The Discrete Pulse Transform and Applications

Jacques Pierre du Toit

Thesis presented in partial fulfillment of the requirements for the degree of Master of Science at the University of Stellenbosch.

Supervisor: C.H. Rohwer

March 2007

Abstract

Data analysis frequently involves the extraction (i.e. recognition) of parts that are important at the expense of parts that are deemed unimportant. Many mathematical perspectives exist for performing these separations, however no single technique is a panacea as the definition of signal and noise depends on the purpose of the analysis. For data that can be considered a sampling of a smooth function with added 'well-behaved' noise, linear techniques tend to work well. When large impulses or discontinuities are present, a non-linear approach becomes necessary.

The *LULU* operators, composed using the simplest rank selectors, are non-linear operators that are comparable to the well-known median smoothers, but are computationally efficient and allow a conceptually simple description of behaviour. Defined using compositions of different order *LULU* operators, the discrete pulse transform (*dpt*) allows the interpretation of sequences in terms of pulses of different scales: thereby creating a multi-resolution analysis. These techniques are very different from those of standard linear analysis, which renders intuitions regarding their behaviour somewhat undependable.

The *LULU* perspective and analysis tools are investigated with a strong emphasis on practical applications. The *LULU* smoothers are known to separate signal and noise efficiently: they are idempotent and co-idempotent. Sequences are smoothed by mapping them into smoothness classes; which is achieved by the removal, in a consistent manner, of block-pulses. Furthermore, these operators preserve local trend (i.e. they are fully trend preserving). Differences in interpretation with respect to Fourier and Wavelet decompositions are also discussed. The *dpt* is defined, its implications are investigated, and a linear time algorithm is discussed. The *dpt* is found to allow a multi-resolution measure of roughness. Practical sequence processing through the reconstruction of modified pulses is possible; in some cases still maintaining a consistent multi-resolution interpretation. Extensions to two-dimensions is discussed, and a technique for the estimation of standard deviation of a random distribution is presented. These tools have been found to be effective in the analysis and processing of sequences and images.

The *LULU* tools are an useful alternative to standard analysis methods. The operators are found to be robust in the presence of impulsive and more 'well-behaved' noise. They allow the fast design and deployment of specialized detection and processing algorithms, and are possibly very useful in creating automated data analysis solutions.

Opsomming

Data analise behels gereeld die skeiding van dit wat belangrik is van dit was as onbelangrik beskou word. Baie wiskundige perspektiewe bestaan wat metodes verskaf vir die uitvoer van sulke skeidings, maar geen enkele tegniek is 'n panasee aangesien die definisie van sein en geraas afhanklik is van die doel van die analise. Lineêre metodes is geneig om goed te werk vir data wat beskou kan word as 'n monstering van 'n gladde funksie. Die teenwoordigheid van groot impulse of diskontinuïteite noodsaak 'n nie-lineêre benadering.

Die *LULU* operatore, saamgestel vanuit die eenvoudigste rang-orde selektors, is nie-lineêre operatore wat vergelykbaar met die bekende mediaan gladstrykers is, maar is doeltreffend berekenbaar en laat 'n konseptueel eenvoudige beskrywing van hul gedrag toe. Die diskrete puls transform (*dpt*), wat saamgestel is uit verskillende orde *LULU* operatore, gee 'n interpretasie van 'n ry in terme van pulse van verskillende skale, en skep so deur 'n multi-resolusie analise. Hierdie metodes van analise is baie anders as die van standaard lineêre analisie, wat intuïsies rakende hul gedrag ietwat onbetroubaar maak.

Die *LULU* perspektief en analise gereedskap word ondersoek met 'n klem op praktiese toepassings. Die *LULU* gladstrykers is bekend daarvoor dat hul sein en geraas doeltreffend skei: hulle is idempotent en ko-idempotent. Rye word gladgestryk deur afbeelding op gladheids-klasse; hierdie afbeelding word uitgevoer deur die verwydering, in a konsekwente manier, van blok-pulse. Verder behou hierdie operators lokale orde (hulle is vol-orde-behoudend). Die *dpt* word gedefinieër, sy implikasies ondersoek en 'n lineêre tyd algoritme word bespreek. Daar word gevind dat die *dpt* 'n multi-resolusie maatstaf van grofheid toelaat. Praktiese ry verwerking deur die rekonstruksie van veranderde pulse word ondersoek. Uitbreidings na twee dimensies en 'n tegniek vir die skatting van die standaard afwyking van 'n lukrake verspreiding word ook bespreek. Hierdie gereedskap is gevind as effektief in die analise en verwerking van rye en beelde.

Die *LULU* perspektief is 'n nuttige alternatief tot standaard analise metodes. Die operatore is robuust in die teenwoordigheid van goed-geaarde en impulsiewe geraas. Hulle maak die vinnige ontwerp van gespesialiseerde opsporing en verwerkings algoritmes moontlik, en is baie nuttig vir die skep van geoutomatiseerde data analise oplossings.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Acknowledgments

I would like to thank my supervisor Dr. Carl Rohwer for his continued support and guidance during my studies, and for proofreading this thesis. I also wish to express my gratitude to Prof. Hans Eggers for introducing me to the Physical Mathematical Analysis program, and for the many informative conversations over the years.

The financial assistance of my parents, the National Research Foundation, and the HB and MJ Thom Trust is gratefully acknowledged.

Contents

Introduction

"*We don't see things as they are. We see them as we are.*"
Anais Nin

Populations of micro-organisms can survive immense environmental changes due to a fast evolutionary response made possible by short reproduction time and genetic diversity. With an average generation of about 30 years, humankind on the other hand cannot absorb the pressures of a changing environment through biological evolution alone. However, man has evolved to possess the ability to gain knowledge (i.e. to learn), allowing the changing of behaviour as a response to environmental changes. Through language (which is itself a form of knowledge) and imitation, knowledge is transferred from generation to generation. This knowledge is dynamic; constantly being refined and built upon. Analogous to how only successful organisms survive (due to natural selection), some ideas die out while others survive. We call this evolution of ideas, cultural evolution.

Whereas the currency of biological evolution can be said to be physical resources and the efficient manipulation thereof, one of the most important currencies of cultural evolution is information. Efficient and useful ways of recognizing, manipulating, representing and combining information are cornerstones of cultural evolution.

At first, most uses of information related directly to survival. As the knowledge and infrastructure inherited from previous generations grew, survival became easier. The opportunities for the creation of knowledge not directly linked to survival thereby increased, causing a profound shift in perspective: ideas became the 'food' of man.

The shift from a direct perception of reality to a perception mediated by concepts has not been without problems. In many cases interpretations and representations came to be regarded as reality, which often results in conflicts about which interpretation is the 'true' one. Though this is unfortunately still occurring today, many now recognize the subjectivity of interpretations and the impossibility of establishing absolute truths; the so-called post-modern viewpoint. The language of Mathematics has mostly managed to stay separate from the politics of information by rooting itself in the abstract.

Currently we live in what is called the information age. Technological advancements have provided us with an overabundance of information; too much, in fact, for it all to get personal human attention.

We are almost ready to introduce the subject of this thesis, but first it is useful to highlight a few important points that are implicit in the above discussion:

- Information usage and volume is increasing and its manipulation becoming more important in today's world. The increase in quantity necessitates more automated data analysis tools.
- There is no absolute way to compare the value of different informations (or interpretations). Different mathematical tools (or information processors) are useful under different sets of assumptions and for different purposes.
- New interpretations and representations often provide a fresh eye on old data. In essence, data has no meaning without a suitable perspective. For automated analysis this perspective needs to be explicit (in contrast with unconscious human data analysis).

Data analysis frequently involves the extraction (i.e. recognition) of parts that are important at the expense of parts that are deemed unimportant; in other words, a separation into signal and noise. Many mathematical perspectives exist for performing said separations, however no single technique is a panacea as the definition of signal and noise depends on the purpose of the analysis. The effectiveness of each depends on how well their implicit assumptions mesh with the analysis goals.

In this thesis we cover theory and applications relating to non-linear multi-resolution decompositions based on the *LULU* operators. Our thesis being that they are an useful, natural and computationally efficient alternative in many areas of practical data analysis. As the usefulness of a tool is closely coupled with what one wishes to achieve, we cannot prove this categorically. Instead our aim is to illuminate the *LULU* perspective by stating mathematical consequences, demonstrating the associated tools on constructed and real data, and contrasting it with other perspectives. The idea being that through understanding how this interpretation differs from others and seeing the ideas applied on a variety of representative problems, the usefulness becomes apparent. We shall see that with the addition of impulsive noise, a non-linear approach is naturally more robust. As far as computational efficiency is concerned, worst case calculation time is shown to scale linearly with data size.

Besides a comprehensive reformulation and discussion of *LULU* theory as a practical tool, this document contains the following contributions to the field: a comparison of the bias of *LULU* smoothers, an analysis of the effects of some of the median smoothers in terms of the *LULU* operators, discussion of differences between linear filters and *LULU* smoothers with an example of a hybrid approach, proof of basic consistency of the *dpt* based on L_n and U_n, an algorithm for the discrete pulse transform with average case run-time of order approximately $N^{1.2}$ and modest memory requirements, properties of the discrete pulse transform and methods related to its use (including a technique for edge detection), a test of the accuracy and extension to 2 dimensions of the technique of estimation of standard deviation, and finally three problems are analyzed using the theory and tools from the first part. All proofs contained within are new.

Some of the ideas contained here are similar to those in Mathematical Morphology [15], but followed a different route in its conception. It was born out of practical needs and later strengthened by a strong theoretical structure [9]. These techniques are an additional instrument in the data analyst's toolbox, and is most effective used in conjunction with existing tools.

We will start with the theory on which these practical techniques are built and the tools that they provide for the data analyst in part 1. In part 2 the focus will shift to applying these techniques on a few real-world problems: trend estimation, noise analysis, and image processing.

Part 1

Theory and Tools

CHAPTER 1

The LULU framework

This chapter serves as an introduction into the underlying theory from which the LULU structure emerges. Proofs for theorems in this chapter, that have already been provided by Rohwer [8], were omitted.

The *LULU* operators were primarily motivated by the smoothing of sequences using a well defined set of operations. The smoothing comes as a results of viewing a sequence as the sum of many seperate pulses and then removing the high resolution pulses. A multi-resolution analysis of a sequence that relies on these smoothing operations later emerges along with other helpful tools. Before we can move on to a definition of the operations involved we need to define our theoretical framework in which these exist. In this way we will move from the general to the specific.

1.1. Sequences and norms

We are presenting tools for the analysis of data where the data is in the form of sequences. For our purposes a sequence is just a list of numbers. The data sequence contains that which of interest, the signal, and that which is not currently of interest, the noise.

Definition 1.1. Let $\mathcal{X}$ be the set of all bi-infinite sequences of real numbers:

$$\mathcal{X} = \{x = \{x_i\} : i \in \mathcal{Z}, \quad x_i \in \mathcal{R}\}$$

In some cases the index of a sequence can be directly correlated to some external variable, like time. In other cases there is only a loose connection between variable and index. In all cases this is defined by circumstances and specifics not related to our analysis methods and thus Somebody Else's Problem.

In real problems these sequences are generally finite, but may be extended with zeros on both sides to make them infinite in length. We will also assume that all sequences are in ℓ_1, i.e. $\sum_i |x_i|$ is bounded. Sometimes we may weaken these assumptions, requiring only a sequence in which the total variation $\sum_i |x_i - x_{j-1}|$ is bounded. In other words, the sequence of local differences Δx is in ℓ_1.

We define addition and scalar multiplication for sequences in $\mathcal{X}$, with which $\mathcal{X}$ becomes a vector space.

Definition 1.2. For sequences $x, y \in \mathcal{X}$ and a real number α.

(1) $x + y = x \oplus y = \{x_i + y_i\}$ Addition
(2) $\alpha x = \alpha \odot x = \{\alpha x_i\}$ Scalar Multiplication

Some sequences can be compared with other sequences using a relation, R:

DEFINITION 1.3. The relation R results in a partial order on $\mathcal{X}$, with

$$x \, R \, y \quad \iff \quad x_i \, R \, y_i, \quad \forall i, \quad x, y \in \mathcal{X}$$

and $R \in \{\leq, =, \geq\}$. Note that this relation does not result in a total order as not all pairs of elements, $x, y \in \mathcal{X}$, are generally comparable: sometimes neither (x, y) nor (y, x) is in R.

When dealing with data in multiple dimensions it is often useful to be able to quantify properties about the data in less dimensions in order for two distinct data sets to be compared. There are many ways to do this, some more useful than others. We will define a few that are useful in our analysis methods. The p-norms map a sequence into a single number and have a geometric meaning. The 1-norm of a sequence being the absolute area between that sequence and the zero sequence. The 2-norm is the Euclidean distance between a sequence and the zero sequence. By using the standard p-norms we can make $\ell_1 \subset \mathcal{X}$ into a normed space.

DEFINITION 1.4. We define the usual p-norms, with $p \geq 1$ a real number or $p \to \infty$:

$$\begin{aligned} \|x\|_p &= \left(\sum_i |x|^p \right)^{\frac{1}{p}} \\ \|x\|_\infty &= \sup_i \{|x_i|\} \end{aligned}$$

A measure which will prove useful later on is the total variation operator. The total variation of an infinite sequence exists as long as $\sum_i |x_i - x_{j-1}|$ is bounded; it is a semi-norm for the set of all such sequences. It is a norm for the set of sequences in ℓ_1. The total variation of a sequence turns out to be a natural norm in the *LULU* framework, as we shall see later.

DEFINITION 1.5. The total variation operator, T, sums the local variation of a sequence and is defined by:

$$Tx = \|\Delta x\|_1 = \sum_{-N}^{N} |x_{i+1} - x_i|$$

1.2. Operators

We would like to consider the problem of smoothing a sequence. Before we can consider finding an operation that does what we intend we need to consider the set of all possible operations. Furthermore, the ways these operations can be combined and manipulated must be defined. We consider all mappings of sequences in $\mathcal{X}$ to sequences in $\mathcal{X}$ as operators, and then define the set of these operators:

DEFINITION 1.6. Let $F(X)$ be the set of all operators on $\mathcal{X}$:

$$F(X) = \{A : \mathcal{X} \to \mathcal{X}\}$$

The field of linear filters is well developed and has provided many tools for the data analyst. Fourier analysis, wavelet decompositions and general linear filters have proved their worth in practice over the years.

DEFINITION 1.7. An operator $A \in F(\mathcal{X})$ is linear if $A(x+y) = Ax + Ay$ and $A(\lambda x) = \lambda Ax$ for all $x, y \in \mathcal{X}$ and $\lambda \in \mathcal{R}$.

The left distributivity of linear operators allows one to decompose an operation on a complicated argument into the sum of operations on less complicated arguments. This allows more straight-forward algebraic manipulation. We need to consider a larger class.

The operators which form part of the *LULU* framework are all *non-linear*. It is useful to list the identities and basic operator definitions available for use. The notation $(Ax)_i = a_i$ gives a definition of an operator on sequences in terms of each element of the sequence; i.e. the sequence $[x_i, \ldots, x_j]$ is transformed to $[a_i \ldots a_j]$.

DEFINITION 1.8. For every $\mathrm{A}, \mathrm{B} \in F(\mathcal{X})$ and $x \in \mathcal{X}$:

(1) $(\mathrm{A}+\mathrm{B})\,x = \mathrm{A}x + \mathrm{B}x$	Sum of operators
(2) $Ix = x$	Identity operator
(3) $(Ox)_i = 0$	Zero operator
(4) $(\alpha \mathrm{A})\,x = \alpha\,(\mathrm{A}x), \quad \alpha \in \mathcal{R}$	Scalar associativity
(5) $(\mathrm{AB})\,x = \mathrm{A}\,(\mathrm{B}x)$	Operator composition
(6) $(Ex)_i = x_{i+1}, \quad \forall i$	Shift operator
(7) $Nx = -x$	Negative operator
(8) $(\mathrm{A}+\mathrm{B})\,\mathrm{C} = \mathrm{AC} + \mathrm{BC}$	Right distributivity
(9) $\mathrm{A}^0 = I, \quad \mathrm{A}^{n+1} = \mathrm{AA}^n, \quad n \in \mathcal{Z}$	Operator powers

Right distributivity of the operators (def 1.8.8) follows directly from the definition of addition (def 1.8.1). The commutativity of operator exponents can be proved from the above definitions using standard induction arguments.

THEOREM 1.9. *For $A \in F(\mathcal{X})$, we have that operator exponents commute:*

$$A^m A^n = A^n A^m, \quad n, m \in \mathcal{Z}$$

In the previous section we defined how a relation can be used to define a partial order on $\mathcal{X}$. We can now extend this concept to the set of operators, $F(\mathcal{X})$. This does not form a total order relation as not all operators can be meaningfully compared. For example: neither $N \geq I$ or $N \leq I$ is true in general.

DEFINITION 1.10. For a relation R and two operators in $F(\mathcal{X})$, A and B, we define

$$A \; R \; B \;\text{ to mean }\; (Ax) \; R \; (Bx) \;\text{ for all } x \in \mathcal{X}$$

An operator that preserves partial order relations between sequences is called *syntone,* or *monotone* by Rohwer and Wild [13], and *increasing* by Serra [15].

DEFINITION 1.11. An operator P on $\mathcal{X}$ is syntone (increasing, monotone) if and only if, for all sequences $x, y \in \mathcal{X}$,

$$x \geq y \Rightarrow (Px) \geq (Py)$$

Clearly, any composition of syntone operators is in turn also syntone. One can apply a syntone operator to each side of an inequality without changing the inequality. This is not true for operators in general. The negative operator, for example, reverses the inequality when applied on each side.

DEFINITION 1.12. Any operator, P, is called variation diminishing if:

$$T(Px) \leq T(x)$$

1.3. Smoothers and Separators

It is useful to now limit the set of available operators by choosing properties that we would like our smoothing operators to have [9]. It is, for instance, reasonable to assume that the indexing of a sequence or the measurement scale should not influence the smoothing. The following axioms for a *smoother* formalizes this.

DEFINITION 1.13. An operator P on $\mathcal{X}$ is a *smoother* if it satisfies the following translation and scale invariance axioms:

- $PE = EP$, Translation Invariance: x-axis
- $P(x + c) = P(x) + c$, Translation invariance: y-axis
  for each x, $c \in \mathcal{X}$ and c constant
- $P(\alpha x) = \alpha P(x)$, Scale Invariance
  for each $x \in \mathcal{X}$ and scalar $\alpha \geq 0$

This differs slightly from the smoother axioms as specified in Mallows [3]. The scale invariance axiom of Mallows allows all values of α. This is unnecessarily strict for scale independence. Restricting α to non-negative values allows more general operators like the $LULU$-operators.

DEFINITION 1.14. An operator P is a *separator* if it satisfies the following axioms:

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

Rohwer [9] defines a separator as operator that is idempotent, co-idempotent *and* a smoother. In definition 1.14 above, the requirement for the operator to be a smoother is dropped. Hindsight has shown that it is useful to distinguish between operators that are idempotent and co-idempotent without being smoothers and those that are idempotent, co-idempotent and smoothers [10]. As such it may be expedient to change the definition of a separator to not require adherence to smoother axioms. One can now talk of a separating smoother when both sets of axioms are satisfied.

We will call mapping a sequence using an operator for which the smoother axioms hold, *smoothing* the signal. Likewise, mapping a sequence using an operator for which the separator axioms hold is called *separating* the 'signal' and 'noise' in the sequence (or just *separating* the sequence). The meaning of signal and noise in this case is not determined by some outside standard but according to the interpretation of the chosen operator itself. The practical use of a separator then depends on what basis this interpretation is made. As a trivial example, the identity operator is a separator that sees everything as signal and nothing (the zero sequence) as noise, and is as such not useful, except for theoretical consistency.

One can regard a smoother as a (possibly inefficient) machine that separates signal and noise. To achieve better results one can re-smooth the signal part to see if there is additional noise that was not removed in the first pass. One can also put the noise part back into the smoother to see if it yields more signal. This yields the two-stage smoother cascade shown in figure 1. This two-stage process gives the following separation into signal and noise.

$$\begin{aligned} \text{signal} &= \left(P^2 + P\left(I - P\right)\right) x \\ \text{noise} &= \Big(\left(I - P\right)^2 + \left(I - P\right) P\Big)\, x \end{aligned}$$

An operator that is not idempotent is somewhat deficient at noise extraction, because $P^2 \neq P$ implies that $(I - P)\,P \neq 0$. And thus more noise can be extracted by re-smoothing the outputs of the operator. An operator that is not co-idempotent is deficient at signal

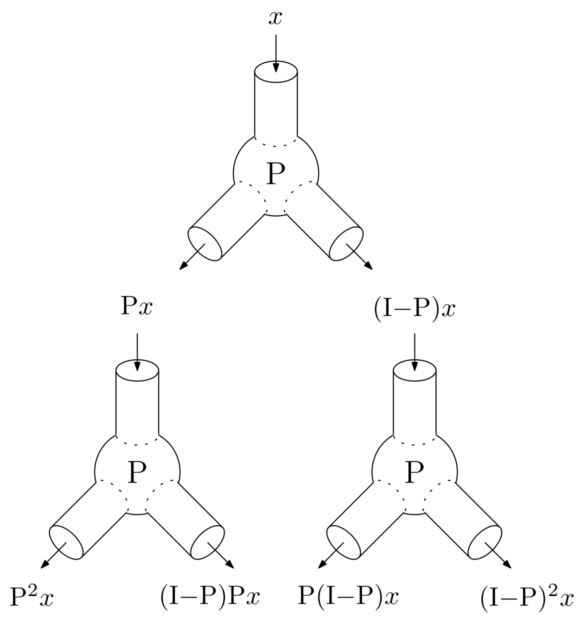


FIGURE 1. Two stage smoother cascade

extraction as $(I-P)^2 \neq (I-P)$ implies $P(I-P) \neq 0$. In other words, more signal can be extracted by re-smoothing the noisy part.

Separators are fully efficient at signal and noise removal (by definition) in the sense that a multiple stage separator cascade will not yield a better separation into signal and noise than using only one stage. This property is needed for an optimally efficient implementation as it will only ever require one application of the smoother. Thus, the separator axioms can be regarded as reasonable demands.

It only makes senses to discriminate between idempotency and co-idempotency for non-linear operators as idempotency and co-idempotency always coincide for linear operators. This may be why the importance of co-idempotence has seldom been recognized in the past [9].

1.4. Smoothing perspectives

Let us recap for a moment. We are finding our way in the dark. An axiomatic framework has emerged. Our target area has been narrowed down slightly from the set of all possible operations on sequences by moving towards some properties and away from others. We are interested in the smoothing of sequences, but what exactly we mean by this has not been defined. There is still a multiplicity of possibilities ahead. Where are we heading? A choice of perspective is needed to light up the way ahead.

To clarify it is helpful to first look at some other related perspectives. In later chapters these perspectives will be revisited to further highlight differences and similarities with respect to the $LULU$ perspective.

The Fourier Transform is one of the most important and ubiquitous analysis tools available. It interprets a function, $f(x)$, as the weighted linear sum of a set of sine and cosine functions of varying frequency. For functions with period 1, we get:

$$f(x) = a_0 + \sum_{n=1}^{\infty} a_n cos(2n\pi x) + \sum_{n=1}^{\infty} b_n sin(2n\pi x).$$

In the Fourier perspective there is an implicit concept of smoothness: a function can be smoothed by removing sinusoids of higher frequency. The degree of smoothing is determined by the chosen cut-off point between low and high frequencies; decreasing the cut-off frequency will result in smoother sequences.

Another perspective is that of the wavelet decomposition: a sequence is decomposed by projections onto a set of orthogonal spaces. We are only interested in pointing out how different analysis tools have different perspectives on smoothing and smoothness, and discuss only the simplest discrete wavelet decomposition: the Haar-wavelet decomposition.

Supposing one has a $N_0 = 2^N$ element sequence, x, which is a sampling of a function, f, such that the coefficients, $\alpha_{k,i}$, describe the sequence in terms of basis functions which are

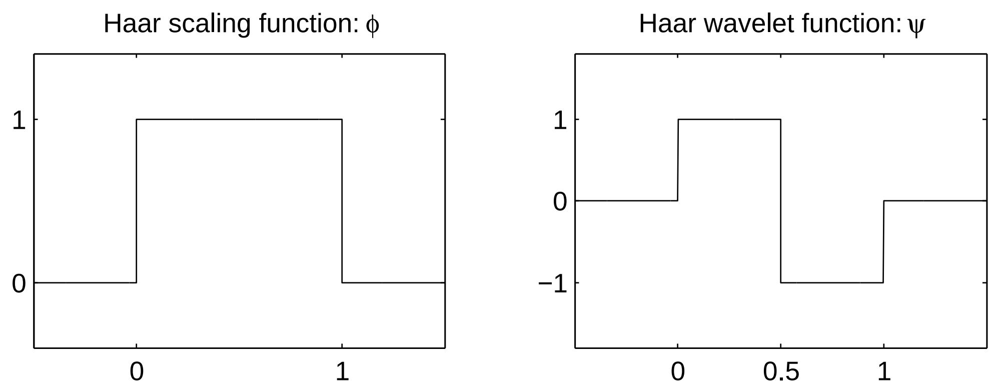


FIGURE 2. Haar wavelet and the associated scaling function

translated versions of the Haar scaling function (figure 2):

$$x_t = \sum_{i=0}^{N_0-1} \alpha_{0,i}\phi\left(t-i\right) \tag{1.1}$$

The coefficients $\alpha_{0,i}$ is then just the sequence x_t. We have that the sequence x is in the space B_0, with the spaces B_j formed by the span of the translated scaling functions at specific scales:

$$B_j = \text{span}\left\{\phi_i : \phi_i(t) = \phi\left(2^{-j}t - i\right)\right\}$$

The Haar wavelet operator, P_k, projects the sequence $x \in B_{k-1}$ to its best least squares estimate in the space B_k. For further smoothing this process is repeated, with each Haar-operator mapping the output of the previous operator into a lower resolution space. Without going into unnecessary details, the best least squares estimate in the next lower subspace is obtained by replacing each pair in the sequence by its average.

The smoothed sequence, $P_k x \in B_k$, at level k then has coefficients in terms of the level $k-1$ coefficients. Notice that the number of coefficients at level k is half that at level $k-1$.

$$\alpha_{k,i} = \frac{1}{2}\left(\alpha_{k-1,2j} + \alpha_{k-1,2j+1}\right) \text{ where } i = 0\ldots N_k - 1 \text{ with } N_k = \frac{N_{k-1}}{2}$$

The coefficients $\alpha_{0,i}$ is either obtained from the sampling of a function or simply come from an arbitrary sequence. The difference between the projected sequence in B_k and the input sequence in B_{k-1} is the part of the sequence peeled off as noise, and can be expressed in terms of the Haar wavelet functions (figure 2), it has coefficients:

$$\begin{aligned}
\beta_{k,i} &= \alpha_{k-1,2i} - \alpha_{k,i} \\
&= \alpha_{k-1,2i} - \frac{1}{2}\alpha_{k-1,2i} - \frac{1}{2}\alpha_{k-1,2i+1} \\
&= \frac{1}{2}\left(\alpha_{k-1,2i} - \alpha_{k-1,2i+1}\right) \text{ where } i = 0\ldots N_k - 1 \text{ with } N_k = \frac{N_{k-1}}{2}
\end{aligned}$$

We now express the smooth and rough part in terms of their basis functions: translated scaled version of the scaling and wavelet functions respectively.

$$\begin{aligned} P_k x(t) \; &= \sum_{i=0}^{N_{k-1}} \alpha_{k,i}\phi\left(2^{-k}t - j\right) \in B_k \\ (I - P_k)\, x(t) &= \sum_{i=0}^{N_{k-1}} \beta_{k,i}\psi\left(2^{-k}t - j\right) \end{aligned}$$

Once again there is an implicit concept of smoothness: a frequency can be associated with the wavelets, $\psi(2^k - \cdot)$; a higher k implies a higher frequency. Although the definition of frequency is different in the Wavelet perspective, this is similar to smoothing with the Fourier Transform as described above. The sequence is smoothed by the removal of the higher frequencies. A concept of resolution is also apparent: the factor by which the scaling and wavelet functions are scaled.

The *LULU* perspective is different. The basic units from which a sequence is constructed are not sinusoids or wavelets, but block pulses.

A discrete sequence is considered as a sum of a collection of block pulses.

$$x = \sum_i \text{blockpulse}_i$$

Definition 1.15. A *block pulse* (hereafter just a *pulse*) is a sequence x with:

$$x_i = \begin{cases} h & i \in [p, p+w-1] \\ 0 & \text{otherwise} \end{cases} = h \sum_{m=1}^{w} \delta_{i,(p+m-1)}$$

It is fully characterized by its position (p), height (h) and width (w). Furthermore, we call it an upwards block pulse if $h > 0$ and a downwards block pulse if $h < 0$.

Figure 3 shows a pulse added to the zero sequence. We see that a pulse has a plateau. While for some applications there may be a reason to exclude negative pulses, there is no reason to do so out of principle. We are working with discrete sequences which implies that the pulse width and position will always be natural numbers. For a sequence of length N (extended before and after with zeros) the possible pulses have the properties:

$$\begin{aligned} \text{position: } & p \in [0, N-1] \subset \mathcal{N} \\ \text{width: } & w \in [1, N] \subset \mathcal{N} \\ \text{height: } & h \in \mathcal{R} \sim \{0\} \end{aligned}$$

At first glance, it seems that the *LULU* perspective is not much different than smoothing with wavelets. The Haar smoothing algorithm, discussed above, smoothes a sequence by the removal of Haar wavelets, which consists of two consecutive pulses (of opposite magnitude). A few negative properties of the Haar operators are investigated to show the need for a fresh perspective.

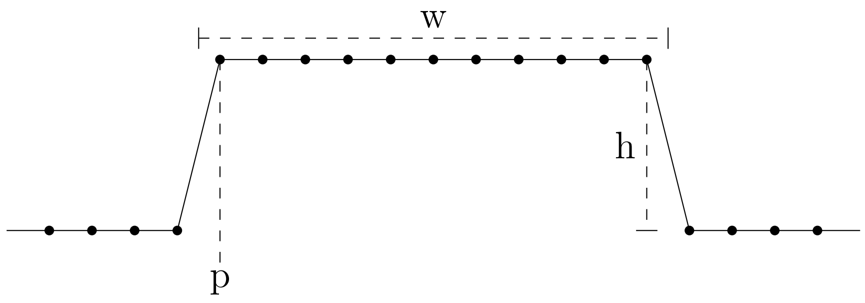


FIGURE 3. Single pulse added to the zero sequence

Since the Haar operators are linear the effect they have on any sequence, x_i with $i = 0 \dots N$, is equal to the sum of the effect it has on the set of sequences $\{\delta_{i,j} x_i : j = 0 \dots N\}$ with $\delta_{i,j}$ the Kronecker-delta function. It is thus illuminating to consider the operator's impulse response (figure 4). Notice that due to the recursive averaging process the impulse energy spreads as the sequence is smoothed. For a sequence with large amplitude impulsive noise this effect may ruin the surrounding signal.

We have stated our basic requirements for a smoother and separator in section 1.3. The Haar wavelet decomposition obeys all the axioms, except translation invariance of the x-axis. Shifting a sequence by one unit left or right changes which elements form the pairs that are averaged. When a structure in the sequence crosses the border between two averaging pairs, the operators are unnecessarily destructive. In practice it is often impossible to align the sequence such that this does not happen. This is the so-called phase problem (figure 5). Notice how one edge of the block pulse is eroded and the other preserved. For certain pulse widths (non powers of two), no amount of shifting the sequence left or right lets the two sides be smoothed similarly.

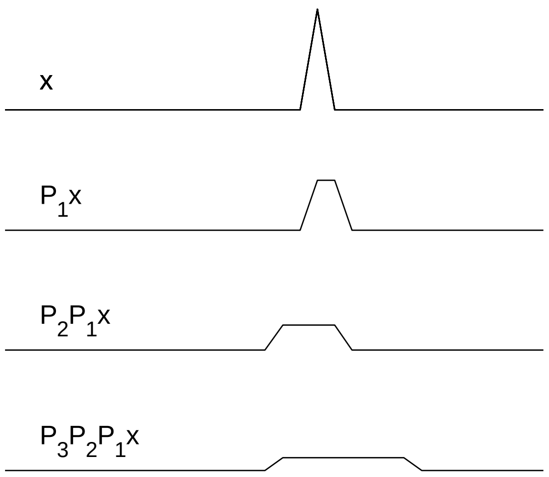


FIGURE 4. Multi-stage Haar smoother impulse response for an impulse at a specific position

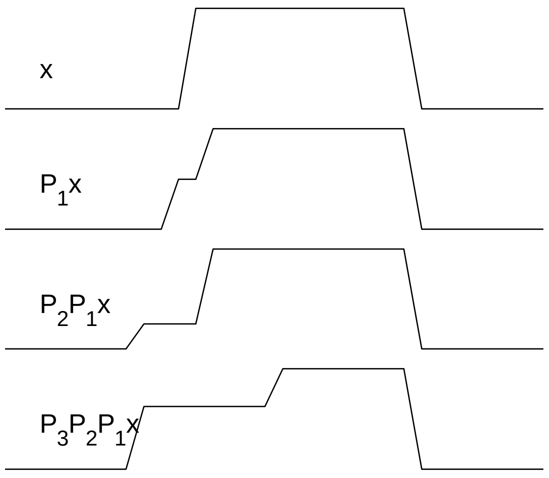


FIGURE 5. The phase problem: one edge of the block pulse is preserved and the other is eroded.

We return to the *LULU* perspective and see what choices are made to avoid the above problems. Choosing *LULU* operators as smoothers (def 1.13), translation invariance is guaranteed and the phase problem avoided.

Linear filters will generally cause a spreading of impulse energy as these filters replace each sequence element with a weighted average of its neighbours [4]. For sequences with low-amplitude unbiased noise this approach tends to work well. For more general signals with possible high-amplitude discontinuities and noise this can lead to unnecessarily large distortions. The *LULU* operators are all non-linear and as a result do not automatically suffer from these drawbacks. They are also separators and smoothers.

To be more specific, the *LULU* operators are all rank based selectors and compositions (we will not use the term concatenation used by Tukey) of these. For a sequence smoothed by a selector, any element in the smoothed sequence must be equal to an element in the input sequence. [9, 4]

DEFINITION 1.16. A rank based selector S maps a sequence $x \in \mathcal{X}$ onto Sx such that every $(Sx)_i = x_k \in W_i$ where W_i is a window of points including x_i. The element selected from the window, x_k , is chosen based on relative ranks of all the points in the window. The number of elements in the window is called the *support* of the operator.

As with the wavelet decomposition a concept of resolution can be defined. In the case of the *LULU* perspective, the width of the pulse is a natural measure of resolution. A narrower pulse is of higher resolution:

$$\text{pulse resolution} \propto \frac{1}{\text{pulse width}}$$

Analogous to the Fourier and Wavelet perspectives, where the high frequency components can be removed to smooth a function, *LULU* smoothing is based on the removal of the higher resolution (narrower) pulses from a sequence. The degree of smoothing is determined by the maximum width of pulses removed. We interpret the effect of the *LULU* operators as pulse removals.

Definition 1.17. A pulse removal is a process whereby a sequence, x, is separated into two sequences, s and n, where $x = s + n$. The sequence n must consist of a single pulse as per definition 1.15 and can thus be characterized by a width, position and height. An operator, P, is a pulse remover if it removes a set of pulses, $\{n_i\}$, from an input sequence x. Then $Px = x - n$ where $n = \sum_i n_i$. Each pulse n_i is characterized by a width, position and height.

1.5. Local Monotonicity

The Fourier and Wavelet perspectives implicitly define measures of smoothness. We want to highlight the related concept in the *LULU* perspective. The concept of local monotonicity (see def. 1.18) is used to classify sequences into different smoothness classes. This classification is done based on the relative local ordering of the elements in the sequence and does not depend on the absolute magnitudes of the elements. We shall see later that the smoothness class of a sequence and the narrowest pulse that are removed by the *LULU* smoothers are related.

A monotone sequence is either non-increasing or non-decreasing. In ℓ_1 the only sequence that is monotone is the zero sequence. This concept of monotonicity is extended to differentiate between global and local monotonicity. We define local monotonicity by:

Definition 1.18. A sequence $x \in \mathcal{X}$ is n-monotone if and only if $\left\{ \begin{array}{cccc} x_i, & x_{i+1}, & \ldots, & x_{i+n+1} \end{array} \right\}$ is monotone for each i.

Definition 1.19. The set of all sequences in $\mathcal{X}$ that are n-monotone is called $\mathcal{M}_n$.

Clearly all sequences that are $n+1$-monotone are also n-monotone. All sequences are at least 0-monotone. We can say that the sets of monotone sequences, $\mathcal{M}_n$, nest (figure 6) such that $\mathcal{M}_{n+1} \subset \mathcal{M}_n$. These sets are our smoothness classes which forms part of the strategy to classify the multi-resolution smoothness of sequences. Standard (globally) monotone sequences can also be said to be ∞-monotone according to definition 1.18.

For a sequence, x, consider the sequence of local differences: $(\Delta x)_i = x_{i+1} - x_i$. For globally monotone sequences the sequence is either non-increasing or non-decreasing, i.e. the local differences are not allowed to change between negative and positive slope. Thus either all $(\Delta x)_i \geq 0$ or all$(\Delta x)_i \leq 0$. In contrast to this, locally monotone sequences can have local differences of any sign as long as there is a constant section of sufficient length between sections of opposite slope. For any sequence $x \in \mathcal{M}_n$, if $(\Delta x)_j (\Delta x)_k < 0$ with $(\Delta x)_i = x_{i+1} - x_i$ then

$$|j-k| > n \text{ and}$$
$$(\Delta x)_i = 0 \text{ for } i \in (j,k)$$

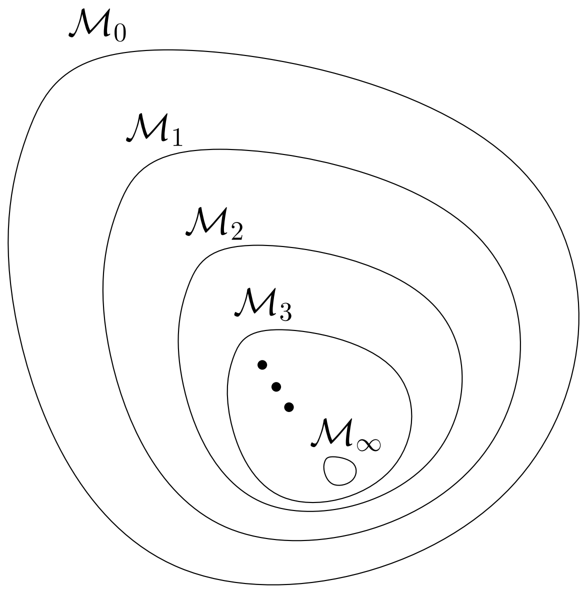


FIGURE 6. Nesting of monotonicity sets

In chapter 2 we shall see how the local monotonicity classes, $\mathcal{M}_n$, are related to some non-*LULU* operators. We will also see that the *LULU* operators map sequences onto these classes. While this mapping cannot be a projection as the operators involved are not linear, they are closer to projections than what one would expect. This will pave the way for creating a multi-resolution decomposition of a sequence: a sequence is split into different sequences that lie in different monotonicity classes, starting from the roughest $\mathcal{M}_0$ to the smoothest $\mathcal{M}_\infty$ (which only contains the zero sequence when in ℓ_1). Using a norm one can compare the total sequence 'energy' in the different resolution levels. These type of decompositions will be discussed in detail in chapters 3 and 4.

It is not immediately clear how the concept of local monotonicity relates to our view of a sequence as the sum of a set of pulses. Before we can answer this question we will need to split the concept of local monotonicity into upwards-monotonicity and downwards-monotonicity.

DEFINITION 1.20. With $k \geq 1$, we call any segment $\left\{ x_i, \quad x_{i+1}, \quad \dots, \quad x_{i+k+1} \right\}$ of a sequence $x \in \mathcal{X}$ a k-upwards arc (also called a cup) if $x_i > x_{i+1} = \cdots = x_{i+k} < x_{i+k+1}$. A k-downwards arc (also called a cap) exists similarly if $x_i < x_{i+1} = \cdots = x_{i+k} > x_{i+k+1}$. The sequence x is then upwards n-monotone (resp. downwards$n$-monotone) if for every k-upwards arcs (resp. k-downwards arcs) it contains, we have $k \geq n+1$. We say that a n-arc has width n, referring to the length of the constant region.

$\mathcal{M}_n^+$ is the set of all upward n-monotone sequences. $\mathcal{M}_n^-$ is the set of all downward n-monotone sequences.

DEFINITION 1.21. A signal is n-monotone if and only if it is both upwards n-monotone and downwards n-monotone:

$$x \in \mathcal{M}_a^+,\ x \in \mathcal{M}_b^- \quad \Rightarrow x \in \mathcal{M}_{\min\{a,b\}}$$

The concepts of upward and downward monotonicity classify sequences based on the minimum width of the upward and downward arcs present in the sequence. Figure 7 shows an example of an n-upwards and n-downwards arc segment. In figure 8 examples of sequences that lie in different monotonicity classes are given. The horizontal and vertical location of the sequences in the grid is their degree of upward and downward monotonicity respectively. The minimum length of all the arcs present in a sequence determines the degree of monotonicity (as per definition 1.21) and thus the sequences on the diagonal lie in $\mathcal{M}_0$, $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ respectively.

In the *LULU* perspective pulses are obtained by the removal of arcs from a sequence. The existence of an n-upwards or n-downwards arc in a sequence x points to the existence of a downwards or upwards pulse of width n respectively (at that position).

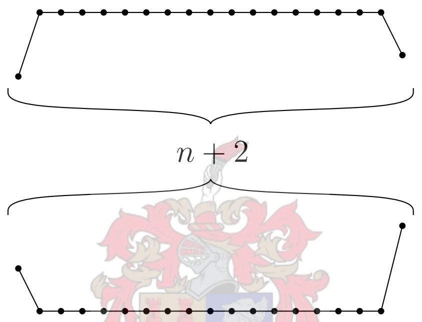


FIGURE 7. A sample n-downwards arc (top) and n-upwards arc (bottom)

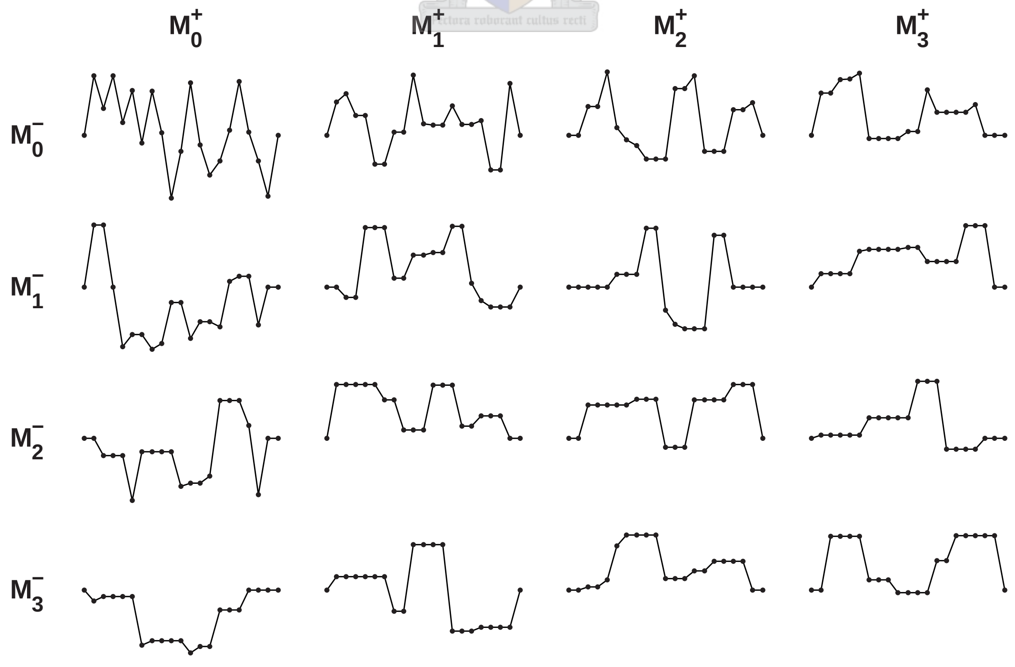


FIGURE 8. Examples of sequences in the different monotonicity classes. The degree of local monotonicity is equal to the minimum arc length.

Definition 1.22. Let $\{x_i, \dots, x_{i+k+1}\}$ be an k-arc. By definition 1.20 we have that $x_i > x_{i+1} = \dots = x_{i+k} < x_{i+k+1}$ (or $x_i < x_{i+1} = \dots = x_{i+k} > x_{i+k+1}$). An arc is removed by setting the valley (or plateau) values $\{x_{i+1}, \dots, x_{i+k}\}$ equal to the neighbouring element that is closest in value. Let x^* be the sequence x but with the arc in question removed, then

$$x_j^* = \begin{cases} x_j & \text{if } k \notin [i+1, i+k] \\ x_i & \text{if } k \in [i+1, i+k] \text{ and } |x_j - x_i| \le |x_j - x_{i+k+1}| \\ x_{i+k+1} & \text{otherwise} \end{cases}$$

Using this definition an arc removal changes the sequence as little is possible while still removing the arc completely. The only parts of the sequence that changes is the valley or plateau part of the arc, which gets replaced by a neighbouring value (a selection operation). The difference sequence $x - x^*$ is then a pulse of width k according to definition 1.15.

It is important to realize that the removal of an arc possibly has side-effects. Another wider arc might be created, or an neighbouring arc can change or be destroyed. As a result not all all arcs present in a sequence will result in a pulse removal. We illustrate these effects with an example. In figure 9 there is a sequence with two downwards arcs and one upward arc. The effect of removing each of these arcs using definition 1.22 is shown. If the first arc is removed, the second arc is destroyed as a side effect. The removal of the second arc destroys both the first and third arc and creates a new width 7 downwards arc. Removing the third arc changes the right edge of the second arc.

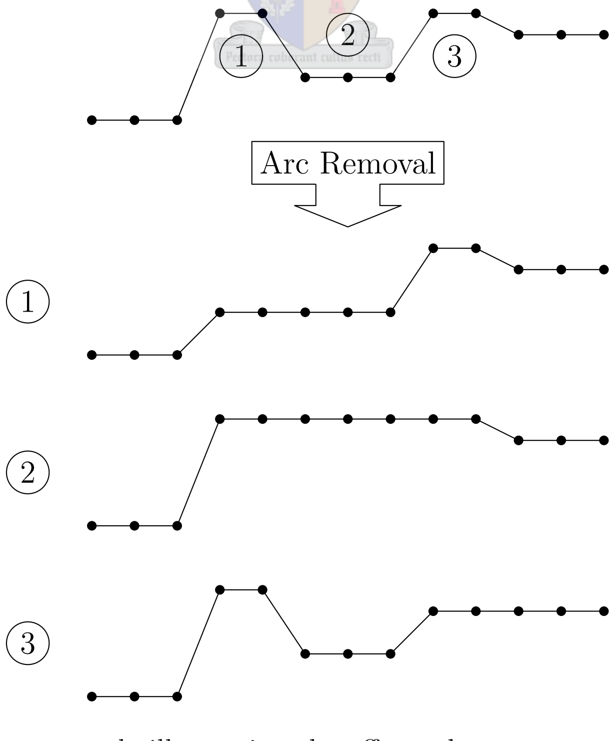


Figure 9. An example illustrating the effects that arc removals can have on other arcs in the sequence.

Neighbouring arcs can only be destroyed or changed if the arc and its neighbour is a downwards and upwards arc that overlaps by two elements. For example, this will occur if we have two arcs $\{x_i, \ldots, x_{i+k+1}\}$ and $\{x_j, \ldots, x_{j+m+1}\}$ such that $j = i+k$ or $j+m+1 = i+1$. A new upward arc (or downward arc) is created if $x_i > x_{i+1}$ and $x_{i+k} < x_{i+k+1}$ (or $x_i < x_{i+1}$ and $x_{i+k} > x_{i+k+1}$) and the result of an arc removal is that $x_{i+1} = \ldots = x_{i+k}$.

We see that the order in which arcs are removed affects what other arcs can be removed. In the next section we will see that the *LULU* operators overcome this by implicitly prioritizing the arcs.

We have discussed how some arcs are only uncovered when others are removed, but these are always wider than the arc whose removal caused its appearance. Therefore the local monotonicity class gives a hard limit on the minimum width of arcs that can be removed: a sequence in $\mathcal{M}_n$ has no arcs (and therefore no pulses) of width n and smaller left for removal. The upwards and downwards monotonicity classes have a similar role, except that they only recognize upward and downward arcs respectively.

The above discussion makes it clear how the concept of local monotonicity relates to the size of pulses that can be removed from a sequence. We have stated that a sequence is smoothed by the removal of higher resolution (narrower) pulses. The local monotonicity class then gives the degree to which this has been accomplished and is thus related to the smoothness of a sequence. We call the monotonicity classes our *smoothness classes*.

1.6. The LULU operators

The *LULU* framework and its ramifications have now been discussed sufficiently for us to move on to the specific *LULU* operators. First the elementary building blocks are defined. These are the protons and electrons with which the neutral atoms (the basic operators) can be built [12] .

Definition 1.23. The elementary operators $\bigwedge$ and $\bigvee$ on $\mathcal{X}$ are:

(1) $(\bigwedge x)_i = min\,\{x_{i-1},\, x_i\}$ erosion
(2) $(\bigvee x)_i = max\,\{x_i,\, x_{i+1}\}$ dilation

It is obvious from definition 1.23 that there exists an asymmetry in the definition of the elementary operators. This asymmetry could easily have been reversed by letting $(\bigwedge x)_i$ be equal to $min\,\{x_i,\, x_{i+1}\}$ and making the corresponding change in $\bigvee$. This is not important as this asymmetry will fall away when we create our basic operators from these elementary building blocks.

Now, let us see what these elementary operators actually do. They are selectors, therefore only values in the input sequence are allowed in the output sequence.

The dilation operator $\bigvee$ widens downward arcs (caps) to the left and narrows upward arcs (cups) from the right. Similarly, the erosion operator $\bigwedge$ widens upward arcs (cups) to the right and narrows downward arcs (caps) from the left. This is visible in figure 10. There are two important effects that these operators have on arcs present in a sequences. First, arcs of unit width are either widened or completely removed from the sequence by these operators. Second, no new arcs can be created. We can call these operators arc-modifiers, as they only change the width of the arcs present in a sequences. These operators are not idempotent as repeated applications will continue to widen some and narrow other arcs in a sequence. The effects of these operators are stated without proof as we will later prove the effect of compositions of these from first principles.

The elementary operators correspond to the erosion and dilation operations common in Mathematical Morphology with a one dimensional structuring element two elements wide [15].

THEOREM 1.24. *Properties of elementary operators. [13]*

(1) $\bigwedge$ *and* $\bigvee$ *are syntone.*
(2) $\bigwedge^m \leq I \leq \bigvee^m$ *for all* $m \geq 0$
(3) $\bigvee^m \bigwedge^m \leq \ldots \leq \bigvee \bigwedge \leq I \leq \bigwedge \bigvee \leq \ldots \leq \bigwedge^m \bigvee^m$ *for all* $m \geq 0$

The syntoneness of the operators implies that the partial ordering of sequences relative to each other will not be destroyed by the application of these operators.

THEOREM 1.25.

(1) $\bigvee^m \bigwedge^m \bigvee^m = \bigvee^m$ *and* $\bigwedge^m \bigvee^m \bigwedge^m = \bigwedge^m$
(2) $(\bigvee^m \bigwedge^m)^2 = \bigvee^m \bigwedge^m$ *and* $(\bigwedge^m \bigvee^m)^2 = \bigwedge^m \bigvee^m$

The following is a direct result of this.

COROLLARY 1.26. $\bigwedge^m$ *and* $\bigvee^m$ *form a Galois connection*

First we define the set of operators that consist of compositions of the elementary operators. Because we are ultimately interested in finding separators as per definition 1.14, we define

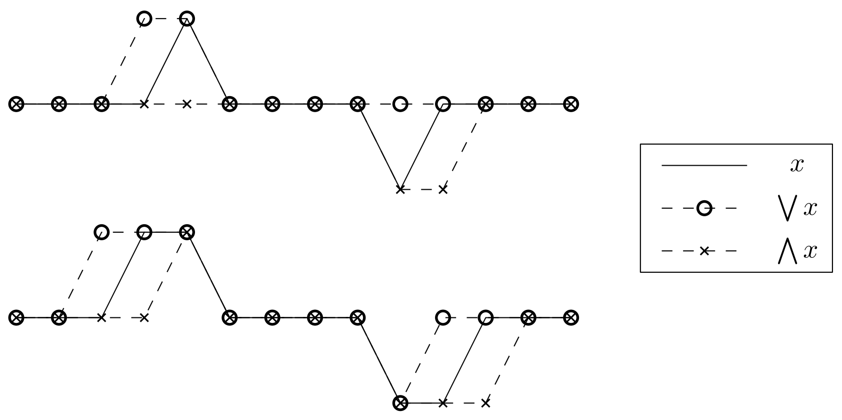


FIGURE 10. Effect of elementary operators on upward and downward pulses of width 1 and 2.

the subset of this set which contains only the idempotent operators. We know that this set is smaller than its superset because the elementary operators, $\bigwedge$ and $\bigvee$, are not idempotent.

Definition 1.27. $\mathcal{C}$ is the set of operators on $\mathcal{X}$ consisting of compositions of the elementary operators $\bigwedge$ and $\bigvee$:

$$\mathcal{C} = \left\{A \,:\, A \text{ is a composition of the operators } \bigwedge \text{ and } \bigvee\right\}$$

Definition 1.28. $\mathcal{L} \subset \mathcal{C}$ is the set of operators in $\mathcal{C}$ that are idempotent.

$$\mathcal{L} = \left\{A \,:\, A \in \mathcal{C}, \quad A^2 = A\right\}$$

We will now define our basic operators. These will form the building blocks with which we create all the subsequent smoothers in the $LULU$ framework

Definition 1.29. The basic operators are

(1) $L_n = \bigvee^n \bigwedge^n$
(2) $U_n = \bigwedge^n \bigvee^n$

In theorem 1.25, we saw that the basic operators are idempotent, and thus in $\mathcal{L}$. The asymmetry in our definition of elementary operators did not carry over to our choice of basic operators as they are asymmetric only in the sense that $L_nN = NU_n$ for all n and are thus 'duals' of each other.

Definition 1.30. An operator A is a dual of an operator B if $AN = NB$.

When a result is proven for one of a pair of dual operators then a related property for the other is also true. For example, by theorem 1.24, we know that

$$U_nx \geq Ix \tag{1.2}$$

for any sequence x. The corresponding rule for L_n is now proven by duality. Apply the negative operator on both sides of equation 1.2 to obtain $NU_nx = L_nNx \leq NIx = INx$. This is valid for any sequence x, therefore it is also valid for $y = Nx$. Then $L_ny \leq Iy$ for any sequence y.

Recall that the elementary operators change the width of arcs in a sequence. Let us consider the operator $L_n = \bigvee^n \bigwedge^n$. After application of $\bigwedge^n$ all upward arcs will be of width $n+1$ or larger. All downward arcs of width n and smaller are filtered from the sequence and the rest are narrower by n units. Following this by $\bigvee^n$ will restore the upward arcs and the remaining downward arcs to their original sizes. The downward arcs (caps) of width n and smaller can not be restored as they were destroyed. This may seem confusing as some of the arcs that are removed are partly obscured by smaller arcs. The following theorem, with its corollary, permits a more illuminating explanation.

Theorem 1.31. *$U_{n+1}U_n = U_{n+1}$ and $L_{n+1}L_n = L_{n+1}$*

Proof. $U_{n+1}U_n = \bigwedge^{n+1}\bigvee^{n+1}\bigwedge^n\bigvee^n = \bigwedge^{n+1}\bigvee(\bigvee^n\bigwedge^n\bigvee^n) = \bigwedge^{n+1}\bigvee^{n+1} = U_{n+1}$ using corollary 1.25. A similar proof exists for L. $\square$

Corollary 1.32. *$U_n = U_nU_{n-1}\ldots U_1$ and $L_n = L_nL_{n-1}\ldots L_1$*

We know that the operator L_n has the same effect as $L_nL_{n-1}\ldots L_1$. Consider any sequence, $x \in \mathcal{M}_0$. Tthe arcs present have a width of at least 1. Performing L_1 will widen all upward arcs by 1 unit before restoring them to their original size. All downward arcs will be made narrower by 1 unit before they are restored to their original size. The downward arcs of width 1 cannot be restored since they were destroyed before they could be restored. Therefore the output sequence will not have any downward arcs remaining of width 1. I.e. the output sequence is in $\mathcal{M}_1^-$. The plateau value of each unit width downward arc is replaced by the maximum of the two neighbouring values. The U_n operator replaces each unit width upward arc with the minimum of the two neighbouring values. The following two theorems play a key role in our understanding of L_n and U_n as pulse removers and the effect they have on the smoothness class of a sequence. Arcs are removed as specified by definition 1.22.

Theorem 1.33. *For any sequence $x \in \mathcal{M}_{n-1}^-$, we have $L_nx \in \mathcal{M}_n^-$. All n-downwards arcs in sequence x are removed:*

$$L_n[\ \cdots,\ \ *,\ \ a\ \ \underbrace{b\ldots b}_{n\ times}\ \ c,\ \ *,\ \ \cdots\ ] = [\ \cdots,\ \ *,\ \ a\ \ \underbrace{b'\ldots b'}_{n\ times}\ \ c\ \ ,*,\ \ \cdots\ ]$$
$$\text{where } b > a, c \text{ and } b' = \max\{a, c\}$$

Other structures in the sequence are left unchanged.

Proof. L_n is a smoother, therefore we can shift the indices of vector x without changing the smoothed result. We will use the notation $\leq k$ to mean a number less than or equal to k when the specific value itself is not important.

From the definition of the elementary and basic operators we get:

$$\begin{aligned}(L_nx)_i &= \left(\bigvee^n\bigwedge^n x\right)_i \\ &= \max\{\min\{x_{i-n}, \ldots x_i\}, \min\{x_{i-n+1}, \ldots x_{i+1}\}, \ldots, \min\{x_i, \ldots, x_{i+n}\}\}\end{aligned}$$

For any monotone section, $\left[\ x_i,\ \ x_{i+1},\ \ \ldots,\ \ x_{i+m}\ \right]$, shift the sequence such that $i = 0$. Suppose that $x_m \geq x_0$ (the other case is similar). Because $x \in \mathcal{M}_{n-1}^-$ all downward arcs have constant part of length $\geq n$. Now assume that there exists no integer $k \in [0, m]$ such that $x_k - x_{k-1} > 0$ and $x_{k+n} - x_{k+n-1} < 0$. In other words, we do not allow the monotone section to contain part of the plateau section of a n-downwards arcs. We then have, $x_j \geq x_m$ for $m \leq j < 1 + m + n$.

We see that x_i with $i = 0\ldots m$ remains unchanged after application of L_n:

$$\begin{aligned}(L_n x)_i &= \max\left\{\min\left\{x_{i-n}, \dots x_i\right\}, \min\left\{x_{i-n+1}, \dots x_{i+1}\right\}, \dots, \min\left\{x_i, \dots, x_{i+n}\right\}\right\} \\ &= \max\{\underbrace{\leq x_i, \dots, \leq x_i}_{n \text{ times}}, x_i\} \\ &= x_i\end{aligned}$$

Now we look at the effect of L_n on arcs. For every m-arc $\begin{bmatrix} x_i, & x_{i+1}, & \dots, & x_{i+m+1} \end{bmatrix}$ shift x such that $i = 0$. We have:

$$\begin{aligned}x_0 &= a \\ x_i &= b \text{ for } i = 1 \dots m \\ x_{m+1} &= c\end{aligned}$$

First suppose it is an upwards arc. Then $b < a, c$. We know that $x \in \mathcal{M}^-_{n-1}$, thus the following must be true:

$$\begin{aligned}&x_j \geq a \text{ when } -n < j < 0, \text{ and} \\ &x_j \geq c \text{ when } m + 1 < j < m + n + 1\end{aligned}$$

Because of this and the fact that $b < a, c$ is in every of the windows we want to calculate the minimum of, we get for $i = 1 \dots m$:

$$\begin{aligned}(L_n x)_i &= \max\left\{\min\left\{x_{i-n}, \dots x_i\right\}, \min\left\{x_{i-n+1}, \dots x_{i+1}\right\}, \dots, \min\left\{x_i, \dots, x_{i+n}\right\}\right\} \\ &= b = x_i\end{aligned}$$

We see that upwards arc are not changed by L_n.

Supposing the arc is a downwards arc, we have $b > a, c$. We also know that $x \in \mathcal{M}^-_{n-1}$, therefore $m \geq n$. The endpoints of the arc are either part of an upwards arc or lies inside a monotone section, either way it has already been proven that they are not changed by L_n.

Now we see what happens, after smoothing with L_n, with the values of the arc plateau. For $i = 1 \dots m$ and $m = n$ we get:

$$\begin{aligned}(L_n x)_i &= \max\left\{\min\left\{x_{i-n}, \dots x_i\right\}, \min\left\{x_{i-n+1}, \dots x_{i+1}\right\}, \dots, \min\left\{x_i, \dots, x_{i+n}\right\}\right\} \\ &= \max\{\underbrace{\leq a, \dots, \leq a}_{n-i \text{ times}}, a, c, \underbrace{\leq c, \dots, \leq c}_{i-1 \text{ times}}\} \\ &= \max\left\{a, c\right\}\end{aligned}$$

When $m > n$, at least one of the sets $\left\{x_{i-n}, \dots x_i\right\}, \left\{x_{i-n+1}, \dots x_{i+1}\right\}, \dots, \left\{x_i, \dots, x_{i+n}\right\}$ contains only elements in the range $x_1 \dots x_m$ and thus the minimum of that set is b. The minimum of the other sets are either $\leq a$ or $\leq b$. Because $b \geq a, c$ we then get, for each $i = 1 \dots m$:

$$(L_n x)_i = \max\left\{\min\left\{x_{i-n}, \dots x_i\right\}, \min\left\{x_{i-n+1}, \dots x_{i+1}\right\}, \dots, \min\left\{x_i, \dots, x_{i+n}\right\}\right\}$$

$$= b = x_i$$

We see that a downwards arc is left unchanged when it is wider than n. When $m = n$ the constant segment is replaced by the maximum value of the endpoints of the arc.

Before application of L_n we have $x \in \mathcal{M}_{n-1}^-$, therefore all downwards arcs are of width n or wider. All n-downward arcs are removed by setting the constant section to one of the endpoints' values, thus creating a new constant section of length $\geq n+1$. This cannot create a new arc of width $\leq n$. Therefore, after every n-downwards arc is removed all the remaining downwards arcs have length $\geq n+1$, and thus $L_n x \in \mathcal{M}_n^-$. $\square$

Since U_n is the dual of L_n, its effect on sequences can be proven in a similar way.

THEOREM 1.34. *For any sequence $x \in \mathcal{M}_{n-1}^+$, we have $U_n x \in \mathcal{M}_n^+$. All n-upwards arcs in sequence x are removed:*

$$U_n[\ \cdots,\quad *,\quad a\quad \underbrace{b\ldots b}_{n\ times}\quad c,\quad *,\quad \cdots\ ] = [\ \cdots,\quad *,\quad a\quad \underbrace{b'\ldots b'}_{n\ times}\quad c\quad ,*,\quad \cdots\ ]$$

$$\text{where } b < a, c \text{ and } b' = \min\{a, c\}$$

Other structures in the sequence are left unchanged.

From the above two theorems we see that the basic operators are pulse removers. These pulses are separated by sections of zero value of length at least 1, because for any sequence $\left[\ \ldots\quad a\quad b\ldots b\quad c\ldots c\quad d\quad \ldots\ \right]$ one cannot have $b > a, c$ and $c > b, d$ both true.

The sequence $(I - L_1)\,x$ will consist of upward pulses of width 1. Now L_2 is applied. We do not need to look at the individual effects of $\bigvee^2$ and $\bigwedge^2$ because we have already proven the effect of L_n for any $n > 0$. This smoother removes all downward arcs of width 2. The sequence $(L_1 - L_2)_n$ contains only upward pulses of width 2. The sequence $L_2 x$ is then in $\mathcal{M}_2^-$.

In this way all upward pulses up to width n are removed by L_n. The argument for U_n is similar, except that it removes downward pulses of length n and smaller. We now have our *LULU* pulse removers. In figure 11 a sequence is smoothed with L_4. Although this is not apparent from the noise sequence $(I - L_4)\,x$ all upward pulse of width 1 to 4 have been removed.

To explicitly show the pulses of each width that were removed by the smoothing process we can split $(I - L_4)\,x$ into four sequences. We do so in figure 12. All removed pulses of width i will be in the sequence $(L_{i-1} - L_i)\,x$. The total noise removed by L_4 is equal to the sum of all these pulses: $(I - L_4)\,x = \sum_{i=1}^{4} (L_{i-1} - L_i)\,x$.

We have seen that the basic operators remove arcs of a specific width if all the smaller arcs have already been removed. The following corollary regarding the range of the basic operators is a direct result of this and corollary 1.32.

COROLLARY 1.35. *If x is any sequence, i.e. $x \in \mathcal{M}_0$. Then $L_n x \in \mathcal{M}_n^-$ and $U_n x \in \mathcal{M}_n^+$.*

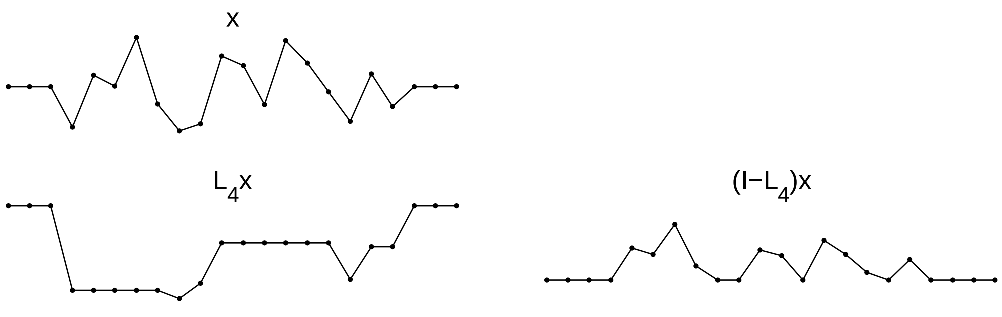


FIGURE 11. Smoothing a sequence with L_4.

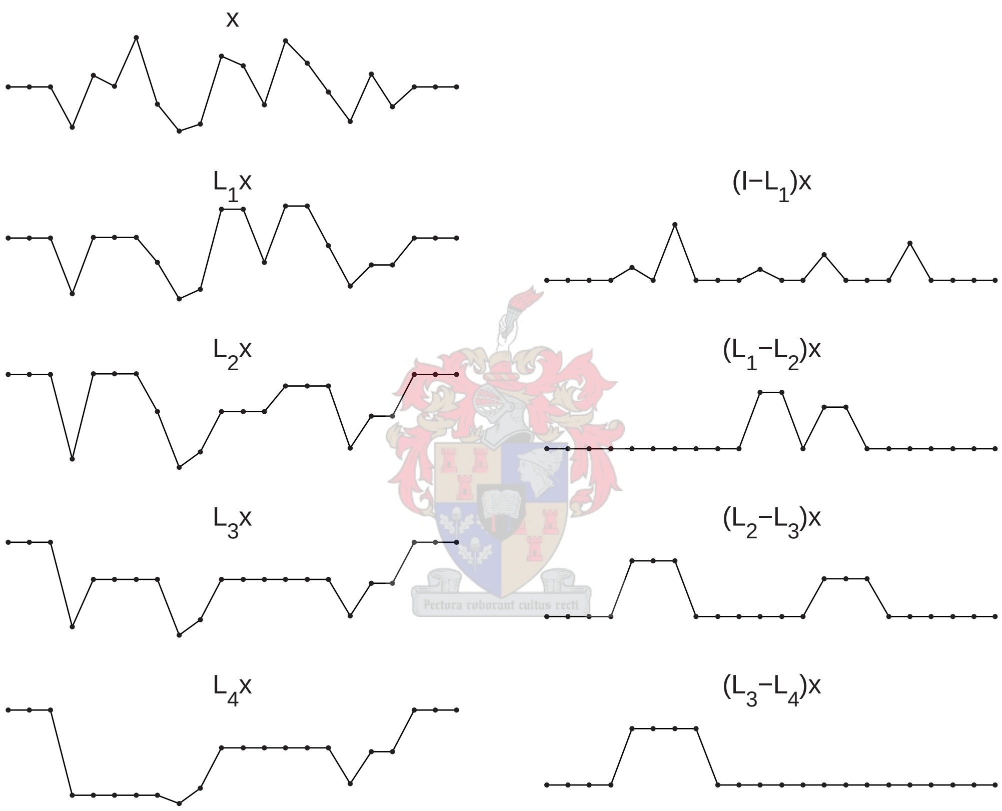


FIGURE 12. Smoothing with L_4 is the removal of upward pulses of width up to 4.

Rohwer and Wild [13] show that all the operators in $\mathcal{C}$ can be reduced to one of four types. Furthermore, if the operator is idempotent, it can always be expressed as a product of the basic operators (the pulse removers). The basic operators and compositions of them will be investigated further in the next chapter.

THEOREM 1.36. *All operators, $A : \mathcal{X} \to \mathcal{X}$, in $\mathcal{C}$ can be reduced to a product of one of the following four types. While all idempotent operators in $\mathcal{C}$, i.e. in $\mathcal{L}$, can only be of type 1 or 3.*

(1) $A = U_{n_1} L_{m_1} U_{n_2} L_{m_2} \dots$

(2) $A = \bigvee^s \bigwedge^t U_{n_1} L_{m_1} U_{n_2} L_{m_2} \ldots$
(3) $A = L_{m_1} U_{n_1} L_{m_2} U_{n_2} \ldots$
(4) $A = \bigwedge^s \bigvee^t L_{m_1} U_{n_1} L_{m_2} U_{n_2} \ldots$

with

$$(m_1 > m_2 > \ldots)\,,\ (n_1 > n_2 > \ldots)\ \textit{and}\ (0 \le s < t) \tag{1.3}$$

The idempotent compositions of the elementary operators, $\bigvee$ and $\bigwedge$, are always in one of two standard forms, and are exactly those compositions where the number of each of the two elementary operators are equal. Accordingly, they consist of compositions alternating between the L_n and U_n operators (hence the name *LULU* theory!).

We can reinterpret the two standard forms of idempotent compositions using the pulse perspective. Expanding the two standard forms as given in theorem 1.36 using theorem 1.31 one gets:

Corollary 1.37. *The operators, $A : \mathcal{X} \to \mathcal{X}$, in $\mathcal{L}$ (i.e. the idempotent operators in $\mathcal{C}$) can be written in one of two possible forms:*

(1) $A = U_{n_1} \ldots U_{n_2+1} L_{m_1} \ldots L_{m_2+1} U_{n_2} \ldots U_{n_3+1} L_{m_2} \ldots L_{m_3+1} \ldots$
(2) $A = L_{m_1} \ldots L_{m_2+1} U_{n_1} \ldots U_{n_2+1} L_{m_2} \ldots L_{m_3+1} U_{n_2} \ldots U_{n_3+1} \ldots$

with $(m_1 > m_2 > \ldots)\,,\ (n_1 > n_2 > \ldots)$.

Using this result one can write any idempotent composition of the elementary operators as the product of $n_1 + m_1$ of the basic operators. This expansion is an unique simple form of an operator in $\mathcal{L}$. Furthermore, the expanded equation now explicitly gives the priorities with which different sized pulses are detected and removed during the smoothing of a signal.

Example 1.38. One can expand the idempotent operator

$$A = U_7 L_5 U_2 L_3$$

uniquely using corollary 1.37 into the composition of 12 basic operators:

$$A = U_7 U_6 U_5 U_4 U_3 L_5 L_4 U_2 U_1 L_3 L_2 L_1$$

The pulse removal priorities can now be read right-to-left: first upward pulses of width 1, 2 and 3 are removed followed by downward pulses of width 1 and 2 and so on. It will be seen later that the expanded form is not necessarily more expensive computationally.

1.7. Trend preservation

The *LULU* operators exhibit strong trend preservation properties [9].

DEFINITION 1.39. An operator A on $\mathcal{X}$ is *neighbour trend preserving* (*ntp*) if, for each sequence $x \in \mathcal{X}$:

$$\begin{aligned} x_{i+1} \leq x_i &\Rightarrow Ax_{i+1} \leq Ax_i, \quad \text{and} \\ x_{i+1} \geq x_i &\Rightarrow Ax_{i+1} \geq Ax_i, \quad \text{for every index, } i \end{aligned}$$

A *ntp* operator cannot invert the local ordering at any place in a signal. This also implies that any constant region in a signal will stay constant (possibly with another value) after application of a *ntp* operator. This is therefore an important shape preserving property.

An important consequence of neighbour trend preservation is that a *ntp* operator cannot map a sequence into a rougher (lower) smoothness class [9].

THEOREM 1.40. *Let $x \in \mathcal{M}_n$ and A ntp. Then $Ax \in \mathcal{M}_n$.*

Compositions and convex combinations of *ntp* operators are also *ntp:*

THEOREM 1.41. *If $A, B \in F(\mathcal{X})$ is ntp, then AB is ntp.*

PROOF. For every index i, if $x_{i+1} \leq x_i$ we have that $Bx_{i+1} \leq Bx_i$. Then $A(Bx_{i+1}) \leq A(Bx_i)$. The same goes when $x_{i+1} \geq x_i$and thus AB is *ntp*.

THEOREM 1.42. *If $A, B \in F(\mathcal{X})$ is ntp then $\alpha A + \beta B$ is ntp when $\alpha, \beta \geq 0$.*

PROOF. For an index i, if $x_{i+1} \leq x_i$ then $Ax_{i+1} \leq Ax_i$ and $Bx_{i+1} \leq Bx_i$. Then $(\alpha Ax + \beta Bx)_{i+1} = \alpha Ax_{i+1} + \beta Bx_{i+1} \leq \alpha Ax_i + \beta Bx_i = (\alpha Ax + \beta Bx)_i$. It is similar when $x_{i+1} \geq x_i$. Thus the combination is also *ntp*. □

□

A further related property is *difference reduction*. This relates the difference between successive elements in the data and the difference between these elements after an operator is applied. Difference reducing operators must preserve or decrease the local variation at any position in a sequence.

DEFINITION 1.43. An operator A on $\mathcal{X}$ is *difference reducing* if, for each sequence $x \in \mathcal{X}$: $|Ax_{i+1} - Ax_i| \leq |x_{i+1} - x_i|$, for every index, i.

These properties come together in *full trend preservation*, which is exhibited by all the standard *LULU* operators.

DEFINITION 1.44. An operator on $\mathcal{X}$ is called *fully trend preserving* (*ftp*) if it is *ntp* and *difference reducing.*

A different but equivalent definition of *full trend preservation* for an operator A is that both A and $I - A$ are *neighbour trend preserving.* A separator splits a sequence into signal and noise efficiently. If a separator is *ftp* then both the signal and the noise mimic the local order structure of the original sequence.

Compositions of *ftp* operators and combinations, $\alpha_1 A_a + \ldots + \alpha_n A_n$, where the coefficients α_i sum to 1 are all *ftp.* The following theorem provides general cases where an operator inherits the *ftp* property [9].

THEOREM 1.45. *If $A, B \in F(\mathcal{X})$ are fully trend preserving, then:*

(1) *AB is ftp*
(2) *$\alpha A + (1-\alpha)B$ is ftp with $\alpha \in [0,1]$*
(3) *$I - A$ is ftp.*
(4) *$A \vee B$, $A \wedge B$, and the morphological center (see page 41), if it exists, are ftp where*

$$(A \vee B)x_i = \max\{Ax_i, Bx_i\} \text{ and } (A \wedge B)x_i = \min\{Ax_i, Bx_i\}$$

CHAPTER 2

Taxonomy of operators

The *LULU* framework provides a general structure in which smoothers (see definition 1.13) can be designed. In section 1.6 we defined the pulse removal operators, U_n and L_n, which remove upward and downward pulses of width $\leq n$ respectively. Using these as our building blocks we can construct many useful smoothers.

This chapter offers a selection of operator sets that are members of the *LULU* family. Each of these operator sets has a different interpetation of noise and signal, although all of them can be understood in terms of the pulse perspective discussed in chapter 1. This does not aim to be an exhaustive list. The aim is rather to provide an introduction to and explanation of those operators which have so far proven most useful and as such it contains ideas which may be exploited further in the design of related operators. Choosing which of these operators to use in a specific case should be made according to the problem at hand and by comparison of the analysis goals with the specific qualities of each operator. After the discussion of the *LULU* operators, we compare them with a few related operators not in the *LULU* family.

2.1. The basic LULU separators: U_n and L_n

The basic LULU operators and their effect as pulse removers were discussed in the previous chapter. We have also seen how any idempotent operator consisting of compositions of the elementary operators can be expressed as a composition of the basic LULU operators with explicit priorities for the removal of different width pulses.

The roots of an operator are those sequences that are left unchanged by application of the operator. Theorems 1.33 and 1.34 implicitly classify the root sequences of U_n and L_n:

Corollary 2.1. *For $j \geq n$, we get that $U_n x = x$ if $x \in \mathcal{M}_j^+$ and $L_n y = y$ if $y \in \mathcal{M}_j^-$.*

The following result follows directly from theorem 1.24.

Corollary 2.2. $L_n \leq \ldots \leq L_1 \leq I \leq U_1 \leq \ldots \leq U_n$

The basic separators form a nested set of intervals, all containing the identity operator. Thus $I \in [L_1, U_1] \subset \ldots \subset [L_n, U_n]$. We call these the *LU*-intervals. A direct consequence of this is that if $(L_n x)_i = (U_n x)_i$ for some i then neither L_j or U_j with $j \leq n$ will change sequence x at element i.

For any sequence, the basic $LULU$ operators L_n and U_n find a lower envelope and upper envelope respectively lying in a specific smoothness class. We illustrate the effect of these operators on a simple sequence for a few values of n (figure 13). The smaller the value of n the closer these envelopes are to the input sequence. As n gets larger more of the smaller scale pulses gets removed, thus mapping the sequence into higher smoothness classes.

The LU-interval also gives a way to compare general smoothers with the best approximation in $\mathcal{M}_n$ for any of the p-norms [9]:

THEOREM 2.3. *Let $x \in \mathcal{X}$. If $P \in [L_n, U_n]$, then for all $y \in \mathcal{M}_n$*

$$\|Px - y\| \leq (2n+1)^{\frac{1}{p}} \|x - y\|_p$$

The basic operators are smoothers. They are also idempotent and co-idempotent and thus separators [9]. The basic separators have strong shape preserving properties. They are neighbour trend preservation and difference reducing and as a result also fully trend preserving. All the operators in $\mathcal{L}$ are thus also fully trend preserving by virtue of theorem 1.45.

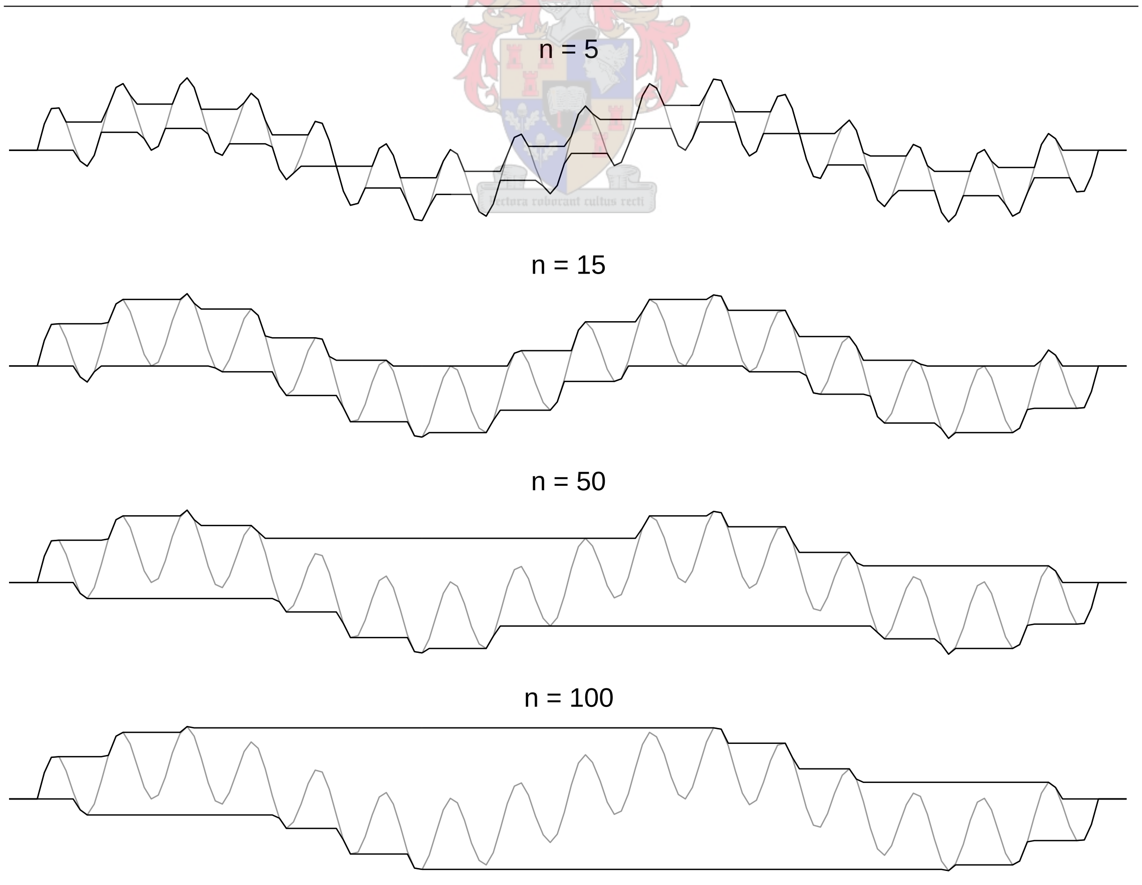


FIGURE 13. Effect of L_n and U_n on a sequence of length 150 for $n = 5, 15, 50$ and 100

2.2. The LULU separators: U_nL_n and L_nU_n

The most straightforward way of combining the basic operators is to follow one by the other. This gives us two possibilities: removing the downward pulses first (U_nL_n) and removing the upward pulses first (L_nU_n). As these are just compositions of the basic operators and in $\mathcal{L}$ they automatically inherit the shape preservation properties of the basic operators: they are *ftp*. They are also smoothers and separators [9]. We call these our standard *LULU* separators.

These operators are in general not equal. In other words the basic operators, U_n and L_n, do not commute. This is due to a *fundamental ambiguity* regarding the classification of pulses. Figure 14 shows an ambiguous sequence and some possible interpretations.

The question is whether the sequence in figure 14 consists of one downward pulse of width n or two upward pulses of width n. The two operators U_nL_n and L_nU_n answer this question differently. L_nU_n gives priority to the removal of downward pulses, so will detect one downward pulse of width n. U_nL_n in turn removes upward pulses before downward pulse and therefore detect two upwards pulses. By choosing one of the interpretations the other one becomes unavailable. The one-dimensional median smoother, M_n, (see section 2.8) gives a different interpretation. In section 2.5, smoothers that rely equally on both of the standard *LULU* separator interpretations are defined.

The difference between these two interpretations gives rise to the LULU-interval $[U_nL_n, L_nU_n]$, which in turn lies in the corresponding *LU*-interval $[L_n, U_n]$.

THEOREM 2.4. $L_n \leq U_nL_n \leq L_nU_n \leq U_n$

The LULU-interval, $[U_nL_n, L_nU_n]$, can be understood as the interval of ambiguity. The magnitude of the ambiguity at a specific element in the sequence is the absolute difference between the interpretations of the two operators: $(|L_nU_nx - U_nL_nx|)_i$. There is no ambiguity when these two operators agree on the smoothed value.

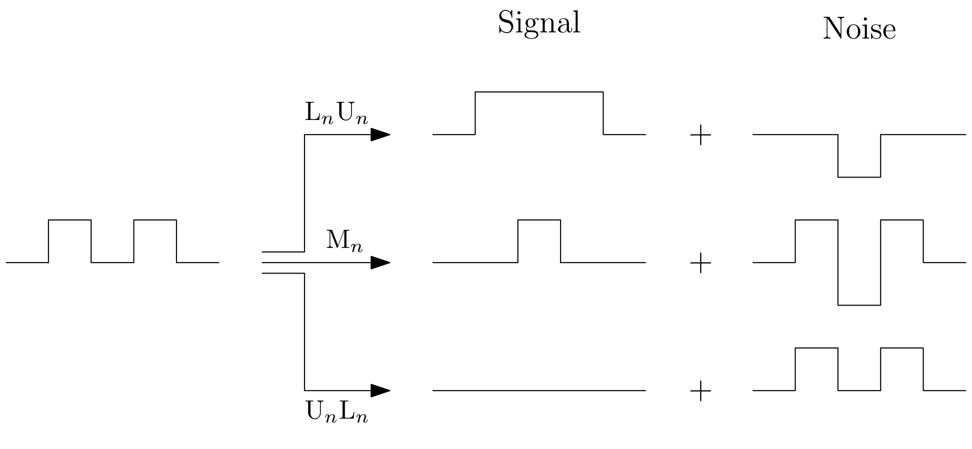


FIGURE 14. Sequence illustrating the fundamental ambiguity associated with removing pulses.

The root sequences of the standard $LULU$ operators, U_nL_n and L_nU_n, are those sequences that are roots of both U_n and L_n. From corollary 2.1 we get:

COROLLARY 2.5. *For $j \geq n$, we have $L_nU_nx = x$ and $U_nL_nx = x$ if $x \in \mathcal{M}_j$.*

The LULU separators remove upward and downward pulses up to a specific width, therefore we can formalize the effect they have on the smoothness class of a sequence x.

THEOREM 2.6. *Let x be any sequence. Then $L_nU_nx \in \mathcal{M}_n$ and $U_nL_nx \in \mathcal{M}_n$.*

PROOF. $x \in \mathcal{M}_0$. From corollary 1.35 we have $U_nx \in \mathcal{M}_n^+$ and $L_nU_nx \in \mathcal{M}_n^-$. From theorem 1.34 we know that L_n cannot create arcs of width less than or equal to n, thus also $L_nU_nx \in \mathcal{M}_n^+$. Then from definition 1.21 we have $L_nU_n \in \mathcal{M}_n$. The case for U_nL_n is similar. $\square$

By theorems 2.3, 2.4 and 2.6 the LULU separators are near-best approximations in $\mathcal{M}_n$.

The operators U_nL_n and L_nU_n are biased towards the detection of upward and downward pulses respectively. We would like to quantify this 'bias' (this is not the strict definition of bias involving the difference between an expectation value and a true value).

An unbiased operator is self-dual, i.e. it commutes with the negative operator: $NP = PN \Rightarrow PN - NP = 0$. This suggests a way of quantifying the bias. Calculate and compare the quantity $\|PNx - NPx\|_2$ for different operators and a representative set of sequences. Unfortunately, all information about the direction of the bias is lost in the process. So rather use, for a sequence x of length N:

$$\text{sign}\left(\frac{1}{N}\sum_{i=0}^{N-1}(PNx - NPx)\right)\|PNx - NPx\|_2 \tag{2.1}$$

Though the 2-norm is used here, other norms would work as well. U_n and L_n are the most biased LULU operators as they can only remove either upward or downward pulses, and are therefore used for comparison. We create a sequence of length 100 consisting of uniformly distributed noise in the range $[-0.5, 0.5]$. This sequence is smoothed with the separators, U_n, L_n, U_nL_n and L_nU_n for $n = 1 \ldots N$. Now use equation 2.1 to quantify the bias for each of these operators and every n. To get a result more representative of any random sequence the experiment was repeated for 1000 random sequences and the results averaged. We will use this technique again in section 2.7 to compare the bias of the rest of the $LULU$ operators.

See figure 15. For ease of exposition we call the average of the quantified bias, calculated using equation 2.1, the bias of the operator. As expected the absolute bias of U_n and L_n is always larger than that of U_nL_n and L_nU_n. Also, the bias of U_{n+1} is greater than that of U_n. The operators that are duals of each other have bias of equal magnitude but different direction. This also follows from the definition of duality as

$$AN - NA = NB - BN = -(BN - NB)$$

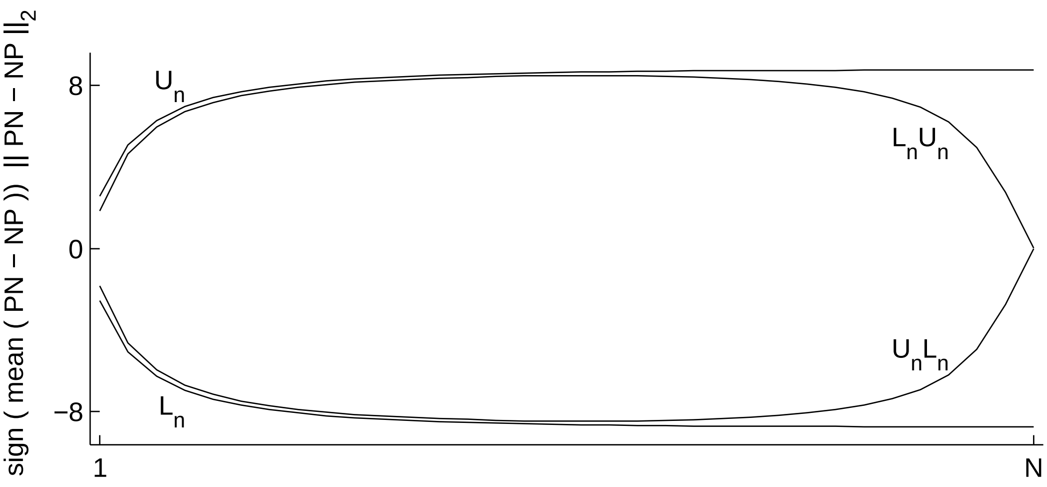


FIGURE 15. Quantifying the bias for the basic and standard LULU operators.

if A and B are duals. The average size of the LU- and $LULU-$interval for a specific n can be inferred from the distances in figure 15 between the bias of L_n and U_n and between the bias of U_nL_n and L_nU_n respectively.

For very small n the standard operators are somewhat less biased than the basic operators. As n increases the difference in bias between U_n and L_nU_n (or L_n and U_nL_n) becomes less pronounced. Recall that $U_n = U_nU_{n-1}\dots U_1$ and $L_n = L_nL_{n-1}\dots L_1$. Then:

$$L_nU_n = L_nL_{n-1}\dots L_1U_nU_{n-1}\dots U_1 \tag{2.2}$$

We apply the operators from right to left. All downward pulses up to width n will be removed before any upward pulses are removed. There is then less chance of removing upward pulses of length $1\dots n$ making it harder for L_n to correct the bias of U_n.

A sequence x of length N can be extended on both sides with zeros to get it in $\mathcal{X}$. This limits the maximum width of pulses to N. Then, we have $L_NU_Nx = 0$. But $U_Nx \neq 0$ if x contains values larger than zero. There must be wide upward pulses left that are removed by L_nU_n for $n = m\dots N$, with $m \leq N$ generally the same order of magnitude as N.

The bias of L_nU_n and U_nL_n and the size of the $LULU$-interval that is a result of this bias are sometimes useful (see section 2.6). Sometimes though, it would be preferable if we could narrow this LULU-interval to arrive at less biased estimates There are a couple of ways we could attempt to do this. The first method is to construct operators where narrower pulses have higher priority than wider pulses, irregardless of their sign. Another idea is to combine the two extreme interpretations and create an unbiased smoother out of them. Operators based on both of these ideas are discussed below.

2.3. Recursive LULU separators: C_n and F_n

DEFINITION 2.7. The operators C_n and F_n are defined recursively by:

(1) $C_{n+1} = L_{n+1}U_{n+1}C_n, \qquad C_0 = I$
(2) $F_{n+1} = U_{n+1}L_{n+1}F_n, \qquad F_0 = I$

According to this definition, these operators start by removing pulses of single width before removing pulses of width 2, which in turn are removed before pulses of width 3, and so on. In other words, narrower pulses of any sign are removed with higher priority.

The recursive $LULU$ operators are smoothers and separators [9]. They are in $\mathcal{L}$ and thus also fully trend preserving. Recall that this means that for any n the local ordering of the input sequence is preserved by the smoothed sequence and the extracted rough part.

The following theorem states that these two operators form an interval (which we will call the recursive $LULU$-interval) that always lie inside the standard $LULU$-interval [9]. In section 2.7 we compare the bias of the recursive $LULU$ operators with the bias of some other operators. The relative sizes of the intervals can then be gauged from their difference in bias. In practice we find that this interval is much narrower than the standard interval.

THEOREM 2.8. $L_n \leq U_nL_n \leq F_n \leq C_n \leq L_nU_n \leq U_n$

A direct result of theorem 2.6 is:

COROLLARY 2.9. *Let x be any sequence. Then $C_nx \in \mathcal{M}_n$ and $F_nx \in \mathcal{M}_n$.*

The discrete pulse transform, which is one of the most powerful tools in the $LULU$ framework, makes extensive use of the recursive $LULU$ separators. It is described in chapter 3.

2.4. Alternating bias separators: Z_n^+ and Z_n^-

The recursive $LULU$ operators lessens the bias of the standard $LULU$ operators by first removing smaller pulses of any sign before moving on to larger pulses. For each pulse width the recursive operators still have a preference for either upward or downward pulses.

The alternating bias separators alternately bias pulses of different signs. At one pulse width there is a preference for pulses of one direction and for the next pulse width the preference is for the other direction. This is an attempt to balance the bias somewhat.

DEFINITION 2.10. The alternating bias separators, Z_n^+ and Z_n^-, are defined recursively:

$$(1)\ Z_{n+1}^- = \begin{cases} L_{n+1}U_{n+1}Z_n^- & \text{if } n \text{ is even} \\ U_{n+1}L_{n+1}Z_n^- & \text{if } n \text{ is odd} \end{cases}$$

$$(2)\ Z_{n+1}^+ = \begin{cases} L_{n+1}U_{n+1}Z_n^+ & \text{if } n \text{ is odd} \\ U_{n+1}L_{n+1}Z_n^+ & \text{if } n \text{ is even} \end{cases}$$

with $Z_0^+ = Z_o^- = I$.

The positive and negative sign in Z_n^+ and Z_n^- refers to the bias these operators have for pulses of width 1. Z_n^+ has a preference for removing upward pulses of single width and Z_n^- has a preference for downward pulses of single width.

These operators map sequences into $\mathcal{M}_n$:

THEOREM 2.11. *Let x be any sequence. Then $Z_n^+ x \in \mathcal{M}_n$ and $Z_n^- x \in \mathcal{M}_n$.*

There exists no partial order relating these two operators, so they do not form an interval like the standard and recursive LULU operators. However they are smoothers, separators and ftp as the others.

2.5. Unbiased smoothers: G_n, G_n^∞ and H_n

The standard LULU operators are both biased equally, but one towards positive pulses and the other towards negative pulses. One can create an unbiased smoother from them by taking their average.

DEFINITION 2.12. $(G_n x)_i = \frac{1}{2}\left((L_n U_n x)_i + (U_n L_n x)_i\right)$

This set of operators satisfies all the smoother axioms and is fully trend preserving (using theorem 1.45). Unfortunately, idempotence is lost, as there may still be arcs of width n or smaller left in the sequence after application of G_n. In general, one can apply the operator repeatedly until it converges. Figure 16 demonstrates how after one application of G_1 arcs of width 1 remain. In this example, applying the operator again yields an unbiased smoothed sequence that is unchanged by further applications of G_1.

DEFINITION 2.13. The operator G_n^∞ is defined as repeating the operator G_n until convergence.

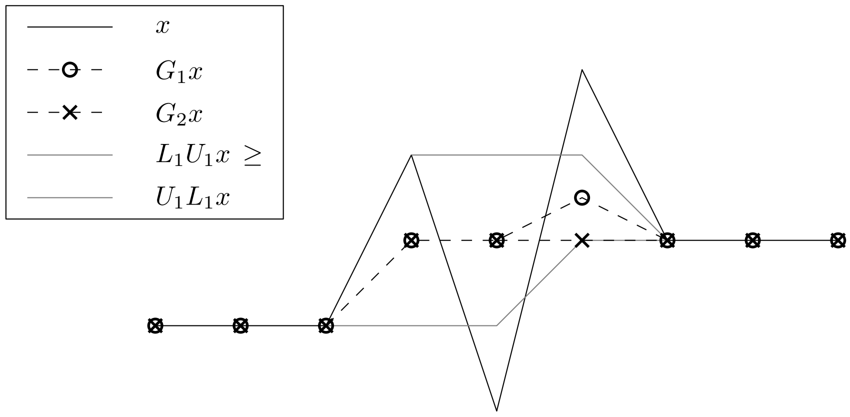


FIGURE 16. Example showing how $G_1 x \notin \mathcal{M}_1$. Another application yields, $G_1^2 x \in \mathcal{M}_1$, an unbiased smoothed sequence.

G_n^∞ is an unbiased smoother that smoothes a sequence by removing arcs of width n and smaller. It inherits full trend preservation due to theorem 1.45. The following theorem gives the guaranteed rate of convergence [9].

THEOREM 2.14. $\|L_nU_nG_nx - U_nL_nG_nx\|_\infty \leq \frac{1}{2}\|L_nU_nx - U_nL_nx\|\infty$

Similar to how the recursive LULU separators in section 2.3 are defined with respect to the standard LULU separators, it is possible to define the recursive unbiased smoother, H_n:

DEFINITION 2.15. The unbiased smoother, H_n, is defined recursively as $H_{n+1} = G_{n+1}^\infty H_n$ with $H_0 = I$.

As this operator is a composition of *ftp* operators, it is also *ftp*. The recursive unbiased operator lies in the recursive $LULU$-interval $[F_n, C_n]$. This interval is generally very narrow and as such this operator does not differ much from C_n and F_n in practice. The operators G_n^∞ and H_n are mappings into the smoothness class $\mathcal{M}_n$.

THEOREM 2.16. *Let x be any sequence. Then $H_nx \in \mathcal{M}_n$ and $G_n^\infty x \in \mathcal{M}_n$.*

Both of these operators are idempotent, but neither is co-idempotent since $G_n^\infty(I - G_n^\infty)$ and $H_n(I - H_n)$ are not zero in general (which experimentation with random sequences will quickly show), and as such they do not satisfy the separator axioms.

2.6. Minimally destructive smoothers: A_n and B_n

For some purposes one may wish to smooth a sequence without distorting the input sequence excessively. For example, one may want to remove impulsive noise while not losing all high frequency content in the process.

The operator B_n, which is the morphological center of L_nU_n and U_nL_n [15], does this by keeping elements of the input sequence unchanged if they fall in the $LULU$-interval. At sequence elements where the input sequence lies outside of the $LULU$-interval the values are clipped to the edges of the interval. The width of the $LULU$-interval can be controlled by varying the parameter n. Refer to figure 15 for an idea of the size of the $LULU$-interval for noisy sequences. Although this depends on the input sequence in question, for long sequences the range of values for n that give good (and similar) results tends to be quite wide.

DEFINITION 2.17. $$(B_nx)_i = \begin{cases} x_i, & \text{if } x_i \in [U_nL_n, L_nU_n] \\ (U_nL_nx)\,x_i, & \text{if } x_i < (U_nL_nx)_i \\ (L_nU_n)\,x_i, & \text{if } x_i > (L_nU_nx)_i \end{cases}$$

We can also define a variant of the morphological centre. Instead of clipping the values that lie outside the $LULU$-interval, one can replace these by a value directly in the middle of the $LULU$-interval. The operator G_n is the average of the two biased LULU separators U_nL_n and L_nU_n and lies in the middle of said interval.

DEFINITION 2.18. $(A_n x)\, i = \begin{cases} x_i, & \text{if } x_i \in [U_n L_n, L_n U_n] \\ (G_n x)_i & \text{otherwise} \end{cases}$

There exists variations of these operators that may be useful. We will not define all possibilities, but one can use the operator or G_n^∞ instead of G_n or base these operators on the recursive $LULU$-interval.

Sometimes one does not need to have a smoothed value at every point in a sequence. Then, instead of defining behaviour for when the sequence lies outside of the LULU interval, one can regard those segments as undefined. This makes it possible to use only those values which has a high likelihood of not being spoiled by noise. It can be be argued that this is superior to attempting to recreate those data points. A downside of this is that one cannot use techniques which require a fully defined sequence.

2.7. Bias comparison

We now compare the bias for some of the above operators using the same procedure as in section 2.1. Noise from an unbiased uniform distribution is smoothed with some of the biased operators defined in this chapter for different values of n. Unbiased operators are self-dual, so we quantify the bias of an operator by testing how far it is from being self-dual. This is repeated 1000 times in order to calculate averages. The results are in figure 17. For comparison the results for the basic and standard $LULU$ operators are plotted again. Their bias is much larger than that of the recursive and alternating bias operators, which decreases relatively quickly as n is increased.

We can estimate the size of the recursive $LULU$ interval for random sequences from this graph, by calculating the distance between C_n and F_n. This tends to be less for larger n.

It seems that the alternating bias operators have equal and opposite bias. This is true for any sequence, but what the test does not show is that there are sequences where Z_n^- is biased towards below and Z_n^+ towards above. This is because the sequences used to generate the statistics are random sequences, which tend to have many unit width pulses. Therefore the bias towards pulses of this width tends to dominate this measure. We see that they are less biased than the recursive $LULU$ operators.

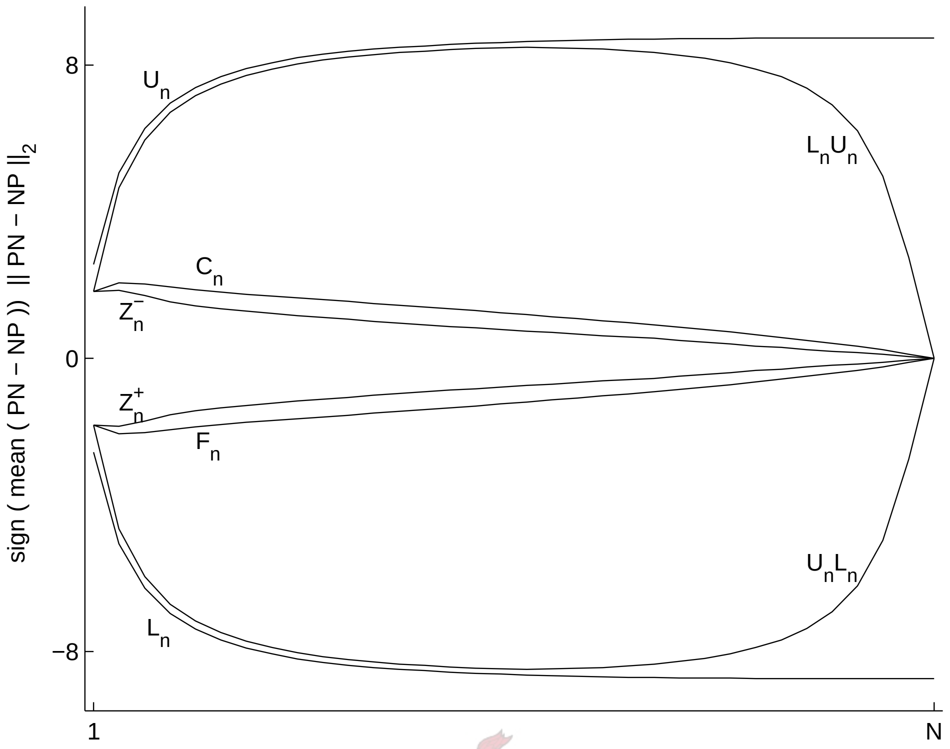


FIGURE 17. Quantifying the bias of some of the LULU operators.

2.8. Comparison with the median smoothers

The median smoothers, a popular family of non-linear smoothers based on the median operator, were popularised by Tukey [17]. The median of an odd length sequence is the middle element when the sequence is sorted in non-decreasing order. We define the median smoothers $\{M_n\}$:

DEFINITION 2.19. The 1-d median smoother, M_n, replaces each element in a sequence by the median of the $2n + 1$ window of sequence elements centered around the particular element:

$$(M_n x)_i = \text{median}\left\{x_{i-n}, \ldots, x_i, \ldots, x_n\right\}$$

The median operators satisfies the translation invariance axioms, and the stricter scale invariance axiom as specified by Mallows [3] making it a smoother. It is not optimally efficient as it is not idempotent nor co-idempotent, and thus not a separator.

The roots of an operator in $F(\mathcal{X})$ are those sequences that are left unchanged by application of the operator, i.e. the eigen-sequences w.r.t. to eigenvalue 1. For the median smoothers the roots that lie in $\mathcal{X}$ can be characterized in terms of the monotonicity classes.

THEOREM 2.20. *Let $x \in \mathcal{X}$ be a sequence that lies in the n-th monotonicity class, $\mathcal{M}_n$. Then x is a root sequence of the median smoothers M_j with $j \leq n$. I.e.*

$$M_j x = x \text{ with } j \leq n$$

PROOF. Let $W_i = \{x_{i-j}, \ldots, x_i, \ldots, x_{i+j}\}$ be the support window for the operator M_j at each index i. The sequence x is n-monotone, thus any sub-sequence $\{x_k, \ldots, x_{k+n+1}\}$ is monotone. The support of M_j is $2j+1$. The elements in the window are either monotone or m-monotone with $n \leq m < 2j-1$. If the elements in the window are monotone, the theorem follows trivially.

Now, assume in turn that $W_i \in \mathcal{M}_m \sim \mathcal{M}_{m+1}$ for $n \leq m < 2j-1$. Then there exists a sub-sequence in W_i such that either $x_k > x_{k+1} = \ldots = x_{k+m+1} < x_{m+k+2}$ or $x_k < x_{k+1} = \ldots = x_{k+m+1} > x_{m+k+2}$. There is thus a constant section of length $m+1$. We know that $m \geq n$ and $n \geq j$, therefore $m+1 \geq n+1 > \frac{2n+1}{2} \geq \frac{2j+1}{2} = \frac{1}{2}\text{support}\,(M_j)$. The constant section has a length of more than half of the support of the median smoother, hence the point x_i is always part of the constant section. The median function finds the middle value in an ordered set. If more than half of the elements have the same value, the median is equal to that value. This proves the theorem. $\square$

The only other roots are infinite in length and not in ℓ_1 [18]. In general it can take an arbitrary number of applications of the median operators to reach a root. Furthermore, repeating an operator until convergence extends the support of the operator allowing a change in one part of the input sequence to affect the smoothing arbitrarily far away.

When applying the median smoother, M_n, to sequences in $\mathcal{M}_{n-1}$ the effect can be described in terms of the basic LULU operators, L_n and U_n [9]. This is not valid when applying the median operators to general sequences.

THEOREM 2.21. *Let $x \in \mathcal{M}_{n-1}$, then $M_n x = (U_n + L_n - I)x$*

To understand the working of the median smoother with respect to the *LULU* operators, we compare the effect of M_1 with that of the basic *LULU* separators L_1 and U_1. To characterize the effects of these opperators on a sequence element it is only necessary to consider the window of length 3 around the element in question, since these operators all have a support window 3 elements wide. We divide this set of sub-sequences into four classes, based on the local ordering of the elements. Using the definitions of M_1, L_1 and U_1 we calculate what these operators do to the center element of each class of sub-sequences (table 1). The total effect of these operators on a full sequence can be determined by calculating the local effect at each sequence element using this table.

In table 1 we see that, as expected, none of these operators change the center element when the elements in the window are monotone. It seems that L_1 removes upward pulses of width 1 (and U_1 downward pulses of width 1), by replacing the center element with the closest neighbour element. It is obvious that they are both biased. It seems that M_1 is unbiased and removes both pulses in exactly the same way that $L_1$and U_1 removes upward and downward pulses respectively. Forgetting for the moment the fact that M_1 is not a separator, one might be lead to think that M_1 is preferable. We shall see that this is not the whole story.

x_{i-1} x_i x_{i+1}	$(M_1x)_i$	$(L_1x)_i$	$(U_1x)_i$
$\leq$ $\leq$	x_i	x_i	x_i
$\leq$ $\geq$	$\max\{x_{i-1}, x_{i+1}\}$	$\max\{x_{i-1}, x_{i+1}\}$	x_i
$\geq$ $\leq$	$\min\{x_{i-1}, x_{i+1}\}$	x_i	$\min\{x_{i-1}, x_{i+1}\}$
$\geq$ $\geq$	x_i	x_i	x_i

TABLE 1. Comparison of effect of median smoother M_1 and basic $LULU$ separators L_1 and U_1.

The median smoothers always lie in the corresponding $LULU$-interval [9]:

$$M_n \in [U_nL_n, L_nU_n] \subset [L_n, U_n] \tag{2.3}$$

The median operator M_n and the smoother G_n (section 2.5) are both unbiased non-idempotent smoothers that lie in the $LULU$ interval. It is illuminating to compare them.

We smooth a short sequence with the smoothers G_1 and M_1. This specific sequence (the same one used in figure 16) was chosen because there is ambiguity regarding the pulses, and neither of the operators can map it directly onto $\mathcal{M}_1$. See figure 18. Both G_1x and M_1x are unbiased and in the $LULU$ interval. However, it seems that M_1 is biased at individual sequence elements with respect to the $LULU$ interval $[U_1L_1, L_1U_1]$ (but in varying directions) while G_1 is, by definition, in the middle of said interval. We investigate this further. As an aside, notice that M_1 lacks the trend preservation properties of the $LULU$ operators as some of the local orderings was inverted.

G_1 is the average of the standard $LULU$ separators, L_1U_1 and U_1L_1. We compare the effect of the operators M_1, L_1U_1 and U_1L_1 on all sub-sequences of length 5, since L_1U_1

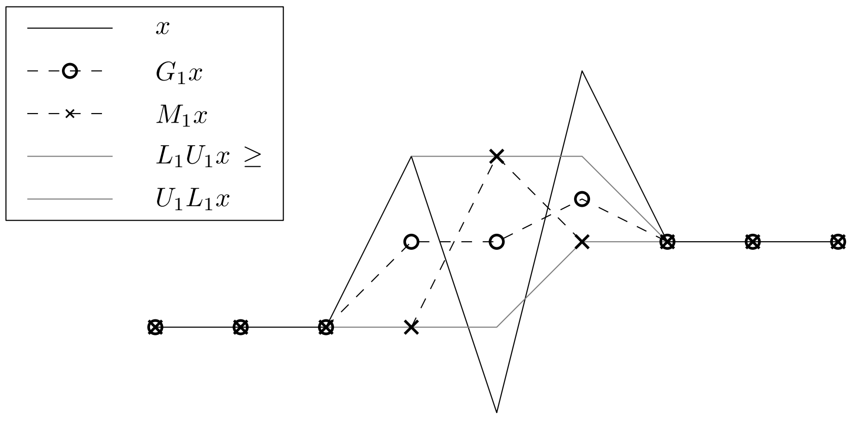


FIGURE 18. Comparison of G_1 and M_1 on an ambiguous sequence.

$x_{i-2}\quad x_{i-1}\quad x_i\quad x_{i+1}\quad x_{i+2}$	$(M_1x)_i$	$(U_1L_1x)_i$	$(L_1U_1x)_i$
$\leq\quad\leq\quad\leq\quad\leq$	x_i	x_i	x_i
$\leq\quad\leq\quad\leq\quad\geq$	x_i	x_i	x_i
$\leq\quad\leq\quad\geq\quad\leq$	$\max\{x_{i-1}, x_{i+1}\}$	$\max\{x_{i-1}, x_{i+1}\}$	$\max\{x_{i-1}, \min\{x_i, x_{i+2}\}\}$
$\leq\quad\leq\quad\geq\quad\geq$	$\max\{x_{i-1}, x_{i+1}\}$	$\max\{x_{i-1}, x_{i+1}\}$	$\max\{x_{i-1}, x_{i+1}\}$
$\leq\quad\geq\quad\leq\quad\leq$	$\min\{x_{i-1}, x_{i+1}\}$	$\min\{\max\{x_{i-2}, x_i\}, x_{i+1}\}$	$\min\{x_{i-1}, x_{i+1}\}$
$\leq\quad\geq\quad\leq\quad\geq$	$\min\{x_{i-1}, x_{i+1}\}$	$\min\{\max\{x_{i-2}, x_i\}, \max\{x_i, x_{i+2}\}\}$	$\min\{x_{i-1}, x_{i+1}\}$
$\leq\quad\geq\quad\geq\quad\leq$	x_i	x_i	x_i
$\leq\quad\geq\quad\geq\quad\geq$	x_i	x_i	x_i
$\geq\quad\leq\quad\leq\quad\leq$	x_i	x_i	x_i
$\geq\quad\leq\quad\leq\quad\geq$	x_i	x_i	x_i
$\geq\quad\leq\quad\geq\quad\leq$	$\max\{x_{i-1}, x_{i+1}\}$	$\max\{x_{i-1}, x_{i+1}\}$	$\max\{\min\{x_{i-2}, x_i\}, \min\{x_i, x_{i+2}\}\}$
$\geq\quad\leq\quad\geq\quad\geq$	$\max\{x_{i-1}, x_{i+1}\}$	$\max\{x_{i-1}, x_{i+1}\}$	$\max\{\min\{x_{i-2}, x_i\}, x_{i+1}\}$
$\geq\quad\geq\quad\leq\quad\leq$	$\min\{x_{i-1}, x_{i+1}\}$	$\min\{x_{i-1}, x_{i+1}\}$	$\min\{x_{i-1}, x_{i+1}\}$
$\geq\quad\geq\quad\leq\quad\geq$	$\min\{x_{i-1}, x_{i+1}\}$	$\min\{x_{i-1}, \max\{x_i, x_{i+2}\}\}$	$\min\{x_{i-1}, x_{i+1}\}$
$\geq\quad\geq\quad\geq\quad\leq$	x_i	x_i	x_i
$\geq\quad\geq\quad\geq\quad\geq$	x_i	x_i	x_i

TABLE 2. Comparison of effect of median smoother M_1 and the standard *LULU* separators U_1L_1 and L_1U_1.

---

and U_1L_1 have support windows of width 5. The sequences are divided into classes based on the local ordering of the elements. There are now 16 classes. Refer to table 2.

The only cases where M_1, U_1L_1 and U_1L_1 change the center element of the sub-sequence $\{x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}\}$ are when either $x_i > x_{i-1}, x_{i+1}$ or $x_i < x_{i-1}, x_{i+1}$. Notice that these inequalities are strict, because if any of these is an equality the entries in table 2 for that class of sequences reduce to x_i. In other words, these operators only change x_i if there is an arc of width 1 centered on x_i. Recall that there is a fundamental ambiguity associated with the removal of pulses. The removal of upward pulses can destroy neighbouring downward pulses and vice versa. But, we saw in table 1 that M_1 removes both in an unbiased way. Is this not a contradiction? In table 2 we can see how this is achieved. When $x_i > x_{i-1}, x_{i+1}$, the median interpretation is the same as that of U_1L_1 and when $x_i < x_{i-1}, x_{i+1}$, the median interpretation is the same as that of L_1U_1. It now becomes clear that the lack of bias of the median smoother came at a price: it is achieved by choosing one $LULU$ interpretation for half of the ambiguous sub-sequences and the other $LULU$ interpretation for the rest. In contrast to this, the smoother G_1 is unbiased because at every sequence element it is always halfway between the two biased $LULU$ interpretation (at the cost of no longer being a selector).

This argument is valid for the rest of the median smoothers M_k only when they are applied to a sequence $x \in \mathcal{M}_{k-1}$. When a sequence in smoothness class $\mathcal{M}_{k-1}$ is smoothed using M_k, the median interpretation is selected from the two $LULU$ interpretations, L_kU_k and U_kL_k.

THEOREM 2.22. *For $x \in \mathcal{M}_{k-1} \subset \mathcal{X}$ and all i, if $(M_kx)_i \neq (L_kU_kx)_i$ then $(M_kx)_i = (U_kL_kx)_i$*

PROOF. Let $x \in \mathcal{M}_{k-1}$. Then,

$$M_kx = (U_k + L_k - I)x \tag{2.4}$$

The operators L_k and U_k only remove arcs of width k when applied to $x \in \mathcal{M}_{k-1}$. By the definition of an arc, the plateau of a downwards arc cannot overlap with the valley of an upwards arc. In other words, if L_n changes sequence element x_i then U_n cannot. Therefore, for $x \in \mathcal{M}_{k-1}$,

$$U_kx_i \neq x_i \Rightarrow L_kx = x_i \tag{2.5}$$

Assume that $(U_kx)_i \neq x_i$, then $(L_kx)_i = x_i$ by equation 2.5. We get.

$$(M_kx)_i = (U_kx)_i + (L_kx)_i - x_i = (U_kx)_i \geq (L_kU_kx)_i$$

But from equation 2.3 we know that $M_k \leq L_kU_k$ and thus $(M_kx)_i = (L_kU_kx)_i$ which contradicts the theorem conditions. Therefore if $(M_kx)_i \neq (L_kU_kx)_i$, then $(U_kx)_i = x_i$.

Using this, equation 2.4 and theorem 2.4, we get:

$$(M_kx)_i = (U_kx)_i + (L_kx)_i - x_i = (L_kx)_i \leq (U_kL_kx)_i$$

This together with $M_k \geq U_kL_k$ (from equation 2.3) proves the theorem. □

The characterization of M_n in terms of the $LULU$ operators for a general sequence $x \in \mathcal{X}$ is an open problem.

2.9. Comparison with linear smoothing techniques

We now discuss some linear smoothing procedures and see how they are all interpretable as the convolution of a sequence and a mask.

DEFINITION 2.23. The convolution of a sequence $x \in \mathcal{X}$ and a mask $m = \{m_{-L}, \ldots, m_L\}$ is:

$$(x * m)_i = \sum_{l=-L}^{L} m_l x_{i-l}$$

The running average linear filter R_n replaces each sequence element by the average of the $2n+1$ points around that element:

DEFINITION 2.24. The running average filter R_n of a sequence $x \in \mathcal{X}$ is

$$\begin{aligned}(R_n x)_i &= \frac{1}{2n+1} \sum_{j=i-n}^{i+n} x_j \\ &= x * m\end{aligned}$$

with $m_i = \frac{1}{2n+1}$ for $i = -n, \ldots, n$ and 0 elsewhere.

The Haar projections smooth a sequence by mapping it onto a sequence of resolution half that of the input sequence.

DEFINITION 2.25. The Haar operators P_k are projections of a sequence, $x(i) = \alpha_{0,i}$ with $i = 0 \ldots 2^N - 1$, from the space B_0 onto the spaces B_k with $k > 0$. With ϕ the Haar scaling function (see figure 2 in chapter 1) :

$$\begin{aligned}B_j &= \text{span}\left\{\phi_i : \phi_i(t) = \phi\left(2^{-j}t - i\right)\right\} \\ P_k x(t) &= \sum_{i=0}^{2^{N-k}-1} \alpha_{k,i}\phi\left(2^{-k}t - j\right) \in B_k \\ \alpha_{k,i} &= \frac{1}{2}\left(\alpha_{k,2j} + \alpha_{k,2j+1}\right) \text{ where } i = 0 \ldots 2^{N-k} - 1\end{aligned}$$

The Haar projection operators replaces the coefficients with their pairwise averages. This is guaranteed to double the length of sub-sequences that are locally monotone. Thus, the Haar projections map sequences into a subset of the monotonicity classes $\mathcal{M}_n$. The operator P_1 maps from $\mathcal{M}_0$ to $\mathcal{M}_1$ and P_2 maps from $\mathcal{M}_1$ to $\mathcal{M}_3$. In general $P_k x(t) \in \mathcal{M}_{2^k-1}$.

The Haar projections are equivalent to calculating one convolution at odd indices and another at even indices:

$$(P_n x)_i = \begin{cases} \left(x * m_n^{'}\right)_i & \text{if } i \text{ is odd} \\ (x * m_n)_i & \text{if } i \text{ is even} \end{cases}$$

with

$$\begin{aligned}(m_n)_i &= \frac{1}{2^n} \text{ when } i = -2^N + 1, \ldots, 0 \\ \left(m_n^{'}\right)_i &= \frac{1}{2^n} \text{ when } i = 0, \ldots, 2^N - 1\end{aligned}$$

For each Haar projection operator, these masks m'_n and m_n can be exchanged. These two possibilities are called the phases of operator P_n. For a n-level Haar smoothing, one needs to make n phase choices, leading to 2^n possible smoothings. Some phase choices may provide better results for some parts of the sequence. Often no choice gives optimal results for all parts (recall figure 5).

The Discrete Fourier Transform decomposes a sequence into its harmonic frequency components. A low-pass filter is created by reconstructing the sequence using only the low frequency sinusoids. By the definition of smoothness in the Fourier perspective, removing the higher frequencies will result in a smoother sequence. The convolution theorem states that convolving two sequences has the same effect as multiplying their Discrete Fourier Transforms element-by-element. This implies that a Fourier low-pass filter is equivalent to convolution with a particular mask sequence.

A relatively smooth test sequence is constructed with two impulses and one sharp edge. This is 'smoothed' using the linear operators discussed above, and with the $LULU$ separators U_1L_1 and L_1U_1. Recall that the Haar projections are not translation invariant (the phase problem; discussed above and in section 1.4), and is thus not a smoother according to definition 1.13. The running average does satisfy the smoother axioms; we can talk of R_n smoothing a sequence. The smoothed sequences are displayed in figure 19 and 20.

It is clear that the Haar wavelet and running average smear high pulses into the surrounding data values and that they erode the large edge discontinuity. In contrast to this, both C_1

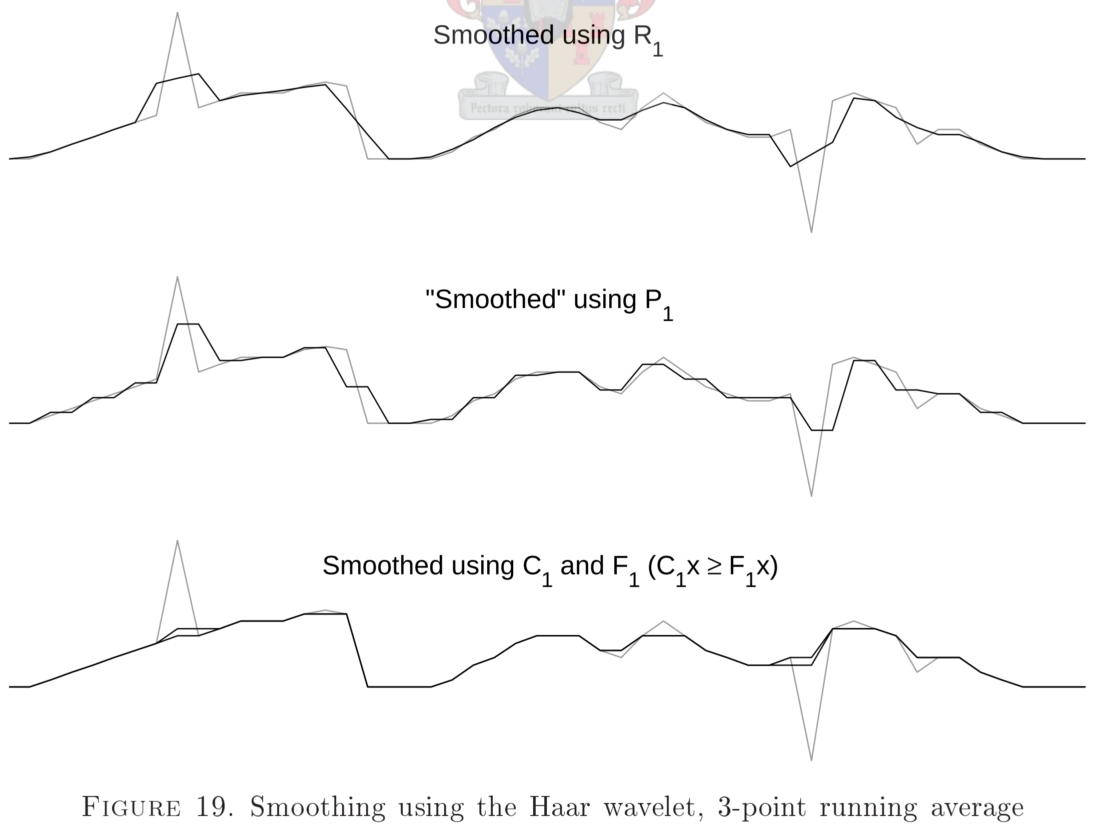


FIGURE 19. Smoothing using the Haar wavelet, 3-point running average and the LULU filters.

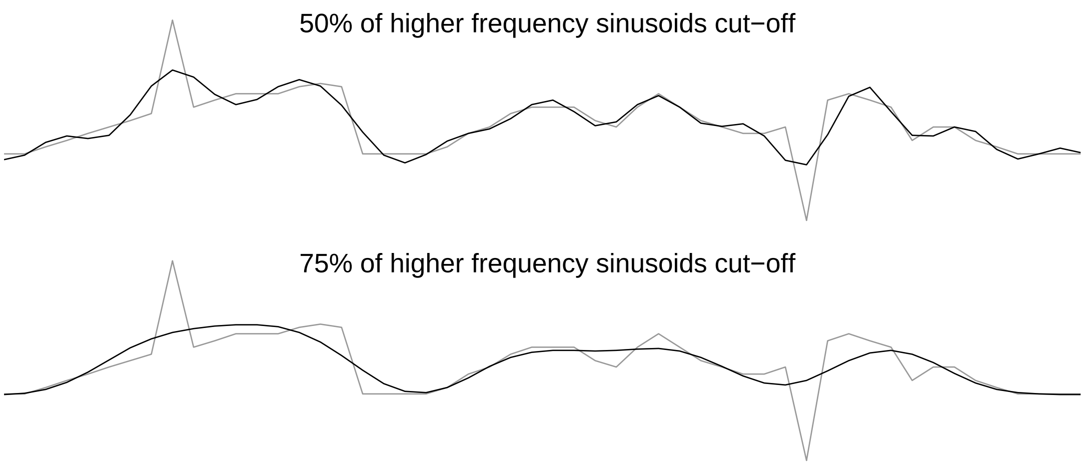


FIGURE 20. Smoothing using a Fourier low-pass filter

and F_1 remove the impulses without distorting the surrounding sequence, and preserves the edge. This is due to their strong shape preserving properties (refer to section 1.7 on trend preservation).

The sequence was smoothed with a Fourier low-pass filter using two cut-off frequencies: 25% and 50% of the lower frequencies were retained in each case. The more high frequencies that are ignored in the reconstruction, the smoother the reconstructed sequence. Once again there is a spreading of 'energy' into surrounding elements, causing an erosion of edges. Around the impulses we see that the filter over compensates in the opposite direction. This is the Gibbs phenomenon. We do not see this effect with the non-linear *LULU* operators.

All three linear smoothing techniques discussed here can be expressed in terms of convolutions of the input sequence with a mask sequence. This implies that a sequence is smoothed by replacing every element with a linear combination of elements in a window around it. This has the effect of averaging out discontinuities in a sequence, causing impulses to spread into the surrounding sequence and once sharp edges to erode. In other words, noise is 'removed' from a sequence by spreading it around.

Let us analyze why this tends to work well for some sequences. Assume that a sequence x is formed by the addition, at each element, of a signal component, s, and an i.i.d. noise component, n: $x_i = s_i + n_i$. This is then processed using a linear filter (i.e. a convolution):

$$\begin{aligned}(x*m)_i &= \sum_{l=-L}^{L} m_l x_{i-l} \\ &= \sum_{l=-L}^{L} m_l \left(s_{i-l} + n_{i-l}\right) \\ &= \sum_{l=-L}^{L} m_l s_{i-l} + \sum_{l=-L}^{L} m_l n_{i-l} \\ &= (s*m)_i + (n*m)_i\end{aligned}$$

When the underlying signal s is sufficiently smooth (i.e. continuous derivatives) and the mask is sufficiently local one can approximate $(s * m)_i \approx s_i$. For a discontinuous signal this is very inaccurate, and thus we see erosion of edges. If the noise is unbiased, identically independently distributed and the mask is wide enough then $(n * m)_i \approx 0$. Large magnitude impulsive noise will cause deterioration of the accuracy of this approximation. Also, the accuracy requirements of the two approximations conflict: the first approximation tends to be more accurate for narrow masks, while the second approximation is more accurate for wider masks.

We see that there are certain conditions that need to apply for the linear filter to accurately remove noise from a sequence. Now we want to analyze the effect of a single impulse on the output of a linear filter. In the bottom graph of figure 20, we see how the impulse on the left 'pulled' the smoothed curve up towards it. If the size of that impulse were to increase, this effect would also increase. We perturb a sequence with an impulse of amplitude α at position a. We analyze what effect this has on a sequence smoothed using a convolution operation. Assuming that $m_l = 0$ when $l \notin [-L, L]$, we get:

$$\begin{aligned}
((x + \alpha\delta_{i,a}) * m)_i &= \sum_{l=-L}^{L} m_l \left(x_{i-l} + \alpha\delta_{i-l,a}\right) \\
&= \sum_{l=-L}^{L} m_l x_{i-l} + \alpha \sum_{l=-L}^{L} m_l \delta_{i-l,a} \\
&= (x * m)_i + \alpha m_{a+i}
\end{aligned}$$

All values that are closer than L elements away from an impulse are contaminated. If the impulse is of large magnitude this can render these values useless. The operators P_1 and R_1 are affected by this, but due to their limited support the damage is contained. With P_n and R_n there are 2^n and $2n + 1$ affected sequence elements respectively. The Fourier low-pass filter has a corresponding mask of length equal to the length of the sequence. A perturbation at a single sequence element can therefore affect the whole sequence. See figure 21. The left impulse was replaced by one an order of magnitude larger. The resulting smoothed sequence is seriously damaged by oscillations around this impulse (the Gibbs effect). Since the *LULU* smoothers are non-linear (and thus not equivalent to a convolution) they can remove these impulses without spreading the noise 'energy' into the sequence.

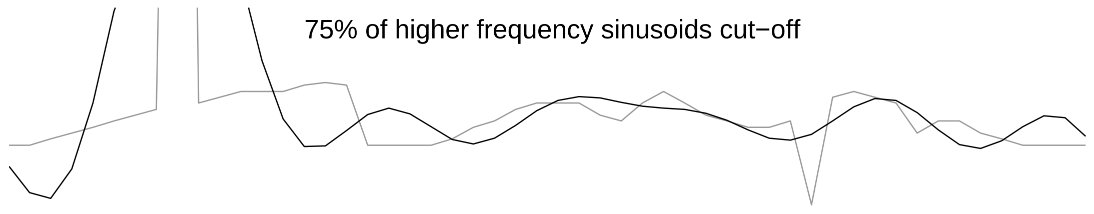


FIGURE 21. Changing the magnitude of one impulse completely changed the Fourier low-pass interpretation.

2.10. A hybrid approach

The running average and Fourier low-pass filter have a smoothing effect on both a sequence and its derivatives. In the *LULU* framework, the smoothness class of a sequence is its degree of local monotonicity. The *LULU* separators are selectors and thus smooth a sequence by replacing values in the sequence by other values that lie in their support window, thereby increasing the degree of local monotonicity of the sequence.

There are times when one might prefer that the sequence and its derivatives become smoother as in the linear approach, while still avoiding the pitfalls associated with large impulses and edge discontinuities. It is possible to achieve this by combining the *LULU* separators with standard linear filters. Pre-smoothing a sequence with a *LULU* smoother to remove impulsive noise is effective but ignores the problem of edge discontinuities. A better way of combining linear filters and non-linear smoothers is to only apply the linear filter when it is not too destructive and to pre-smooth with a *LULU* smoother only when it can reduce the destructiveness below a threshold.

As an example, we define a non-linear hybrid smoother using the linear running average filter R_1 and the unbiased *LULU* smoother G_1^∞. Filter a sequence element with R_1 if the change is smaller than a threshold. If not, we consider pre-smoothing with G_1^∞. If filtering the pre-smoothed result still results in too large a change, we do not use the linear filter at that element at all. We get for a constant threshold T:

$$(Sx)_i = \begin{cases} (R_1 x)_i & \text{if } \left|(R_1 x)_i - x_i\right| \le T \\ (R_1 G_1^\infty x)_i & \text{else if } \left|(R_1 G_1^\infty x)_i - (G_1^\infty x)_i\right| \le T \\ (G_1^\infty x)_i & \text{otherwise} \end{cases}$$

Applying this kind of approach to the same test sequence as in figure 21, we get the results in figure 22. We see that this was effective in removing the impulsive noise and preserving the sharp edge discontinuity. In addition, the other parts of the sequence were smoothed satisfactorily.

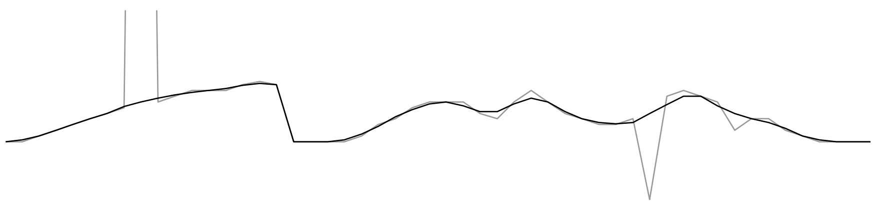


FIGURE 22. Smoothing the sequence with a hybrid linear filter / non-linear smoother approach.

CHAPTER 3

A multi-resolution decomposition

There exists well-known perspectives for decomposing sequences and functions into components. The Fourier Transform finds the underlying frequencies in a periodic function by decomposing the sequences into its 'inherent' sinusoid components. This implies a specific definition of frequency. The Fourier Transform assumes that the frequency content is constant for the whole length of the sequence, so for discontinuous or non-stationary functions the extracted frequencies may not be accurate. Techniques to deal with this limitations (like the Windowed Fourier Transform) have been designed over the years, but that is not our focus.

The perspective of the Haar wavelet decomposition was discussed in chapter 1. The focus then was on using the Haar projection operators to smooth sequences by the removal of small scale structures. The Haar projection operators also define a multi-resolution interpretation of a sequence. The difference between the 'smoothed' sequence at one level and the even 'smoother' sequence at the next level contains the higher resolution detail that had to be removed. These difference sequences (the noise at each level) are expressed in terms of translated scalings of the Haar wavelet function, $\psi(t)$. The noise coefficients $\beta_{k,i}$ at a specific level are expressed in terms of the coefficients of the projection of the input sequence onto the previous subspace.

$$\begin{aligned}\left(P_{k-1}-P_k\right)x(t) &= \sum_{i=0}^{N_{k-1}} \beta_{k,i}\psi\left(2^{-k}t-j\right) \text{ and}\\ \beta_{k,i} &= \frac{1}{2}\left(\alpha_{k-1,2i}-\alpha_{k-1,2i+1}\right)\end{aligned}$$

where

$$i = 0\ldots N_k - 1 \text{ with } N_k = \frac{N_{k-1}}{2}$$

The sequences $\left(P_{k-1}-P_k\right)x(t)$ are the detail levels of a multi-resolution decomposition. Each level consists of the detail at a specific scale and lie in the monotonicity classes $\mathcal{M}_{2^{k-1}}$. The original sequence can be reconstructed by adding all the detail levels to the residual sequence, $P_N x(t)$.

The detail levels for the Haar multi-resolution decomposition will always sum to zero as the basis function (the Haar wavelet) is an upward pulse followed by an equal magnitude downward pulse. This implies that any impulse (which by definition is either positive or negative) in a sequence will be characterized in the detail levels by an Haar wavelet centered around zero. As a result, there is a negative value in the detail level next to a positive value at all positions where an impulse was smoothed ('ringing').

Some other problems with the Haar wavelet decomposition, like the spreading of impulse energy and the phase problem, were already discussed. There may exist techniques that attempt to deal with these problems, but that is not important for our discussion.

We want to define a decomposition using the *LULU* operators that is analogous to the Fourier and Wavelet decompositions (this was hinted to in figure 12 of chapter 1). The *LULU* operators removes pulses up to a specific width from a sequence. L_nU_n and U_nL_n removes all upward and downward pulses of width $\leq n$. This leads one naturally to consider decomposing a sequence of length N into N sequences of length N, where each output sequence contains only the pulses at a specific resolution. We are working with absolutely summable bi-infinite sequences, so with a sequence of length N it is meant that the non-zero part is of length N. The zeros on either side extend to infinity.

In essence, we want to decompose a sequence fully into its component pulses, which are then grouped according to pulse width. We do so using the discrete pulse transform (or *dpt*) [11, 9]. The *dpt* is now defined followed by a discussion on what exactly it does and its implications.

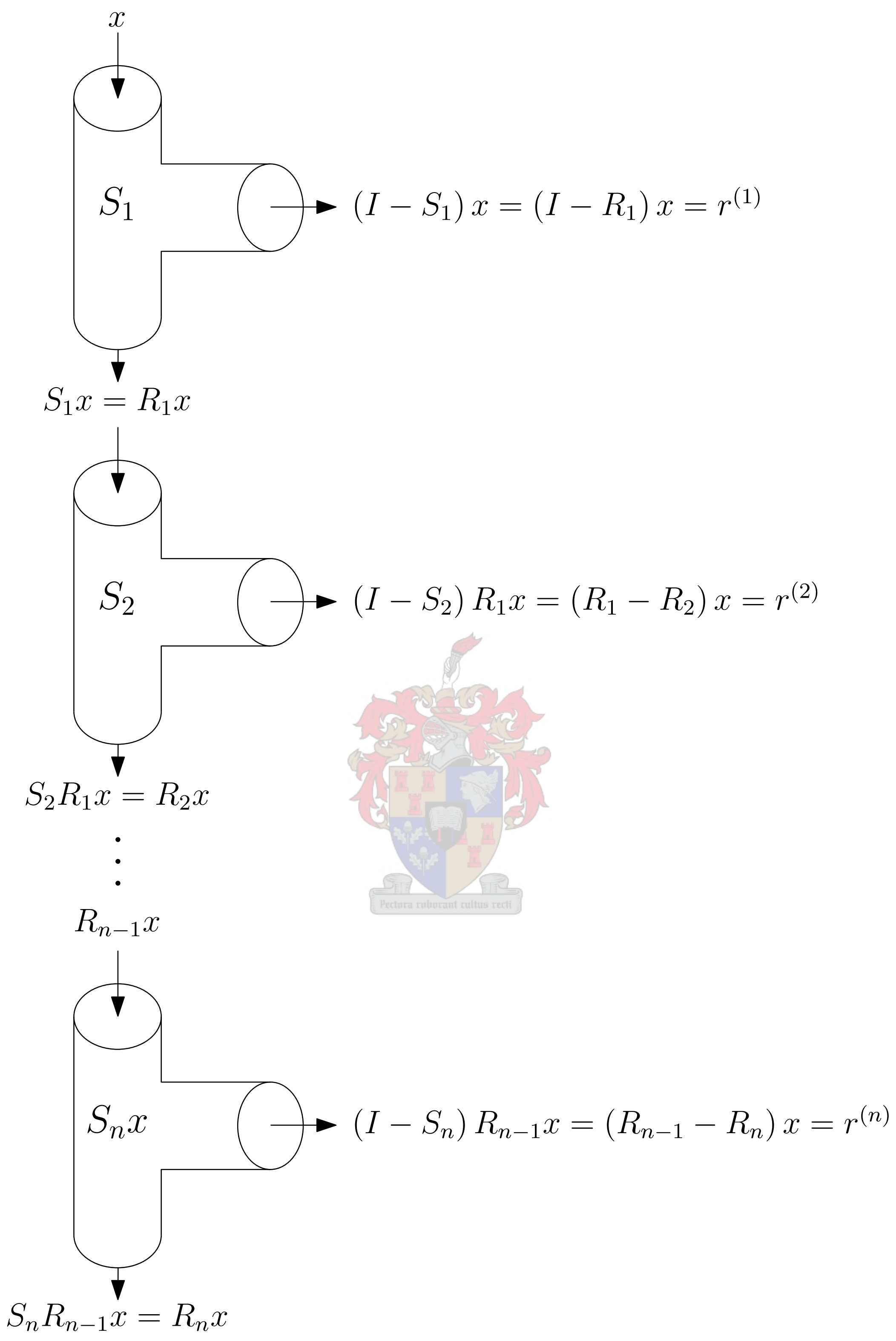
DEFINITION 3.1. The discrete pulse transform of a sequence x with respect to a set of operators $[S_1, \ldots, S_N]$, $\text{DPT}_{S^N}\ x$, is a mapping of x to the vector of sequences $\left\{r^{(n)}\right\}$, where:

(1) The set of operators $\{R_n\}$ are defined recursively by $R_n = S_nR_{n-1}$ with $R_0 = I$.
(2) $r^{(n)} = D_nx = (I - S_n)\,R_{n-1} = (R_{n-1} - R_n)x$ for $0 < n \leq N$ are the individual resolution levels.
(3) $r^{(0)} = D_0x = S_NS_{N-1}\ldots S_1 = R_Nx$ is the residual sequence remaining after the last operator is applied.
(4) $[S_1, \ldots, S_N]$ is a set of fully trend preserving LULU operators that are smoothers and separators (i.e. $S_i \in \mathcal{L}$). Also, they must separate a sequence $x \in \mathcal{M}_{n-1}$ into a smoother part $S_nx \in \mathcal{M}_n$ and a rougher part $(I - S_n)x \in \mathcal{M}_{n-1}$.

In property 2 we see that the resolution level $r^{(i)}$ is just the difference between the consecutive smoother interpretations R_{i-1} and R_i. It consists of the pulses of width i, which had to be removed to map the sequence to the smoothness class $\mathcal{M}_n$.

In the discrete pulse transform the set of separators $\{S_n\}$ are applied one after the other. For convenience we define the recursive operators $\{R_n\}$ in terms of these separators. The discrete pulse transform is basically a cascade of smoothing operators. The component of the sequence removed at each level is the corresponding detail level. This is pictured in figure 23.

The first separator smooths the sequence to $\mathcal{M}_1$. The resolution level $r^{(1)}$ then consists of what was removed in that step: the pulses of width 1. Now it is the second separator's



x
S_1
$(I - S_1)\, x = (I - R_1)\, x = r^{(1)}$
$S_1 x = R_1 x$
S_2
$(I - S_2)\, R_1 x = (R_1 - R_2)\, x = r^{(2)}$
$S_2 R_1 x = R_2 x$
$R_{n-1} x$
$S_n x$
$(I - S_n)\, R_{n-1} x = (R_{n-1} - R_n)\, x = r^{(n)}$
$S_n R_{n-1} x = R_n x$

FIGURE 23. Discrete pulse transform cascade diagram

turn. S_2 separates the signal remaining into a smooth part $R_2x \in \mathcal{M}_2$ and the width 2 pulses , $r^{(2)} \in \mathcal{M}_1 \sim \mathcal{M}_2$. And so on. We see that after each stage the smoothed part cascades on to the next separator, forming a multi-stage process that terminates at the last separator S_N. At each stage another separator gets a chance to extract more noise from the sequence. Each separator has a different interpretation of noise. This results in a multi-resolution decomposition of a sequence which is unique for each set of operators $\{S_n\}$.

The decomposition procedure, as defined by the *dpt,* stops at a level N. There is possibly still signal remaining, which is called the residual sequence, $r^{(0)} = R_Nx$. When N is chosen large enough the residual is always zero.

THEOREM 3.2. *For an absolute summable sequence, x, of length N and the discrete pulse transform $DPT_{S^N}x$, the residual $r^{(0)} = R_Nx$ is zero.*

PROOF. Sequence x is of length N. Let i be such that the non-zero part of our input sequence form the sub-sequence $[x_i, \dots, x_{i+N-1}]$. The operator S_N is fully trend preserving, therefore the zero parts before and after this sub-sequence must remain unchanged: $x_j = 0$ for $j \notin [i, i+N-1]$.

Because $S_Nx \in \mathcal{M_N}$ the set $[x_{i-1}, \dots, x_{i+N}]$ is monotone. The first and last element of this set is zero, therefore every element of this set is zero. Thus sequence x is zero everywhere. □

Until now the assumption was that sequences are in ℓ_1. In chapter 1 it was mentioned that sometimes we will be interested in analyzing sequences where only the total variation $\sum_i |x_i - x_{j-1}|$ is bounded. An infinite sequence where the total variation is bounded can be seen as a section of finite length N, preceded and followed by constant sections. The length of the sequence is defined as the length of the non-constant section. We can apply the discrete pulse transform to such sequences. The residual $r^{(0)}$ is not necessarily zero, even when many levels are decomposed. The following theorem can be proven in the same way as theorem 3.2.

THEOREM 3.3. *For a sequence, x, of length N with bounded total variation and the discrete pulse transform $DPT_{S^N}x$, the residual $r^{(0)} = R_Nx$ is monotone.*

This implies that for a sequence of length N :

$$\dots, \quad a, \quad x_0, \quad \dots, \quad x_{N-1}, \quad b, \quad \dots$$

The residual after a N level decomposition is a monotone sequence with a value of a extending to the left and a value of b extending to the right of the sequence. For all R_m with $m \geq N$, one gets $R_mx = R_Nx$ (theorem 2.5).

From property 4 we see that a set of operators $\{S_i\}$ map sequences into progressively higher smoothness classes by separating the sequence into a smoother and a rougher part (figure

24). This is analogous to how the wavelet decomposition decomposes a sequence into a set of sequences of different resolutions. An important difference is that in the wavelet decomposition the resolution of every level is half that of the previous one (the decrease is geometrical), whereas in the discrete pulse transform the resolution decreases linearly.

Now a discrete pulse transform based on a specific set of operators is created. The sets of operators $\{L_nU_n\}$ and $\{U_nL_n\}$ are a natural choice as both of these sets satisfy property 4 of definition 3.1. We will use L_nU_n, i.e. create the discrete pulse transform: $\mathrm{DPT}_{L_NU_N}$. The *dpt* based on U_nL_n is conceptually identical, the only difference is that the bias is opposite to that of the *dpt* based on L_nU_n. Recall that the idea is to separate a sequence into pulses with different resolutions. Following the definition of the *dpt,* this is done by iteratively applying the L_nU_n operator to remove pulses of each width starting with those of single width.

For the set of operators $\{L_nU_n\}$ the recursive operators will be the operators C_n defined in chapter 2. After application of C_n all arcs of width n and smaller have been removed from the input sequence and the sequence is in $\mathcal{M}_n$. When this is followed by $L_{n+1}U_{n+1}$ all arcs of width $n+1$ are also removed. The pulses of width $n+1$ can then be found on the $(n+1)$-th resolution level:

$$r^{(n+1)} = (I - L_{n+1}U_{n+1})\, C_n x = (C_n - C_{n+1})\, x$$

In figure 25 a sequence x is decomposed using $\mathrm{DPT}_{L_3U_3}$ into the detail levels $r^{(1)}$, $r^{(2)}$, $r^{(3)}$ with residual sequence $r^{(0)} = C_3x$. At each level in the decomposition, the remaining smooth part was split into a smoother and noisier part such that $C_ix = C_{i+1}x + r^{(i+1)}$. The detail levels are the pulses that had to be removed to map the sequence to the next smoothness class.

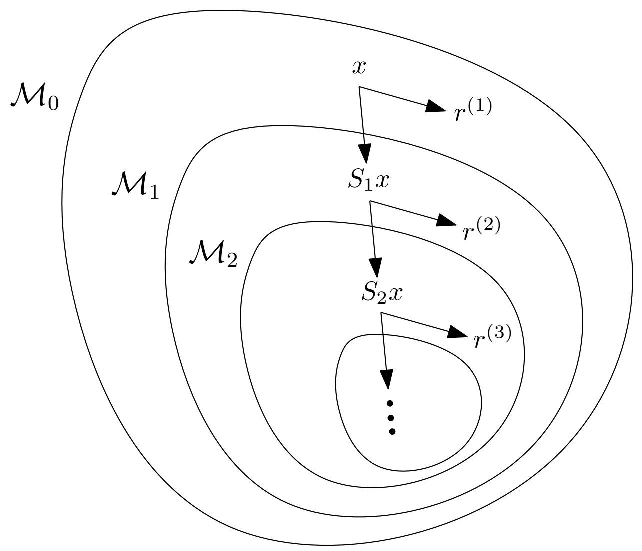


FIGURE 24. The resolution levels are in different smoothness classes.

One can slightly relax the conditions of definition 3.1 by using either $\mathcal{M}_n^+$ or $\mathcal{M}_n^-$ as our smoothness classes; the sequence is then decomposed into resolution levels consisting of upward pulses or of downward pulses respectively.

In figure 12 of chapter 1, a sequence was smoothed with L_4 by removing all upward pulses of width up to 4. The four sequences $(L_{i-1} - L_i)\, x$ with $i = 1, 2, 3, 4$ contain the pulses removed of each width and are the resolution levels of a discrete pulse transform based on L_4. The sequence $L_4 x$ is the residual sequence of DPT_{L_4}.

The input sequence to the discrete pulse transform can be reconstructed by summing all the detail levels and the residual. In chapter 5 we discuss practical uses of sequence reconstruction.

$$x = (I - (R_1 - R_1) \ldots - (R_N - R_N))\, x = R_N x + \sum_{i=1}^{N} (R_{i-1} - R_i)\, x = r^{(0)} + \sum_{i=1}^{N} r^{(i)}$$

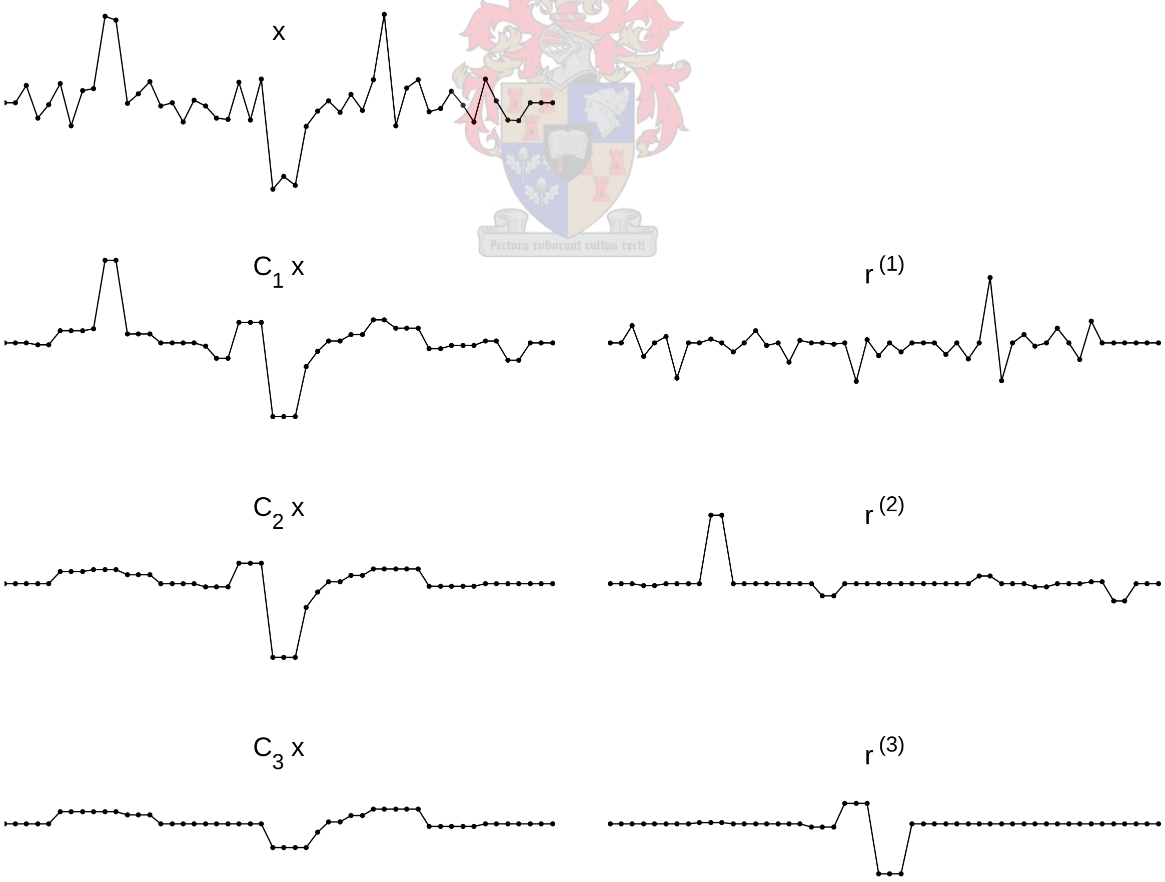


FIGURE 25. Discrete pulse transform $\mathrm{DPT}_{L_3 U_3}$ of a sequence x. The residual $r^{(0)} = C_3 x$.

If N is larger or equal to the length of the sequence or if $R_N x = 0$ the sequence is decomposed by simply adding all the resolution levels together.

$$x = \sum_{i=1}^{N} r^{(i)}$$

When processing a sequence one may wish to only use some of the resolution levels in the reconstruction or use different weights for some levels. We will see that under certain conditions one can do reconstructions like this and even have them decompose consistently again. This is remarkable for a procedure built on non-linear operators.

3.1. Consistency

A set of pulses obtained from a discrete pulse transform is modified and then the sequence is reconstructed by adding together the pulses. With consistency it is meant that if this reconstructed sequence is decomposed using the same discrete pulse transform as before the extracted list of pulses will be the exact same pulses used in the reconstruction.

Definition 3.4. A *dpt* has basic consistency if and only if for $\mathrm{DPT}_{S_N} x \to \left\{ r^{(i)} \right\}$ the reconstruction

$$x^* = \sum_{i=0}^{N-1} \alpha_i r_i \quad \text{where } \alpha_i \geq 0$$

decomposes consistently. I.e. $\mathrm{DPT}_{S_N} x^* \to \left\{ \alpha_i r^{(i)} \right\}$

When a dpt has basic consistency one can scale each resolution level by its own non-negative constant and still have the reconstruction decompose consistently. In other words, scaling the resolution levels like that will not change the interpretation of what is signal and what is noise. First the basic consistency of the discrete pulse transform based on U_n must be proven, the case for L_n follows by duality. . The following theorem is proved by Rohwer [9].

Theorem 3.5. *Let $x \in \mathcal{M}_{n-1}$, and A, B ntp. For all α, $\beta \geq 0$, if A commutes with U_n then:*

$$U_n \left(\alpha A + \beta B U_n \right) x = \alpha U_n A x + \beta B U_n x$$

The following lemma is a direct result of theorem 1.31 and corollary 1.32.

Lemma 3.6. *$U_m = U_m U_n$ for $n \leq m$.*

We now prove the basic consistency of U_n.

Theorem 3.7. *(U_n dpt basic consistency theorem). Let $x \in \mathcal{M}_0$, and $DPT(x) = [D_1 x,\ D_2 x,\ \ldots,\ D_N x,\ D_0 x]$ with $D_0 x = U_N x$ and $D_i x = \left(U_{i-1} - U_i \right) x$. If $\alpha_i \geq 0$, then $z = \sum_{i=1}^{n} \alpha_i D_i x$ is decomposed consistently, for $n \leq N$*

PROOF. Let $z = \sum_{i=1}^{n} \alpha_i D_i x$ with

$$D_i = (I - U_i)\, U_{i-1} = U_{i-1} - U_i \tag{3.1}$$

all ftp.

Since $U_{i-1} = U_{i-1} U_m$ for all $i \geq m+1$

we have

$$\sum_{i=m}^{n} a_i D_i x = \left(\sum_{i=m}^{n} \alpha_i (I - U_i)\, U_{i-1} \right) x = A_m U_{m-1} x$$

where

$$A_m = \left(\sum_{i=m}^{n} \alpha_i (I - U_i)\, U_{i-1} \right) \tag{3.2}$$

is ntp since it is a convex combination of ftp operators.

Assume that $U_{j-1} z = \left(\sum_{i=j}^{n} \alpha_i D_i \right) U_{j-1} x$ for $j < n$. This is clearly true for $j = 1$.

$$\begin{array}{rcll}
U_j z & = & U_j \left(\alpha_j D_j + \left(\sum_{i=j+1}^{n} \alpha_i D_i \right) \right) U_{j-1} x & \\
& = & U_j (\alpha_j D_j + A_{j+1} U_j)\, x & \text{using eq. 3.2} \\
& = & (\alpha_j U_j D_j + A_{j+1} U_j)\, x & \text{using theorem 3.5} \\
& = & (\alpha_j U_j (I - U_j)\, U_{j-1} + A_{j+1} U_j U_{j-1})\, x & \text{using eq. 3.1 and lemma 3.6} \\
& = & (\alpha_j U_j (I - U_j) + A_{j+1} U_j)\, U_{j-1} x & \text{using lemma 3.6} \\
& = & 0 + A_{j+1} U_j x & \text{using lemma 3.6 and } U_j (I - U_j) = 0 \\
& = & \left(\sum_{i=j+1}^{n} \alpha_i D_i \right) U_j x & \text{using eq.3.2}
\end{array}$$

Then $D_j z = (U_{j-1} - U_j)\, z = \alpha_j D_j x$

The theorem now follows by mathematical induction. □

The proof for the discrete pulse transform based on L_n is very similar due to the duality of L_n and U_n.

It is also possible for a *dpt* to have an even stronger consistency property:

DEFINITION 3.8. A *dpt* has full consistency if and only if for $\mathrm{DPT}_{S_N} x \rightarrow \{ r^{(i)} \}$ the reconstruction

$$x^* = r^{(0)} + \sum_{i=1}^{N} A_{r^{(i)}} r^{(i)}$$

where $A_{r^{(i)}}$ is an operator that multiplies each pulse by its own non-negative constant, decomposes consistently. I.e. $\mathrm{DPT}_{S_N} x^* \to \left\{A_{r^{(i)}} r^{(i)}\right\}$

This differs from basic consistency as one is now allowed to multiply every single pulse by its own non-negative constant. This allows one to selectively enhance pulses without changing the signal/noise interpretation. See chapter 5 for a discussion on how this is useful for the processing of sequences.

The full consistency of the discrete pulse transform based on L_nU_n and U_nL_n was conjectured in Rohwer [9] as 'The Highlight Conjecture'. The term highlight refers to how full consistency enables the selective highlighting of single pulses or groups of pulses without distorting the sequence or its decomposition. Rohwer and Harper [10] provide a proof for this conjecture. Laurie has given an alternative proof.

3.2. Efficient implementation

When calculating the discrete pulse transform of a sequence it is necessary to calculate S_nx for $x \in \mathcal{M}_{n-1}$ at each level n. The operators S_i are in $\mathcal{L}$ and as such consists of compositions of the basic operators, L_n and U_n. A straightforward implementation of the discrete pulse transform using the definition of the operators (def 1.29) results in a computational complexity of $O(N^3)$ for a full decomposition where N is the length of the non-zero part of a sequence. By making use of some higher order properties of the LULU-operators and clever uses of data structures it is possible to construct $O(N^2)$ and $O(N)$ algorithms.

Recall that, U_n is the dual of L_n and U_nL_n is the dual of L_nU_n therefore:

$$U_nx = -L_n(-x) \tag{3.3}$$

$$U_nL_nx = -L_nU_n(-x) \tag{3.4}$$

Only the operator L_n and the pulse transform using U_nL_n are mentioned with regard to improving the running times because equations (3.3) and (3.4) provide an easy way to convert to the dual versions.

3.2.1. Faster versions of the basic smoothers.

The basic operator, L_n, is defined by:

$$\begin{aligned} (L_nx)_i &= \left(\bigvee^{n}\bigwedge^{n} x\right)_i \\ &= \max\left\{\min\left\{x_{i-n},\ldots x_i\right\}, \min\left\{x_{i-n+1},\ldots x_{i+1}\right\},\ldots,\min\left\{x_i,\ldots,x_{i+n}\right\}\right\} \end{aligned} \tag{3.5}$$

Implementing this as two loops results in a running time of $O(nN)$ for a length N sequence. For the N-level discrete pulse transform one has to calculate $L_1, L_2 \ldots L_N$. The worst case running time is: $O(1N + 2N + \ldots + N^2) = O\left(\frac{1}{2}N^2\,(N+1)\right) = O\left(N^3\right)$.

In chapter 1 we proved that the operators L_n and U_n remove upward and downward pulses of width n respectively if narrower pulses have already been removed.

The pulse transform removes the pulses from the signal starting with pulses of width 1 up to pulses of the width of the whole signal. When L_n must be calculated all upward pulses with width less than n have been removed already. We thus have a guarantee on the minimum monotonicity class of the sequence to be smoothed. It is now only necessary to test if each upward pulse remaining in the signal is of length n and then remove it if it is. Testing if an upward pulse is of length n is much simpler than implementing equation 3.5. This next corollary follows directly from theorem 1.33

COROLLARY 3.9. *When $x \in \mathcal{M}_n^+$ smoothing with L_n implies that for each i, if the condition*

$$x_i < x_{i+1} > x_{i+1+n} \tag{3.6}$$

holds then a pulse of width n and height $(x_{i+1} - \max\{x_i, x_{i+1+n}\})$ can be removed at position $i+1$:

$$(L_n x)_i = \max\{x_i, x_{i+1+n}\} \text{ for } j \in [i+1, i+n] \tag{3.7}$$

L_n leaves the sequence unchanged at other places.

Using this we can calculate L_n in $O(N)$ time if the sequence $x \in \mathcal{M}_n^+$. The operator U_n can be calculated in a similar way: only the inequalities change direction and the maximum operator is replaced by a minimum.

3.2.2. $O(N^2)$ time pulse transform.

Implementing the discrete pulse transform using the more efficient versions of the basic operators will result in a worst and average case running time of $O\left(N^2\right)$. We can however still improve on this.

We know that if an upward pulse exists, its left edge will be at a position where $x_i > x_{i-1}$. It is clear from corollary 3.9 that every pulse removal destroys at least one difference in the sequence and never creates a new difference: after a pulse removal there is either one or two more values of i where $(\Delta z)_i = z_{i+1} - z_i = 0$ for the smoothed sequence. A sequence of length N has at most $N+1$ differences. We also note that the last pulse removed will always destroy two differences. Thus, a sequence of length N has at most N pulses total in all the resolution levels.

By keeping a list of positions where $x_i < x_{i+1}$ and removing items from these lists as the sequence is smoothed there are less positions where pulses can possibly be removed. The condition 3.6 can only be true for positions i in this list. As the pulse transform progresses into higher levels there are fewer differences left and therefore less work to do.

It is thus possible to improve upon the naive implementation by using the above ideas together with the fast versions of the basic smoothers, U_n and L_n. We will create a more efficient version of $\mathrm{DPT}_{U_N L_N}$.

Create a list i to hold the positions in the vector where the sequence differs from its successor. For every element x_i of the input data series x: if x_i differs from its successor then i is added *in order* to the list of positions, $\{p_i\}$, such that:

$$p_{j+1} > p_j$$

Now for each resolution level in the decomposition, n, we traverse through the sequence removing n-pulses as we find them. If two n-pulses of different sign are next to each other in the sequence we remove the upward pulse first because of the bias of $U_n L_n$ (downward pulse first for $L_n U_n$). Algorithm 1 gives the necessary steps to do this in detail.

The running time for this algorithm is faster in the worst and average case scenarios as one only needs to test for pulses where differences still exist in the sequence. The worst case running time of $O(1+2+\ldots+N) = O\left(\frac{1}{2}\left(N^2+N\right)\right)$ takes place when there is one pulse on each level, forcing every pulse of width n to be tested for removal on the $n-1$ preceding resolution levels.

In real-world signals most of the pulses are extracted in the first few levels, reducing the amount of unnecessary calculations dramatically. For sequence with random noise, about half of the pulses lie in the first resolution level. Hence, the real expected running time is much more promising. Experimentation yields an average running time of $O\left(N^{1.2}\right)$.

It is also important to note that the worst case running time of $O\left(\frac{1}{2}\left(N^2+N\right)\right)$ is for a full N level decomposition of a length N sequence. In practice one often only need to

Algorithm 1 $U_n L_n$ discrete pulse transform

```
(1) Create list, $\{p_i\}$, that holds positions of differences in input sequence, $x$.
(2) Define: (with reference to equation 3.6)
    (a) $\mathrm{cap}_n(j)$ is true if and only if $(x_{p_i} < x_{p_i+1} > x_{p_i+1+n})$
    (b) $\mathrm{cup}_n(j)$ is true if and only if $(x_{p_i} > x_{p_j+1} < x_{p_i+1+n})$
    (c) Removal of a pulse happens as specified by equation 3.7 and its dual. Information on removed pulses are added to some type of register.
(3) For each level, $n$, in decomposition:
    (a) Create new empty position list $\{p_i^*\}$.
    (b) For each, $j$, in position list $\{p_i\}$:
        (i) If $\mathrm{cap}_n(j)$ then remove $n$-pulse starting at $p_j+1$ from $x$.
        (ii) If $\mathrm{cup}_n(j)$ and not $\mathrm{cap}_n(j+1)$ then remove $n$-pulse starting at $p_j+1$ from $x$.
        (iii) If $\mathrm{cup}_n(j)$ and $\mathrm{cap}_n(j+1)$ then remove $n$-pulse starting at $p_{j+1}+1$ from $x$. If $\mathrm{cup}_n(j)$ is still true then remove $n$-pulse starting at $p_j+1$ from $x$.
        (iv) If no pulse was removed add $p_j$ to the end of the position list $\{p_i^*\}$.
    (c) Replace $\{p_j\}$ with new position list $\{p_i^*\}$.
```

decompose up to a resolution level M which can possibly be much smaller than N. The worst case running time for the truncated version is:

$$\begin{aligned} O\left(N+(N-1)+\ldots+(N-M+1)\right) &= O\left(\frac{1}{2}\left(N^2+N-(N-M)^2-(N-M)\right)\right) \\ &= O\left(\frac{M}{2}\left(2N+1-M\right)\right) \end{aligned}$$

The only temporary computer memory required is for the list of positions. This lists changes, but it is possible to update it in place. The maximum number of differences is equal to the length of the sequence, thus the temporary storage required is of order N.

3.2.3. Linear time pulse transform.

The linear time algorithm is based on many of the same ideas as above but uses more advanced data structures to prevent unnecessary work from being done. In algorithm 1 in section 3.2.2 the main problem is that all remaining differences are checked at every level to see if a pulse must be removed, which directly leads to the worst case running time of $O\left(N^2\right)$.

Dirk Laurie designed an algorithm where a clever use of data structures prevents the need to test a pulse for removal more than once (he dubbed this the Roadmaker's algorithm)[11]. We now discuss this algorithm in detail. A data structure acts as a *schedule of features* to be removed. The *features* are pulses as defined in chapter 1. The pulses in the schedule are ordered by priority; those with highest priority need to be removed from the sequence first. As the high priority pulses are removed new ones may be created, which are then added to the schedule. Other pulses may be destroyed by the removal of pulses (a direct consequence of the fundamental ambiguity) and must then be removed from the schedule. Recall that in the discrete pulse transform, a set of operators are used to smooth a sequence into progressively smoother smoothness classes. These operators fully decide the priority of the different width pulses. For $\text{DPT}_{L_N U_n}$ the highest priority pulses are the downward pulses of width 1 followed by the upward pulses of width 1, and so on.

A feature cannot exist at any place in the sequence. From corollary 3.9 we see that an arc, where a feature can be removed, is characterized by two local differences of opposite sign with a section of zero difference in-between. A *catalogue of non-zero differences* is created to keep track of all possible arc edges. A catalogue entry, with position p, is initially created where the local difference d is non-zero:

$$d = \Delta x_p = x_{p+1} - x_p \neq 0$$

Figure 26a depicts a catalogue entry as consisting of:

- The position of the difference, p.

- The difference value, $d \neq 0$.
- Pointers to the next and previous catalogue entries, defining a linked list of catalogue entries. This allows the removal and insertion of entries in $O(1)$ time. The order in the list is determined by the position value: non-zero differences later in the sequence have entries later in the catalogue.
- A pointer to a feature in the schedule is present if the catalogue entry forms part of the left edge of an arc.

A feature consists of (figure 26b):

- Pointers to the next and previous features scheduled for removal. The previous feature has higher removal priority and the next feature has a lower priority. Features can be inserted or deleted from this linked list in $O(1)$ time.
- A pointer to the catalogue entry corresponding the left edge of the arc. Recall from corollary 3.9 that the presence of an arc shows where a feature (pulse) can be removed.

The schedule holds the prioritized list of features. There are two levels of priorities. Smaller width features have higher priorities than larger width features. Furthermore, for equal width features those closer to the front of the priority list have higher priority than those near the back. The schedule can be implemented as a list of double-ended queues: one for each resolution level. When a feature of a certain width is detected, it can be added to either the front or back of the corresponding dequeue in constant time. Whether a feature is added to the front or back of this queue depends on the priorities as implied by the operator R_N (refer to example 1.38 for a discussion on pulse removal priorities).

We will now see how the properties of a pulse can be determined for a specific feature using the associated catalogue entry. A feature only exists where there is an arc in the sequence.

p	$\in \mathcal{Z}$	position
d	$\in \mathcal{R}$	difference
prev	●	previous catalogue entry
next	●	next catalogue entry
f	●	corresponding feature

(a) Data structure for a catalogue entry

prev	●	previous feature
next	●	next feature
c	●	corresponding catalogue entry

(b) Data structure for a feature in the schedule

FIGURE 26. Data structures for features and catalogue entries. A dot indicates a pointer to another structure.

Therefore, for feature f with corresponding catalogue entry, $a = f.c$, we have

$$(a.d) \times (a.\text{next}.d) < 0$$

because an arc is characterized by two consecutive differences of opposite sign. The corresponding pulse has position of its left edge, p, equal to the position of the catalogue entry.

$$p = a.p$$

The pulse is upward if the left edge of the arc slopes upward, and negative if the left edge slopes downward. The magnitude is determined using corollary 3.9 and its dual version. The associated pulse then has height, h, where:

$$\begin{aligned} \operatorname{sign}(h) &= \operatorname{sign}(a.d) \\ |h| &= \min\{|a.d|, |a.\text{next}.d|\} \end{aligned}$$

The pulse width, w, is determined by the distance between the two non-zero differences which form the arc:

$$w = a.\text{next}.p - a.p$$

Before any features can be removed, the catalogue of non-zero differences and schedule of features must be initialized. This involves traversing the original data sequence, calculating the finite difference $\Delta x_i = x_{i+1} - x_i$ and adding a catalogue entry whenever this difference is non-zero. Every time a catalogue entry is added its sign is compared with the sign of the previous catalogue entry. When they differ it points to the existence of an arc. A feature is then added to the schedule. This initialization can be done in $O(N)$ time as it is only necessary to consider each element of the input sequence once.

The main decomposition algorithm finds the feature with highest priority and removes the associated pulse from the sequence. The catalogue and schedule are updated to reflect the changes this makes to the sequence. This is repeated until there are no more features left in the schedule. Every arc in the sequence has an associated feature, and thus when there are no more features there are no arcs left in the sequence. The sequence remaining is then the residual sequence, $r_0 = R_N x$, as per definition 3.1 and is globally monotone, or zero if the input sequence was in ℓ_1.

Now we discuss how the removal of a feature from the sequence can change the catalogue and schedule. Refer to figure 27. A feature, f, with highest priority is removed. This removal extracts a pulse by widening an arc. This destroys either the left edge of the arc (when $|a.d| < |b.d|$), or right edge (when $|a.d| > |b.d|$) or both (when $|a.d| = |b.d|$). The corresponding catalogue entries must then be removed. The linked list of catalogue entries allows removals to take place in $O(1)$ time.

The features connected with removed catalogue entries must also be removed from the schedule. These catalogue entries are marked with a cross in figure 27. At some of these places (marked with a square) a new feature might be created. These newly created features

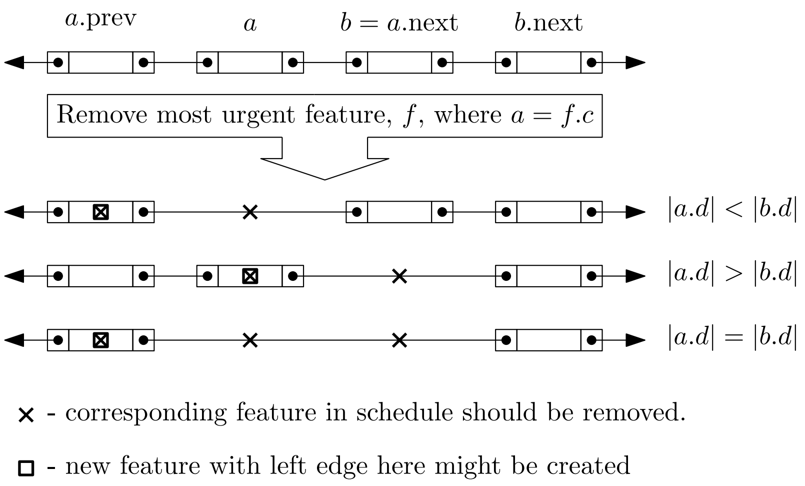

a.prev
a
$b = a$.next
b.next
Remove most urgent feature, f, where $a = f.c$
$|a.d| < |b.d|$
$|a.d| > |b.d|$
$|a.d| = |b.d|$
× - corresponding feature in schedule should be removed.
□ - new feature with left edge here might be created

FIGURE 27. Demonstrates how the removal of a feature will update the catalogue (corresponds with step 5 of algorithm 2)

will have a larger width and thus lower priority than the features removed prior to this. Each of the schedule updates takes $O(1)$ time.

At most two catalogue entries can be affected by the removal of a feature. Therefore, the amount of changes that must be made to the schedule and catalogue is bounded. Every time a feature is removed, at least one difference in the catalogue is destroyed. A signal of length N can have at most $N - 1$ items in the catalogue. Thus, the amount of features in a signal is bounded by the length of the signal. All the features can be removed in $O(N)$ time and thus the amortized running time for each removal is $O(1)$.

The features removed from the signal are added to a table of extracted features, which merely collects the positions, widths, and amplitudes of the pulses. Algorithm 2 describes in detail the steps necessary to remove the most urgent feature in the schedule.

EXAMPLE 3.10. To illustrate the working of the algorithm we will demonstrate the first couple of feature removals for $\text{DPT}_{U_n L_N}$ on a simple data sequence, x_i with $i = 0 \ldots 13$. Figure 28 on page 69 shows the input sequence and the sequences modified by removing the first two features. The corresponding catalogue and schedule entries for each sequence are also given.

Catalogue 1 and Schedule 1 are determined by traversing the input sequence as described above. The schedule shows the positions of the entries in the catalogue which forms the left edges of the features. Priority is ordered from highest to lowest if you read from left-to-right top-to-bottom. There are only four positions in the catalogue where the sign differs for two consecutive differences and thus only four features in the first schedule. The feature at the front of the level 1 double-ended queue is at position 8. Although it was added to the schedule last, it was added to the front because upward pulses have higher priority in $\text{DPT}_{U_n L_N}$. This is removed as per algorithm 2.

Two of the differences in the catalogue was destroyed as a result and are removed as the catalogue is updated (cat. 2). Furthermore, one new feature was created by this process.

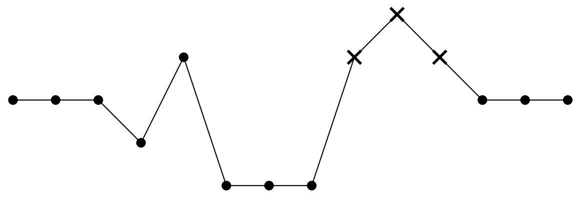
Algorithm 2 Roadmaker's algorithm: removal of most urgent feature

```
(1) Retrieve the most urgent feature, $f$, from the schedule.
(2) Get the corresponding catalogue entry, $a = f.c$,
    and its successor, $b = a.next$
(3) Get the properties of the pulse:
    $w = a.\text{next}.p - a.p$
    $h = \text{sign}(a.d)\ \min\{|a.d|, |b.d|\}$
    $p = a.p$
    and record the pulse in the table of extracted features.
(4) Update the differences $a.d$ and $b.d$
    $a.d \leftarrow a.d - h$   and   $b.d \leftarrow b.d + h$
(5) Either $a.d$ or $b.d$ is now zero, now:
    (a) If $b.d$ is zero: Delete $b.f$ from schedule and $b$ from catalogue and let $b \leftarrow b.next$
    (b) If $a.d$ is zero: Delete $a.f$ from schedule and $a$ from catalogue and let $a \leftarrow a.prev$.
    (c) Delete $a.f$ from the schedule if it exists.
(6) If $(a.d) \times (b.d) < 0$: Insert feature starting at $a$ into the schedule with priority
    determined by pulse width and direction. This pulse has width: $b.j - a.j$, and is
    an upward pulse if $a.d$ is positive, and a downward pulse otherwise.
```

It is an upward arc of width 3 at position 7, so is added to the front of the level 3 dequeue in schedule 2. This finishes the update process.

The next feature to remove with the highest priority is at position 3 according to schedule 2. This removal causes the destruction of one difference (position 3) and the modification of another (position 4). The feature at position 2 does not exist anymore and is removed from the schedule. This leaves us with four differences and two features as is shown in catalogue and schedule 3. The feature with highest priority is now the width 3 arc starting at position 7.

The appendix lists Matlab code that implements the above algorithm. Since Matlab is specialized for the use of matrices, this implementation uses a matrix to play the part of the catalogue and schedule. Each catalogue entry and its associated feature is represented in the matrix by a column vector of 6 elements: position, difference, next catalogue entry, previous catalogue entry, next feature and previous feature. The pointers are indices to columns in the matrix. To implement a separate double ended priority queue for each resolution level, two additional vectors keep indices to the matrix columns corresponding with the first and last feature in every resolution level. The temporary memory requirements for this implementation of the linear time pulse transform is then of the order $8N$ where N is the length of the input sequence.



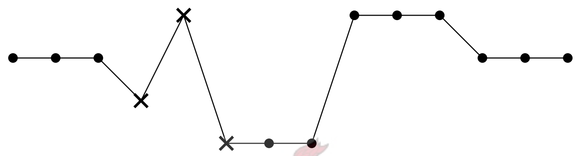
Catalogue 1

position	2	3	4	7	8	9	10
difference	-1	2	-3	3	1	-1	-1

Schedule 1

Level 1:	positions	8	3	2
Level 3:	positions	4		

Remove highest priority arc $\{x_8, x_9, x_{10}\}$, then update catalogue and schedule.



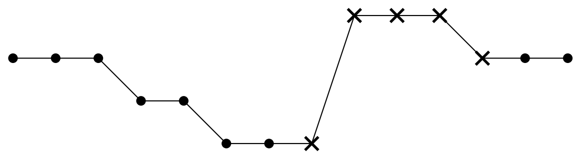
Catalogue 2

position	2	3	4	7	10
difference	-1	2	-3	3	-1

Schedule 2

Level 1:	positions	3	2
Level 3:	positions	7	4

Remove highest priority arc $\{x_3, x_4, x_5\}$, then update catalogue and schedule.



Catalogue 3

position	2	4	7	10
difference	-1	-1	3	-1

Schedule 3

Level 3:	positions	7	4

Remove highest priority arc $\{x_7, \ldots, x_{11}\}$, then update catalogue and schedule.

$\vdots$

FIGURE 28. Extraction of two highest priority pulses from a sequence. The corresponding catalogue and schedule entries are also shown.

3.2.4. Comparison of fast decomposition algorithms.

We shall see that the $O\left(N^2\right)$ algorithm has an average running time of order $N^{1.2}$ for a full decomposition. Furthermore, its temporary memory requirements, of order N, is modest. It might be preferred over the linear time transform when a truncated decomposition of only a few levels are required or when the length of a sequence makes the linear time algorithm's memory requirements excessive.

The linear time pulse transform is much more efficient than the other transforms and should be used in most cases. Its memory requirements of order $8N$ is acceptable for smaller signals. For a truncated decomposition it falls to $6N + 2M$ where $M \leq N$ is the number of resolution levels to decompose.

We know the worst case running times of the various algorithms, but would like to compare performance differences for more typical sequences. We count the number of comparisons needed by the two algorithms to decompose random sequences of different length. To calculate the average number of comparisons we repeat the test for 5000 sequences of each length.

The results are displayed in figure 29. A curve was fitted for each algorithm. On average the $O\left(N^2\right)$ algorithm needs about $12.3N^{1.2}$ comparisons and the linear time algorithm needs about $22.5N$ comparisons. The average case for the $O(N^2)$ algorithm is much better than the worst case. For short sequences ($N < 100$) the two algorithms are about equally efficient.

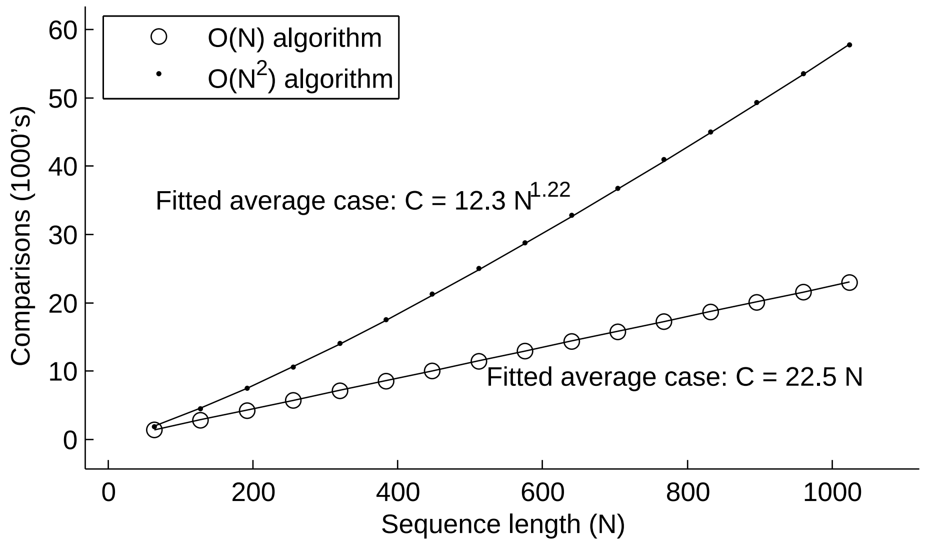


FIGURE 29. Average number of comparisons needed to calculate discrete pulse transform for random sequences.

CHAPTER 4

A multi-scale description of a sequence

We have seen in the previous chapter how the discrete pulse transform decomposes a sequence into resolution levels, with each resolution level consisting of the sum of pulses of a specific width. In this chapter we will further flesh out ideas on how this decomposition gives one a new perspective on a sequence. We do this by focusing on the resolution levels as extracted by the dpt and seeing how they relate to the input sequence.

The discrete pulse transform decomposes a sequence into a set of resolution level sequences. Each resolution level consists of the sum of all the pulses of a specific width. Each pulse in turn is characterized by its position, width and height. We get:

$$\begin{tikzcd}
\mathrm{DPT}_{S_N} x \arrow[d] \\
\text{resolution levels } \left\{ r^{(j)} \right\} \text{ with } j = 1 \dots N \text{ and residual } r^{(0)} \arrow[d] \\
\text{pulses } p_{j,k}, \quad w_{j,k} = j \quad h_{j,k} \text{ with } k = 1 \dots N_j
\end{tikzcd}$$

where the set $\{N_j\}$ specifies how many pulses exist on each resolution level.

DEFINITION 4.1. Every pulse is a sequence, $r^{(j,k)}$, such that

$$r^{(j)} = \sum_{i=1}^{N_j} r^{(j,k)}$$

where we define the pulse sequence (using definition 1.15) as:

$$r_i^{(j,k)} = h_{j,k} \sum_{m=1}^{k} \delta_{i,(p_{j,k}+m-1)}$$

4.1. Constraints on pulses

This section looks at what constraints there are on the pulses of a discrete pulse transform given the existence of another pulse. In other words: if a pulse, i, exists what limits does this place on the properties, w_k, h_k, and p_k, of all pulses $k \neq i$.

The decomposition procedure can only extract pulses at positions where there is a local difference between successive elements in the sequence.

THEOREM 4.2. *If a sequence x is such that $x_i = x_{i-1}$ then no pulse has position i.*

PROOF. The discrete pulse transform is based on operators in $\mathcal{L}$. These operators remove pulses from a sequence by the removal of arcs. An arc as defined in 1.20, is characterized by a segment where either $x_j > x_{j+1} = \cdots = x_{j+k} < x_{j+k+1}$ or $x_j < x_{j+1} = \cdots = x_{j+k} > x_{j+k+1}$. The arc removal operators, L_n and U_n, removes an arc by setting the value of the arc plateau to the value of one of the endpoints (theorems 1.33 and 1.34). The removed pulse then has position, j. This process can only remove differences from the sequence. So if $x_i = x_{i-1}$ then no segment $\left\{ x_{i-1}, \quad x_i, \quad \ldots, \quad x_{i+m} \right\}$ is an arc and no arc removals can change that. Therefore no pulse be extracted at position i. □

The general proof technique we will use is to prove that in certain cases a specific resolution level will be non-zero, whereas the consistency of the discrete pulse transform implies the opposite case, i.e. that the same resolution level is zero. This will yield a contradiction and thereby rule out certain combinations of pulses in the *dpt* of a sequence.

LEMMA 4.3. *If a sequence x is such that $x \in \mathcal{M}_{n-1} \sim \mathcal{M}_n$ then resolution level $D_n x = r^{(n)}$ of the discrete pulse transform is non-zero.*

PROOF. $D_n x = (R_{n-1} - R_n)\, x$ by the definition of the discrete pulse transform.

But, $R_{n-1}x = x$ because $x \in \mathcal{M}_{n-1}$ and R_{n-1} ntp.

From theorem assumptions we have $x \notin \mathcal{M}_n$ but we know that $R_n x \in \mathcal{M}_n$ by definition of *dpt*. Therefore $R_n x \neq Ix$. Thus, $D_n x = (I - R_n)\, x \neq 0$. □

The first theorem disproves the existence of pulses of the same sign immediately following each other.

THEOREM 4.4. *Let x be a sequence with a consistent dpt, $\left\{r^i\right\}$. Choose any two pulses in the dpt. Let the left pulse and the right pulse be described by the tuples, (w_a, h_a, p_a) and (w_b, h_b, p_b) respectively. Then if $\operatorname{sign} h_a = \operatorname{sign} h_b$, then $p_b \neq p_a + w_a$.*

PROOF. Assume that $p_b = p_a + w_a$.

We have already argued why pulses of the same width are not allowed immediately following each other when discussing the effects of L_n and U_n on arcs in section 1.6. Now assume that $w_a \neq w_b$.

Let y be the scaled sum of resolution levels w_a and w_b:

$$y = \frac{1}{h_a} r^{w_a} + \frac{1}{h_b} r^{w_b}$$

with $dpt(y) \to \left\{s^i\right\}$. According to the consistency of the discrete pulse transform, the two pulses $(w_a, 1, p_a)$ and $(w_b, 1, p_b)$ must exist in $\left\{s^i\right\}$.

Looking at $\{y\}_k$ with $k = p_a - w_a, \dots, p_a + 2w_a + w_b$,

$\{y\}_k = [\underbrace{*, \dots, *}_{>w_a}, \underbrace{1, \dots, 1}_{w_a+w_b}, \underbrace{*, \dots, *}_{>w_a}]$, with all $* < 1$.

This is an $(w_a + w_b)$-downward arc (cap). There will be no pulse in the dpt of y starting at position p_b, because there is no difference at that position in the sequence y by theorem 4.2. This is a contradiction. Therefore $p_b \neq p_a + w_a$.. The case where the pulses are negative are handled in exactly the same way. $\square$

THEOREM 4.5. *Let x be a signal with a consistent dpt. No two pulses on any two levels can overlap such that $i + m - n < j < i + m$, where the pulses have positions i and j and lengths m and n respectively.*

PROOF. Assume that the first pulse has position i, length m and height a, and the second pulse has position j, length n and height b. Let $m \geq n$ such that the first pulse is longer or equal to the second one.

Calculate the consistent dpt of x: $dpt(x) \to \begin{bmatrix} r^1 \\ \vdots \\ r^N \end{bmatrix}$

According to the consistency, the sequence

$$y = \frac{1}{a} r^m + \frac{1}{b} r^n$$

must have the dpt: $y \to \frac{1}{a} r^m,\ \frac{1}{b} r^n,\ r^t = 0$ with $t \neq m, n$.

First look at the case where $\text{sign}(a) = \text{sign}(b)$.

Assume that the pulses overlap on the right side of the first pulse, then:

$$i + m - n < j < i + m \tag{4.1}$$

$\{y\}_k = [\dots, \underbrace{*, \dots, *}_{>m}, 1, \dots, 1, \underbrace{2, \dots, 2}_{\alpha}, 1, \dots, 1 \underbrace{*, \dots, *}_{>m}, \dots]$, with all $* < 1$.

The length of the overlap is given by

$$\alpha = i + m - j \tag{4.2}$$

Using this and the assumption 4.1, we get: $0 < \alpha < n$.

Therefore, y contains an α-downwards arc and thus $y \in \mathcal{M}_\alpha \sim \mathcal{M}_{\alpha+1}$. By lemma 4.3 and $0 < \alpha < n$ there exists a whole number $k < n$ such that $r^k \neq 0$ in dpt of y. This is a contradiction of the consistency and thus the assumption 4.1 is false.

Now look at the case where $\text{sign}(a) \neq \text{sign}(b)$. Assume again equation 4.1.

$\{y\}_k = [\dots, \underbrace{*, \dots, *}_{>m}, 1, \dots, 1, 0, \dots, 0, \underbrace{-1, \dots, -1}_{\beta} \underbrace{*, \dots, *}_{>m}, \dots]$, with all $* < 1$.

Due to the specifics of the pulses we get,

$$\beta = n + j - m - i$$

Using this and the assumption, equation 4.1, we get $0 < \beta < n$.

This implies that y is β-monotone and thus $y \in \mathcal{M}_\beta$. There then exists a whole number $k < n$ such that $r^k \neq 0$ in dpt of y. This is a contradiction and thus the assumption 4.1 is false. □

THEOREM 4.6. *Let x be a signal with consistent dpt. Assume there exists two pulses: the first with position i, length m and height a and the second with position j, length n and height b. Assume further that sign$(a) \neq$ sign(b), $m > n$, and that the smaller pulse is located somewhere in the larger pulse, i.e. $i \leq j \leq i + m + n$. Then:*
(a) $i + n \leq j \leq i + m - 2n$
(b) $m \geq 3n$
These constraints are illustrated in figure 30.

PROOF. Calculate the consistent dpt of x: $x \rightarrow \begin{bmatrix} r^1 \\ \vdots \\ r^N \end{bmatrix}$

According to the consistency, the sequence

$$y = \frac{1}{a}r^m + \frac{1}{b}r^n$$

must have the dpt: $y \rightarrow \frac{1}{a}r^m,\ \frac{1}{b}r^n,\ r^t = 0$ with $t \neq m, n$.

$\{y\}_k = [\ldots, \underbrace{*, \ldots, *}_{\geq m}, \underbrace{1, \ldots, 1}, \underbrace{0, \ldots, 0}_{n}, \underbrace{1, \ldots, 1}_{\beta}, \underbrace{*, \ldots, *}_{\geq m}, \ldots]$, with all $* < 1$.

From the specifics of the pulses, we get

$$\alpha = j - i \tag{4.3}$$

$$\beta = i + m - j - n \tag{4.4}$$

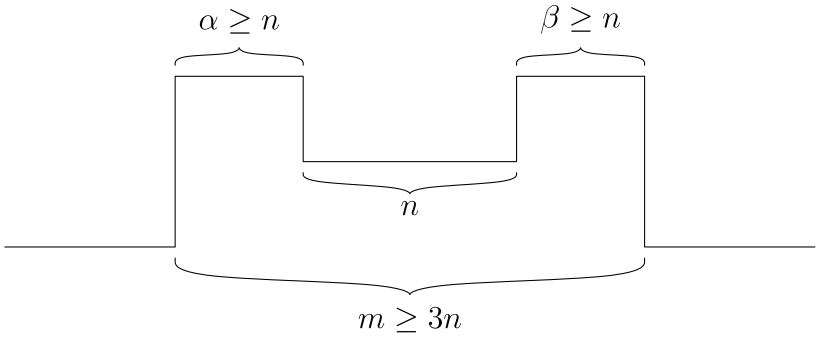


FIGURE 30. Constraints on a pulse within a larger pulse

Because the *dpt* operators are neighbour trend preserving, if either α or β is zero then there can exist no positive pulse at position i in r^m. This contradicts the consistency of the dpt. Therefore $\alpha, \beta > 0$

Now, assume that either $\alpha < n$ or $\beta < n$. Let $c = \min\{\alpha, \beta\}$. Then y has an c-arc and thus $y \in \mathcal{M}_{c-1} \sim \mathcal{M}_c$. By lemma 4.3 we have that $r^c \neq 0$ in the dpt of y. This is a contradiction because all $r^{(i)} = 0$ for $i < n$ by consistency of the dpt. Therefore $\alpha, \beta \geq n$.

(a) now follows directly from this and equations 4.3 and 4.4:

$$\begin{aligned} \alpha = j - i \geq n \quad &\Rightarrow \quad j \geq i + n \\ \beta = i + m - j - n \geq n \quad &\Rightarrow \quad j \leq i + m - 2n \end{aligned}$$

(b) follows from equations 4.3, 4.4 and $\alpha, \beta \geq n$:

$$m - n = \alpha + \beta \geq n + n \quad \Rightarrow \quad m \geq 3n$$

The case where the larger pulse is negative and the smaller pulse is positive is proven in the same way. $\square$

4.2. Properties of the resolution level operators

The resolution levels consist of pulses centered around the zero sequence, and as such the resolution level operators are not axis invariant (i.e $D_k(x + c) = D_k(x) \neq D_k(x) + c$) and thus not smoothers. They are separators.

THEOREM 4.7. *The resolution level operators D_k of a consistent discrete pulse transform are separators.*

PROOF. We first prove idempotency using the fact that all S_i are co-idempotent with root sequences lying in $\mathcal{M}_i$:

$$\begin{aligned} D_k D_k &= (I - S_k) R_{k-1} (I - S_k) R_{k-1} \\ &= (R_{k-1} - R_k)(I - S_k) R_{k-1} \\ &= R_{k-1}(I - S_k) R_{k-1} - R_k (I - S_k) R_{k-1} \\ &= S_{k-1} \dots S_1 (I - S_k) R_{k-1} - S_k S_{k-1} \dots S_1 (I - S_k) R_{k-1} \\ &= (I - S_k) R_{k-1} + S_k (I - S_k) R_{k-1} \\ &= D_k + 0 = D_k \end{aligned}$$

Co-idempotency follows directly from the basic consistency of the discrete pulse transform [10]:

$$\begin{aligned} D_k (I - D_k) x &= D_n \left(\sum_{i \neq k} D_i x + 0 D_k x \right) \\ &= 0 \end{aligned}$$

This proves the theorem. □

All the resolution levels $r^{(i)}$ have pulses of length i if $r^{(i)} \neq 0$:

THEOREM 4.8. *For a discrete pulse transform, each non-zero resolution level $r^{(i)}$ is guaranteed to be in the smoothness class $\mathcal{M}_{i-1}$ and not in any higher smoothness classes $\mathcal{M}_j$ with $j > i-1$. I.e.*

$$r^{(i)} = D_i x \in \mathcal{M}_{i-1} \sim \mathcal{M}_i$$

PROOF. By the definition of the discrete pulse transform we have that $D_i x = (I - S_n) R_{n-1} \in \mathcal{M}_{n-1}$. Now assume that $D_i x \in \mathcal{M}_n$. Because all sequences in $\mathcal{M}_n$ are root sequences of S_n and we are only considering non-zero resolution levels, we get $S_n D_i x = D_i x \neq 0$. By the definition of the *dpt* S_n is a separator, thus $S_n D_i x = S_n (I - S_n) R_{n-1} x = 0$. This is a contradiction. Therefore $D_i x \notin \mathcal{M}_n$, which proves the theorem. □

The discrete pulse transform resolution level operators $D_i = (I - S_i)R_{i-1}$ are all fully trend preserving because the operators S_i are *ftp* (definition 3.1 and theorem 1.45).

This is a very powerful trend preservation property. Every resolution level on its own mimics the trend of the input sequence. If for any i we have $x_{i+1} > x_i$ in the input sequence, then $x_{i+1} \geq x_i$ in all the resolution levels. So, one cannot have a negative pulse on any resolution level start at a position i if $x_i > x_{i-1}$.

As neighbour trend preservation is inherited by any sum of *ntp* operators, the resolution levels modified by any *ntp* operators still have the same local difference structure as the input sequence. In essence we have, for a set of *ntp* operators, $\{T_i\}$:

$$\left(\sum_i T_i r^{(i)}\right)_i \geq \left(\sum_i T_i r^{(i)}\right)_{i+1} \quad \text{if } x_i \geq x_{i+1} \text{ and}$$
$$\left(\sum_i T_i r^{(i)}\right)_i \leq \left(\sum_i T_i r^{(i)}\right)_{i+1} \quad \text{if } x_i \leq x_{i+1}$$

Due to the consistency of the *dpt*, this modified reconstruction will have the same signal/noise interpretation as the original sequence. See chapter 5 for practical uses of creating modified reconstructions.

4.3. Norms and roughness

The amount of content in a specific resolution level can be quantified using a norm. We can express some of these in terms of the individual pulses on the resolution levels.

THEOREM 4.9. *Let $\left\{r^{(j)}\right\}$ be the resolution levels from a discrete pulse transform with the N_k pulses on the j-th resolution level characterized by position, width and height: $\{p_{j,k}\}$, $\{w_{j,k}\} = k$ and $\{h_{j,k}\}$ where $i = 1 \ldots N_j$. Then:*

(1) $\left\|r^{(j)}\right\|_p = (j\sum |h_{j,k}|^p)^{\frac{1}{p}}$
(2) $T(r^{(j)}) = 2\sum_{i=1}^{N_k} |h_{j,k}|$

Proof. The p-norm of a sequence is given by $\|x_i\|_p = \left(\sum_{i=-\infty}^{\infty} |x_i|^p\right)^{\frac{1}{p}}$. The sequence $r^{(k)}$consists of the superposition of non-overlapping pulses and is zero at locations where there are no pulses. Therefore one can split up the calculation of the norm to add each pulse separately:

$$\begin{aligned}\left\|r^{(j)}\right\|_p &= \left(\sum_{k=1}^{N_j}\left(\sum_{i=p_{j,k}}^{p_{j,k}+w_{j,k}+1}\left|r_i^{(j)}\right|^p\right)\right)^{\frac{1}{p}}\\ &= \left(\sum_{i=1}^{N_j} j\,|h_{j,k}|^p\right)^{\frac{1}{p}}\end{aligned}$$

This proves (1).

We can prove 2 in a similar way. We want to determine the value of the local difference for any place in the sequence. As pulses cannot overlap, differences only exists in the sequence at the endpoints of pulses. Also, the only time two pulses on a resolution level can immediately follow each other (i.e. $p_{k,i+1} = p_{k,i} + w_{k,i}$) is if their sign differs. We use this information to get an equation for the local difference at any position i in resolution level $r^{(j)}$:

$$\left|r_i^{(j)} - r_{i-1}^{(j)}\right| = \begin{cases} |h_{j,k}| & \text{if } \exists j \text{ s.t. } i = p_{j,k} \neq p_{j,k-1} + w_{j,k-1} \\ & \quad \text{or } i = p_{j,k} + w_{j,k} \neq p_{j,k+1} \\ |h_{j,k-1}| + |h_{j,k}| & \text{if } \exists j \text{ s.t. } i = p_{j,k} = p_{j,k-1} + w_{j,k-1} \\ 0 & \text{otherwise when no such } j \text{ exists} \end{cases}$$

(2) follows from this for any j:

$$\begin{aligned}T\left(r^{(j)}\right) &= \sum_{i=-\infty}^{\infty} |x_i - x_{i-1}|\\ &= \sum_i |x_i - x_{i-1}| \text{ with } i \in \{p_{k,i}\} \cup \{p_{k,i} + w_{k,i}\}\\ &= 2\sum_{k=1}^{N_j} |h_{j,k}|\end{aligned}$$

□

A direct result of this, is a relation between the total variation of a resolution level and its 1-norm.

Corollary 4.10. $T\left(r^{(j)}\right) = \frac{2}{j}\left\|r^{(j)}\right\|_1$

Recall that the resolution level $r^{(j)} \in \mathcal{M}_{j-1} \sim \mathcal{M}_j$. Corollary 4.10 does not hold for any general sequence in $\mathcal{M}_{j-1}$, instead the factor $\frac{2}{j}$ acts as an upper bound on the ratio of total variation to 1-norm of a sequence [9]:

THEOREM 4.11. *For any non-constant sequence $x \in \mathcal{M}_j$, we have*

$$\frac{T(x)}{\|x\|_1} \leq \frac{2}{j+1} \tag{4.5}$$

The above ratio (equation 4.5) reaches the upper bound of $\frac{2}{j+1}$ for the resolution levels $r^{(i-1)}$. Call this ratio the *'roughness ratio'* of a sequence [5]. The LULU operators smooth a sequence by removing the roughest parts at a specific resolution, causing the roughness ratio to decrease. As measured by this ratio, the part of the sequence removed as noise is as rough as possible for that resolution. There is thus a natural link between the monotonicity class and total variation of a sequence and the concept of roughness. In functional analysis, the total variation is also related to monotonicity and a concept of smoothness: functions of Bounded Variation are almost everywhere differentiable and can be expressed as sums of monotone functions [5].

4.4. A roughness profile

The smoothness of a sequence is a widely used concept but not one that has found one satisfactory definition. Different analysis perspectives have different ideas about what smoothness means. In the LULU and some other related perspective, smoothing a sequence means separating the noise from the signal. Clearly one could argue that a sequence that is left unchanged by what one chooses as your smoother is smooth. Similarly, a sequence where noise can be removed could be argued to possess a degree of roughness. The roughness is quantifiable by the amount of noise removed. It is important to remember the subjectivity of the concepts of signal and noise: every smoothing operator has its own interpretation of the roughness of a sequence.

The Parseval Identity relates the sum of the squares of the Fourier coefficients of a function to the integral of the squared function. For a function with period 1 we have the Fourier series expansion:

$$f(x) = a_0 + \sum_{n=1}^{\infty} a_n cos(2n\pi x) + \sum_{n=1}^{\infty} b_n sin(2n\pi x).$$

The Parseval Identity is then:

$$2\int_0^1 |f(x)|^2 \, \mathrm{d}x = \frac{1}{2}a_0^2 + \sum_{n=1}^{\infty} \left(a_n^2 + b_n^2\right) \tag{4.6}$$

In essence, the integral of the squared function can be understood as a measure of the signal power. The Fourier transform splits a signal into its component harmonics, each of these then contains part of the total signal power. By equation 4.6, this signal power is conserved as the sum of the powers of the individual frequencies must sum to the total. A plot of this quantity versus the frequency is the well-known power spectrum of a sequence.

In a wavelet decomposition we have, by the Pythagoras Theorem for Projections, that the squares of the 2-norms of the resolution levels are preserved:

$$\|r_1\|_2^2 + \|r_2\|_2^2 + \ldots = \|x\|_2^2 \text{ with } r_i = (I - P_i)\, P_{i-1} \ldots P_1 x \tag{4.7}$$

One can understand the 2-norm of a sequence as a measure of the 'energy' content of the sequence, which the wavelet decomposition then separates into the 'energy' content at different scale levels. Equation 4.7 is then a law of conservation of 'energy'.

Both the Fourier and Wavelet decompositions techniques supply some kind of information about the multi-scale structure of the input signal, the exact nature of said information being determined by the choice of perspective. The 'energy' vs scale spectrum then indicates what part of the total signal energy exists at each scale.

The general pattern that emerges is:

$$N(x) = \sum_i N(r_i) \tag{4.8}$$

For some decompositions there exists an associated 'norm' which allows one to see the decomposition as removing information from the sequence layer by layer and that one can quantify the amount of information ('energy') in every layer. The total 'energy' in the sequence is then the sum of the 'energies' of the individual layers, i.e. the total energy is conserved. The word energy is only used to give a flavour of what is happening, the true definition depends on the decomposition in question. After a full decomposition one can see what layers (sinusoid frequency or scale level) contain the largest parts of the information in the input sequence.

Does something similar exist in the LULU perspective? In chapter 1 it was mentioned that the total variation is a natural norm in the LULU framework. We see now that this is because, analogous to the decompositions described above, total variation can be used to visualize the discrete pulse transform as the peeling off of discrete resolution levels with an associated conserved energy content.

For any fully trend preserving operator (which includes all the LULU operators) the total variations of the signal and the noise adds up to the total variation of the input sequence [9].

THEOREM 4.12. *Let S be a fully trend preserving operator. Then, for any sequence $x \in \mathcal{X}$,*

$$T(x) = T(Px) + T((I - P)x)$$

This conservation of total variation is illustrated in figure 31. A sequence is decomposed into three resolution levels using the discrete pulse transforms based on L_nU_n and U_nL_n. At each decomposition level the total variation is split into one part corresponding with

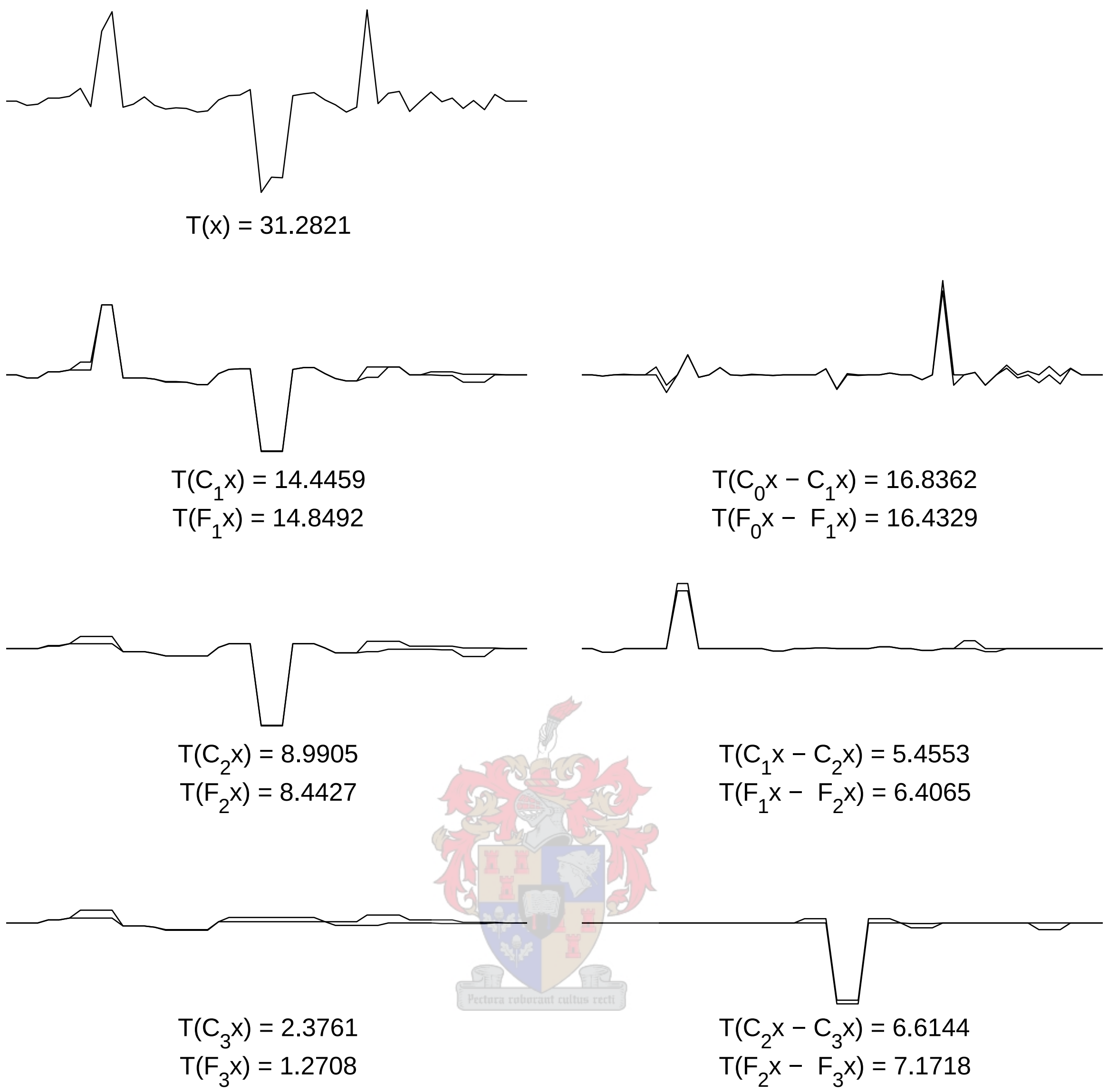


FIGURE 31. 4 level discrete pulse transform, $\mathrm{DPT}_{L_4U_4}$ and $\mathrm{DPT}_{U_4L_4}$ showing how $T(x) = T(Px) + T((I - P)x)$

signal and one part corresponding with noise. For this example we get

$$T\left(R_n x\right) = T\left(S_{n+1} R_n x\right) + T\left(\left(I - S_{n+1}\right) R_n x\right) = T\left(R_{n+1} x\right) + T\left(R_n x - R_{n+1} x\right)$$

where $S_n = L_n U_n$ (or $S_n = U_n L_n$) and $R_n = C_n$ (or $R_n = F_n$) for the dpt based on $L_n U_n$ (or $U_n L_n$).

The following theorem gives the wanted result that the variation 'energy' in the input sequence is conserved with respect to variation 'energy' in the resolution levels. This allows one to compare the contribution of each resolution level to the total variation content of the input sequence.

THEOREM 4.13. *For any sequence $x \in \mathcal{X}$ and a discrete pulse transform DPT_{S_N}, the total variation of the input sequence is equal to the sum of the total variation of the resolution* *levels.*

$$T\left(x\right)=\sum_{i=0}^{N}T\left(D_ix\right)=T\left(D_0x\right)+\sum_{i=1}^{N}T\left(D_ix\right)$$

PROOF.

$$\begin{aligned}
T\left(x\right) &= T\left(S_1x\right)+T\left(\left(I-S_1\right)x\right)\\
&= T\left(R_1x\right)+T\left(D_1x\right)\\
&= T\left(S_2R_1x\right)+T\left(\left(I-S_2\right)R_1x\right)+T\left(D_1x\right)\\
&= T\left(R_2x\right)+T\left(D_2x\right)+T\left(D_1x\right)\\
&\vdots\\
&= T\left(R_Nx\right)+T\left(D_Nx\right)+\ldots+T\left(D_{1x}\right)\\
&= T\left(D_0x\right)+T\left(D_Nx\right)+\ldots+T\left(D_{1x}\right)\\
&= \sum_{i=0}^{N}T(D_ix)
\end{aligned}$$

□

By the consistency of the dpt, this is also true for reconstructions of only a subset of the levels.

COROLLARY 4.14. *The total variation of the sum of a subset of resolution levels equals the sum of the total variations of the levels.*

$$T\left(\sum_{j\in S}D_jx\right)=\sum_{j\in S}T\left(D_ix\right)$$

We see that the LULU operators peel off variation in a sequence to map it to a smoother monotonicity class. We have argued a link between the total variation of a sequence and its roughness. The resolution level versus total variation plot is thus our multi-scale roughness profile. This allows us to compare the roughness content at different scales. Other roughness profiles are also possible, although they will probably lack the above results.

The maximum number of pulses on lower (wider) resolution levels are lower because the pulses are wider and pulses cannot overlap. Thus the total variation of the first levels are often much larger than the later levels. An useful alternative for the roughness profile that corrects for this is the resolution level versus 1-norm plot. Corollary 4.10 gives an equation relating the total variation of the resolution levels to the 1-norm. We see that the 1-norm of the resolution level is the same as the total variation but with a bias proportional to the width of pulses on that level ($\left\|r^{(i)}\right\|_1=\frac{i}{2}T\left(r^{(i)}\right)$). Unfortunately, as expected the above results do not apply in general and therefore the contribution of the resolution levels to the 1-norm of the input sequence can not be defined.

An useful special case where we get a result of the form of equation 4.8 for the p-norms is when all pulses in a discrete pulse transform are of the same sign (when using U_n and L_n for example). We will state the following theorem, whereby an alternative profile can be defined, without proof. For the 1-norm, this measures the contribution of each resolution level to the total area of the input sequence.

THEOREM 4.15. *Let $r^{(i)} = D_i x$ be the resolution levels of a discrete pulse transform DPT_{S_N} of a sequence $x \in \ell_p$. If all the pulses have the same sign (i.e. $r^{(i)} \geq 0$ for all i, or $r^{(i)} \leq 0$ for all i) and the residual $r^{(0)}$ is zero, then:*

(1) $\|x\|_p = \sum_{i=0}^{N} \|D_i x\|_p = \|D_0 x\|_p + \sum_{i=1}^{N} \|D_i x\|_p$
(2) $\left\|\sum_{j\in S} D_j x\right\|_p = \sum_{j\in S} \|D_i x\|_p$

These roughness profiles will now be illustrated with a couple of examples. Figure 32 shows the total variation profile (and its logarithm) for a sequence of uniformly distributed random noise. The total variation is largest in the first resolution level. Thereafter it quickly falls to zero. The logarithm plot allows us to compare the relative magnitudes of the total variation values when they are close to zero.

We analyze what is happening here. Let x be a sequence with a slowly changing or zero underlying trend and added identically independently distributed noise. Now we look at a part of this data series where there is a section of values that are larger than the elements neighbouring the section:

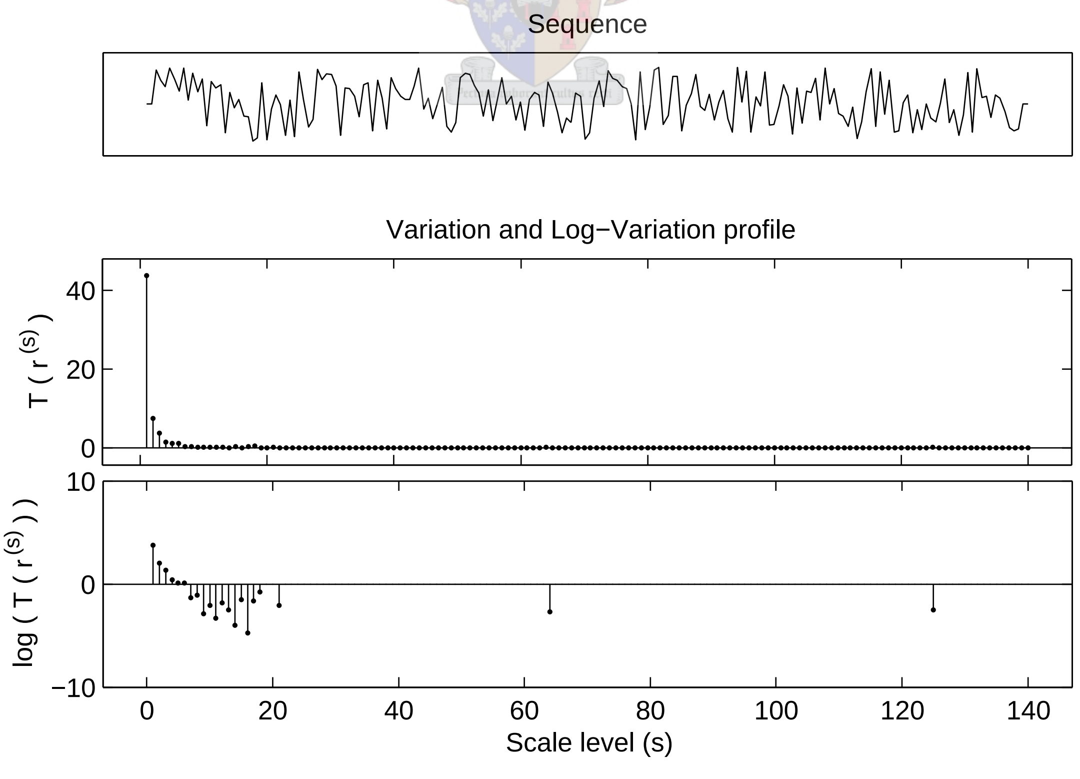


FIGURE 32. Total variation profile

$[\ \dots\ \ *, \alpha,\ \ \gamma_1,\ \ \dots\ \ , \gamma_n\ \ , \beta, *\ \ \dots\ ]$

with, $\gamma_i > \max\{\alpha, \beta\}, \quad i = 1 \dots n$

If the probability of there being one such point (i.e $n = 1$) is p. Then the probability of having n such points in a row is p^n, if the noise is identically independently distributed. The discrete pulse transform will remove a pulse of width n of height $\min\{\gamma_i - max\{\alpha, \beta\}\}$ at any position with probability

$$P(\text{pulse of width } n \text{ at position } i) = sp^n < p^n,$$

where $0 \le s \le 1$ is determined by the $2n$ data points surrounding the area in question on each side. Thus one can expect most of the pulses for a random signal to be in the first detail levels of the discrete pulse transform. The large width pulses that are found in higher levels will most likely have a small height, since the probability p is inversely related to the height of the pulse, $min\{\gamma_i - max\{\alpha, \beta\}\}$.

As another example we decompose a sequence consisting of a fort-like structure. Since all the pulses in the decomposition were positive we show both the total variation and 1-norm profiles (figure 33).

There are four peaks in the profiles, corresponding with pulses of size 1, 16, 95 and 127. There are many width 1 peaks and thus the variation profile shows the largest peak there.

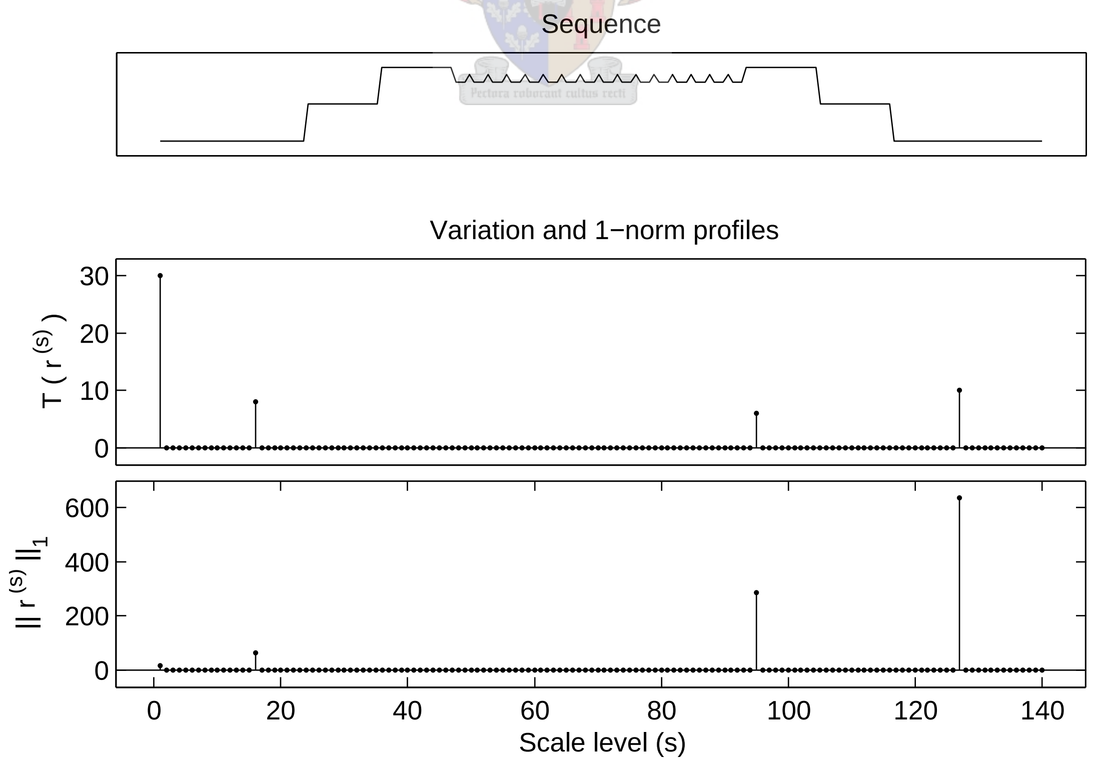


FIGURE 33. Total variation and 1-norm profiles

The 1-norm profile measures contributions to total sequence area and has highest peak for the pulse of width 127.

These profiles are useful for determining what resolutions levels contribute most to the roughness or area of a sequence. When processing sequences, one can use this information to select what levels are important and what can be discarded.

CHAPTER 5

Practical reconstruction

The discrete pulse transform, DPT_{S_N}, decomposes a sequence into a set of different resolution pulses, $\left\{r^{(j,k)} : j \in [1,N]\,,\ k \in [1,N_j]\right\}$ and a residual sequence $r^{(0)}$. The input sequence, x, can be obtained by adding all the pulses to the residual sequence $r^{(0)}$. We call this reconstructing the sequence:

$$x = r^{(0)} + \sum_{j=1}^{N}\sum_{k=1}^{N_j} r^{(j,k)} \tag{5.1}$$

A straight reconstruction (i.e. after transmission) of the extracted pulses may be useful. Generally we rather want to modify the pulses and then add them together. Under certain operations on the pulses the discrete pulse transform is consistent, i.e. modifying the pulses does not change the multi-resolution interpretation of the pulse transform. The resolution level operators are neighbour trend preserving, which implies that all the resolution levels mimic the local ordering of the input sequence. These properties are extremely useful because one can highlight certain structures or scales in the sequence and know that the sequence will not be distorted beyond what is intended. In this chapter, some pulse modification techniques and their uses will be discussed.

5.1. Level based highlighting

The LULU discrete pulse transforms all have basic consistency. This means that is possible to multiply each resolution level by its own non-negative constant and still have the reconstructed sequence decompose into the same set of pulses (with modified pulses heights of course). I.e. the pulses, $\left\{r^{(j,k)}\right\}$, of a discrete pulse transform are modified by multiplying all the pulses in each resolution level, j, y a weight, α_j with $j = 0 \ldots N$:

$$s^{(j,k)} = \alpha_j r^{(j,k)} \text{ with all } \alpha_j \geq 0 \tag{5.2}$$

The sequence is then reconstructed using the modified pulses. Decomposing this reconstructed sequence using the same discrete pulse transform, will always yield the pulses $\left\{s^{(j,k)}\right\}$. The implication being that one can highlight certain resolution levels to enhance their contribution to the reconstruction and suppress the influence of other levels without distorting the pulse interpretation.

At the end of chapter 4 we analyzed the variation profile of a random sequence and discussed why most of the pulses are often found on the first resolution levels. For general noise removal one can then set the level weights of the first few levels low or zero. Setting the first n level weights to zero and then reconstructing is equivalent to smoothing with the recursive operators R_n as defined in the definition of the *dpt*. One can also combine this level weighting strategy with others in order to further narrow down the set of structures one is interested in.

Now we demonstrate how this kind of level weighting allows one to quickly design a reconstruction procedure that retains the information one is interested in (the signal) and gets rid of other parts (the noise). This example is meant to give an idea of the ease and effectiveness of creating a reconstruction like this, so it does not matter if the example is a bit contrived. The ideas contained here are generally relevant.

We construct a class of test sequences of length 1000. These sequences consist of width 10 pulses (of height 1), some wider pulses of width anywhere between 20 and 500 (also of height 1), and superimposed noise from a Gaussian distribution. Figure 34 shows a sample sequence in this class with various degrees of superimposed noise.

We are interested in locating all the width 10 pulses, so everything else is regarded as noise. Regard the sample sequence with added Gaussian noise of standard deviation 0.25, notice that at most places it is possible for one to exactly identify the pulses visually. For the next sequence, where the noise has standard deviation of 0.5, this becomes harder. While the true pulses are mostly visible, there are places where one may falsely guess a pulse existed (false positives) if one did not have the noiseless sequence available for comparison.

We calculate the variation profile of the middle sequence in figure 34 from the discrete pulse transform $DPT_{U_N L_N}$. The total variation of the different scale levels have a large range of magnitudes. It is therefore helpful to plot the logarithm of the total variation profile. Figure 35 shows this. We are not interested in the resolution levels that contribute very little to the total variation of the input sequence, therefore the bottom part of the profile is cut-off.

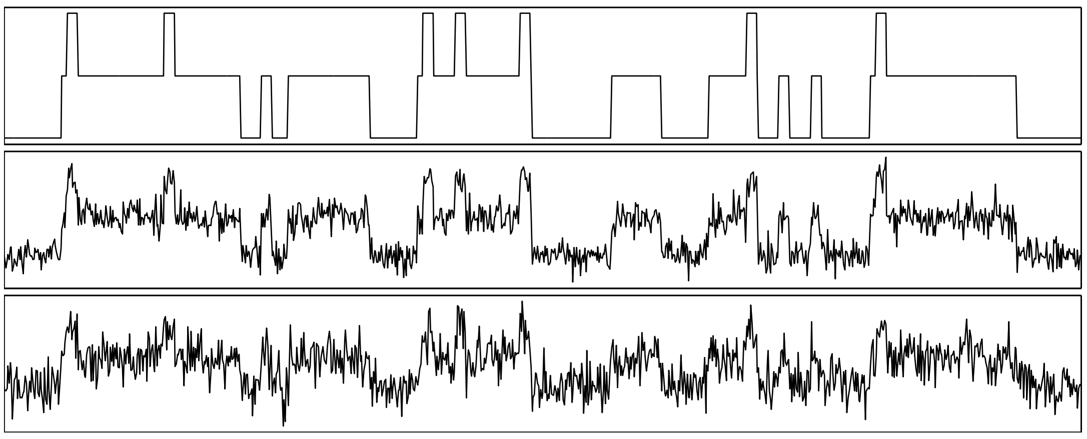


FIGURE 34. Sample sequence with Gaussian noise of standard deviation 0, 0.25 and 0.5 respectively.

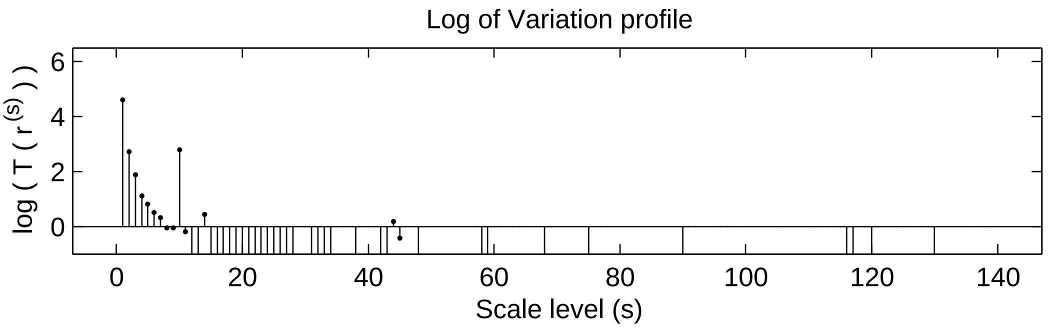


FIGURE 35. Log-Variation profile of middle sequence in figure 34.

We see that the variation in the sequence due to noise is highest at the first level, but then decreases quite quickly. The effect of larger magnitude noise on the total variation profile will be higher, and therefore more of the detail levels will be influenced. Ideally most noise should be left out in the reconstruction. There is a high peak corresponding with pulses of width ± 10; this is our area of interest. There are a few other peaks in the profile at the scale levels where the larger pulses are found; these scale levels should be ignored.

Our reconstruction coefficients (with respect to equation 5.2) are thus

$$\alpha_j = \begin{cases} 1 & \text{for } 10 - c \leq j \leq 10 + c \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

The constant c specifies how many levels around 10 should be included. This is necessary since the presence of noise can affect the width of pulses. The level weights for the resolution levels of interest were all chosen as 1 in equation 5.3. It is only necessary that these weight be non-negative for consistency and trend preservation. One might thus obtain better results by choosing the level weights inversely proportional to the difference between the width of pulses on each level and 10. Our aim is to show what is achievable with a level-weighted reconstruction using a simple example, so we will do no more than mention such possibilities.

We can now reconstruct our sequence. This reconstruction has a pulse decomposition consistent with the original decomposition. Also, the local differences between successive sequence elements in the reconstructed sequence will not have sign opposite to the corresponding differences in the input sequence (trend preservation).

This reconstructed sequence is used to detect the pulses. The presence of noise affects the height of other pulses in a sequence. We thus make use of a threshold to determine which parts of the retained sequence point to the existence of signal (pulses of width 10) and which do not. This threshold is determined using the average value of the upward pulses in the first scale level (in chapter 7 we discuss why this is effective). The number of sequence elements where there are true and false matches are then counted. A true match at a sequence element is when our detection procedure correctly specifies that part of our signal exists at that location. A false match occurs when the procedure detects signal when there is none. Figure 36 shows obtained results for $c = 0, 1$ and 3 (in equation 5.3). The detection results for 500 sequences, from the class described above, were used to calculate average detection rates. The standard deviations of these rates are shown as error-bars.

The true and false matches are normalized by the total number of pulses. A true match ratio r_t of 1 implies that all the signal pulses were correctly detected, while a value of 0 implies none of the signal was detected (1 is optimal). A false match ratio r_f of 0 means that no signal was detected where none exists, and a ratio of 1 means that for every real pulse a false one was detected (0 is optimal).

The pulses were correctly identified for all three different ranges of kept levels in the no- and low-noise cases. With higher amplitude noise, we get less than perfect detection. Keeping only the 10th resolution yielded the least amount of false matches but at the expense of not detecting all the real pulses in the high noise cases. Using more of the surrounding levels results in the detection of more of the true pulses, but the number of false matches increases as well. The best overall results were obtained by using only levels 9 to 11 to detect pulses.

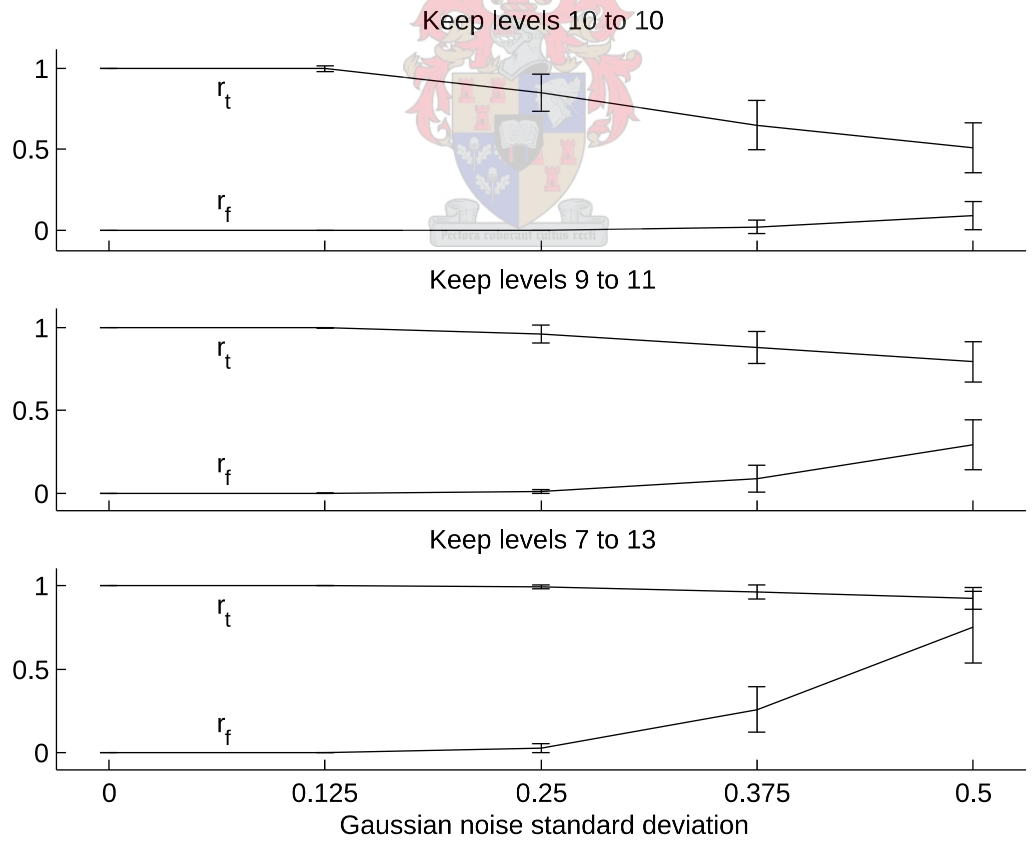


FIGURE 36. Ratio of true matches to number of real pulses, r_t, and ratio of false matches to number of real pulses, r_f, for various amounts of added Gaussian noise and different sets of levels kept in the reconstruction.

The detection algorithm was not designed for optimal results as the idea was only to show how, with a minimal amount of effort, the retention of selected resolution levels can separate the important information from a sequence and thereby aid the automated analysis of data. Further improvements should be possible by revising the reconstruction procedure (as mentioned above), or tweaking the threshold used to determine whether a pulse is detected (i.e. by finding an optimal threshold experimentally for a training data set, or from considerations related to the source of the data). More advanced detection techniques than mere thresholding can also be advantageous.

We see that a simple weighted reconstruction and a thresholding operation was effective in the detection of pulses up to a moderate noise level (Gaussian noise with standard deviation of 0.375). When the features of interest can be characterized with respect to resolution levels, a level weighting scheme provides a straightforward method to separate signal from noise.

5.2. Pulse based highlighting

In the above section, information on the width of important features were used to create a trend preserving reconstruction that eliminated most of the unwanted information. When unwanted structures appear in the highlighted resolution levels, one needs to make use of additional criteria to discriminate between these and the wanted features. The pulses in a decomposition are individually highlighted based on their agreement with this criteria. This is pulse based highlighting.

The pulses, $\left\{r^{(j,k)}\right\}$, of a discrete pulse transform are modified by multiplying each pulse in every resolution level, j, by a weight, $\alpha_{j,k}$ with $j = 0 \dots N$:

$$s^{(j,k)} = \alpha_{j,k} r^{(j,k)} \text{ with all } \alpha_{j,k} \geq 0 \tag{5.4}$$

Due to full consistency the modified reconstruction $x' = r^{(0)} + \sum_{j=1}^{N} \sum_{k=1}^{N_j} s^{(j,k)}$ will decompose into the set of pulses $\left\{s^{(j,k)}\right\}$ when using the same discrete pulse transform used to find the pulses $\left\{r^{(j,k)}\right\}$. From theorem 1.42 and the fact that all the pulse weights are non-negative, the trend of the input sequence is preserved by the reconstruction.

A few examples of pulse weighting rules are now discussed. These are to serve as an introduction to the creation of this type of weighting system.

The first example is the concept of multi-resolution support. Intuitively, elements of the input sequence that have a larger supporting base will have a larger support value. We count, at each sequence element, the number of consecutive resolution levels (starting from the first) with the same sign pulse (i.e. all upward or all downward). If the pulses are positive the support is positive, and if the pulses are negative the support is negative. Specifically, an element of the input sequence that has a corresponding positive pulse at that location in the first N resolution levels, has support equal to N. With negative pulses in the first N resolution levels at a specific sequence element, the support is $-N$.

The support is only non-zero at sequence elements where there exists a pulse in the first resolution level (i.e. at places where the input sequence has local extremas), and thus the only non-zero pulse weights are for pulses in the first resolution level. We get the following pulse weights (in equation 5.4):

$$\alpha_{j,k} = \begin{cases} \frac{n_k}{|h_{j,k}|} & \text{if } j = 1 \\ 0 & \text{otherwise} \end{cases}$$

where n_k is the largest integer such that $(h_{j,k})\left(r^{(m)}_{p_{j,k}}\right) > 0$ for all $1 \leq m \leq n_k$.

The reconstructed sequence $\sum_{k=1}^{N_1} \frac{n_k r^{(1,k)}}{|h_{1,k}|}$ quantifies the support of the input sequence at each element. Supposing that one has multiple peaked structures in a sequence, the support at each local extremum is the width of the multi-resolution structure supporting it. This useful for discriminating between values caused by noise which will have low support and wide peaked structures (for example, a sampled Gaussian function) with higher support. Before we illuminate this with an example, we discuss another pulse weighting rule: pulse normalization.

Basically, we want each element in the reconstructed sequence to specify whether the resolution levels consist mostly of positive or negative pulses at that position. This is useful in similar situations as the measure of support above as it will be larger for wider peaked structures, than smaller ones or noise.

For each sequence element we count the number of positive pulses and negative pulses at every resolution level. This is equivalent to normalizing all the pulse heights to an absolute magnitude of 1. It is sometimes useful to only consider a specific subset of resolution levels. Considering then only the pulses of width $\in [a, b]$, the weights in equation 5.4 are:

$$\alpha_{j,k} = \begin{cases} \frac{1}{|h_{j,k}|} & \text{for } a \leq j \leq b \\ 0 & \text{otherwise} \end{cases}$$

This pulse weighting scheme normalizes all pulse heights. This has the effect of maintaining the multi-resolution structure. The specific difference values found in the input sequence are replaced by whole numbers reflecting only the relative ordering of the elements of the sequence and not their original value.

We now test the effect of these pulse weighting rules on a sequence consisting of sampled Gaussian functions of varying widths. The support of a sequence element is a measure of the number of consecutive levels (starting at the first resolution level) with the same sign pulse. A well supported structure consists of pulses in many resolution levels. In figure 37 we see that the wider Gaussian functions has higher support as expected. The pulse normalized reconstruction is also displayed. This is similar to the support of a sequence with the difference that all the resolution levels contribute to the value, not just the first

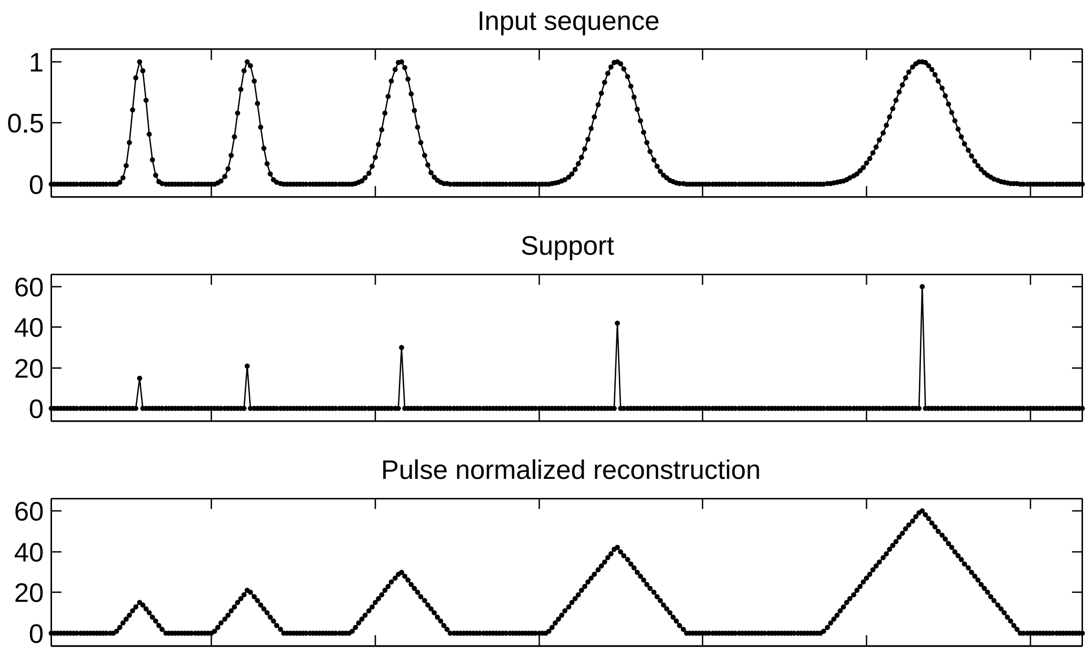


FIGURE 37. A noise-less sequence and some pulse-based reconstruction.

N levels with a same sign pulse at a specific location. The net result is a normalization of the slope of the sequence: the curved sides of the Gaussian functions are straightened.

In figure 38 we recalculate these measures with added normally distributed noise. The magnitude of support depends on the number of consecutive resolution levels (starting at the first level) with the same sign pulse at a specific sequence element. This is not robust. When noise is present, the invertion of local ordering at any place on the peaks will cause a smaller pulse to be created at the expense of a wider pulse (recall that the total number of pulses in a sequence is bounded by the number of non-zero local differences). The probability of this is inversely proportional to the magnitude of the slope of the peaked structure, as it takes a larger value to change the local ordering when the sides of the structure are steep. The two narrowest Gaussian shaped structures have larger support. The wider structures had a some of their low level pulses destroyed and have a low support as a result.

The pulse normalized reconstruction was calculated using all pulses (figure 38 middle), and using only pulses of width between 4 and 60 (figure 38 bottom). This reconstruction is much more robust as a change in one resolution level cannot completely change its value. The relative difference in width of the different peaks can be inferred from the height of the peaks. These values are smaller than the noise-less example because the noise around the edges of the Gaussian functions caused fewer wide pulses to be identified. Using only some of the resolution levels in the reconstruction results in a smoother reconstruction, which may be easier to process procedurally.

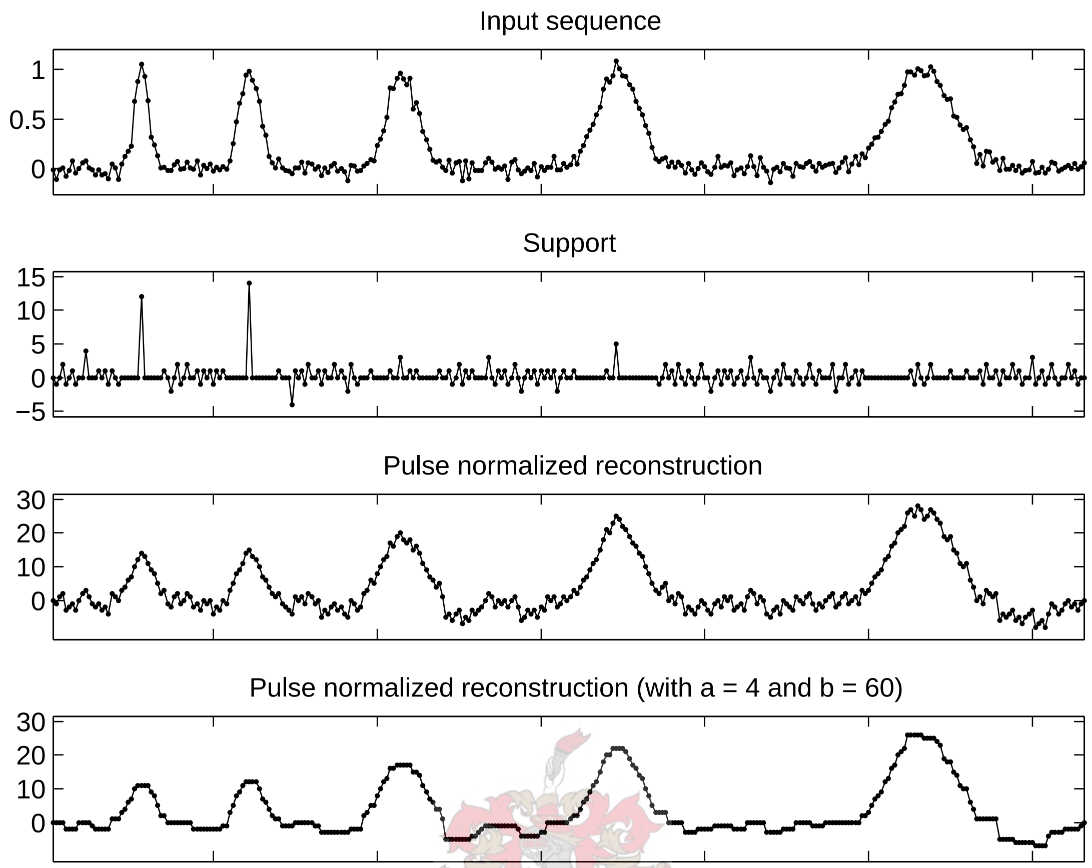


FIGURE 38. A noisy sequence and some pulse-based reconstructions.

A pulse weighting scheme allows pulses to be highlighted or suppressed individually. This is useful when additional constraints provides a way of determining the importance of specific pulses. The above techniques employed this to create sequences that characterizes the multi-resolution support (or structure) of a sequence. Other uses also exist. In chapter 6 we discuss using the *LULU* operators (and the discrete pulse transform) for the analysis of two-dimensional images. Knowing the vertical and horizontal size of important features in an image then allows one to suppress all pulses which do not satisfy both of these constraints.

5.3. Edge detection

In the above techniques the sequence was reconstructed using weighted pulses. It also possible to replace each pulse sequence by another sequence before reconstructing, at the cost of losing consistency and trend preservation. We now see how this can be useful for robust edge detection.

Decomposing a sequence using the discrete pulse transform results in a set of pulses. As the resolution level operators are trend preserving, there will only be pulses at positions where there are differences in the input sequence. In essence, the edges of pulses correspond with edges in the input sequence. The idea now is to reconstruct the sequence such that only the edges remain. Before the sequence is reconstructed, each pulse is replaced by 2 pulses of width 1. These pulses are placed at the edges of the original pulse and has height equal to the absolute height of the original pulse. This is illustrated in figure 39. I.e. replace each pulse sequence

$$r_i^{(j,k)} = h_{j,k} \sum_{m=1}^{k} \delta_{i,(p_{j,k}+m-1)}$$

with the edge sequence

$$r_i^{(j,k)} = |h_{j,k}| \left(\delta_{i,p_{j,k}} + (1-\delta_{j,1})\, \delta_{i,p_{j,k}+k-1}\right)$$

This technique is demonstrated on a fort-like sequence (figure 40). This sequence consists of pulses of width 1, 16, 95 and 127. All the edges were detected.

For noisy sequences, the edges caused by noise can easily overwhelm. Ignoring smaller width pulses when calculating this reconstruction is effective in reducing edges caused by noise, at the price of no longer being able to detect edges caused by very small features. See figure 41. Gaussian noise was added to the sequence in figure 40. The edge detection reconstruction was first calculated using pulses of all widths. This was effective in detecting the edges of the widest structures, but edges of the medium size structures (those to the left and right of the series of single width pulses in figure 40) are not visible amongst the noise. When only pulses of width 16 and wider are used in the edge detection reconstruction,

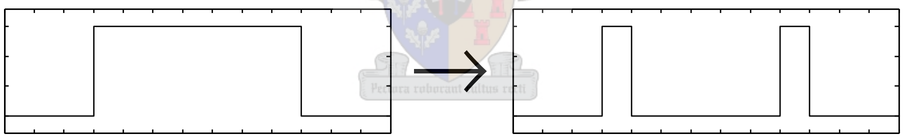


FIGURE 39. Standard width n block-pulse and its replacement in the reconstruction.

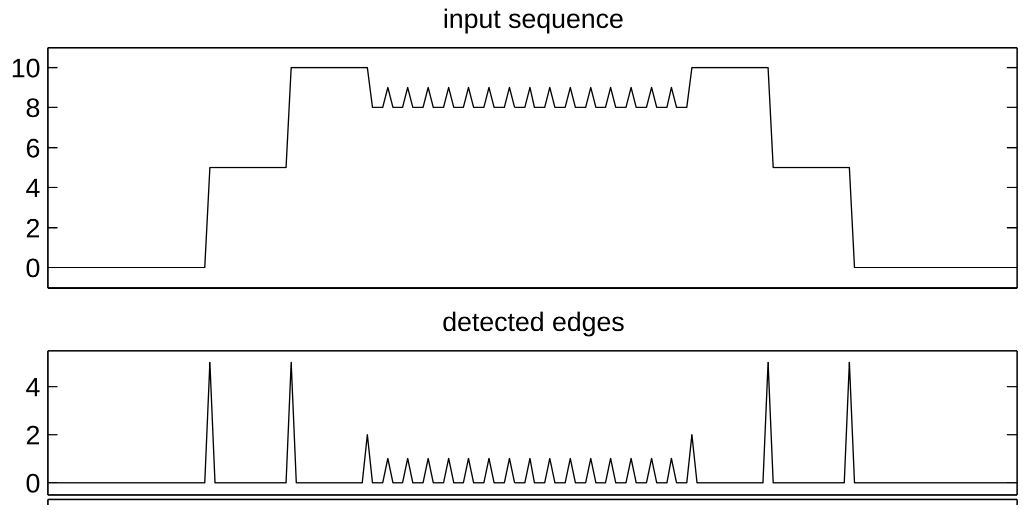


FIGURE 40. A fort-like sequence (top) and its detected edges (bottom)

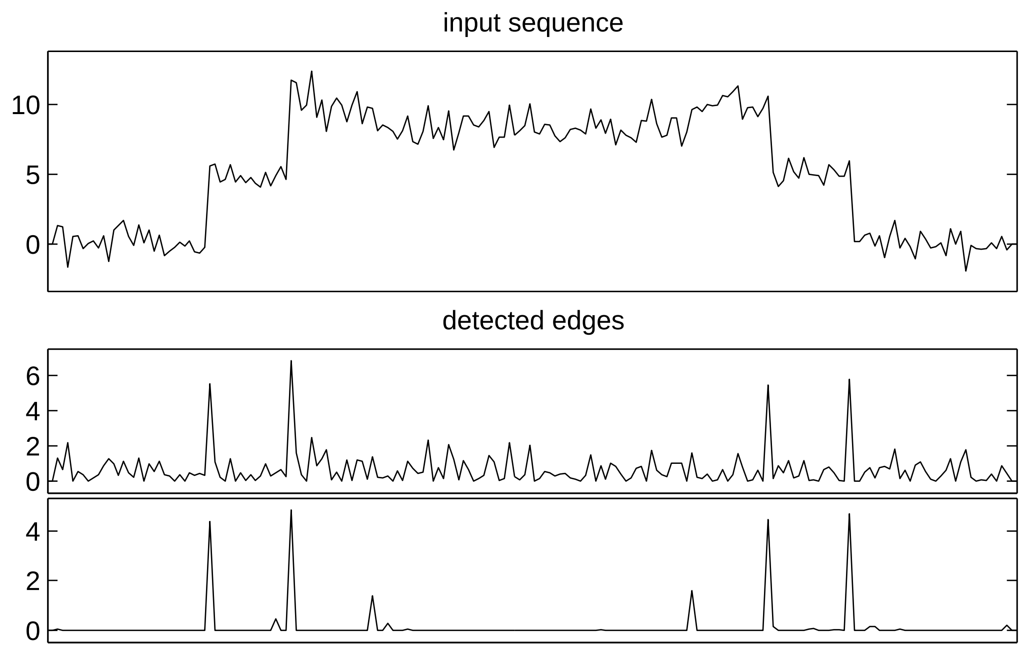


Figure 41. Fort sequence with noise (top). Detected edges using all the pulses (middle), and detected edges using only pulses of width 16 and wider (bottom)

most of the edges caused by noise disappears. One can now clearly see some extra edges not previously detected.

CHAPTER 6

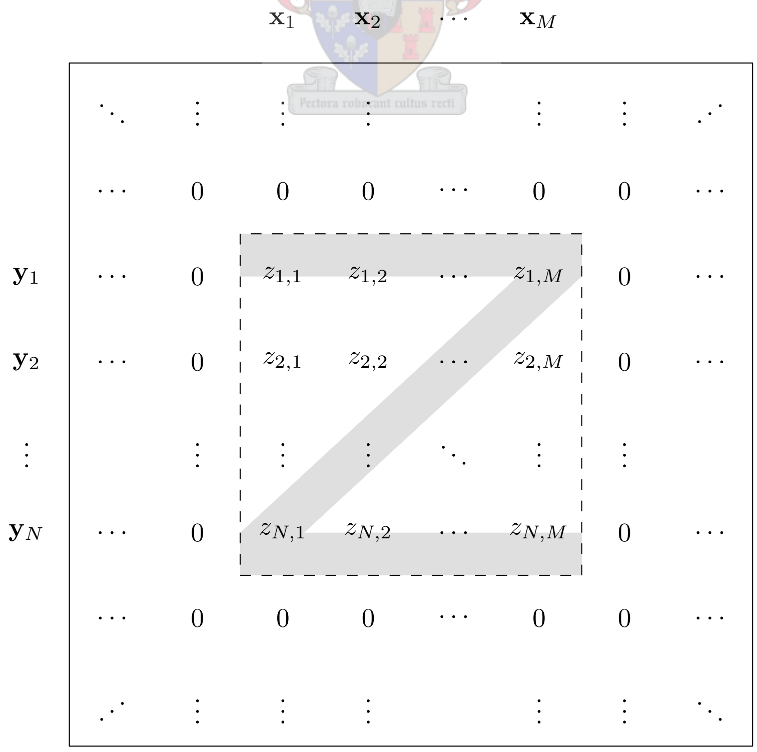
Two dimensional analysis

Only 1-dimensional sequences have been considered so far. The question is now whether the ideas and concepts in the LULU framework can be extended to two dimensions. We will investigate ways in which one can do this and see what the benefits and drawbacks of the different approaches are.

DEFINITION 6.1. A two-dimensional image Z is a set of double indexed real data

$$Z = \{z_{i,j} : i, j \in [1, N] \times [1, M] \subset \mathcal{Z} \times \mathcal{Z}, \quad z_{i,j} \in \mathcal{R}\}$$

An image Z can be considered as a set of bi-infinite sequences (rows or columns) by appending zeros. Figure 42 shows an image Z extended on all sides with zeros. The row and column sequences which form the image are labeled.



$\mathbf{x}_1$ $\mathbf{x}_2$ $\cdots$ $\mathbf{x}_M$
$\mathbf{y}_1$
$\mathbf{y}_2$
$\vdots$
$\mathbf{y}_N$
$z_{1,1}$ $z_{1,2}$ $\cdots$ $z_{1,M}$
$z_{2,1}$ $z_{2,2}$ $\cdots$ $z_{2,M}$
$z_{N,1}$ $z_{N,2}$ $\cdots$ $z_{N,M}$
0

FIGURE 42. An image $Z = \{z_{i,j}\}$, with constituent row and column sequences y_i and x_i.

6.1. 2-d extension of basic operators

First we recall how L_n and U_n are calculated:

$$
\begin{aligned}
(L_n x)_i &= \left(\bigvee^{n} \bigwedge^{n} x\right)_i \\
&= \max\left\{\min\left\{x_{i-n}, \ldots x_i\right\}, \min\left\{x_{i-n+1}, \ldots x_{i+1}\right\}, \ldots, \min\left\{x_i, \ldots, x_{i+n}\right\}\right\}
\end{aligned}
$$

$$
\begin{aligned}
(U_n x)_i &= \left(\bigwedge^{n} \bigvee^{n} x\right)_i \\
&= \min\left\{\max\left\{x_{i-n}, \ldots x_i\right\}, \max\left\{x_{i-n+1}, \ldots x_{i+1}\right\}, \ldots, \max\left\{x_i, \ldots, x_{i+n}\right\}\right\}
\end{aligned}
$$

We call the sub-sequences $\{x_{i-n}, \ldots x_i\}, \{x_{i-n+1}, \ldots x_{i+1}\}, \ldots, \{x_i, \ldots, x_{i+n}\}$ the 'leaves' (as in leaves of the operation tree; called structuring elements in Mathematical Morphology [15]) of $(L_n x)_i$ and $(U_n x)_i$. The leaves of these operators for $n = 1, 2$ are displayed in figure 43. The dot denotes the sequence element x_i, while elements n units to the left and right of the dot correspond with the sequence elements x_{i-n} and x_{i+n} respectively. To show what sequence elements influence the value of $(L_n x)_i$ and $(U_n x)_i$ the combined leaves are also displayed. This is simply the leaves drawn on top of each other such that the element x_i lies at the center.

In one dimension the only variable aspects for a leaf consisting of consecutive sequence elements are its width and its position relative to the element x_i. The only difference between the operators L_i and L_j are the number and width of the leaves. We see that the leaves and thus the operators are symmetric with respect to the current sequence position.

To define two dimensional $LULU$ operators one must create 2-d leaf configurations analogous to those in figure 43. At the moment we are looking for 2-d versions of the basic

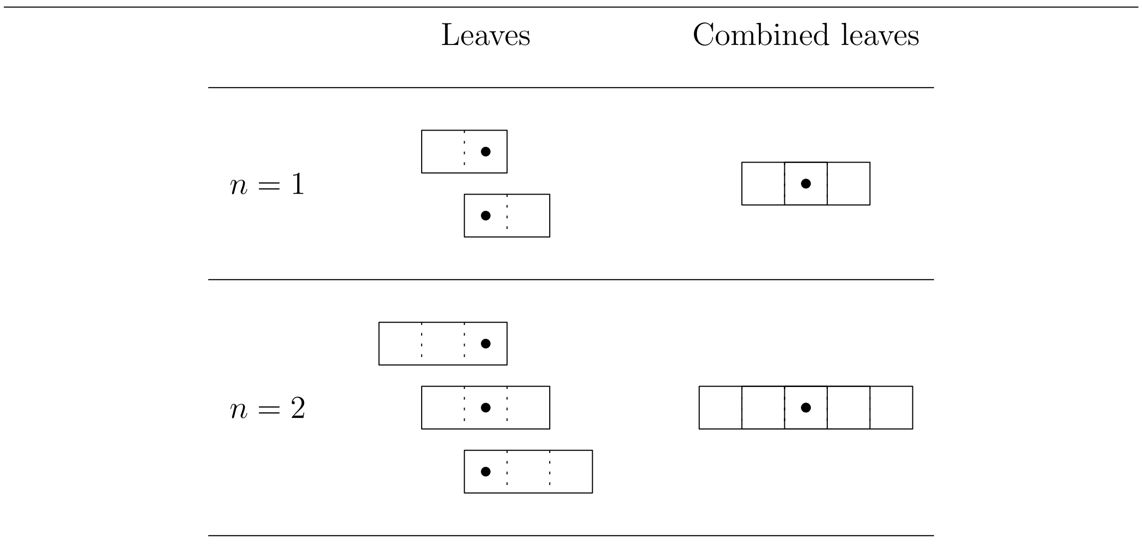


FIGURE 43. Leaves of basic operators L_n and U_n for $n = 1, 2$

LULU smoothers L_1 and U_1. Figure 44 shows four possible leaf configurations and their combined leaf diagram. Configurations (a), (b) and (c) are all 90 degree rotationally symmetric, while (d) is 45 degree rotationally symmetric. Using configurations (a) or (b) (as well as configurations (c) or (d)) will result in smoothing decisions based on the same set of elements, as can be seen from the combined leaf diagrams, but because the leaves themselves are not the same in the different configurations the decisions will also differ.

Each of these leaf configurations (sets of structuring elements) imply a specific combination of the erosion and dilation operators that are common in Mathematical Morphology [15, 16].

The problem is now to choose among the many leaf configuration possibilities. The difficulty lies in the fact that in two dimensions a pulse is harder to define. For the operators

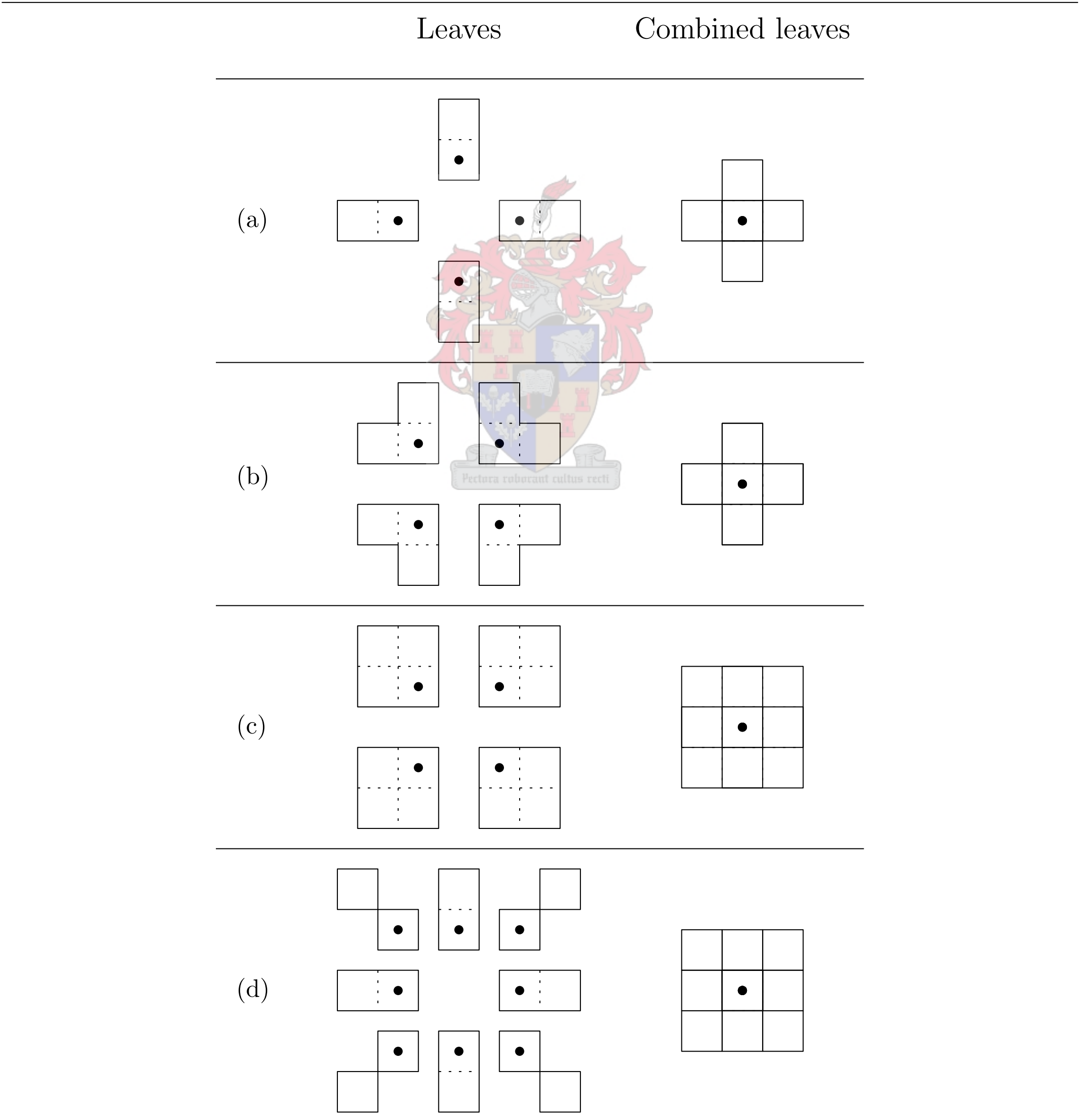


FIGURE 44. Possible leaf sets for 2-dimensional LULU operators

L_1 and U_1 there is less of a problem, but for 2-d versions of a general L_n and U_n one must deal with such questions. Though the full problem is out of the scope of this thesis, we mention some approaches taken by others.

Kao [2] uses configuration (c) to define two dimensional analogies of the operators L_1 and U_1 (and thereby the compositions L_1U_1 and U_1L_1). These are used to remove impulsive noise from images. Kao finds these operators too destructive of fine image details (like width 1 lines) and remedies the problem by choosing the least destructive of either the 2-d L_1U_1 or U_1L_1 at each image element.

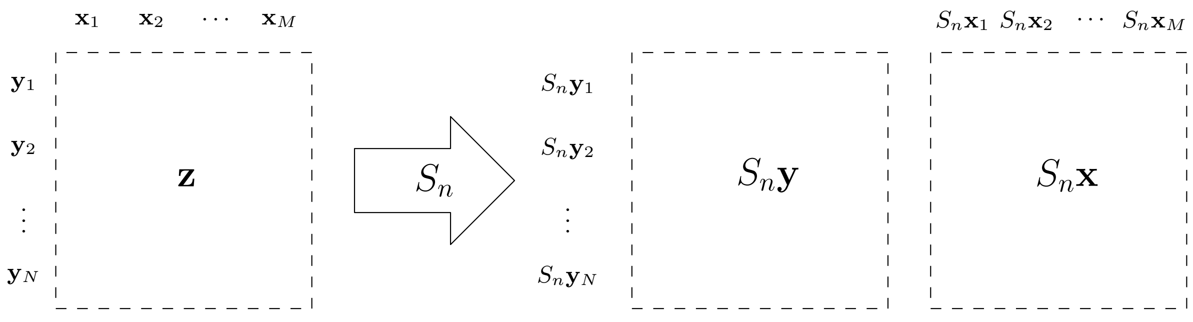
Rohwer [7] approaches the problem by analyzing the properties of the $LULU$-like operators based on some of the leaf configurations in figure 44. It is found that using (a) results in separators with a LULU structure [9, 7]. Basically this means that of all compositions of U and L, only U, L, UL and LU are unique and there exists a full order relation between these: $L \leq UL \leq LU \leq U$. For configuration (b) the resultant operators are not even idempotent and as such cannot form a $LULU$ structure. For leaf configuration (c) there are two further compositions besides U, L, UL and LU that are unique: LUL and ULU. When using this configuration, the 2-d analogies of the basic separators form an extended $LULU$ structure [7].

When extending the general operators L_n and U_n to two dimensions for large values of n the number of possible leaf configurations is very large. While it is possible to define extensions for larger n of the leaf configurations in figure 44, these are not guaranteed to perform as well as the simple $n = 1$ case. Instead of further focusing on this problem, we discuss an alternative method that allows us to use all the existing tools in the $LULU$ framework on images.

6.2. Row and column based decompositions

Recall figure 42. An image consists of a set of row (or column) sequences. It is possible to apply $LULU$ operators to these sequences and thereby extract multi-resolution information from the image. One can also apply these operators along other lines in an image (like diagonals). Suppose one decides to smooth with an operator S_n. Then, at each image element $z_{i,j}$ one has a smoothed value for each of the smoothing directions (i.e. horizontal $(S_ny_i)_j$ and vertical $(S_nx_j)_i$), resulting in an output image for each smoothing direction (see figure 45).

The output images can be the input to another smoothing step. For example, after smoothing the columns with S_n and obtaining the image $S_n\mathbf{x}$ one can smooth the rows of this output image with S_n as well (or vice versa). Analogous to how choosing to remove upward pulses (U_nL_N) or downward pulses (L_nU_n) first affects the smoothing process, the choice of smoothing rows or columns first will affect how the images are smoothed.



$$z_{i,j} = (x_j)_i = (y_i)_j$$

FIGURE 45. Directional smoothing of an image results in a output image for each direction

All the *LULU* operators discussed in previous chapters can be applied to an image on a line by line basis. Performing the *dpt* on an image is particularly useful. As mentioned in chapter 5, one can use the extra information obtained from taking decompositions in more than one direction to choose pulse weights in a pulse-based reconstruction.

To illustrate the power of directional decompositions of images we demonstrate it on the problem of removing positive impulsive noise. We test our technique on a standard image processing test image (figure 46a). Let $z = \{z_{i,j}\}$ be an image with positive impulsive noise added (figure 46b), with $\left\{a_i^{(n)}\right\}$ the resolution levels of the row-by-row discrete pulse transform and $\left\{b_i^{(n)}\right\}$ the resolution levels of the column-by-column *dpt* :

$$\mathrm{DPT}_{U_3L_3}\ y_i \rightarrow \left\{a_i^{(n)}\right\},\ n = 0, 1, 2$$
$$\mathrm{DPT}_{U_3L_3}\ x_j \rightarrow \left\{b_j^{(n)}\right\},\ n = 0, 1, 2$$

Create a 'resolution level image' for each smoothing direction (horizontal and vertical) and each resolution level n, consisting of all the pulses on that level.

$$\alpha_{i,j}^{(n)} = \left(a_i^{(n)}\right)_j$$
$$\beta_{i,j}^{(n)} = \left(b_j^{(n)}\right)_i$$

The residual images consisting of the residual sequences $U_3L_3x_i$ (vertical) and $U_3L_3y_j$ (horizontal) are then $\alpha^{(0)}$ and $\beta^{(0)}$ respectively (figure 46c and d). Note that while they remove most of the impulsive noise, they are very destructive of fine details (the hair for example), and therefore one would rather not use them directly.

The image is contaminated with impulsive noise, thus remove at each image element $z_{i,j}$ the largest of the width 1 pulses along each direction found at that position.

$$z_{i,j} - \max\left(\alpha_{i,j}^{(1)}, \beta_{i,j}^{(1)}\right) \tag{6.1}$$

While this should get rid of most the impulsive noise, it will also destroy small image details (similar to but less extreme than figure 46c and d). One would rather keep an image element unchanged if smoothing there is too destructive.

We know the impulsive noise is positive, therefore we only remove a pulse at an image element if a positive pulse of width 1 was found in both smoothing directions. Call this condition C. The horizontal and vertical residual images consist of what remains after horizontal and vertical pulses of width up to 3 respectively have been removed. If, at an image element, there is more agreement between these two images than between these two images and the input image, there is probably noise which can safely be removed. Call this condition D.

$$\begin{aligned} C_{i,j} \text{ is true} &\iff \alpha_{i,j}^{(1)}, \beta_{i,j}^{(1)} > 0 \\ D_{i,j} \text{ is true} &\iff \left|\alpha_{i,j}^{(0)} - \beta_{i,j}^{(0)}\right| < \left|\alpha_{i,j}^{(0)} - z_{i,j}\right|, \left|\beta_{i,j}^{(0)} - z_{i,j}\right| \end{aligned}$$

If both $C_{i,j}$ and $D_{i,j}$ are true then we assume it is safe to remove the largest width 1 pulse using equation 6.1:

$$(Sz)_{i,j} = \begin{cases} z_{i,j} - \max\left(\alpha_{i,j}^{(1)}, \beta_{i,j}^{(1)}\right) & \text{if } C_{i,j} \text{ and } D_{i,j} \\ z_{i,j} & \text{otherwise} \end{cases}$$

The result of de-noising the test image at every pixel using this equation is displayed in figure 46e. By comparison with figure 46a we see that most of the impulsive noise was removed and the fine image details were not distorted unreasonably. By applying this smoothing procedure again, even more of the impulsive noise is removed with hardly any (if at all) further distortion (figure 46f). Incorporating decompositions along diagonal lines in deciding which pixels to smooth should lead to a de-noising procedure that is even less destructive.

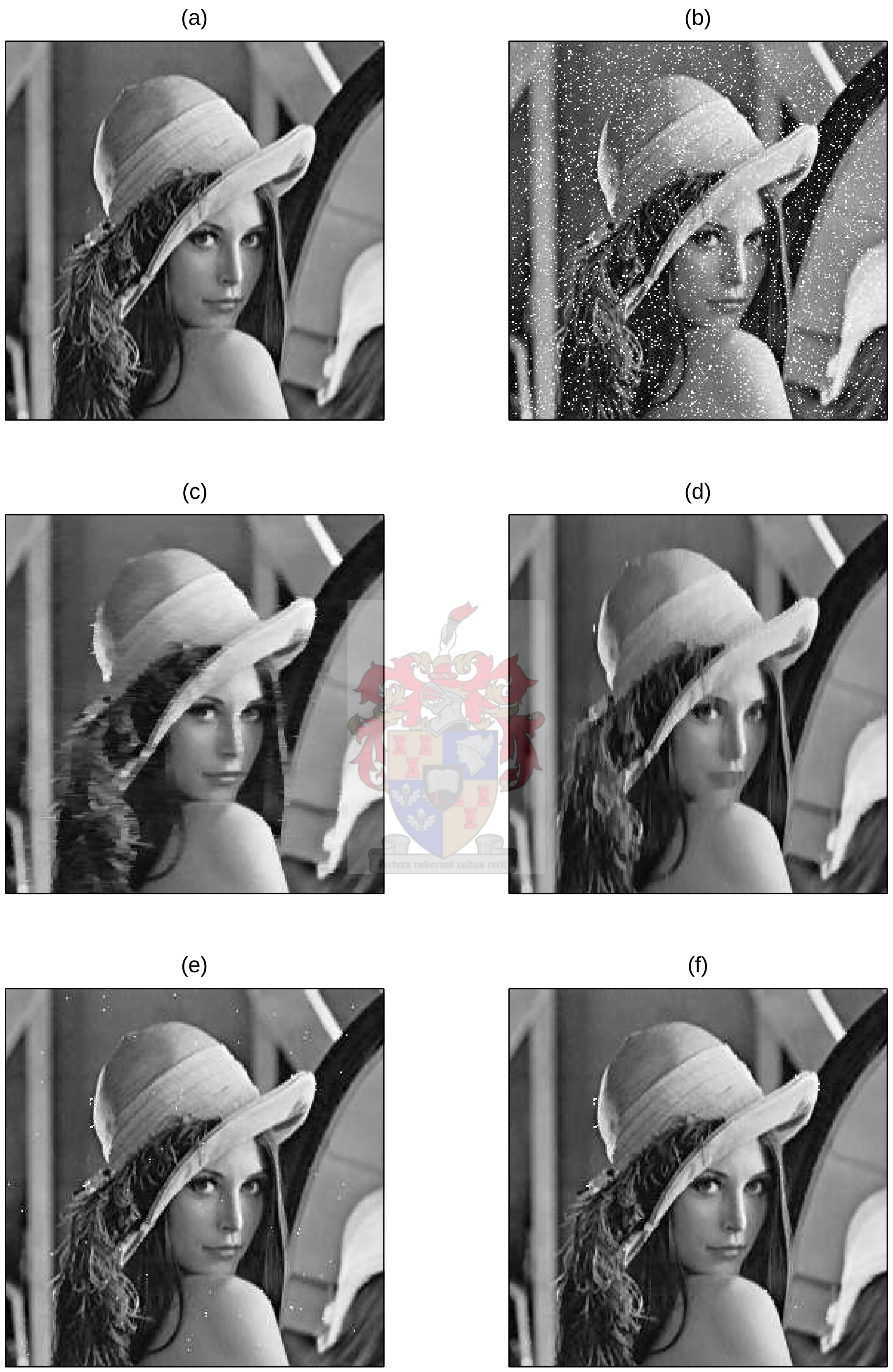


FIGURE 46. (a) True image (b) Image with added impulse noise at 4000 pixels (c) Horizontally smoothed image (d) Vertically smoothed image (e) De-noised image (f) De-noising again using the same procedure

CHAPTER 7

Estimation of moments

In most cases where one obtains a data sequence from a physical apparatus, the incoming data is contaminated with noise. When coding images it can be economical to code a noisy region (a high-frequency texture for instance) as a distribution with particular parameters instead of an inherently incompressible random stream. Also, the introduction of rounding processes to data introduces quantizing errors. The distribution of these errors are well approximated by the B-splines or a Gaussian [6].

The specific noise values are less important than the distribution from which the noise comes. It is useful to be able to estimate these parameters (the moments of the distribution). Rohwer [6] gives a method for calculating the second moment (variance) of an unknown random distribution contaminating measurements. This is estimated from the first level LULU decomposition of the data sequence. Critical to this estimation is the assumption that the noise is identically independently distributed (i.e. noise at each sequence element comes from the same distribution but is independent from noise at other elements). In this chapter, we will follow the derivation of this technique and test the accuracy. We will also see how we can extend this to two dimensions.

7.1. Standard deviation from average pulse height

First of all, we assume that our underlying sequence is of a lower resolution than the sampling interval. Otherwise there is little hope of separating the noise from the equally scaled signal. Using the B-splines as an example, we will show how one can estimate the standard deviation ($\sqrt{\text{variance}}$) of a symmetric distribution contaminating measurements using the average value of the negative pulses in the first resolution level for a signal decomposed with L_1U_1. This same technique can be applied to many other distributions. The first moment (the mean) of a random distribution contaminating measurements is not obtainable as it is indistinguishable from trend (if there is trend).

Let $P(x_i = z) = f_n$ be a B-spline of order n. We will illustrate the following steps with the B-splines of order $n = 1,\ 2$ and 16.

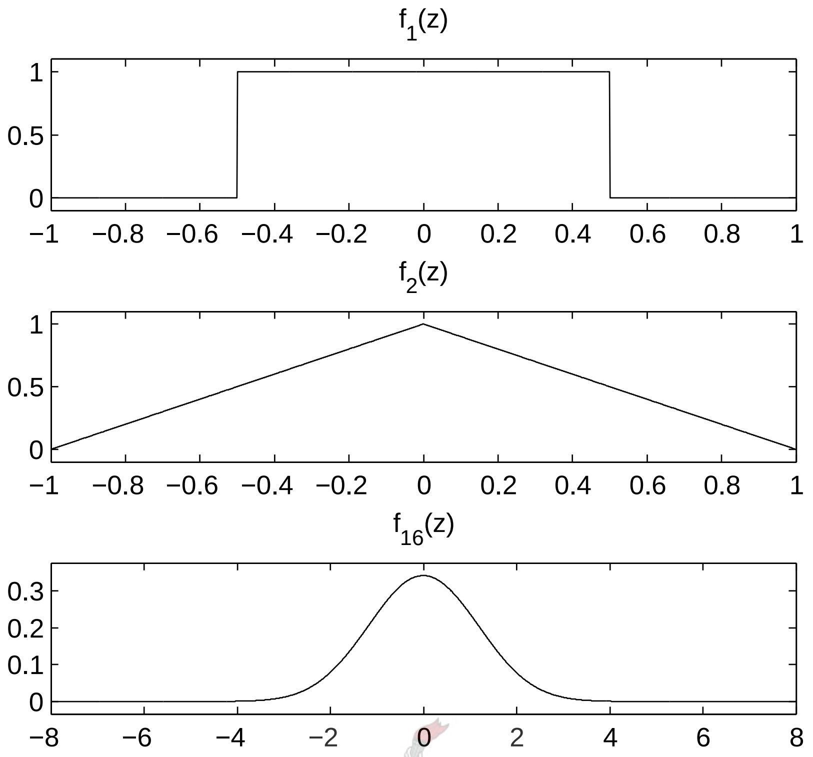


FIGURE 47. Probability density functions of b-splines of order 1, 2 and 16.

The distribution function is given by: $F_n(z) = \int_0^z f(z)\ dz$

We want to calculate the probability density function for a pulse in the first level to have a certain value, z. Let $C_n(z) = P(x_i - Ux_i = z)$. Then,

$$\begin{aligned} & C_n(z) \\ = \quad & P(x_i = z) * P(Ux_i = z) \\ = \quad & 2f_n(z) * F_n(z) f_n(z) \\ = \quad & 2\int_{-\infty}^{\infty} f_n(z-t) f_n(t) F_n(t)\ dt \end{aligned}$$

because:

$$\begin{aligned} & P(\min\{x_{i-1}, x_{i+1}\} < z) \\ = \quad & P(x_{i-1} < z) \,.\, P(x_{i+1} < z) \\ = \quad & F_n^2(z) \end{aligned}$$

and

$$\begin{aligned} & P(Ux_i = z) \\ = \quad & P(\min\{x_{i-1}, x_{i+1}\} = z) \\ = \quad & \frac{d}{dz} P(\min\{x_{i-1}, x_{i+1}\} < z) \end{aligned}$$

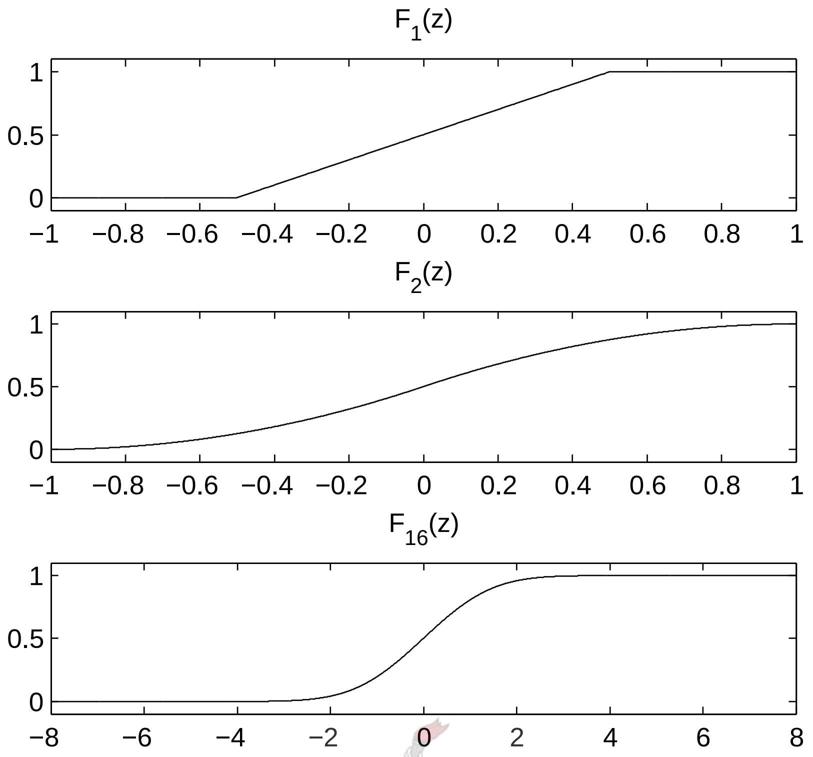


FIGURE 48. Probability distribution functions of b-splines of order 1, 2 and 16.

$$\begin{aligned} &= \frac{d}{dz}F_n^2(z) = 2F_n(z)\,\frac{dF_n(z)}{dz} \\ &= 2F_n(z)\,f_n(z) \end{aligned}$$

This results in a probability density function, $C_n(z)$ (see figure 49), for the B-spline of each order, n. $C_n(z)$ is then integrated to calculate the probability of finding a negative pulse in the first resolution level:

$$P(x_i < 0) = \int_{-\infty}^{0} C_n(z)\,dz \tag{7.1}$$

U_1 removes local minimas, therefore equation 7.1 is equal to $\frac{1}{3}$ irrespective of the particular distribution [6]. Now use this to calculate the average value of the negative pulses:

$$a_n = \frac{\int_{-\infty}^{0} zC_n(z)\,dz}{\int_{-\infty}^{0} C_n(z)} = 3\int_{-\infty}^{0} zC_n(z)\,dz \tag{7.2}$$

The standard deviation of a B-spline of order n is given by

$$\sigma_n = \sqrt{\frac{n}{12}} \tag{7.3}$$

The B-spline distribution can be normalized to a general standard deviation. First we calculate what effect changing the standard deviation of f will have on C_n. The constant

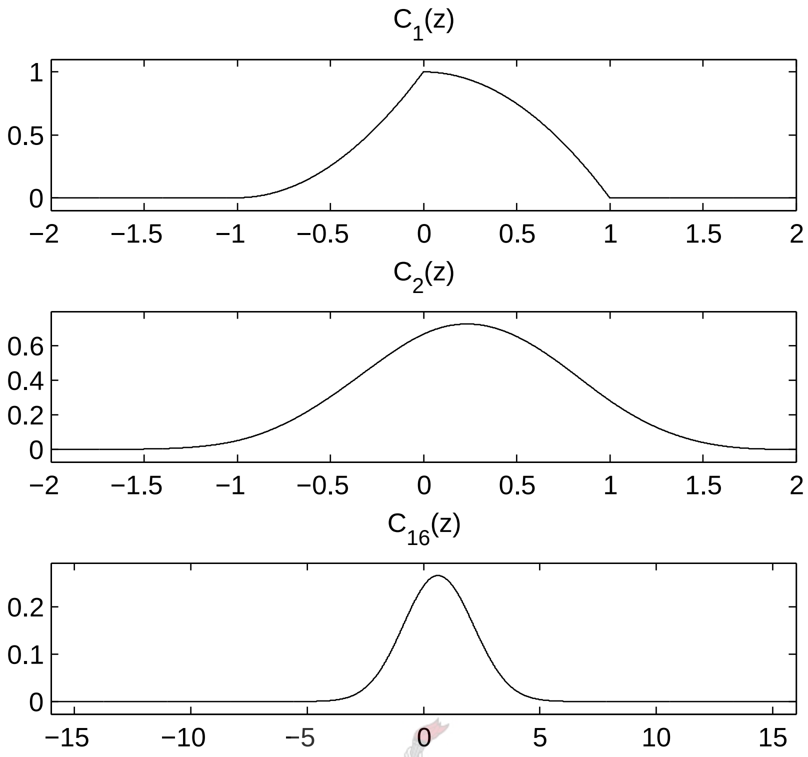


FIGURE 49. Probability density function for $x_i - Ux_i = z$

$k = \frac{\sigma_n}{\sigma'}$ in $f(kz)$ will change the standard deviation of f to σ'.

$$\begin{aligned} C_n'(z) = & \quad 2\int_{-\infty}^{\infty} f(kz-kt)\, f(kt)\, F(kt)\; dt \\ = & \quad \frac{2}{k}\int_{-\infty}^{\infty} f\left(kz-t'\right) f\left(t'\right) F\left(t'\right)\, dt' \\ = & \quad \frac{1}{k}C(z) \qquad \text{with } k=\frac{\sigma_n}{\sigma'} \end{aligned}$$

This, together with equations 7.2 and 7.3 gives the average value of the negative pulses in the first resolution for a general standard deviation, σ':

$$a_n^{'} = \quad 3\int_{-\infty}^{0} zC_n'(z)\, dz = \sigma' 3\sqrt{\frac{12}{n}}\int_{-\infty}^{0} zC_n(z)\, dz \tag{7.4}$$

$$= \quad \alpha_n\, \sigma' \tag{7.5}$$

The factors, α_n, which relate the average values of the negative pulses in the first resolution level to the standard deviation of the underlying distribution was calculated numerically for $n = 1 \ldots 30$. The resulting curve is shown in figure 50. These constants all fall in the small interval of 0.025 (a maximum difference of about 3%) and approach the constant $\alpha_\infty = \frac{-3}{2\sqrt{\pi}}$ asymptotically as n increases (Rohwer [6]). This limit is reached exactly for a Gaussian distribution.

The standard deviation of a B-spline or Gaussian like distribution can now be estimated by dividing the average value of the negative pulses in the first resolution level by the

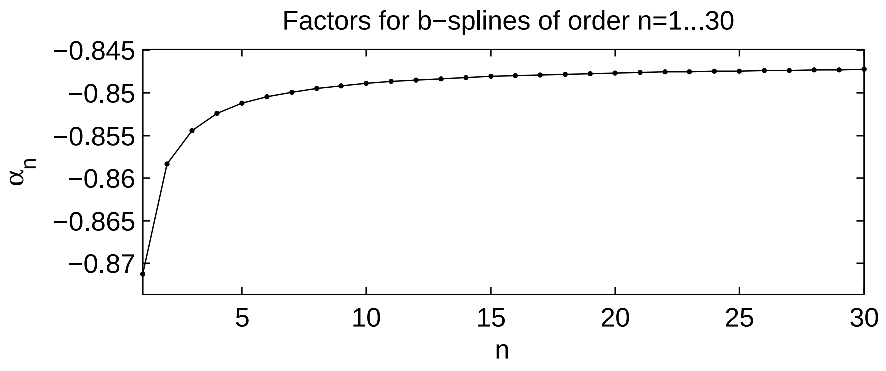


FIGURE 50. Constant factors which relate the average value of the negative pulses on the first level and the standard deviation of the underlying distribution

constant α_∞. Since the operators L_n and U_n are duals of each other, one can also use the average of the positive values in the first resolution level divided by the constant $-\alpha_\infty$ if one decomposed the signal with U_1L_1 instead of L_1U_1.

7.2. Accuracy

The LULU operators are non-linear, therefore the underlying signal will influence the first level decomposition. A low resolution signal will have little effect on the first level decomposition, whereas a signal with a high slope will have the most dramatic effect. For this experiment we choose the signal: $y_i = \beta i + \epsilon_i$. Where ϵ_i is our noise from the random distribution we wish to describe. We vary the steepness of the slope by varying the parameter β. For each value of β, 1000 sequences of length 10000 are generated to gather statistics on the accuracy of the standard deviation estimation (see figure 51).

The shorter a sequence, the less pulses there are in the first resolution level. Therefore the length of the sequence will influence the accuracy of our estimation, we would like to see how much this is so. The standard deviation of a normally distributed random

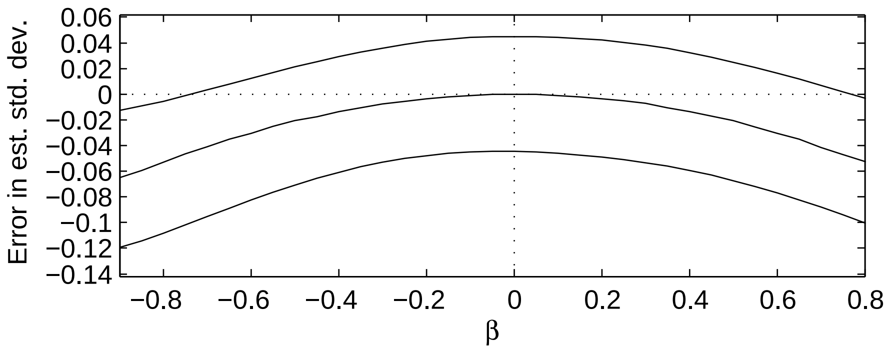


FIGURE 51. Error in estimated standard deviation of random noise as a function of the steepness of underlying signal. Middle line is average error. Top and bottom line is maximum and minimum error respectively.

sequence is estimated using the above-mentioned technique. The signal length is varied from a length of 20 elements to about 3000. For each signal length, we create 1000 signals so that statistics about the accuracy of the estimation can be gathered.

The results are graphed in figure 52. The standard deviation used to generate the data is 1.0. Although the estimation was accurate most of the time for a short signal length, there is no guarantee that the estimated standard deviation is within a certain distance of the true value. The maximum absolute error in the estimation goes down quickly as the signal length increases. With long signals, the estimation is very accurate most of the time but about 1 percent out in the worst cases.

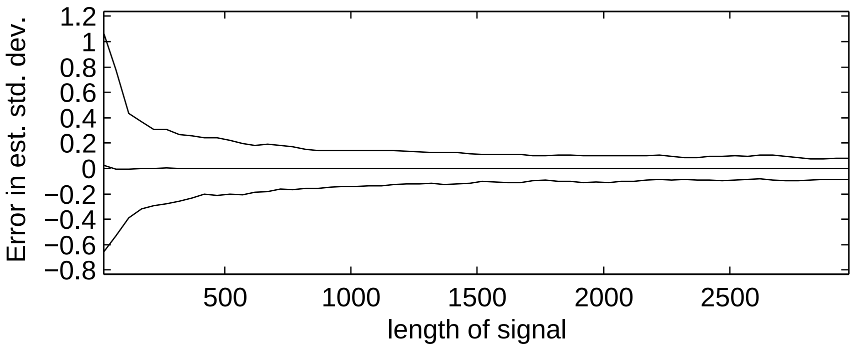


FIGURE 52. Error in estimated standard deviation of random noise as a function of the length of the signal. The middle line is the average error. The top and bottom line is the experimental worst case errors.

7.3. Extension to 2 dimensions

The standard deviation of i.i.d. distributed noise contaminating a 1-d measurement can be estimated using the average value of the positive pulses on the first level resolution level when decomposing with U_nL_n. Instead of finding the unit-width pulses of a 1-d sequence one can find these pulses for a 2-dimensional image as described in section 6.2. One now has a set of pulses for differently oriented lines on the image. For example: pulses for each row, column and diagonal. The average of these pulses around a specific area can be used to calculate a local standard deviation, using one of the conversion constants α_n. The larger the area the more accurate but less local the estimation is.

A set of horizontal lines in an image will register pulses in every column for the column-based decomposition and in every diagonal for the diagonal decomposition. To prevent structures like these from influencing the calculated standard deviation one can leave out pulses that do not occur in all directions when calculating the average.

We now test this technique. A 2d image of normal distributed noise is created with a standard deviation that varies with the x-axis of the image. Figure 53(a) shows the standard deviation of the noise, (b) shows the generated noise. The test image (d) is created by adding this noise to a background image consisting of some superimposed rectangles (c).

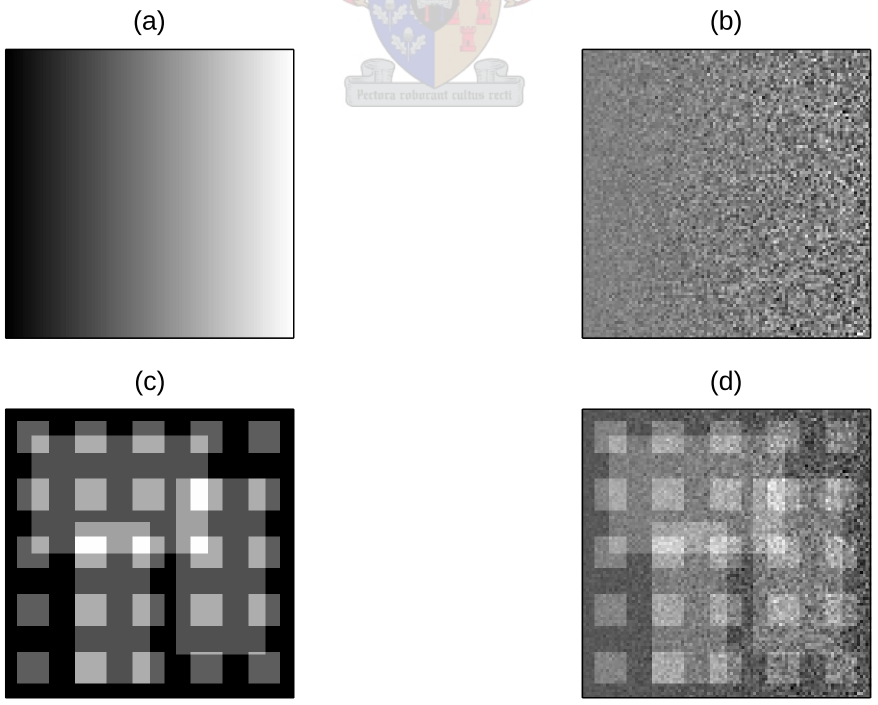


FIGURE 53. (a) Standard deviation of added noise. (b) Added Noise. (c) Background Image. (d) Test image formed by adding noise and background images.

Row- and column-based decompositions are done using L_n and U_n. This gives four decompositions. Each of these yields an image of pulses for the first resolution level. On each of these images use a disk-shaped structuring element to average the non-zero elements around each point. This is divided by the constant α_∞ for decompositions based on U_n and by $-\alpha_\infty$ for decompositions based on L_n. We now have a estimate for the standard deviation at every pixel in each of these four images. To get the final estimate the average of these four images are calculated. We repeat this for averaging disks of four different radii so that we can compare the effect of the size of the averaging disk. The results are displayed in figure 54. With small disks the estimation is very local. As the disk size increases more points are used to make the estimate which results in a standard deviation estimate closer to the real one used to create the noise.

In figure 55 we plot the error images for the noise estimations. These were calculated by subtracting the estimated noise for each disk radius from image of standard deviations used to generate the noise. With larger averaging disks the global accuracy is better, but local variations in the standard deviation are no longer detected.

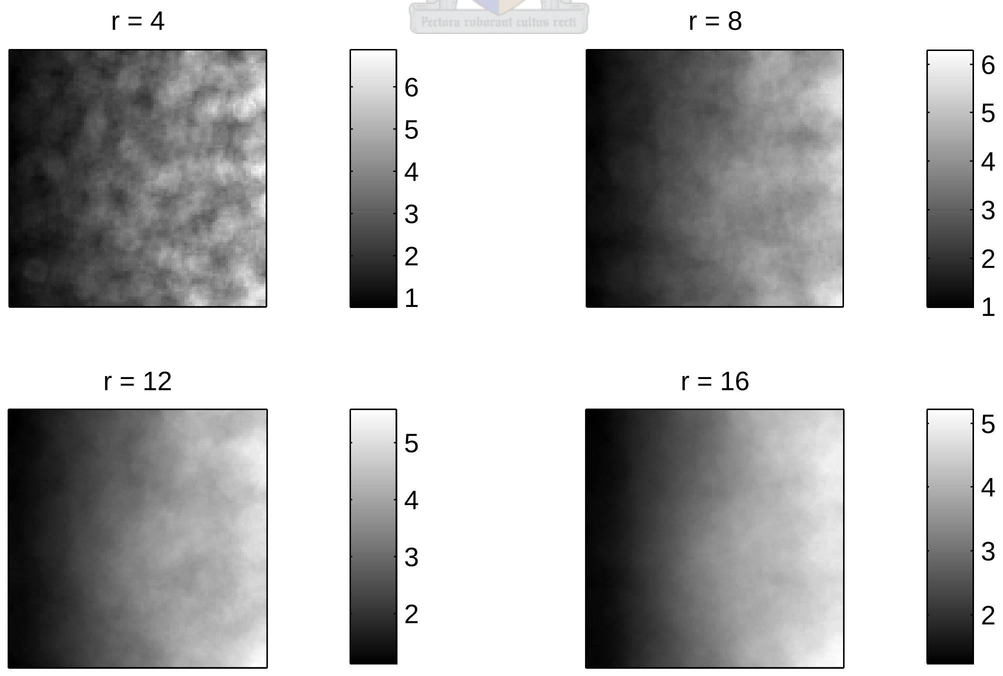


FIGURE 54. Estimated standard deviation for averaging disks of radii 4, 8, 12 and 16.

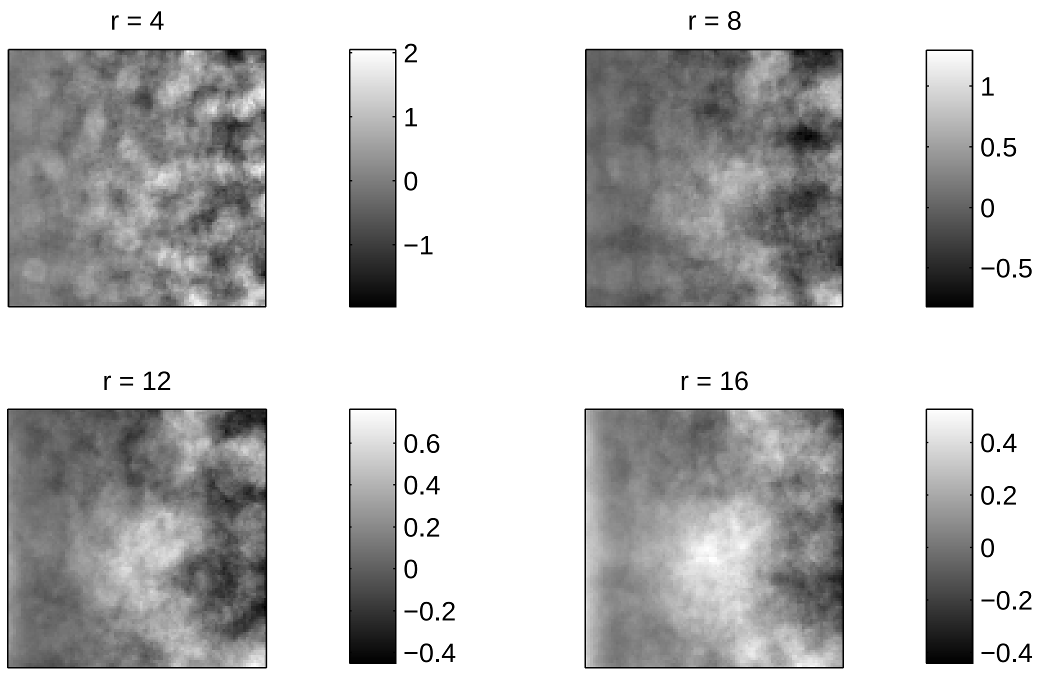


FIGURE 55. Errors in estimations for averaging disks of radii 4, 8, 12 and 16 calculated by subtracting estimation from noise standard deviation in figure 53(a).

Part 2

Applications

CHAPTER 8

Noise analysis

We want to use the LULU framework to analyze the data in figure 56, which shows the time-dependent second harmonic signal from a pure undoped Si sample under femtosecond (80 fs pulse length, 80 MHz repetition rate) irradiation at 782 nm[1]. The second harmonic signal is a measure of the electric field that exists across the interface. The time-dependence is a result of the induced charge transfer that occurs between the Si and the thin (< 5 nm) native oxide layer covering the Si. A photomultiplier tube is used to detect the (very small) signal.

The data exhibits an initial sharp rise, attributed to electron trap sites being created in the Si, and the subsequent movement of electrons to these trap sites. Thereafter, the signal declines slowly due to the creation, and the filling with holes, of hole trap sites.

Superimposed on the signal is noisy-looking high frequencies with large impulses in the positive direction. We want to see what we can determine regarding the nature of the noise. Is it completely random or is it possible to infer some order?

First we separate the high frequency noisy parts of figure 56 from the rest of the signal. The discrete pulse transform will remove all small width pulses in the first resolution levels. Thus it should be possible to separate the noise from the original signal by removing the first N resolution levels. To do this, we have to choose our N.

Figure 57 shows the 1-norm and total variation of C_n as a function of n. The function values in these graphs at a specific n are the values of the 1-norm and the total variation in the remaining sequence after the first n detail levels are removed. The total variation

[1]Laser group - Department of Physics - University of Stellenbosch

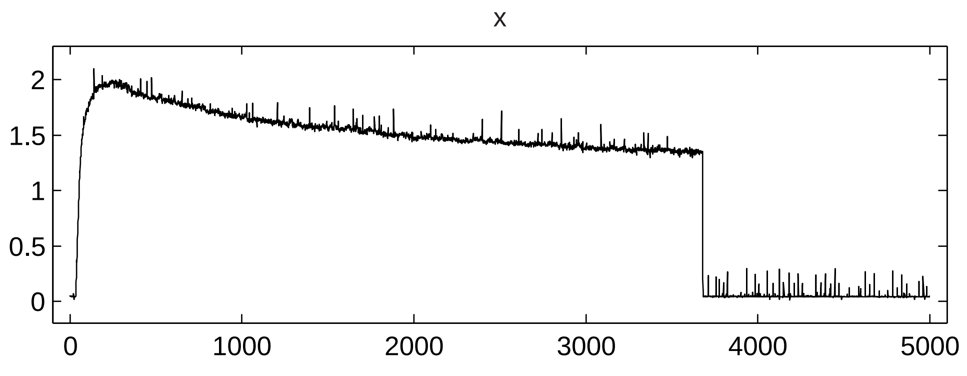


FIGURE 56. Signal to analyze

decreases monotonically as more of the detail levels are removed (section 4.4). After the high frequency noisy parts are removed the total variation and 1-norm fall off very slowly. This is because only a few small pulses are found in these levels. The separation level, N, must be chosen such that the high frequency sequence components fall in the detail levels $1 \ldots N-1$ and the rest of the signal in the detail levels N and above. Choose:

$$N = 20$$

Choosing N somewhat larger or smaller will have very little effect on our removal of the high frequency parts due to the lack of large pulses in the detail levels around $n = 20$. This can be inferred from the flatness of the graphs around this value of n.

Using our chosen N we split the original signal, x, into two: (see figure 58)

$$\begin{aligned} \text{smooth part:} &\quad C_N x \\ \text{noisy part:} &\quad (I - C_N)\, x \end{aligned}$$

After the sudden cutoff in the smooth part the nature of the noise changes. Therefore from now on we regard these two parts as two regions with different properties. There are more large positive impulses in the second region but the low amplitude noisy part is smaller.

The standard deviation of a Gaussian shaped distribution can be found using the average height of either the positive pulses (found using L_1) or negative pulses (found using U_1) of width 1 (refer to chapter 7). The noisy part extracted from the original signal looks like it contains noise from a distribution with added positive impulsive noise. It is possible to

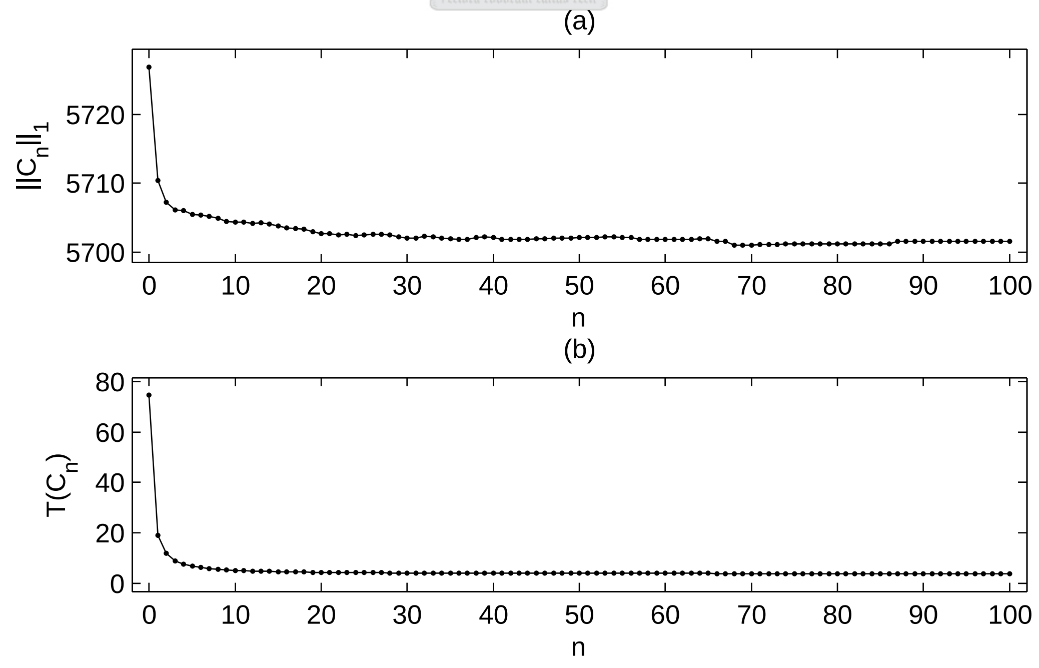


FIGURE 57. (a) 1-norm of C_n vs n (b) Total variation of C_n vs n

threshold our collection of pulses and only use those smaller than the threshold when calculating our standard deviation. This makes it possible to estimate the standard deviation of the first distribution without the interfering presence of the impulsive noise.

A normally distributed random sequence (with standard deviation of 0.02) is generated and decomposed using L_1 and U_1. All pulses higher than a threshold value are ignored when calculating the standard deviation using equation 7.5. Figure 59 shows the standard deviation estimated for different threshold values. The estimated standard deviation is very close to the real value when the threshold is higher than all the extracted pulses (i.e. no pulses were ignored). As the threshold decreases, more of the larger pulses are left out when estimating the standard deviation and therefore the estimation begins to fall.

The same technique is now used on the two regions in extracted noisy part. The results are displayed in figure 60. In both regions we initially see that the standard deviation vs.

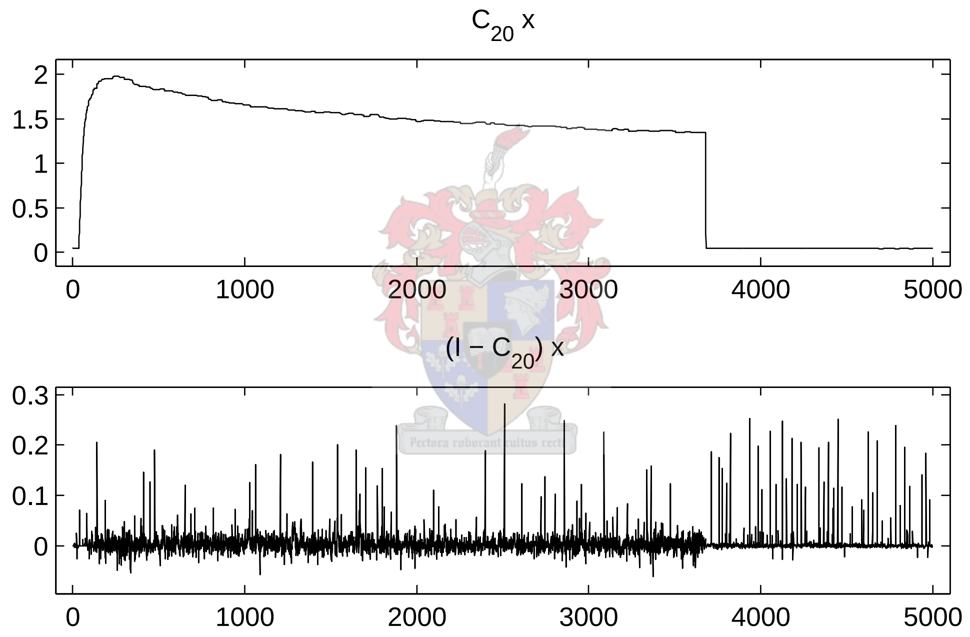


FIGURE 58. Separation of input signal into smooth and noisy part.

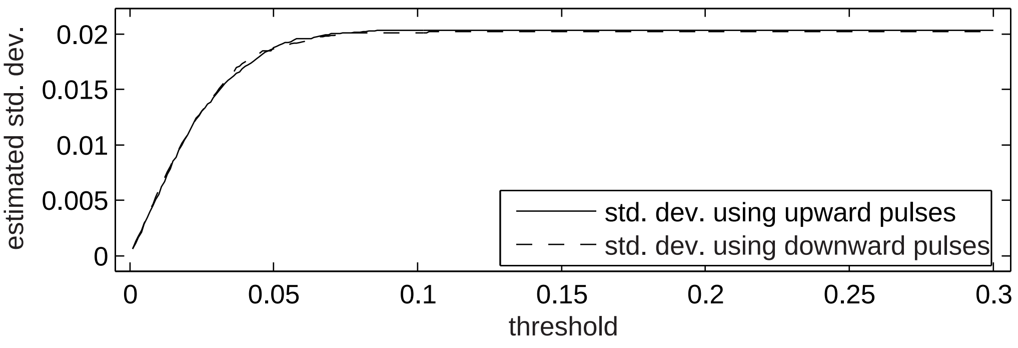


FIGURE 59. Estimated standard deviation for a normally distributed random sequence and different threshold values.

threshold curve has the same shape as in figure 59. This hints that there is a symmetric noise distribution present. Before the curves can flatten out as with the constructed example, the two estimations start to diverge. The estimation using the upward pulses is much higher in both cases. There is therefore some type of biased influence. The exact nature of the upward impulsive noise can be inferred from the difference in the estimations using the upward and downward pulses.

It is reasonable to assume that the noise in the data is from an unbiased distribution with added positive impulsive noise from another source. The unbiased noise is proportional to the signal strength, while the amount of impulsive noise seems to increase after the signal drops to zero. The source of the impulsive noise was later found to be the photomultiplier tube which, being quite old, sometimes generates an incorrect signal.

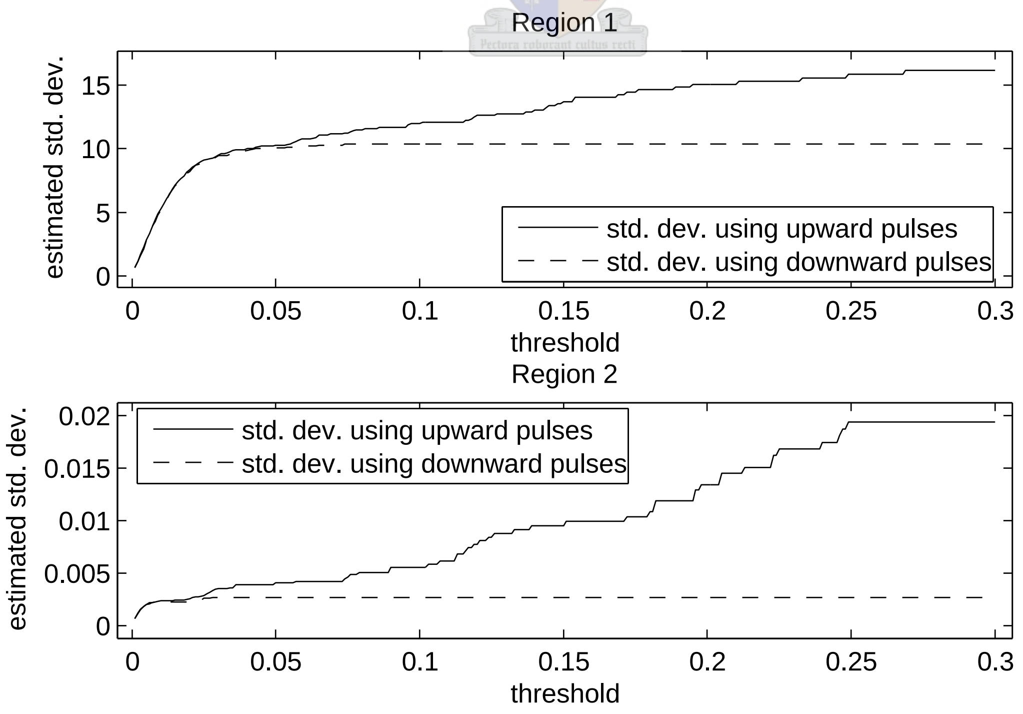


FIGURE 60. Estimated standard deviation for the two noisy regions in our sequence.

CHAPTER 9

Share price smoothing

Conradie et al. [1] introduce the application of *LULU* smoothers on financial data to the econometrical and statistical literature, and compare the properties of these smoothers with other operators commonly used on financial data. We take a different approach: focusing instead on interpreting the interval of ambiguity as a measure of time-scale dependent stock price volatility, and using the smoother G_n^∞ as an unbiased estimator of trend. In previous chapters we have discussed what smoothing means in the *LULU* perspective. To recap, a sequence is smoothed by mapping it into a higher smoothness class (local monotonicity class) by removing the narrower arcs. We wish to demonstrate this on some real data: the daily closing prices for a stock in the New York Stock Exchange. Figure 61 shows a 10 year segment of the daily closing prices of some stock.

Stock prices are not a fixed quantity; a buyer and a seller need to be matched to create a trade. Sellers set prices for the shares they want to get rid of, while buyers specify how much shares they want to purchase and the amount they are willing to pay. Not all offers generate a successful trade, as shares on offer go to the highest bidder. When the demand rises for a stock, sellers can increase their asking price and still get buyers. Many agents, with their own individual agendas, interact with this system at the same time. These many individual supply and demand forces 'push' the share price around; the total effect being observable as trend (or lack thereof). The time evolution of share prices has been observed to posses a fractal nature, i.e. fluctuations (and trends) exist at all scale levels.

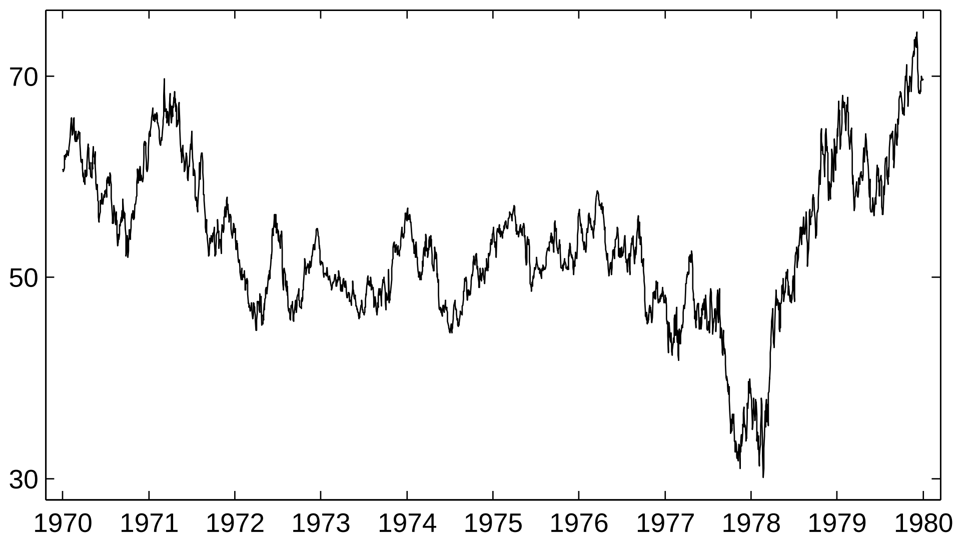


FIGURE 61. Daily share price of a stock for ten years from 1970 to 1980

It makes sense to speak not of a single share price, but of an interval of prices. For example, the daily low and high prices form an interval which contains all the price fluctuations during that day. When this interval is larger, one could interpret the share price as being more volatile. On the other hand, if said interval is smaller one can come closer to pinpointing a single share price. This is similar to the *LULU* concept of an interval of ambiguity (mentioned first in section 2.2). Ambiguity can be understood as a measure of the magnitude of fluctuations. A narrower interval of ambiguity implies more certainty about a value, whereas a wider interval of ambiguity points to uncertainty regarding the multi-resolution structure of data.

Stock markets are only open on weekdays; each week then has five closing prices (ignoring public holidays). A month is defined as four weeks (i.e. 20 market days), and a year as 52 weeks (i.e. 260 market days). We want to investigate the larger scale trend in stock price. Specifically, we are interested in extracting all fluctuations from the data that have a duration shorter than specific time-spans: a week, a month, and a year. Having automated procedures to do this is useful.

The *LULU* operators L_nU_n and U_nL_n are a natural choice. Calculating both of them will provide us with an interval of ambiguity. We do so now for $n = 5$ (a week), $n = 20$ (a month), and $n = 260$ (a year). The output sequences with all fluctuations of duration shorter than a week, a month, and a year removed are shown in figure 62. Data outside of the range displayed were also used to smooth the data: for a smoothing time-span of length t, data for times $2t$ before and after the displayed section was needed. For each smoothing width (time-span) there are two output data sequences: $L_nU_nx \geq U_nL_nx$. It seems that the *LULU* operators were successful in removing the shorter scale trends. Sometimes the removal of shorter scaled fluctuations highlights ambiguities regarding the larger scale trend, which is represented by the size of the interval of ambiguity. The larger (time-) scale fluctuations have a larger absolute magnitude, and we thus see that the interval of ambiguity is larger when more of the fluctuations are removed.

The operator G_n^∞ (section 2.5) is used to create an unbiased estimator of trend. This, together with the (biased) interval of ambiguity boundaries calculated using L_nU_n and U_nL_n is plotted in figure 63. To present results that are more visually appealing, we apply linear running average filters to the interval boundaries and the unbiased trend estimation. The unbiased operator G_n^∞ follows the slow-scale trends of the underlying share price. The size of the interval of ambiguity $|L_nU_n - U_nL_n|$ (the distance between the two lines above and below the unbiased trend estimation) measures the uncertainty of the share-price for a time-scale of either one month, or one year. In essence, this uncertainty is larger at a data element when there is fluctuations of width (a time-span in this case) smaller than the operator width n (in L_nU_n etc.) at that element. Notice that the yearly trend estimations extracted the large dip around year 1978 as it was not of sufficient duration. We see that the combination of the standard *LULU* operators and the unbiased smoother G_n^∞ allows one to extract longer time-scale trends and obtain an interval which quantifies the associated uncertainty.

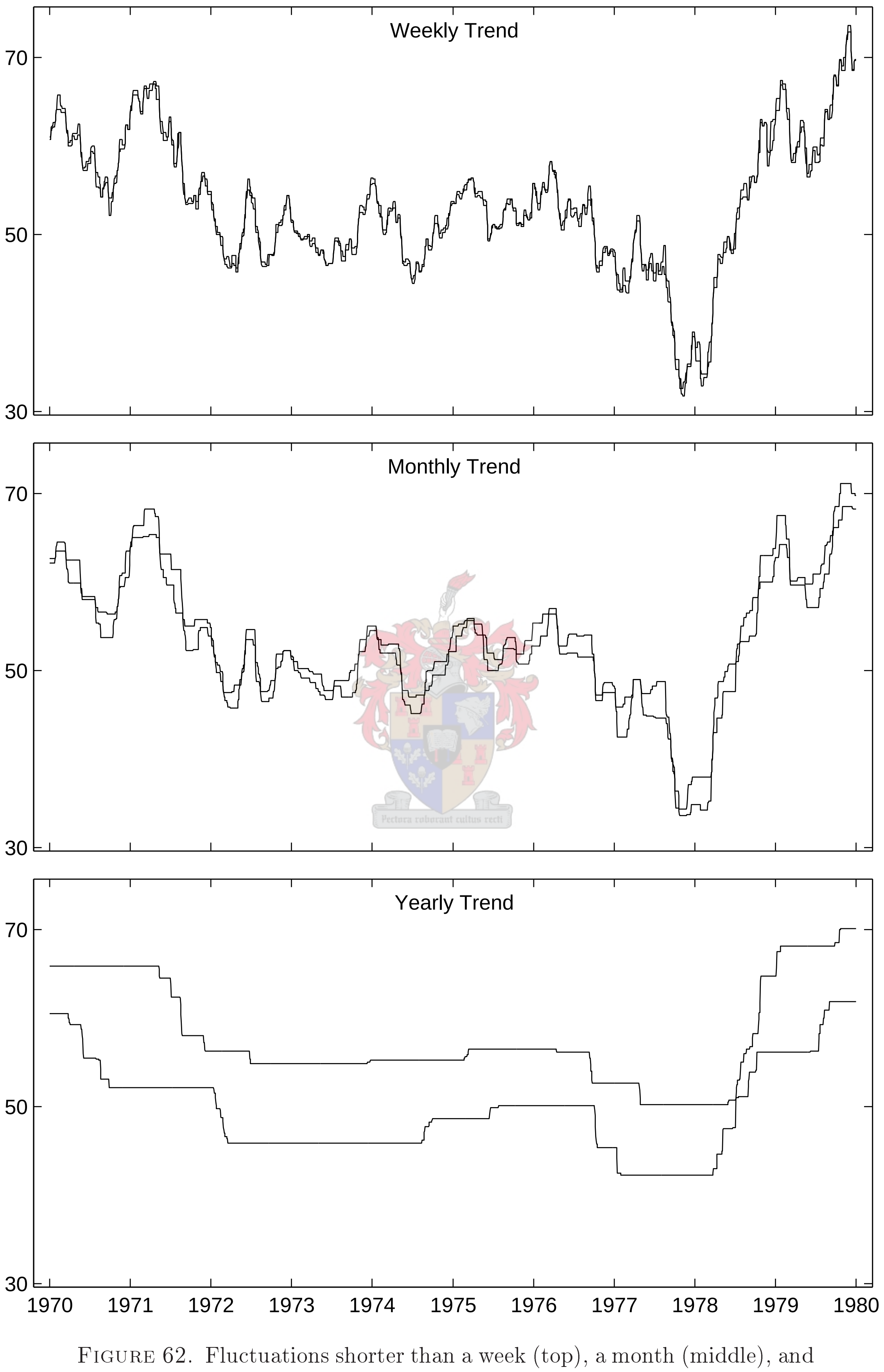
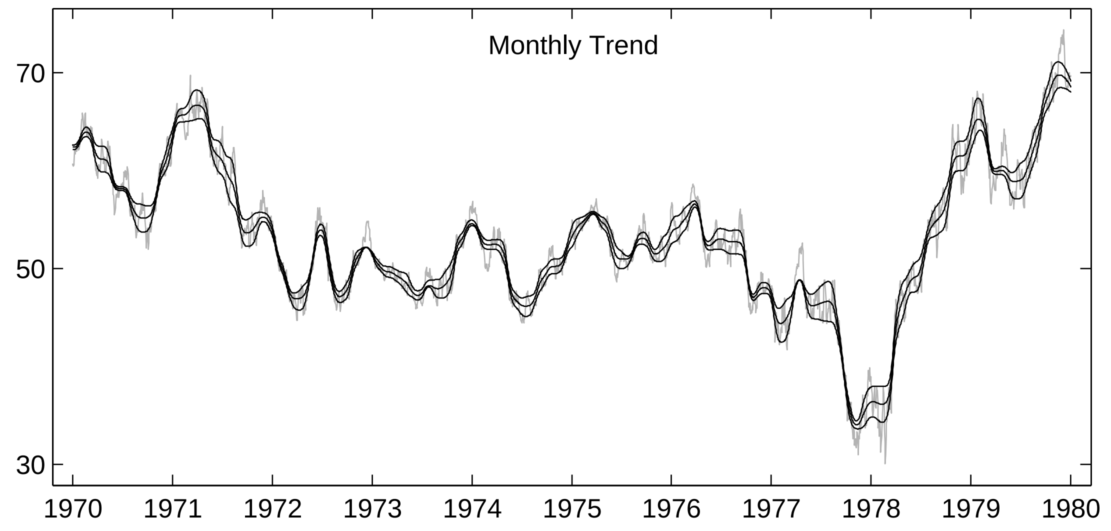


FIGURE 62. Fluctuations shorter than a week (top), a month (middle), and a year (bottom) removed using the operators L_nU_n and U_nL_n.



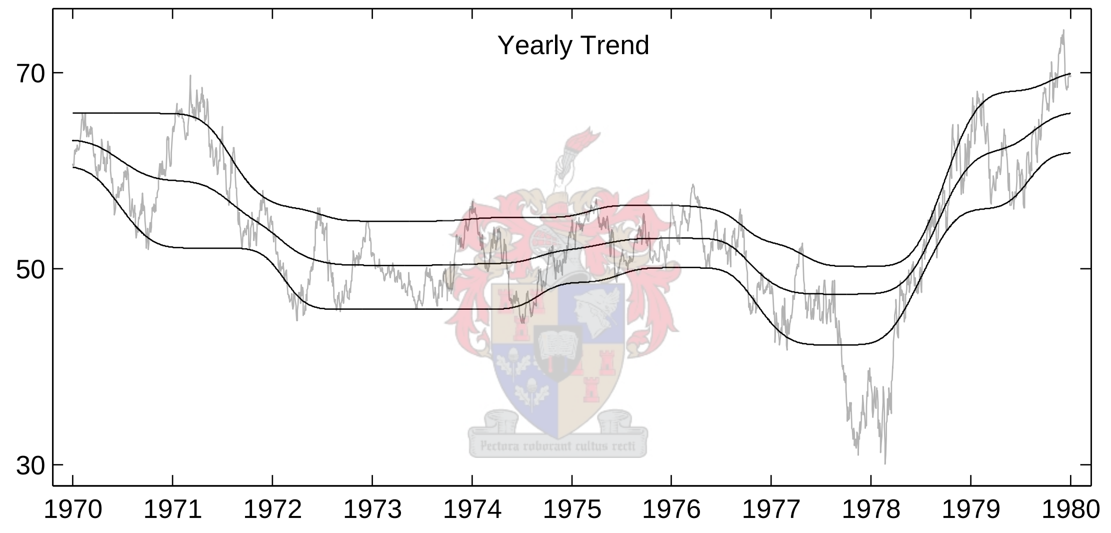


FIGURE 63. Trend and intervals of ambiguity smoothed using repeated linear running average filter

CHAPTER 10

Image processing

Scheidt et al. [14] employ optical second harmonic imaging in the analysis of $Pb_xCd_{1-x}Te$ ternary alloys. The sample wafers are prepared by the vertical Bridgman method from a homogenized melt of pure Pb, Cd and Te and show two distinct segregated phases consisting mainly either of Pb-rich or Cd-rich crystalline material. Areas in the Pb-rich phase enclose fine Cd-rich microcrystals due to PbTe acting as a solvent for CdTe during the solidification process of the melt. The Cd-rich phase occurs in two different crystalline growth directions. Large area grains (several mm^2) of (111) and (411) crystalline growth direction are identified by second harmonic (SH) imaging at different azimuthal angles and characterized by a $\sim 30°$ phase shift in the rotational SH anisotropy curves. The Cd-rich micro-crystal present in the Pb-rich phase are strongly aligned in the (111) direction. The SH response of pure PbTe (rock-salt crystal structure) is found to be at least two orders of magnitude weaker than that of pure CdTe (zinc-blende crystal structure), indicating that the Cd-rich phases dominate the SH response of the $Pb_xCd_{1-x}Te$ ternary alloy.

We investigate employing the *LULU* perspective in automated image analysis. First examine at the images as measured using optical second harmonic imaging. The two images in figure 64 show the same area but with a rotation of the azimuthal angle by approximately 30° in the right image. There are two large scale Cd-rich phases present which are characterized by a large SH response in one of the two images, due to a phase shift ($\sim 30°$) in the rotational SH anisotropy curves of the two crystal growth directions. The Pb-rich areas show a weak signal superimposed with a large variance random distribution which is attributed to the presence of Cd-rich micro-crystals [14].

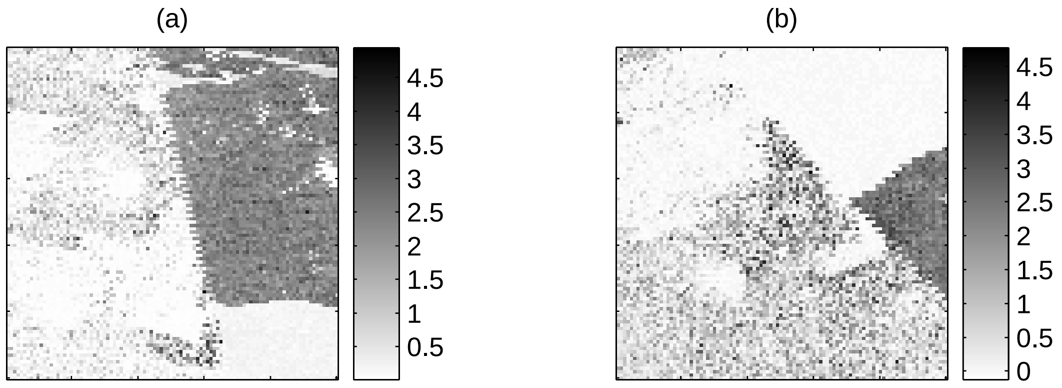


FIGURE 64. SH images of $Pb_{0.2}Cd_{0.8}Te$ (recorded in p-p polarization)

The discrete pulse transform and the LULU filters on which it is based usually operate on 1 dimensional sequences. It is necessary to define how this is extended to two dimensions. To analyze these images we make use of row- and column-based decompositions as discussed in section 6.2. Each row of the image will be regarded (and decomposed) separately. After processing is done on the rows it is possible to reconstruct the image using the processed rows. The same is done for the columns. Keeping the two decomposition directions separate, results in two images (refer to figure 45). For some purposes a single image is calculated by finding the average of these two images.

Using the tools in the *LULU* framework along with other methods we try to create automated procedures for identifying the features present in the images. We would also like to integrate the detected information from each image into a composite.

10.1. Highlighting of Cd-rich crystals

The human visual processing system has no trouble in recognizing the high response Cd-rich areas in both images in figure 64. We want to automate this process. We do this by suppressing the contribution of small structures and leaving large scale structures intact.

Each of the differently oriented images are decomposed row-by-row and column-by-column using $\mathrm{DPT}_{L_{10}}$(which extracts only positive pulses). The input image is then reconstructed using a selection of pulses. Pulses of width up to 10 form part of to the reconstruction only if it has a supporting base higher than the pulse itself. Specifically, a pulse only contributes to a corresponding image element in the output image if either the vertical or horizontal residual images have a higher value at that location than the pulse height (which implies the existence of a structure of wider than 10 units in one of the decomposition directions).

Figure 65 shows the result of applying the procedure on each of the two images in figure 64. Compared to the original images, the Cd-rich crystals were highlighted by the removal of most of the high variance noise found in the Pb-rich areas. A simple thresholding operation can be used to find the set of pixels comprising the Cd-rich crystal in each image.

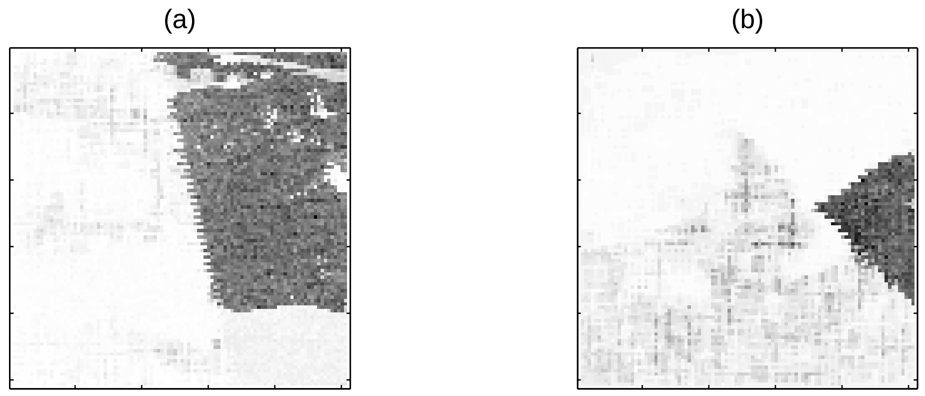


FIGURE 65. The images with the Cd-rich phases enhanced by the removal of small pulses.

Determining the crystal regions automatically should be more accurate if information from both images are used. From [14] we know that the Cd-rich phase is characterized by a high response at one azimuthal angle (the one image) and a zero response when the azimuthal angle is rotated by $\sim 30°$ (the other image). With equations relating a position in one image to a position in the other image, one can use this characterization to determine which crystal each image element belongs to. This is exactly what we aim to do next.

The crystal boundaries form straight lines. Knowing the equation of these lines will allow us to determine the transformations needed to find corresponding points in the two images. The goal is an automated algorithm that works for these type of crystal structures, which rules out simply calculating it by hand. For this, we need to know what parts of the image form part of the crystal boundaries.

10.2. Edge detection using the discrete pulse transform

Doing conventional edge detection can be problematic due to the noisy nature of the SH images. The high variance noise caused by the large amount of micro-crystals will yield many edges that we actually want to ignore when trying to find the crystal boundaries. Using the highlighted images from the previous section with conventional edge detection techniques will provide much better results.

An alternative is to use the edge detection technique discussed in section 5.3. The highlighted image is decomposed (this time using $DPT_{U_N L_N}$) along each horizontal and vertical line. For the purpose of recognizing edges, replace each block-pulse with the border elements of that pulse and reconstruct. Wherever a pulse exists in the decomposition, only the edges of that pulse shall remain in the reconstruction. We are only interested in finding the edges of large scale structures, therefore ignore all pulses smaller than width 10 in the reconstruction. This gets rid of edges caused by the graininess of the images. Figure 66 shows this procedure to be effective in finding the edges of the crystal boundaries when applied to the highlighted images in figure 65.

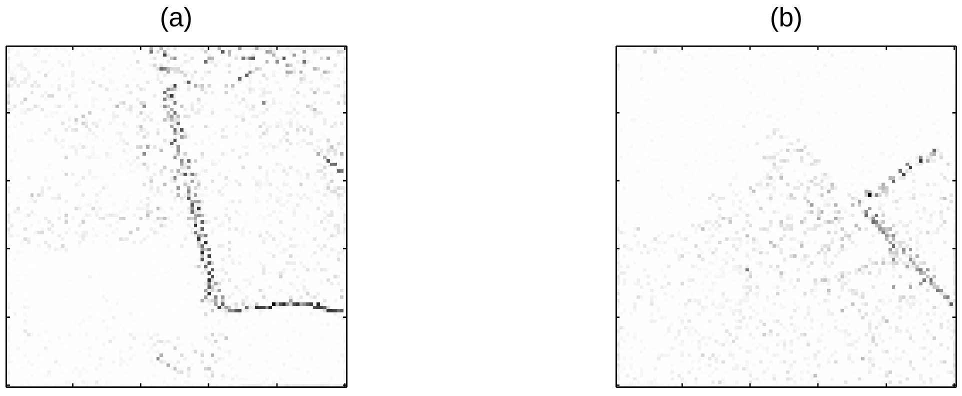


FIGURE 66. Edge pixels extracted by skipping the first 5 resolution levels.

It is now possible to find the equations that describe the boundary lines between the crystals. A standard technique to identify straight lines in an image is the Hough Transform. The Hough Transform maps every image element in the $x - y$ plane into a curve in $\rho - \theta$ space, defined by equation 10.1. The curves are weighted by the magnitude of the corresponding image element. A straight line consisting of many pixels will correspond to a large peak in the Hough Transform of the edge image.

$$x \sin\theta + y \cos\theta = \rho \tag{10.1}$$

The Hough Transform is calculated using the edge pixels of figure 66. By taking a weighted average of the area around the highest peak it is possible to find the parameters ρ and θ. This yields the line which is most strongly suggested by the collection of edge pixels. All edge pixels close to the this line are then removed before the Hough Transform is recalculated. Again, a weighted average around the highest peak is used to obtain the line parameters. As there are only two prominent crystal boundaries in the data, we stop here. In figure 67 we see that this method is successful in obtaining reasonable estimates of the crystal boundaries.

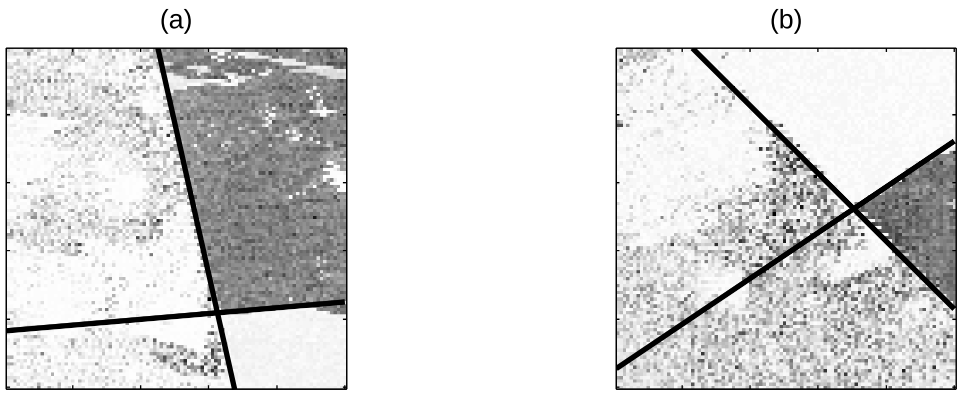


FIGURE 67. SH images with identified crystal boundaries

10.3. Image registration

Image registration refers to finding equations that map each image element of one image to a corresponding image element in another image. We wish to combine the two SH images of figure 64 into one image using these equations. The detected lines as displayed in figure 67 provide enough information to do this.

First determine the point where the lines cross in both the images, as these points must clearly be at the same absolute location. From equation 10.1 get the y coordinate of the intersection point in each image:

$$I_y = \frac{\rho_1 - \rho_2 \frac{\sin\theta_1}{\sin\theta_2}}{\cos\theta_1 - \cos\theta_2 \frac{\sin\theta_1}{\sin\theta_2}}, \tag{10.2}$$

where θ_i and ρ_i are the parameters of the lines found in section 5.3. The x-value of the intersection point is then easy to determine as well. If θ_1 is very close to zero, then the second version of equation 10.3 should be used.

$$I_x = \tfrac{\rho_1 - y\cos\theta_1}{\sin\theta_1} \qquad \text{or} \qquad I_x = \tfrac{\rho_2 - y\cos\theta_2}{\sin\theta_2} \tag{10.3}$$

Equation 10.2 and 10.3 allows us to determine the position of the intersection point in both images. Next, we calculate how much each line in figure 67b must be rotated to fit onto the corresponding lines in figure 67a. Due to approximation errors and noise the two calculated angles will differ slightly. With the data analyzed here the difference is about 3 degrees. The mean of these two values is used as the angle, α, by which the azimuthal angle was rotated before the second SH image was measured. We get:

$$\alpha = 30.65^o$$

This agrees with the approximate value of 30° known from the experimental setup [14]. Now transform the second image to place it in the correct position and orientation with respect to the first image using the composition of three affine transformations.

$$T_{\mathrm{I}} \circ R_\alpha \circ T_0$$

First the image is translated to put its intersection point at $(0, 0)$. Then the image is rotated by α. Finally the image is moved such that the two intersection points lie at the same position. See figure 68. For any image element in the second image, the corresponding element in the first image can be found using this transformation.

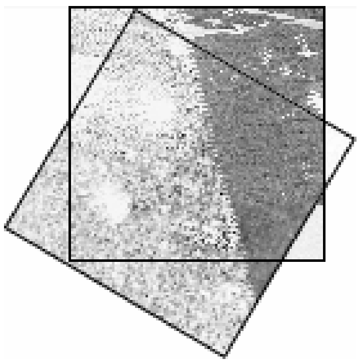


FIGURE 68. Result of image registration

10.4. Results

In the overlapping parts of the registered images there is information from both of the second harmonic images available. This can be used to simplify some data analysis tasks. For example, the Cd-rich crystals are identified by looking for areas with high SH response in one of the highlighted images and close to zero response in the other image. Using the transformation function each image element is tested using this criteria. In the test data, this yields two separate Cd-rich crystals of different phases. The remaining area is classified as Pb-rich. To summarize the information obtained so far a composite image is created (algorithmically), displaying the crystal regions and detected crystal boundaries (figure 69). Only the parts that are visible in both of the SH images are displayed.

The SH imaging of the Pb-rich areas show a weak response with a superimposed random distribution. This is caused by the presence of Cd-rich micro-crystals [14]. Quantifying the standard deviation of the distribution allows one to compare the concentration and growth direction of Cd-rich micro-crystals in the Pb-rich area. In section 7.3 we discussed a way of estimating the standard deviation of a 2d image. Applying this here results in figure 70 (a darker gray-scale implies a higher concentration). As expected from the differences in

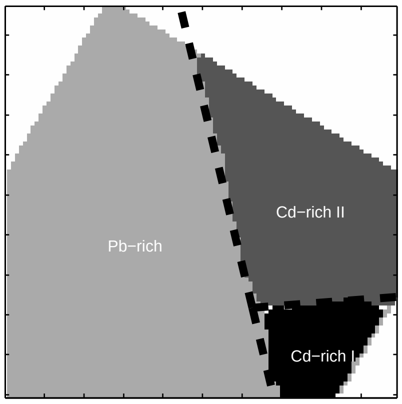


FIGURE 69. Crystal regions and boundaries.

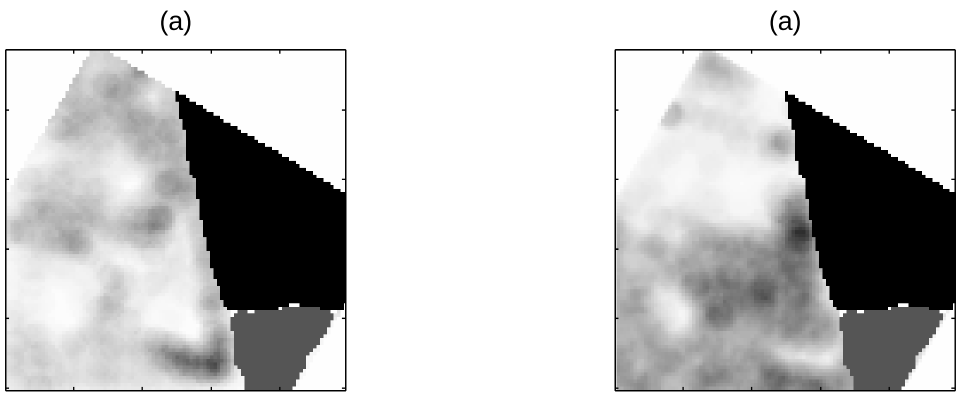


FIGURE 70. Visualizing the variance in the Pb-rich areas.

rotational anisotropy curves of the two Cd-rich phases, when the concentration of micro-crystals is high in one image it is low in the other. Most of the micro-crystals are of the same phase as the bottom Cd-rich crystal.

The constants (i.e. the degree of smoothing applied, and number of levels left out of reconstructions) used in this image processing procedure depends on the scales present in the images and not any specific images features, and should be the same for other images of this type.

The *LULU* framework provided tools that simplified the creation of automated procedures for the efficient and robust processing of real-world data. In practice, systems like this can decrease the amount of human intervention needed. This is especially useful when large sets of similar data need to be processed and analyzed (i.e. quality checks on an assembly line).

Closing remarks

The concepts behind, related tools and practical uses of the $LULU$ perspective and the discrete pulse transform have been discussed, with some focus on differences in viewpoint with respect to the Fourier Transform and the Wavelet Decomposition. Application of the $LULU$ tools on a variety of practical data processing problems was demonstrated. The idea is that, through these discussions and (often simple) demonstrations, an intuition and practical know-how regarding the $LULU$ perspective has developed, allowing the reader to envisage and implement analysis procedures based on these ideas.

Future research avenues include:

(1) Which concepts in the $LULU$ framework generalize to Mathematical Morphology?
(2) Further comparison of median smoothers with the $LULU$ operators.
(3) Determination of necessary and sufficient conditions for the consistency of a discrete pulse transform based on an arbitrary set of operators. Is a *dpt* based on the unbiased smoothers G_n and H_n consistent?
(4) Design of other highlighting procedures.
(5) True extension of the pulse perspective to two dimensions (as discussed in 6.1), i.e. a combination of leaf-based 2-d decomposition (with a $LULU$ structure), and a 2-d characterization of pulses.
(6) Estimation of other moments using a pulse decomposition, and the feasibility of using both upward and downward pulses of L_1U_1 and U_1L_1 to estimate standard deviation.
(7) Compression, transmission and storage of images.

Bibliography

[1] 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.

[2] O. Kao. Modification of the LULU operators for preservation of critical image details. In H. Arabnia et al, editor, *Proceedings of the 2001 International Conference on Imaging Science, Systems, and Technologoy*, pages 280–286. CSREA Press, 2001.

[3] C. L. Mallows. Some theory of non-linear smoothers. *The Annals of Statistics*, 8: 695–715, 1980.

[4] A.E. Marquardt, L.M. Toerien, and E. Terblanche. Applying nonlinear smoothers to remove impulsive noise from experimentally sampled data. *J. SA I Mech.*, 7(1):15–18, 1991.

[5] C. H. Rohwer. Concepts and measures of smoothness in sequences. To be published.

[6] C. H. Rohwer. Estimating moments of an unknown distribution contaminating measurements. To be published.

[7] C. H. Rohwer. LULU operators for two-dimensional data. To be published.

[8] C. H. Rohwer. Multiresolution analysis with pulses. In M. D. Buhmann and D. H. Mache, editors, *Advanced Problems in Constructive Approximation*, volume 142 of *International Series of Numerical Mathematics*, pages 165–186. Birkhauser Verlag, 2002.

[9] C. H. Rohwer. *Multiresolution Analysis with Pulses.* Birkhauser-Verlag, 2005.

[10] C. H. Rohwer and J. P. Harper. On the consistency of a separator. To be published.

[11] C. H. Rohwer and D. P. Laurie. The discrete pulse transform. *SIAM Journal of Mathematical Analysis*, 38(3):1012–1034, 2006.

[12] C. H. Rohwer and M. Wild. Contributions towards a mathematics of vision. To be published.

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

[14] T. Scheidt, E. G. Rohwer, H. M. von Bergmann, E. Saucedo, E. Diéguez, L. Fornaro, and H. Stafast. Optical second-harmonic imaging of $Pb_xCd_{1-x}Te$ ternary alloys. *Journal of Applied Physics*, 97:103104.1–103104.6, 2005.

[15] J. Serra. *Image Analysis and Mathematical Morphology.* Academic Press, 1982.

[16] J.-L. Starck. Non linear multiscale transforms. In T.J. Barth, T. Chan, and R. Haimes, editors, *Multiscale and Multiresolution Methods*, pages 239–278. Springer, 2002.

[17] J. W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, Reading, Mass., 1977.

[18] Zhou Xing-wei, Yang De-yun, and Ding Run-tao. Infinite-length roots of median filters. *Science in China, Ser. A*, 35:1496–1508, 1992.

APPENDIX A

The discrete pulse transform

The discrete pulse transform is one of the most useful tools in the *LULU* framework. We list a Matlab function that performs the decomposition in $O(n)$ time for any sequence.

```
function [residual, lev, width, pos, hgt]=dptstd(f, N, bias)
% [residual, lev, pos, hgt, r] = dptstd(f, N, alternate)
%
% Perform fast discrete pulse transform of sequence f
% up to level N using seperators S_n.
%   bias = 0 : S_n = L_n U_n   (bias downwards pulses)
%   bias = 1 : S_n = U_n L_n   (bias upwards pulses)
%
% Returns:
%   residual - residual sequence r^(0)
%   lev - Nxlength(f) matrix of resolution levels, r^(i) = lev(i, :);
%   M pulses:
%       width      - 1xM vector of pulse widths
%       pos        - 1xM vector of positions of left edge
%       hgt        - 1xM vector of pulse heights

% Default values
if (nargin<3), bias = 0; end;
if (nargin<2), N = length(f)-2; end;
% Require a row vector
if (size(f,1)>size(f,2)), f = f'; end;

% Differences smaller than TOL are assumed to be zero
TOL = eps*10;

%% Initialize catalogue/schedule/table of extracted features
diffval = diff(f);
diffpos = find(diffval~=0);
numD = length(diffpos);  % number of differences in f
% We require at least two differences to find a pulse
if (numD <= 1)
    residual = f;
    lev   = zeros(0, length(f));
    width = zeros(1, 0);
    pos   = zeros(1, 0);
    hgt   = zeros(1, 0);
    return;
end;
```

```

% Number of pulses removed from signal
numPulse = 0;

% Arrays holding extracted pulse information
pos = zeros(1, numD);
hgt = zeros(1, numD);
width = zeros(1, numD);

% Data matrix holding catalogue and schedule
d = [diffpos; diffval(diffpos); 0:length(diffpos)-1; ...textcolorcomment
    [2:length(diffpos) 0]; zeros(2,length(diffpos))];
% Indices into data matrix
cPos = 1; cVal = 2; cPrev = 3; cNext = 4;
fPrev = 5; fNext = 6; fLev  = 7;

% First and last entries in feature priority queues
fPosF = zeros(1,N);
fPosL = zeros(1,N);

%% Helper functions
    function addFeature(highPriority, leftIndex, featSize)
        if (featSize>N) return; end;
        % Add feature of size [featSize] at
        % index [leftIndex] to the the feature scheduler
        if (highPriority)
            % Bumps are placed in beginning (for UL)
            if (fPosF(featSize)>0)
                d(fPrev, fPosF(featSize)) = leftIndex;
                d(fNext, leftIndex) = fPosF(featSize);
            else
                fPosL(featSize) = leftIndex;
            end
            d(fLev, leftIndex) = featSize;
            fPosF(featSize) = leftIndex;
        else
            % Pits in the end (for UL)
            if (fPosL(featSize)>0)
                d(fNext, fPosL(featSize)) = leftIndex;
                d(fPrev, leftIndex) = fPosL(featSize);
            else
                fPosF(featSize) = leftIndex;
            end
            d(fLev, leftIndex) = featSize;
            fPosL(featSize) = leftIndex;
        end
    end

    function deleteFeature(where)
        % Delete feature at index [where] from schedule
        if (d(fLev, where)>0)
            if (d(fPrev, where)>0)
                d(fNext, d(fPrev, where)) = d(fNext, where);
            else
                % if that was first feature in row, update fPosF
                fPosF(d(fLev, where)) = d(fNext, where);
            end
            if (d(fNext, where)>0)
                d(fPrev, d(fNext, where)) = d(fPrev, where);
            else
                % if that was last feature in row, update fPosL
                fPosL(d(fLev, where)) = d(fPrev, where);
            end
            d(fLev,  where) = 0;
            d(fNext, where) = 0;
            d(fPrev, where) = 0;
        end
    end

    function deleteDifference(where)
        % Delete difference at index [where] from catalogue
        if (d(cPrev, where)>0)
            d(cNext, d(cPrev, where)) = d(cNext, where);
        end
        if (d(cNext, where)>0)
            d(cPrev, d(cNext, where)) = d(cPrev, where);
        end
        d(cPrev, where) = 0;
        d(cNext, where) = 0;
    end

    function extractPulse(feat, level)
        % Extract feature at position [feat] and update differences
        numPulse = numPulse + 1;

        width(numPulse) = level;
        pos(numPulse) = d(cPos, feat) + 1;

        height = sign(d(cVal, feat)) * ...
            min(abs(d(cVal, feat)), abs(d(cVal, d(cNext, feat))));
        hgt(numPulse) = height;

        f(d(cPos, feat)+1:d(cPos, feat)+level) = ...
            f(d(cPos, feat)+1:d(cPos, feat)+level)-height;
        d(cVal, feat) = d(cVal, feat) - height;
        d(cVal, d(cNext, feat)) = d(cVal, d(cNext, feat)) + height;

    end

    function [Shalf] = removeFeatures(curLev, order)
```

```
        % If order is true, remove upward pulses first
        % Depending on wheter S_n = L_nU_n or U_nL_n, start at
        % different ends of priority queue
        if (order)
            tfNext = fNext;
            tcNext = cNext;
            tfPrev = fPrev;
            tcPrev = cPrev;
            curFeat = fPosF(curLev);
            lastPulseSign = 1;
        else
            tfNext = fPrev;
            tcNext = cNext;
            tfPrev = fNext;
            tcPrev = cPrev;
            curFeat = fPosL(curLev);
            lastPulseSign = -1;
        end;
        nextPulseSign = lastPulseSign;
        Shalf=f;
        stored = 0;
        while (curFeat ~= 0)
            nextDiff = d(tcNext, curFeat);
            nextPulseSign = sign(d(cVal, curFeat));
            if (~stored && lastPulseSign ~= nextPulseSign)
                % Store copy of signal after one type of pulse removed
                % (Available for future enhancements)
                Shalf = f;
                stored = 1;
            end
            extractPulse(curFeat, curLev);
            % Position of current difference after pulse is removed
            curDiffAfter = curFeat;
            % Position of next difference after pulse is removed
            nextDiffAfter = nextDiff;

            % Was right difference removed
            if (abs(d(cVal, nextDiff)) < TOL)
                numD = numD-1;
                nextDiffAfter = d(tcNext, nextDiff);
                deleteFeature(nextDiff);
                deleteDifference(nextDiff);
            end

            % Was left difference removed
            if (abs(d(cVal, curFeat)) < TOL)
                numD = numD-1;
                curDiffAfter = d(tcPrev, curFeat);
                deleteDifference(curFeat);
            end
            nextFeat = d(tfNext, curFeat); % next feature to process

            % Possibly create new features
            if (curDiffAfter>0)
                if (nextFeat == curDiffAfter),
                    nextFeat = d(tfNext, curDiffAfter);
                end
                % Remove from schedule if feature already exists at this location
                deleteFeature(curDiffAfter);

                % Add to feature list if pulse found
                if  (nextDiffAfter>0 && ...
                     d(cVal, curDiffAfter) * d(cVal, nextDiffAfter) < 0)
                    addFeature(d(cVal, curDiffAfter)>0, curDiffAfter, ...
                        d(cPos, nextDiffAfter) - d(cPos, curDiffAfter));
                end
            end
            curFeat = nextFeat;
        end
        if (lastPulseSign == nextPulseSign)
            Shalf = f;
        end

    end

%% Scan for initial features
curDiff = 1;
nextDiff = d(cNext, curDiff);
smallestFeat = 999999999;
while (curDiff ~= 0 && nextDiff ~= 0)
    % Feature characterized by two differences of opposite sign
    if (d(cVal, curDiff) * d(cVal, nextDiff) < 0)
        featSize = d(cPos, nextDiff) - d(cPos, curDiff);
        addFeature(d(cVal, curDiff) > 0, curDiff, featSize)
        smallestFeat = min(smallestFeat, featSize);
    end
    curDiff = nextDiff;
    nextDiff = d(cNext, curDiff);
end

%% Skip levels where no pulses exist
df = zeros(N+1,length(f));
df(1,:)=f;
for i=2:min(smallestFeat,N);
    df(i,:) = f;
end

%% Remove features starting from those with highest priority
for curLev=smallestFeat:N
    removeFeatures(curLev, bias);
    df(curLev+1,:) = f;
    % Stop if no pulses left
    if (numD<=1), break; end;
end
df = df(1:curLev+1, :);


%% Set output variables
residual = f;
lev = -diff(df);
width = width(1:numPulse);
pos = pos(1:numPulse);
hgt = hgt(1:numPulse);
end
```