthed connected set in Figure A.10 and column 57 in Figure A.11 contains a 4-lengthed connected set.

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure A.10: Matrix form of column 3 from `C`.

```
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  1  1  0  0  0  0
0  0  0  0  0  1  1  0  0  0  0
0  0  0  0  0  0  1  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
```

Figure A.11: Matrix form of column 57 from `C`.

```
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  1  1  0  0  0  0
0  0  0  0  0  2  1  0  0  0  0
0  0  0  0  0  1  1  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
```

Figure A.12: Resultant connected set from adding columns 3 and 57 of `C`.

We add the connected set in Figure A.12 if it is unique to the set `All_C` by setting all nonzero elements in the matrix to one (this is done for testing uniqueness), reshaping the matrix to vector form and appending it to `All_C`.

This process is repeated for the rest of the 2 permutations, then for all 3 and 4 permutations until `All_C` contains all possible 5-lengthed connected sets.

Note that `N_Connect` may be run for all `N` > 1, however, because the programs are computationally intensive, it is not recommended for `N` > 9.

An empirical study of image pixels [MATLAB]

Introduction

In this section we describe the process of obtaining the tables in Table 3.1 from MATLAB. In addition to variables like scalars and matrices, MATLAB also offers structure variables where we can store *fields* at each of its elements. For example, if `dog` is a structure variable, then it have fields like `gender`, `weight`, `height`, `breed` etc. Thus, structure variables are super-variables where we can store many facets of a particular variable of interest.

Functions

Here we describe the purpose of each function and how they relate to each other:

- Auxiliary functions

  1. `Map_Maker`

     `Map_Maker` is used to created the dependence structure defined in section 3.2 of Chapter 3 for different values of K. Its purpose is for convenience since we have generalised the MATLAB code to not only accept the dependence structure in sectrion 3.2, but also others as well.

  2. `Diamond_Maker`

     Similar to `Map_Maker`, but with no filling. Thus, in using maps from `Diamond_Maker` as input `map_in`, we only focus on the *border* pixels of the dependence structure - limiting the focus of our study and decreasing the amount of computations required for a full run of a video.

- Main functions

  1. `TSA_Image`

     For a video stream as input, `TSA_Image` models, for each pixel (i, j) within a frame, as a AR process and estimates its p parameters with CLS using 500 observations - that is, the video must contain at least 500 frames. The number p is such that the residuals are uncorrelated (according to the Durbin-Watson test of uncorrelated residuals) and thus is a white noise process. We also perform a test for normality with the Anderson-Darling test on the residuals.

  2. `TSA_Correl_Image`

     `TSA_Correl_Image` finds the covariance and correlation between the centre pixel and the other pixels present in an input dependence structure. In addition, we also determine its mean (with respect to the AR model estimated by `TSA_Image`). Its purpose is to understand the video better with sample statistics, more importantly, the correlations between the centre pixel and the rest in a dependence structure.

3. `TSA_HypoTest`

   `TSA_HypoTest` uses the AR model estimated by `TSA_Image` to obtain residuals for each pixel in the dependence structure. Then we perform Haugh's test of independent univariate time series [12] with the centre pixel and the rest defined in the dependence structure.

- Supplementary functions

1. `Ave_ARpar`

   For the input image used in `TSA_Image`, `Ave_ARpar` finds the average of the AR regressive parameters in the video stream.

2. `TSA_FAverage`

   For the input image used in `TSA_Correl_Image`, `TSA_FAverage` finds the average of the correlations in the dependence structures for each pixel in the video stream.

We now begin explaining the functions in detail.

```
function [ map ] = Map_Maker( lim )

n = 2*lim+1;
map = zeros(n,n);
for i = 1 : n
    if i < lim + 1
        map(i,(lim+2-i:lim+i)) = 1;
    elseif i > lim + 1
        k = n - i + 1;
        map(i,(lim+2-k:lim+k)) = 1;
    else
        map(i,(1:n)) = 1;
    end
end

end
```

Input variable

`lim`: 1×1 The number of connected elements to extend from the centre.

Return variable

`map`: $(2 * \texttt{lim} + 1) \times (2 * \texttt{lim} + 1)$ A matrix representation of the dependence structure defined in section 3.2 of Chapter 3.

Method `Map_Maker` creates the dependence structure defined in section 3.2 with `lim` $= K$. It assigns a value of one where the elements of the matrix corresponds to the form of the dependence structure. For example, for $M^{(3)}$, we have that `map = Map_Maker(3)` produces the following matrix in MATLAB:

```
map =

     0     0     0     1     0     0     0
     0     0     1     1     1     0     0
     0     1     1     1     1     1     0
     1     1     1     1     1     1     1
     0     1     1     1     1     1     0
     0     0     1     1     1     0     0
     0     0     0     1     0     0     0
```

```
function [ Diamond ] = Diamond_Maker( lim )

out = Map_Maker( lim );
in = Map_Maker( lim-1 );

s_out = size(out);
s_in = size(in);

in = [ zeros(s_in(1)+1,1) , [ zeros(1,s_in(2)) ; in ] ];
[I,J] = ind2sub(size(in),find(in == 1));
ind = sub2ind(s_out,I,J);

Diamond = out;
Diamond(ind) = 0;
Diamond(lim+1,lim+1) = 1;

end
```

Input variable

`lim`: 1×1 The number of elements to extend from the centre.

Return variable

`Diamond`: $(2 * \texttt{lim} + 1) \times (2 * \texttt{lim} + 1)$ A matrix representation of a diamond.

Method `Diamond_Maker` creates a 'hollow' dependence structure by calling `Map_Maker` twice - first to define the full structure, second to subtract from the first to obtain to remove all the 1's in the filling. For example, calling `Di = Diamond_Maker(3)` produces the following output from MATLAB:

```
Di =

     0     0     0     1     0     0     0
     0     0     1     0     1     0     0
     0     1     0     0     0     1     0
     1     0     0     1     0     0     1
     0     1     0     0     0     1     0
     0     0     1     0     1     0     0
     0     0     0     1     0     0     0
```

```
function [ TSA ] = TSA_Image( Image )

%Obtaining image dimensions
[m , n] = size(Image(1,1).frames);

%Initialising structure classes for speed
TSA(m,n).normal = 0;
TSA(m,n).rescorr = 0;
TSA(m,n).ARlag = 0;
TSA(m,n).ARpar = zeros(1,10);
for i = 1 : m
    for j = 1 : n
        f = ones(500,1);

        for k = 1 : 500
            f(k) = Image(k,1).frames(i,j);
        end

        dw = 0;
        lag = 0;
        stop = 0;
        %Estimating time series parameters with CLS
        while dw < 0.05 && stop == 0
            lag = lag + 1;
            y = f(lag+1:500);
            X = [];
            for l = 1 : lag;
                X = [ f(l:500-(lag-(l-1))) , X ];
            end
            X = [ones(n-lag,1) , X];
            b = inv(X'*X)*X'*y;
            r = y - X*b;

            %Durbin-Watson test for uncorrelated residuals
            dw = dwtest(r,X);
            if lag > 10
                stop = 1;
            end
        end
        %Anderson-Darling test for normality of residuals
```

```
41          ad = adtest(r);
42
43          TSA(i,j).normal = ad;
44          TSA(i,j).rescorr = dw;
45          TSA(i,j).ARlag = numel(b) - 1;
46          TSA(i,j).ARpar(1:numel(b)) = b;
47      end
48  end
49
50  end
```

Input variable

`Image`: $k \times 1$ A structure variable containing an image sequence (video) with each structure element holding one frame in a field called `frames`: $m \times n$.

Return variable

`TSA`: $m \times n$ A structure variable containing the following fields at each element for each pixel (i, j), $i = 1, 2, ..., \mathtt{m}$ and $j = 1, 2, ..., \mathtt{n}$:

`normal`: 1×1 The p-value from the Anderson-Darling test for normality of residuals.

`rescorr`: 1×1 The p-value from the Durbin-Watson test for uncorrelated residuals.

`ARlag`: 1×1 The number of AR parameters required for the time series model.

`ARpar`: 1×10 The vector containing the estimated AR parameters from conditional least squares. Note that the maximum number of AR parameters allowed is 10.

Method `TSA_Image` begins by initialising fields `normal, rescorr, ARlag` and `ARpar`. This is done to improve on computing speed. Then for each pixel (i, j) within the frame we extract 500 observations from the video stream to estimate our AR parameters. The procedure of estimation is as follows [`lines 11 - 48`]:

1. Start `p`, the number of AR parameters, at 1.
2. [`lines 25 - 31`] Formulate the vector of dependent observations $\mathbf{y}$ and design matrix $\mathbf{X}$ as

$$\mathbf{y} = \mathbf{x}_n = [x_n \;\; x_{n-1} \;\; ... \;\; x_{p+1}]',$$

and

$$\mathbf{X} = [\mathbf{1} \;\; \mathbf{x}_{n-1} \;\; \mathbf{x}_{n-2} \;\; ... \;\; \mathbf{x}_{n-p}],$$

with x_i the ith observation in the sample of pixels. Estimate the AR parameters by CLS as

$$\mathbf{y} = \mathbf{X}\mathbf{b},$$

with $\mathbf{b} = [c \;\; \phi_1 \;\; \phi_2 \;\; ... \;\; \phi_p]'$.

3. [lines 33 & 35] Compute the residuals $\boldsymbol{\varepsilon} = \mathbf{y} - \mathbf{Xb}$ then apply the Durbin-Watson test for uncorrelated residuals (the function readily available in MATLAB).

4. If the p-value of the test is less than 0.05 then redo steps 2 and 3 with new p equal to old p plus 1, else proceed to step 5.

5. [lines 41 - 46] Apply the Anderson-Darling test for normality (the function readily available in MATLAB) on the residuals, store the p-values from the Anderson-Darling and Durbin-Watson tests, and the AR parameters in the appropriate fields. Move onto the next pixel.

```
function [ TSACorr ] = TSA_Correl_Image( V_range, H_range, map_in, Image , TSA )

%Determining image size
s_map = size(map_in);
cen = s_map(1) - (s_map(1) - 1)/2;

%Centering the dependence structure map_in
[I,J] = find(map_in == 1);
M_full = transpose([I-cen,J-cen]);

D_H = (H_range(1) : H_range(2));
D_V = (V_range(1) : V_range(2));

%Preallocating structure fields for speed
sam = 500;
TSACorr(V_range(2)-V_range(1)+1,H_range(2)-H_range(1)+1).Corr = zeros(s_map(1),s_map(1));
TSACorr(V_range(2)-V_range(1)+1,H_range(2)-H_range(1)+1).Cov = zeros(s_map(1),s_map(1));
TSACorr(V_range(2)-V_range(1)+1,H_range(2)-H_range(1)+1).Map = zeros(s_map(1),s_map(1));
TSACorr(V_range(2)-V_range(1)+1,H_range(2)-H_range(1)+1).Mean = zeros(s_map(1),s_map(1));
for i = V_range(1) : V_range(2)
    for j = H_range(1) : H_range(2)
        %Creating maps to suit current (i,j) location. Here, if the indices
        %of TestM are contained in D_H and D_V, then we choose those
        %indices as our current map.
        TestM = M_full + repmat([i;j],1,size(M_full,2));
        loc_i = unique(intersect(TestM(1,:),D_V)) - i;
        loc_j = unique(intersect(TestM(2,:),D_H)) - j;

        map = [];
        for ii = 1 : size(loc_i,2)
            for jj = 1 : size(loc_j,2)
                if map_in(loc_i(ii)+cen, loc_j(jj)+cen) ~= 0
                    map = [ map , [loc_i(ii) ; loc_j(jj)] ];
                end
            end
        end


        X = zeros(sam,size(map,2));
        X_bar = zeros(1,size(map,2));
```

```
41          for m = 1 : size(map,2)
42              for n = 1 : sam
43                  X(n,m) = Image(n,1).frames(i+map(1,m),j+map(2,m));
44              end
45
46              par = TSA(i+map(1,m),j+map(2,m)).ARpar;
47              X_bar(m) = par(1)/( 1 - sum( par(2:numel(par)) ) );
48          end
49
50          %Sample statistics.
51          X0 = X - repmat(X_bar,size(X,1),1);
52          X_cov = transpose(X0)*X0/(sam-1);
53          D = real(sqrt(inv(eye(size(X_cov,1)).*(X_cov))));
54          X_cor = D*X_cov*D;
55
56          %Storing results in structures. With (0,0) as a reference point, we
57          %can recreate the map_in using our temporary map.
58          i0 = find(map(1,:) == 0);
59          j0 = find(map(2,:) == 0);
60          loc0 = intersect(i0,j0);
61          for m = 1 : size(map,2)
62              TSACorr(i,j).Corr(map(1,m)+cen,map(2,m)+cen) = X_cor(loc0,m);
63              TSACorr(i,j).Mean(map(1,m)+cen,map(2,m)+cen) = X_bar(m);
64          end
65          TSACorr(i,j).Cov = X_cov;
66          TSACorr(i,j).Map = map;
67      end
68  end
69
70  end
```

Input variables

`V_range`: 2×1 A vector storing the range of vertical pixels to consider when running `TSA_Correl_Image` with first and last element the minimum and maximum of the pixel location respectively. The intersection of `V_range` and `H_range` then defines a rectangular region for the function to operate on.

`H_range`: 2×1 Similar to `V_range` but for horizontal considerations.

`map_in`: $m_{\texttt{map}} \times m_{\texttt{map}}$ An odd matrix with 1 at the centre and zeros and ones elsewhere (user-defined). Note that a value of 1 at element (i, j) corresponds to finding the correlation and covariance between element (i, j) and the centre for each pixel in the frame.

`Image`: $k \times 1$ See *Input variables* in function `TSA_Image`.

`TSA`: $m \times n$ See *Return variable* in function `TSA_Image`.

Return variable

`TSACorr:` `V` × `H` A structure variable containing the correlation matrix, covariance matrix and the mean for the set of pixels defined by the dependence structure[2] `map` in fields `Corr, Cov` and `mean` respectively. Furthermore, we also store the location of these image pixels in a matrix called `Map`. Note that the number of rows `V` of `TSA_Corr` is determined by `V_Range` as `V = V_range(2) - V_range(1) + 1`. This goes similarly for the number of columns `H`.

Method [lines 3 - 9] For input dependence structure `map_in`, we obtain the *subscripts* of where the value of 1's occur and centralise them. Thus we obtain, for each value of 1, the total vertical and horizontal *displacement* required to travel from the centre to another point within `map_in` - these are stored in `M_full`. This implies that, for each pixel (i, j), we can recreate the dependence structure centred at (i, j) simply by adding the displacements to (i, j).

[lines 11 - 12] The variables `D_H` and `D_V` contain all possible (i, j) pixel locations to consider (as dictated by `V_range` and `H_range`). These are used to prevent the dependence structures from extending beyond feasible limits of the frame or the user-defined limits. For example, for the dependence structure defined in FILLMEIN, we have that centering the structure at the top-left hand corner pixel $(1, 1)$ places parts of the structure beyond the scope of the frame. Intersecting `D_H` and `D_V` with the dependence structure at $(1, 1)$, we will get usable pixel information.

[lines 39 - 66] For each pixel within the dependence structure centred at (i, j), we extract 500 samples from the video stream. Then we compute the sample covariance matrix and sample correlation matrix. After which we store these results with respect to the input dependence structure. This is repeated for all (i, j) dictated by `V_range` and `H_range`.

```
1  function [ Ave_pval , TSA_HypoTest ] = TSA_HypoTest( V_range , H_range , map_in ,
2                                                        alpha , TSA , Image)
3
4  %Determining image size
5  [m , n] = size(TSA);
6
7  %Centering the dependence structure map
8  s_map = size(map_in);
9  cen = s_map(1) - (s_map(1) - 1)/2;
10 [I,J] = find(map_in == 1);
11 M_full = transpose([I-cen,J-cen]);
12
13 %Determining all possible pixel values
14 D_H = (H_range(1) : H_range(2));
15 D_V = (V_range(1) : V_range(2));
16
17 sam = 500;
18 M = round(log(sam));
```

[2] Not a MATLAB structure, but the dependence structure defined in FILLMEIN

```
19
20  %Preallocating structure fields for speed
21  TSA_HypoTest(m,n).p_value = zeros(s_map(1),s_map(1));
22  TSA_HypoTest(m,n).Haugh = zeros(s_map(1),s_map(1));
23  TSA_HypoTest(m,n).RejH0 = zeros(s_map(1),s_map(1));
24  Ave_pval = zeros(s_map(1),s_map(1));
25  Nval = Ave_pval;
26  for i = V_range(1) : V_range(2)
27      for j = H_range(1) : H_range(2)
28          %Creating maps to suit current (i,j) location. Here, if the indices
29          %of TestM are contained in D_H and D_V, then we choose those
30          %indices as our current map
31          TestM = M_full + repmat([i;j],1,size(M_full,2));
32          loc_i = unique(intersect(TestM(1,:),D_V)) - i;
33          loc_j = unique(intersect(TestM(2,:),D_H)) - j;
34
35          map = [];
36          for ii = 1 : size(loc_i,2)
37              for jj = 1 : size(loc_j,2)
38                  if map_in(loc_i(ii)+cen, loc_j(jj)+cen) ~= 0
39                      map = [ map , [loc_i(ii) ; loc_j(jj)] ];
40                  end
41              end
42          end
43
44          %Creating a temporary observation matrix X to find the residuals
45          %from the AR model defined in TSA
46          X = zeros(sam,size(map,2));
47          for nn = 1 : sam
48              for mm = 1 : size(map,2)
49                  X(nn,mm) = Image(nn,1).frames(i+map(1,mm),j+map(2,mm));
50              end
51          end
52
53          i0 = find(map(1,:) == 0);
54          j0 = find(map(2,:) == 0);
55          loc0 = intersect(i0,j0);
56
57          %Computing residuals for centre pixel
58          lag = TSA(i,j).ARlag;
59          y0 = X(lag+1:sam,loc0);
60          X0 = [];
61          for l = 1 : lag;
62              X0 = [ X(l:sam-(lag-(l-1)),loc0) , X0 ];
63          end
64          X0 = [ones(sam-lag,1) , X0];
65          b = TSA(i,j).ARpar(1:lag+1)';
66          u0 = y0 - X0*b;
67
68          %Computing residuals for the other pixels defined in the dependence
69          %structure. In addition, apply Haugh's test of independence.
70          for mm = 1 : size(map,2)
71              i1 = i + map(1,mm);
72              j1 = j + map(2,mm);
```

```
73
74              if mm ~= loc0 && TSA(i1,j1).ARlag < 10
75                  lag = TSA(i1,j1).ARlag;
76                  y1 = X(lag+1:sam,mm);
77                  X1 = [];
78                  for l = 1 : lag;
79                      X1 = [ X(l:sam-(lag-(l-1)),mm) , X1 ];
80                  end
81                  X1 = [ones(sam-lag,1) , X1];
82                  b = TSA(i1,j1).ARpar(1:lag+1)';
83                  u1 = y1 - X1*b;
84
85                  s = 0;
86                  for k = -M : M
87                      s = s + r_uv(k,u0,u1)^2;
88                  end
89                  s = s*min(numel(u0),numel(u1));
90
91                  i_map = cen + map(1,mm);
92                  j_map = cen + map(2,mm);
93                  TSA_HypoTest(i,j).p_value(i_map,j_map) = 1 - chi2cdf(s,2*M+1);
94                  TSA_HypoTest(i,j).Haugh(i_map,j_map) = s;
95              end
96          end
97
98          %Extracting p-values from each test
99          p_val = TSA_HypoTest(i,j).p_value;
100         if sum(sum(isnan(p_val))) == 0
101             s = TSA_HypoTest(i,j).Haugh;
102             TSA_HypoTest(i,j).RejH0 = ( p_val < alpha ).*(s > 0);
103
104             m1 = size(p_val,1);
105             n1 = size(p_val,2);
106             Ave_pval(1:m1,1:n1) = Ave_pval(1:m1,1:n1) + p_val;
107             Nval(1:m1,1:n1) = Nval(1:m1,1:n1) + (s > 0);
108         end
109     end
110 end
111 Ave_pval = Ave_pval.*(Nval > 0)./Nval;
112
113 end
```

Input variables

`V_range`: 2×1 See *Input variables* in function `TSA_Correl_Image`.

`H_range`: 2×1 See *Input variables* in function `TSA_Correl_Image`.

`map_in`: $m_{\texttt{map}} \times m_{\texttt{map}}$ See *Input variables* in function `TSA_Correl_Image`.

`alpha`: 1×1 The significance level for Haugh's [12] test of independence between two univariate time series.

`TSA`: $m \times n$ See *Return variable* in function `TSA_Image`.

`Image`: $k \times 1$ See *Input variables* in function `TSA_Image`.

Return variables

`Ave_pval`: $m_{\texttt{map}} \times m_{\texttt{map}}$ A matrix containing the average of the p-values of the tests with its structure defined by `map_in`.

`TSA_HypoTest`: $m \times n$ A structure variable with each element (i, j) containing the hypothesis tests results for corresponding pixel at location (i, j). The following fields are embedded within each structure element:

`p_value`: $m_{\texttt{map}} \times m_{\texttt{map}}$ A matrix containing the p-values of the tests with its structure defined by `map_in`.

`Haugh`: $m_{\texttt{map}} \times m_{\texttt{map}}$ A matrix containing the test statistics of the tests with its structure defined by `map_in`.

`RejH0`: $m_{\texttt{map}} \times m_{\texttt{map}}$ A matrix containing 0's and 1's of the tests with its structure defined by `map_in`. Note that 1 implies we have rejected the null hypothesis of independence and 0 otherwise.

Method Same as `TSA_Correl_Image`, `TSA_HypoTest` begins by developing the displacements from the centre to points in the input `map_in` and determining the range of i and j values so that the dependence structure centred at each pixel (i, j) is in context.

`[lines 44 - 51]` At pixel (i, j), create a temporary observation matrix `X` to store 500 observations of the pixel variables present within the input dependence structure.

`[lines 53 - 66]` To begin with Haugh's test of independence between two univariate time series [12], we begin by calculating the residuals for the centre pixel and fixing it constant for the particular i^{th} and j^{th} iteration of the two vertical and horizontal pixel loops. This is because these residuals remain constant throughout the test for each pixel within the dependence structure. To obtain the residuals, we use the information from the structure variable `TSA` to recreate our design matrix $\mathbf{X}$ from CLS (since `TSA` contains the estimated AR parameters) and vector of regressors $\mathbf{y}$, and set the residuals as $\boldsymbol{\varepsilon} = \mathbf{y} - \mathbf{X}\mathbf{b}$, with $\mathbf{b}$ the vector containing our AR parameters.

`[lines 68 - 96]` This process is repeated for all pixel present within the dependence structure. In addition, we implement Haugh's test of independent univariate time series [12] with the centre pixel and the others. We store the p-values and their test statistics in fields `p_value` and `Haugh` respectively.

`[lines 98 - 107]` In these lines we keep a running total of the p-values in the dependence structure where, at the end of the code, we divide by the total number of occurences to get the average p-value of the whole dependence structure.

```
function [ Ave_ARpar ] = Ave_ARpar( TSA )

[m,n] = size(TSA);
N = 0;
Ave_ARpar = zeros(11,1);
for i = 1 : m
    for j = 1 : n
        ARpar = TSA(i,j).ARpar(2:numel(TSA(i,j).ARpar));
        ARpar = reshape(ARpar,numel(ARpar),1);

        if sum(isnan(ARpar)) == 0 && sum(isinf(ARpar)) == 0
            Ave_ARpar(1:numel(ARpar)) = Ave_ARpar(1:numel(ARpar)) + ARpar;
            N = N + 1;
        end

    end
end

Ave_ARpar = Ave_ARpar/N;

end
```

Input variable

`TSA`: $m \times n$ See *Return variable* in function `TSA_Image`.

Return variable

`Ave_ARpar`: 11×1 A vector containing the average of the AR parameters within the video from `TSA_Image`.

Method For each pixel (i, j), `Ave_ARpar` keeps a running sum of the AR parameters of the pixel in the video stream. The final sum is then divided by the total number of occurences to obtain the average.

```
function [ Corr_Ave ] = TSA_FAverage( TSACorr )

Corr_Ave = zeros(size(TSACorr(1,1).Corr,1),size(TSACorr(1,1).Corr,1));
N = Corr_Ave;

for i = 1 : size(TSACorr,1)
    for j = 1 : size(TSACorr,2)
        rho = TSACorr(i,j).Corr;

        if sum(sum(isnan(rho))) == 0
            size_rho = size(rho);
            Corr_Ave(1:size_rho(1),1:size_rho(2)) = Corr_Ave(1:size_rho(1),1:size_rho(2))
                    + rho;
            N(1:size_rho(1),1:size_rho(2)) = N(1:size_rho(1),1:size_rho(2)) + (rho ~= 0);
        end
```

```
    end
end

Corr_Ave = Corr_Ave ./ N;
end
```

Input variable

`TSACorr`: $m \times n$ See *Return variable* in function `TSA_Correl_Image`.

Return variable

`Corr_Ave`: $m_{\text{map}} \times m_{\text{map}}$ The average of the correlations within each dependence structure.

Method We sum over all correlations within each dependence structure and divide the result by the total number of occurences.

Noise removal in images [MATLAB]

Introduction

In this section we provide the codes used to obtain results in Chapter 4. A summary of the functions is given below:

`LCG`: Simulates uniform variates using the LCG algorithm.

`Box_Muller`: Simulates standard normal variates using the Box-Muller algorithm (uses `LCG`).

`Noisy`: Simulates random variates from distributions specified in Table 4.1 of Chapter 4 (uses `LCG` and `Box_Muller`).

`L_n`: Applies the L_n operator on an image f.

`U_n`: Applies the U_n operator on an image f.

`PP_plotv2`: Finds the PP plot co-ordinates for input noise sample.

```
function [ U ] = LCG( a, c, m, x0, dim )

n = dim(1)*dim(2);

x = [x0 ; zeros(n,1)];
for i = 2 : (n+1)
    x(i) = mod(a*x(i-1) + c, m);
end
x = x(2:n+1,:)/m;

U = reshape(x,dim(1),dim(2));

end
```

Input variables

`a`: 1×1 The multiplier.

`c`: 1×1 The increment.

`m`: 1×1 The modulus.

`x0`: 1×1 The initial value (seed).

`dim`: $s \times t$ The dimension of the matrix in which we populate with random uniform variates.

Return variable

`U`: $s \times t$ A matrix containing independent uniform variates.

Method With an initial value x_0 (seed), the function obtains a sequence of integers $x_1, x_2, ..., x_{st}$ with the recursive formula given by

$$x_n = (ax_{n-1} + c)(\text{mod } m).$$

Thus x_n is the remainder of $ax_{n-1} + c$ after dividing by m. The integers $x_1, x_2, ..., x_{st}$ are then multiplied by $1/m$ to obtain mn random numbers on the interval $[0, 1)$.

```
function [ N , U ] = Box_Muller( a, c, m, x0, dim )

n = dim(1)*dim(2);
if mod(n,2) ~= 0
    n = n + 1;
end

U = LCG (a, c, m, x0, [n 1]);

U_v = reshape(U,n,1);

odd_idx = (1:2:n);
even_idx = (2:2:n);

z1 = sqrt(-2*log(U_v(odd_idx,:))) .* cos(2*pi*U_v(even_idx,:));
z2 = sqrt(-2*log(U_v(odd_idx,:))) .* sin(2*pi*U_v(even_idx,:));

N = zeros(dim);
N(odd_idx) = z1;
select = intersect(even_idx,(1:dim(1)*dim(2)));
N( select ) = z2( 1:numel(select) );

end
```

Input variables

`a`: 1×1 The multiplier.

`c`: 1×1 The increment.

`m`: 1×1 The modulus.

`x0`: 1×1 The initial value (seed).

`dim`: $s \times t$ The dimension of the matrix in which we populate with standard random normal variates.

Return variables

`N`: $s \times t$ A matrix containing independent standard normal variates.

`U`: $s \times t$ A matrix containing the independent uniform variates used to simulate the normal variates in `N`.

Method Since the Box-Muller algorithm generates an even number of standard normal variates, `[lines 3 - 6]` ensures that we simulate an even number of variates. For each pair of uniform variates (u_i, u_{i+1}) in $\mathbf{u}$, $i = 1, 3, 5, ..., st - 1$ generated by the LCG (`[line 8]`), compute,

$$z_i = \sqrt{-2 \ln u_1} \cos(2\pi u_2), \text{ and}$$

$$z_{i+1} = \sqrt{-2 \ln u_1} \sin(2\pi u_2).$$

Note that z_i and z_{i+1} are independent standard normal variates and are stored in `z1` and `z2` respectively. Next, store all random standard normal variates in a matrix `N` by selecting from the appropriate elements of `z1` and `z2`.

```
1  function [ Noise , U ] = Noisy ( a , c, m, x0, dim, k, std_dev)
2
3  var = std_dev^2;
4  U = LCG( a, c, m, x0, dim );
5  if k == 1
6      Noise = (U .* (2*sqrt(3)*std_dev)) - sqrt(3)*std_dev;
7  elseif k == 2
8      [ Noise , U1 ] = Box_Muller(a, c, m, x0, dim);
9      Noise = Noise*std_dev;
10 elseif k == 3
11     s = sqrt(3*var/pi^2);
12     Noise = -s*log( U .^ (-1) - 1);
13 elseif k == 4
14     s = 2/(4 - pi)*var;
15     Noise = sqrt( -2*s*log( 1 - U ) ) - sqrt( -2*s*log( 0.5 ) );
16 elseif k == 5
17     beta = sqrt(6/(pi^2)*var);
18     mu = -beta*0.5772;
19     Noise = -beta*log(-log(U)) + mu;
20 elseif k == 6
21     Noise = -std_dev*log(1 - U) + std_dev*log(0.5);
22 end
23
24 end
```

Input variables

`a`: 1×1 The multiplier.

`c`: 1×1 The increment.

`m`: 1×1 The modulus.

`x0`: 1×1 The initial value (seed).

`dim`: $m \times n$ The dimension of the matrix in which we populate with random variates.

`k`: 1×1 The type of noise to generate (see Method).

`std_dev`: 1×1 The standard deviation of the random numbers.

Return variables

`Noise`: $m \times n$ A matrix containing independent variates from the same distribution specified by `k` with standard deviation `std_dev`.

`U`: $m \times n$ A matrix containing the independent uniform variates used to simulate the normal variates in `N`.

Method `Noisy` simulates random variates from distributions given in Table 4.1 of Chapter 4. This is done by setting the mean of the distribution equal to zero and solving for the distribution parameters by setting its variance equal to the input parameter `std_dev`^{2}. If the distribution is not normal, the inverse transform method is used, else the Box-Muller is used. The type of distribution to simulate is determined by the input parameter `k`, if:

`k = 1` `Noisy` simulates random variates from the uniform distribution.

`k = 2` `Noisy` simulates random variates from the normal distribution.

`k = 3` `Noisy` simulates random variates from the logistic distribution.

`k = 4` `Noisy` simulates random variates from the Rayleigh distribution.

`k = 5` `Noisy` simulates random variates from the Gumbel distribution.

`k = 6` `Noisy` simulates random variates from the exponential distribution.

In addition, for Rayleigh and exponential distributions, the whole sample is shifted to the left by its median.

```
1  function [ L_f ] = L_n( f_in, n )
2
3  f = [ zeros(1,size(f_in,2)+2) ;
4        zeros(size(f_in,1),1) , f_in , zeros(size(f_in,1),1) ;
5        zeros(1,size(f_in,2)+2) ];
```

```
6
7   mn = size(f);
8   remove = f;
9   numele = sort(unique(f),'descend');
10
11  for i = 1 : numel(numele)
12      isnumele = (f >= numele(i));
13
14      c = bwconncomp(isnumele, 4);
15      c_struct = cell2struct(c.PixelIdxList,'PixIdx',1);
16
17      for j = 1 : size(c_struct,1)
18          if size(c_struct(j,1).PixIdx) <= n
19              [I,J] = ind2sub(mn,c_struct(j,1).PixIdx);
20              IJ_all = [I + 1 , J     ;
21                        I     , J + 1 ;
22                        I - 1 , J     ;
23                        I     , J - 1];
24
25              check1 = (IJ_all > 0);
26              check2 = (IJ_all(:,1) <= mn(1));
27              check3 = (IJ_all(:,2) <= mn(2));
28              check = (check1 == [check2 , check3]);
29              IJ_all = IJ_all( (sum(check,2) == 2) ,:);
30
31              ind_all = sub2ind(mn,IJ_all(:,1),IJ_all(:,2));
32              ind_set = sub2ind(mn,I,J);
33              ind_sur = setdiff(ind_all, ind_set);
34
35              if max(f(ind_sur)) < numele(i)
36                  remove(ind_set) = max(f(ind_sur));
37              end
38          end
39      end
40  end
41
42  L_f = remove(2:mn(1) - 1, 2:mn(2) - 1);
43
44  end
```

Input variables

`f_in`: $h \times w$ An image f.

`n`: 1×1 The level n at which to apply L_n.

Return variable

`L_f`: $h \times w$ The image $L_n f$.

Method `L_f` removes all pulses of size `n` or less by using the Roadmaker's algorithm. That is, it does not incorporate the definition (this would computationally intense).

`[lines 3 - 5]` appends zeros around the image `f`. This is done so that 4-connectivity can be used. Next we set an empty matrix `remove` equal to `f` so that we can track all the removed pulses and store all the unique elements within `f` in `numele`.

`[lines 11 - 40]` For each element in `numele`, say u, we find the logical matrix `isnumele` where,

$$\texttt{isnumele}_{ij} = \begin{cases} 1 & \text{if the element in } \texttt{f}_{ij} \text{ is greater than } u \\ 0 & \text{otherwise.} \end{cases}$$

Apply `bwconncomp`[3] on `isnumele` to extract all connected sets that have pixel values greater than or equal to u and store all indices in `c_struct` (a structure variable).

For each element in `c_struct` determine its size. Since the size of the element indicates the size of the pulse, we proceed if the size is less than `n` (since L_n removes all pulses of size n or smaller) else move onto the next element in `c_struct`. Next we change the indices to subscripts and determine the indices that are above, below, to the left, and right of them. After locating the indices, set the pulse equal to the maximum of its surrounding elements by changing the appropriate values in `remove`.

Once all the unique elements of `f` have been examined, all upward pulses of size n or less are removed in `removed`. After which we return `L_f` as `removed` without the appended zeros.

```
1  function [ U_f ] = U_n( f_in, n )
2
3  f = [ zeros(1,size(f_in,2)+2) ;
4        zeros(size(f_in,1),1) , f_in , zeros(size(f_in,1),1) ;
5        zeros(1,size(f_in,2)+2) ];
6
7  mn = size(f);
8  remove = f;
9  numele = sort(unique(f),'ascend');
10
11 for i = 1 : numel(numele)
12     isnumele = (f <= numele(i));
13
14     c = bwconncomp(isnumele, 4);
15     c_struct = cell2struct(c.PixelIdxList,'PixIdx',1);
16
17     for j = 1 : size(c_struct,1)
18         if size(c_struct(j,1).PixIdx) <= n
19             [I,J] = ind2sub(mn,c_struct(j,1).PixIdx);
20             IJ_all = [I + 1 , J     ;
21                       I     , J + 1 ;
```

[3] `bwconcomp(f,c)` is a MATLAB function which finds all nonzero connected sets in `f` with `c`-connectivity and stores the information in a structure variable. The most important field of which is `.PixelIdxList` which contains all the indices of the connected sets in cell format.

```
                        I - 1 , J     ;
                        I     , J - 1];

            check1 = (IJ_all > 0);
            check2 = (IJ_all(:,1) <= mn(1));
            check3 = (IJ_all(:,2) <= mn(2));
            check = (check1 == [check2 , check3]);
            IJ_all = IJ_all( (sum(check,2) == 2) ,:);

            ind_all = sub2ind(mn,IJ_all(:,1),IJ_all(:,2));
            ind_set = sub2ind(mn,I,J);
            ind_sur = setdiff(ind_all, ind_set);

            if min(f(ind_sur)) > numele(i)
                remove(ind_set) = min(f(ind_sur));
            end
        end
    end
end

U_f = remove(2:mn(1) - 1, 2:mn(2) - 1);

end
```

Input variables

`f_in`: $h \times w$ An image f.

`n`: 1×1 The level n at which to apply U_n.

Return variable

`U_f`: $h \times w$ The image $U_n f$.

Method Similar to `L_f`.

```
function [ TVf ] = TV2( f )

mn = size(f);

v_diff = f(2:mn(1),:) - f(1:mn(1)-1,:);
h_diff = f(:,2:mn(2)) - f(:,1:mn(2)-1);

TVf = sum( sum( abs(v_diff) ) ) + sum( sum( abs(h_diff) ) );

end
```

Input variable

`f`: An image f.

Return variable

`TVf`: The total variation of image f

Method This function utilises the definition of total variation in section 3.3.1 in Chapter 3. It starts by finding the vertical differences in `v_diff`, all the horizontal differences in `h_diff` and summing the absolute value of the differences.

```
function [ PP_cord ] = PP_plotv2 ( noise , k , std_dev)

sam_n = size(noise,1)*size(noise,2);
X = sort(real(reshape(noise,[sam_n,1])));
X = X(X ~= 0);

Emp_Dist = (1/size(X,1) : 1/size(X,1) : 1)';

var = std_dev^2;
if k == 1
    sam_Dist = (X + sqrt(3)*std_dev)/(2*sqrt(3)*std_dev);
elseif k == 2
    sam_Dist = 0.5 + 0.5*erf((X)/(sqrt(2)*std_dev));
elseif k == 3
    s = sqrt(3*var/pi^2);
    sam_Dist = (1+exp(-(X/s))).^(-1);
elseif k == 4
    s = 2/(4 - pi)*var;
    sam_Dist = 1 - exp(-((X + sqrt( -2*s*log( 0.5 ) )).^2)/(2*s));
elseif k == 5
    beta = sqrt(6/(pi^2)*var);
    mu = -beta*0.5772;
    sam_Dist = exp(-exp(-(X-mu)/beta));
elseif k == 6
    lambda = 1/std_dev;
    sam_Dist = 1 - exp(-lambda*(X - std_dev*log(0.5)));
end

sam_Dist(sam_Dist < 0) = 0;
sam_Dist(sam_Dist > 1) = 1;

QQ_cord = [Emp_Dist , sort(real(sam_Dist)) ];

end
```

Input variables

`noise`: $m \times n$ The matrix containing the noise sample.

`k`: 1×1 The parameter specifying the type of distribution.

`std_dev`: 1×1 The required standard deviation of the distribution.

Return variable

`Noise:` $m \times n$ The matrix containing a sample of the distribution specified by `k`.

Method The function starts by reshaping the matrix `noise` into a vector `X` and sorting the sample in ascending order. This is done so that we can finds its sample cdf. The theoretical cdf is the uniform vector $[0, 1]$, with total number of elements equal to the sample size. For the sample cdf, we determine the cdf values by using the appropriate distribution function dictated by `k`, which follows the exact same call as function `Noisy`. Finally, because the noise is extracted from images it can contain values outside the support of the specified distribution, we set all negative cdf values to 0 and all values greater than one to 1.

Note that for the Rayleigh and exponential distributions we shift the sample to the right by the appropriate distribution median so that the range of the sample is within the support of the distribution.
