UNIVERSITY OF
KWAZULU-NATAL

# Classical Noise in Quantum Systems

by


## Ismail Yunus Akhalwaya


Submitted in fulfillment of the academic requirements for the degree of Doctor of Philosophy in the School of Chemistry and Physics,
University of KwaZulu-Natal, Durban

Supervisor: Prof Francesco Petruccione

December 2013

# Abstract

Quantum mechanics contains a fresh and mysterious view of reality. Besides the philosophical intrigue, it has also produced and continues to inspire tantalizing new technological innovations. In any technological system, the designers must contend with the problem of noise. This thesis studies classical noise in two different quantum settings. The first is the classical capacity of a quantum channel with memory. Adding forgetful-memory, attempts to push the boundaries of our understanding of how best to transmit information in the presence of correlated noise. We study the noise within two different frameworks; Algebraic Measure theory and Monte Carlo simulations. Both tools are used to calculate the capacity of the channel as correlations in the noise are increased. The second classical-quantum system investigated is atomic clocks. Using power spectral density methods we study aliasing noise induced by periodic-correction which includes the Dick Effect. We propose a novel multi-window scheme that extends the standard method of noise correction and exhibits better anti-aliasing properties.

A uniting thread that emerges is that correlations can be put to good use. In the classical capacity setting, correlations occur between uses of the quantum channel. We show that stronger correlations increase the classical capacity. The benefits of correlation are even seen at a meta-level within the framework of Monte Carlo simulations. Correlations are designed into the algorithm which have nothing to do with real-world correlations, but are abstract correlations created by a Markov chain employed in the algorithm to help efficiently sample from a distribution of exponential size. Finally, in the atomic clock setting, correlations in the measured noise are used to help predict and cancel noise on a short time-scale while trying to limit aliasing.

Channel capacity and precise time-keeping are distinct topics and require very different approaches to study. However, common to both topics is their application to communication and other tasks, the need to overcome noise and the benefits of exploiting correlations in the noise.

# Preface

The work described in this dissertation was carried out while the author was

- a research visitor at the University of Cambridge, November 2006 - November 2007
- a part time lecturer at the University of Kwazulu Natal, 2008
- a full time researcher at the Center for High Performance Computing, 2009
- a full time lecturer at the University of Johannesburg, 2010 - Present.

The author first registered as a full-time PhD candidate in 2007 at the School of Physics, University of KwaZulu-Natal, Westville, Durban, under the supervision of Professor Francesco Petruccione. He subsequently switched to part-time status in 2009.

The studies represent original work by the author and have not otherwise been submitted in any form for any degree or diploma to any tertiary institution. Where use has been made of the work of others it is duly acknowledged in the text.

## Declaration 1 – Plagiarism

I, **Ismail Yunus Akhalwaya** declare that

1. The research reported in this dissertation, except where otherwise indicated, is my original research.

2. This dissertation has not been submitted for any degree or examination at any other university.

3. This dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

4. This dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

   a) Their words have been rewritten but the general information attributed to them has been referenced.

   b) Where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced.

5. This dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation and in the Bibliography.

Signed: _____

## Declarations 2 – Publications

### Publication 1

*The Classical Capacity of a Qubit Depolarizing Channel with Memory*, J. Wouters, I. Akhalwaya, M. Fannes, F. Petruccione (Phys. Rev. A 79, 042303, 2009)

Signed: _____

# Contents

# List of Figures

# List of Tables

# Acknowledgements

Bismillahirahmaniraheem

"Whoever does not thank people, has not thanked God." – pithy saying.

Many people have contributed to making this thesis possible*. That contribution may have come in the form of personal or academic support, and both cases were literally crucial and highly appreciated.

I would like to start by acknowledging my family and their immense sacrifices: Ayesha for giving of her herself and giving up so much, Maryam and Asma for freely giving their innocent unreserved affection, Mum and Dad for giving their lives to their kids, Yasmeen and my siblings for giving their respect and assistance and the Omars and Meriam Matenche for giving their well-wishing and assistance. Thank you for your love, patience and prayers.

*This is, of course, a cliche and perhaps what seemingly makes cliches boring, is that many people have experienced them and so there is no useful novel information in subsequent experiences and reporting of the experience. But the view that cliches are boring, perhaps misses their most essential aspect, which probably led to the sayings becoming cliches in the first place. What makes meaningful cliches actually precious, is that they are experienced by a given person for the first time and in a personal, subjective way. This is valuable to the person reporting it and to the people involved in the experience. Furthermore, just by having this view above, even people not directly involved, can reflect on their own experiences and make a special connection between the subjective and the paradoxical *common* experience of the subjective. I say all of this because I really want to express how moved I am by all the assistance.

Next, I would like to mention a list of people/institutions who have been involved in the PhD either directly or indirectly, and leave the descriptive thanks to in-person discussion: Hermann Uys (as well as his collaborators Michael Biercuk and Jarrah Sastrawan), Timothy Winters, Ken Matthis, the Musallees at UKZN and UJ, Brian Vaddan, David Robinson, the NLC (including Alpha Ebrahim and Yingwen Wein), the CHPC (including computing time), Nilanjana Datta, Jeroen Wouters, Mark Fannes, Tony Dorlas, Artur Ekert, Andreas Buchleitner and group, the Applied Math Department at UJ, Thomas Konrad, Mervlyn Moodley, the Quantum group at UKZN (including Ilya Sinayskiy and Mhlambululi Mafu) and Francesco Petruccione, my supervisor.
Finally, I would like to acknowledge funding from the NRF, Canon Collins-Mamphela Ramphele-Chevening Scholarship, UKZN and UJ.

# Chapter 1

# Introduction

## 1.1 Quantum Mechanics

Quantum mechanics is a mathematical description of physical reality [1, 2]. As with any mathematical model, certain assumptions about the fundamental structure of reality are made when the model is mapped to the real world. The verification of these assumptions comes indirectly in the 'tasting'*, that is, how well the theory predicts the outcomes of experiments. Quantum theoretical predictions have been more accurately confirmed than any other man-made physical theory to date. The strange thing about these fundamental quantum axioms and assumptions [3], besides the strangeness of complex *superposition*, the weirdness of *wave-collapse* and the plethora of unintuitive behaviours it implies, is that, almost uniquely amongst scientific theories, we don't know what the axioms actually mean in terms of the underlying structure of reality. There's no shortage of possible interpretations including the canonical observer-based Copenhagen interpretation [4], the pilot-wave Bohmian interpretation [5], the many-world's Everett interpretation [6] and the Consistent Histories interpretation [7], to name a few, but none of them are testable (except, perhaps the *quantum suicide-test* [8] of the many worlds interpretation, which can't really count because the experimenter can't share his/her

*This refers to the saying, 'The proof of the pudding is in the tasting' and is used deliberately since engagement with reality is ultimately a subjective experience, like tasting. The ontological question of reality is made all the more curious within the framework of quantum mechanics.

results and the experiment of course would probably be unethical[†]).

## 1.2 Randomness

One important issue that has fascinatingly different interpretations, is the question of *randomness*. Is randomness intrinsic to reality or merely a reflection of the lack of deterministic information? What is the proper understanding of the transition between the quantum and classical regimes and what more must be added to decoherence in order to arrive at a complete picture? Does wave collapse really collapse some real-world correspondence to the wave-function and really introduce randomness or is it merely a change in our knowledge of the real world? What does Heisenberg's uncertainty principle say about reality? Are complex amplitudes just a way of introducing 'negative' probabilities?[‡] Can negative probabilities make sense? Can probabilities make sense? This last tongue-in-cheek question refers to the oft debated and perhaps still open problem in classical statistics, of the interpretation of probabilities, along the lines of the Bayesian versus Frequentist views. It is interesting that this debate, which is all about states of knowledge and randomness, also hinges on issues of subjectivity.

Besides the above philosophical aspects of randomness, the practical aspects are very important to real world applications of quantum systems. How can decoherence be best managed in order to enable quantum computing? Can metrology benefit from quantum systems in order to beat classical noise? Can quantum noise actually help improve efficiency of the transport of energy in quantum networks? Can correlations in the noise help to combat noise?

[†]For perhaps at least two separate reasons, the first being that it is an experiment that involves suicide, even if one interpretation has it that consciousness will not be destroyed, and secondly even if the many worlds interpretation is correct, parallel consciousnesses will be killed. Actually these considerations, at a meta-level, open up a whole Pandora's box of Schrodinger's cats, by drawing attention to the fact that especially in this case, ethics is relative to one's model of the universe, and specifically of consciousness itself.

[‡]A Quantum Turing Machine can be constructed without complex numbers but with negative numbers.

## 1.3 The Overarching Subject of the Thesis

This thesis does not deal with any of the philosophical questions on randomness, but it does take their importance as motivation to study random noise. The study of randomness includes building mathematical tools that describe it and using these tools in the service of answering some of the practical questions. Randomness has both classical as well as essentially quantum aspects. Classical randomness refers to a lack of knowledge about deterministic states, for example, the outcome of a coin toss. Quantum randomness refers to measurement induced randomness (wave-collapse) or curiously, the equivalent notion of indeterminate subsystems of a larger pure state (tracing). This thesis chooses to focus on classical randomness but in relation to quantum systems.

One of the main practical applications of the ideas in this thesis is in the field of communication. Safe and secure communication is the bloodline of our information technology world. Information theory, the study of information and communication resources, lays the theoretical foundations upon which that technology operates. The most recent major advancement in the field of information theory and technology has been the expansion from a classical to a quantum worldview. There are new unrivalled possibilities such us quantum computing, quantum channels and quantum metrology but also new challenges such as degradation due to quantum decoherence of noisy environments.

The overall aim of this doctoral thesis is to better understand noise and to try and manage it. We look at two different aspects of noise that, for example, affects communication. The first aspect is the study of the capacity of a noisy single-qubit quantum channel. Our specific primary concern here is understanding the capacity's dependence on the correlations in the noise. We approach this part of the study using algebraic [9] as well as numerical Monte Carlo methods [10]. The application of these methods attempts to introduce novel ways of calculating the capacity of quantum channels. The second aspect is the noise in quantum metrology, specifically in an atomic clock setup. This noise places limitations on the precision with which time can be kept which ultimately has a bearing on the speed of communication. Our primary concern here is studying

aliasing and the Dick Effect [11] using Fourier Analysis methods.
It is hoped that this thesis could help, in a small way, to combat noise in real-world applications. It also presents the overall idea that correlations should explicitly be looked out for, perhaps in different guises, as a source of benefit when trying to achieve certain information-related tasks. Finally, it is dreamed that this idea of the emphasis on correlations, may perhaps indirectly, help answer some of the philosophical questions outlined above, but no progress is made towards this goal.

## 1.4 Structure

This thesis consists of eight chapters and an appendix of code. The main topics come in the form of a background chapter followed by a 'paper' chapter. Each paper chapter has either already been accepted by a journal or is in the process of being submitted to a journal. Only the first paper chapter (Ch. 3) has co-authors that were also involved in the actual writing of the paper [12]. My estimated contribution to that paper is 50%. For the remaining chapters, I am the sole writer, with the other participants providing editing and guidance.

# Bibliography


[1] *Modern Quantum Mechanics*, J.J. Sakurai, (Addison-Wesley; 1994)

[2] *The Principles of Quantum Mechanics (fourth edition)*, P.A.M. Dirac, (The International Series of Monographs on Physics 27; Oxford University Press; 1982)

[3] *Quantum Computation and Quantum Information*, M.A. Nielsen and I.L. Chuang, (Cambridge University Press; 2000)

[4] *Who invented the Copenhagen Interpretation? A study in mythology*, D. Howard, (Philosophy of Science; 2004)

[5] *A Suggested Interpretation of the Quantum Theory in Terms of "Hidden Variables"*, D. Bohm, (Physical Review 85; 1952)

[6] *Theory of the Universal Wavefunction*, H. Everett, (Thesis; Princeton University; 1956)

[7] *Properties of Consistent Histories*, F. Dowker and A. Kent, (Phys. Rev. Lett. 75; 3038 - 3041; 1995)

[8] *The Doomsday Device*, H. Moravec, (Mind Children: The Future of Robot and Human Intelligence; Harvard University Press; p. 188; 1988)

[9] *Functions of Markov processes and algebraic measures*, M. Fannes, B. Nachtergaele, and L. Slegers, (Rev. Math. Phys. 4; 39; 1992)

[10] *Monte Carlo Sampling Methods Using Markov Chains and Their Applications*, W.K. Hastings (Biometrika Vol. 57, No. 1 (Apr., 1970) pp. 97-109)

[11] *Minimizing the Dick Effect in an Optical Lattice Clock*, P.G. Westergaard, J. Lodewyck, and P. Lemonde (IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, Vol. 57; No. 3; 2010)

[12] *The Classical Capacity of a Qubit Depolarizing Channel with Memory*, J. Wouters, I. Akhalwaya, M. Fannes, F. Petruccione (Phys. Rev. A 79, 042303, 2009)

# Chapter 2

# Channel Capacity Background

## 2.1 Classical Information Theory

The notions of *knowledge* and *wisdom* are abstract concepts that have intrigued philosophers throughout the ages. A particularly ingenius and less ambitious approach to partially understanding the issue was made by Claude Shannon, the father of information theory. Shannon separated out the notions of meaning and symbols. He left the meaning of meaning to the philosophers and employed the tools of probability, to describe the frequency of occurrence of the symbols. With a random variable treatment of symbols, his work became one of the foundation stones in the digital computing and communications technological age. He even coined the now ubiquitous term 'bit' of information [1].

### 2.1.1 Entropy

In his landmark 1949 paper [1], Shannon mathematically defined the concept of *information entropy* of a discrete random variable $X$,

$$\mathrm{S}(X) = -\sum_x p_x \log p_x.$$

The discrete variable $x$ runs over the set of symbols/letters called the alphabet, $x \in \mathcal{A}$. The value $p_x$ is the probability of occurrence of the letter $x$, for every independent* draw from the random variable $X$, often called the *source*.

*Shannon's results may be extended to stationary random variables.

Entropy is a measure of uncertainty or unpredictability in the source. If the source is an origin of 'annoying' noise, then entropy measures how noisy and variable that noise is. If the source is related to some useful data source, for example from the weather report, then entropy is a measure of how much can be learnt on average with every new piece of information received from the source, that is, the number of random possibilities that are eliminated in learning the actual outcome. Thus, entropy is a measure of the size of the number of probable outcomes (as opposed to the number of possible outcomes). The notions of the uncertainty before a random useful outcome is known, and the information gained once a 'reading' has been taken, are two views of the same concept of entropy.
It is also useful to remember that entropy is an average concept, indeed it can be viewed as the expected, $\langle\cdot\rangle$, logarithm of the a-priori probability of the outcomes,

$$\mathrm{S}(X) = \langle -\log_2 p_x \rangle \,.$$

This notion, of taking random probabilistic events and averaging them to arrive at a deterministic answer is a common theme that runs through this thesis, statistics and even quantum theory in general.
These intuitive views and probably the reason for the definition, come from the first coding theorem that Shannon proved in the 1949 paper. The theorem gives an operational meaning to the definition and makes explicit and literal what is meant by the notion that entropy measures the amount of information.
The theorem states that entropy is the average number of bits needed to record the outcome per draw of the random variable with vanishing error as more and more draws are recorded as a single block of outcomes. Thus entropy is a measure of the space of growing probable outcomes. By growing probable outcomes, we mean the number of probable outcomes as more and more draws are made. By measure of the space, we mean the 'dimension' of the space and hence the appearance of the logarithm in the definition. As the number of possibilities grows exponentially, the logarithm of the number of possibilities grows linearly, capturing the 'dimensional' size of the space.
The extremal case of equally probable outcomes makes the operational meaning clearer.

Let us count the number of outcomes of a sequence of fair coin tosses. Each outcome is equally likely, with probabilities,

$$p_{\mathrm{H}} = 1/2, \qquad p_{\mathrm{T}} = 1/2.$$

One divided by either of the probabilities is equal to the number of possible outcomes, $1/p_{\mathrm{H}} = 1/p_{\mathrm{T}} = 2 = N$. Indeed, in general, for any number of equally likely outcomes, $N$, each probability is $p_i = 1/N$ and $1/p_i = 1/(1/N) = N$. Now the logarithm of the number of possibilities, is a measure of the 'dimensional' size of the space and the probabilities can give us access to $N$

$$\log_2 1/p = -\log_2 p = \log_2 N.$$

To be more precise about what we mean by 'dimensional' size, we ask what is the number of binary variables forming a tuple so that all configurations of the tuple could uniquely label all the possibilities? The number of variables is the *dimension* of the Cartesian product of the spaces that the variables run over. For $N$ equally likely outcomes, it is easy to see that we need $\log_2 N$ number of variables, or so called *bits*, to describe the outcome.

Returning to the coin example, for one flip, $N = 2$ and $\log_2 2 = 1$. For a larger number of flips, $F$, there are $N = 2^F$ possible sequences of outcomes and there are $\log N = F$ bits (two state descriptions, either H or T) needed to single out which outcome occurred. Thus $-\log_2 p$ is the number of bits needed to describe the outcome. The full definition caters for general probability mass distributions. This generalization is achieved by actually turning the arbitrary probability mass distribution into a scenario of equally likely possibilities. To be more precise, the proof introduces equally likely sequences of draws as an asymptotically growing block. The heart of the proof is that the blocks of draws form a set of so called equally likely *typical* sequences, whose total probability of occurrence asymptotically approaches unity. A rough proof below appeals to the law of large numbers.

For a large number, $F$, of independent draws the number of times symbol $x$ appears is $p_x F$. Now, the probability of a particular typical sequence to be drawn is the product of the probability of each specific letter at each location in the sequence. Since letter $x$ appears $p_x F$ times, that letter's contribution to the probability is $p_x^{(p_x F)}$ and the probability, $p_t$, of the full typical sequences is

$$p_t = \prod_x p_x^{(p_x F)}.$$

By the law of large numbers used for a second time, all the typical sequences will have this probability. Therefore, the number of such equally probable typical sequences is

$$N = 1/p_t = \frac{1}{\prod_x p_x^{(p_x F)}}.$$

Now, as we saw above, the number of recording bits needed to specify each of these $N$ outcomes is,

$$\log_2 N = \log_2 \frac{1}{\prod_x p_x^{(p_x F)}} = -F \sum_x p_x \log p_x = F\,\mathrm{S}(X).$$

Finally, the number of bits *per draw* on average is then

$$(\log_2 N)/F = \mathrm{S}(X),$$

non-rigorously proving Shannon's first coding theorem.

### 2.1.2 Classical Capacity

Shannon went on to extend the entropy definition to related quantities. For example, he introduced a second random variable $Y$, which is potentially correlated to $X$. That is, $X$ and $Y$ have a joint probability distribution $p_{x,y}$ and it may be that $p_{x,y} \neq p_x p_y$. With this joint distribution Shannon introduced the joint entropy,

$$H(X,Y) \equiv -\sum_{x,y} p_{x,y} \log(p_{x,y}),$$

and the conditional entropy of $Y$ given $X$,

$$H(Y|X) \equiv -\sum_{x,y} p_x p(y|x) \log p(y|x).$$

The conditional probability is defined as $p(y|x) \equiv p_{x,y}/p_x = p_{x,y}/\sum_y p_{x,y}$.
These correlations do not, in general, imply a causal relationship between $X$ and $Y$. However, for the specific model of communication that Shannon introduced, $X$ models the sender, who sends letters $x$ to the receiver (the causality part) which are received with potential disturbances (the noise part) as letters $y$, according to the random variable $Y$.
If we specify $p_x$ and $p(y|x)$ then we have completely specified the joint probability $p_{x,y} = p_x p(y|x)$, and all the entropic quantities, $H(X)$, $H(Y)$, $H(X,Y)$ and $H(Y|X)$. The sender is free to choose $p_x$ but $p(y|x)$ is a fixed property of the channel, specifying the channel's actions and introduced noise, completely.
$H(Y|X)$ measures the average uncertainty left in random variable $Y$, conditioning on an outcome $x$ of $X$, occurring with probability $p_x$. Putting this together with the full entropy of $Y$, Shannon defined the (symmetric) mutual information between $X$ and $Y$

$$I(X:Y) \equiv H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y) = H(X) - H(X|Y).$$

This measures the common information between $X$ and $Y$. An intuitive sense is gained from the definition by seeing the first term as the total variability in $Y$ and the second term as the variability in $Y$ that does not come from $X$, but, for example, from unrelated noise sources. Hence the difference is the information common to both random variables. The second identity, has a simple Venn diagram interpretation, where area represents uncertainty and set overlap represents commonality.
The third identity has an intuitive reading that pre-empts the channel capacity theorem. $H(X)$ measures the dimension of the space of typical sequences (Shannon's first coding theorem). These typical, equally likely sequences, could potentially be used to encode information. Now, $H(X|Y)$ measures the average uncertainty in $X$ remaining, even after $Y$ is received and known. Such remaining uncertainty implies that the exact $X$ typical sequence that was sent, cannot be perfectly inferred from the corresponding sequence of $Y$'s. Thus the dimension of the space of uncertain remaining sequences subtracts from the dimension of the space of available typical sequences to give the dimension of

the space of *code-words* that can be reliably sent. To maximise the size of the space of code-words, the whole expression needs to be maximized over the choice of $X$, yielding the capacity.

This is made precise in Shannon's second coding theorem,

$$C(X \to Y) = \max_X I(X : Y).$$

The *definition* of the *capacity*, $C(X \to Y)$, is the maximum rate in bits of information per communication symbol that can be transmitted from a sender using an optimized multiple block encoding of $X$'s through a given channel, characterized by $p(y|x)$, to a receiver who decodes the sequence of received $Y$'s. The theorem proves that this capacity is equal to the maximization of the mutual information over $X$.

The maximization is present because the sender has the freedom to select which communication letters to send and how often. The goal is to make the $X$ typical space as large as possible while keeping the post-reception uncertain space as small as possible. The optimal $X$'s letters and probabilities are not what constitutes the message. $X$ sets up the *typical* sequences and an unknown, but special subset of these, called code-words, could be used to encode the message. It is up to the sender and receiver to map messages to code-words in a pre-agreed fashion so that the receiver can decode the message. The decoding step is distinct from the encoding. A sequence of $Y$'s that is received is not immediately recognizable as code-words. The sequence received needs to be mapped to the likely code-word sent and then the code-word can be inverted to give the message.

Shannon's second coding theorem, shows that it is possible to take a noisy channel and use it for asymptotically reliable lossless communication. The irony of his proof that gives assurance of the communication possibilities of the channel, is that it does not actually *construct* a set of code-words, it merely proves its existence.

Finally, an interesting property of the capacity is that it is *additive*. That is, if multiple identical channels are available for parallel use, the capacity of the combined channel is simply the sum of the capacities of the individual channels.

## 2.2 Quantum Information Theory

Information theory and computer science seem to 'start out' as abstract concepts in the mathematical world which then get applied to the real world. For example, the Church-Turing hypothesis [2] takes the abstract Turing machine and conjectures that no-real world computing device can outperform it.

Quantum mechanics as a revolutionary physical theory, brings many unexpected twists to our understanding of reality itself. Not only does it contest the strong version of the Church-Turing hypothesis, it also places physics at the base of information theory. From this view, information theory and computer science through their constructions, are actually modelling the real world at a physical level and as such need to take into account quantum physics to become complete.

Quantum information theory provides a further twist by questioning our notions of physical locality through Bell inequality violations [3] and perhaps even suggesting that the axioms of quantum mechanics somehow have information-theoretic motivations.

Thus information theory and quantum theory are inextricably and so far mysteriously linked. Therefore we can expect Shannon's coding theorems to have quantum analogues, and sometimes with interesting new features. As the quantum world comes with new resources (but also with counter-intuitive restrictions), the generalization is not unique. For example, there are product state or entanglement assisted encoding and decoding possibilities. There is also the quantum capacity of a quantum channel or the classical capacity of a quantum channel. For this thesis, we study the classical capacity of a quantum channel featuring product-state-input encoding and *joint*-output-measurement decoding.

### 2.2.1 Von Neumann Entropy

The quantum analogue of the Shannon entropy is the Von Neumann entropy of a density state, $\rho$,

$$\mathrm{S}_V(\rho) = -\mathrm{Tr}(\rho \log \rho).$$

Since $\rho$ is Hermitian and positive, it is diagonalizable with positive eigenvalues $\{\lambda_i\} = \Lambda$. The Von Neumann entropy is thus related to the Shannon entropy,

$$\mathrm{S}_V(\rho) = \mathrm{S}(\Lambda).$$

Using the definition of Von Neumann entropy, Josza and Schumacher [4] generalized Shannon's first coding theorem to a quantum information setting. They showed that the average number of *qubits* per draw needed to hold a quantum source of potentially overlapping pure states, $\{p_i, \Phi_i\}$ (an extension of the letters of the alphabet), is the Von Neumann entropy of the mixture of the pure states,

$$\#\text{qubits} = \mathrm{S}_V\left(\sum_i p_i \Phi_i\right).$$

### 2.2.2 Classical Capacity of a Quantum Channel

One generalization of Shannon's second coding theorem is to consider sending classical information through a quantum channel, $\Phi(\rho)$, as described by a *completely positive*, *trace preserving* memoryless map.

Since it is classical information we are sending and receiving, the generalization actually involves the original entropic concepts. That is, the concept of mutual information between the initial and final random variables is exactly the same as in Shannon's second coding theorem, because the variables are classical and the overall transfer of classical information is 'oblivious' to the fact that a quantum channel is being used in between.

Whilst the mutual information definition is the same, its calculation is very different. Indeed, it involves a series of mappings from classical to quantum through the channel and back to classical. Finally, to arrive at the capacity, the maximization over *more* choices has to be taken because there are extra levels of mappings.

Another major difference for the quantum case, is that if we allow the receiver to conduct a joint quantum measurement on multiply-received quantum states, that is, we allow the receiver to entangle the output before measuring, then the capacity can be shown to be greater than if this is not allowed. This means that the capacity is not *additive* in the

sense discussed above for the classical capacity. Since the capacity is an asymptotic average quantity of increasing number of uses anyway, we naturally choose to allow joint quantum measurements and over larger and larger block sizes. The full capacity per use should be defined taking into account this extra strategy. Having said this, we treat entanglement in input states as a separate case (entanglement assisted capacity) and prefer to maintain that the sender is only allowed product state inputs.

So the definition of the capacity per use involves a *regularization* procedure,

$$C_{X\to\Phi\to Y} = \lim_{n\to\infty} C^{(n)}/n, \tag{2.1}$$

where $C^{(n)}$ is the classical capacity through a quantum channel where joint measurements are allowed on blocks of size $n$.

Now that we have laid out the setting, we introduce a quantum information entropic quantity called the Holevo $\chi$ quantity [5] based on the output states of the channel for given input states, $\rho_i$, occurring with probability $p_i$,

$$\chi(\{p_i,\rho_i\}) = \mathrm{S}(\Phi(\sum_i p_i\rho_i)) - \sum_i p_i\,\mathrm{S}\left(\Phi\left(\rho_i\right)\right).$$

This is a quantum property because it involves quantum states and channels. However, it refers to our goal of transmitting classical information through the mapping of the classical state $i$ to an assigned quantum state, $i \mapsto \rho_i$.

This quantity captures the size of the carrier quantum state being sent through the channel (in the first term) and the average amount of noise the channel disturbs those carriers by (in the second term).

The difference is related to the amount of classical information that can be reliably sent through the quantum channel. Indeed, Holevo [5] and Schumacher-Westmoreland [6] (HSW) proved that the classical capacity is the maximization of the $\chi$ quantity over input pure states and their probabilities,

$$C_{X\to\Phi\to Y} = \max_{\{p_i,\rho_i\}} \chi(\{p_i,\rho_i\}).$$

This is a remarkable 'single-letter' quantum entropic characterization of the complicated limiting regularization process Eq. (2.1). This is not to say that the capacity can be achieved by code-words of single letters, but that combining the letters and the mapped quantum states with joint measurements and taking the asymptotic limit, leads to a capacity that can be described in terms of the properties of the single letters and their associated quantum states. Indeed, the capacity depends on taking larger and larger blocks of letters, and for example capping the joint measurement size, leads to a lower capacity.

The HSW theorem became a fundamental building block for many other entropic and capacity related quantities. One extension [7] that we use in this thesis, is a weakening of the restriction that the channel must be memoryless. One of the simplest extensions to memoryless channels are called *forgetful* channels. The HSW theorem applies directly to these forgetful channels. In this thesis, we calculate the $\chi$ quantity for a specific simple forgetful channel and study the effects that the strength of the correlations have on the capacity.

# Bibliography


[1] *The Mathematical Theory of Communication*, C. Shannon and W. Weaver (Urbana IL: University Illinois Press; 1949)

[2] *Quantum theory, the Church-Turing principle and the universal quantum computer*, D. Deutsch (Proceedings of the Royal Society of London Ser. A; A400; pg. 97–117; 1985)

[3] *On the Einstein Podolsky Rosen Paradox*, J. Bell, (Physics 1 (3); 195200; 1964)

[4] *A New Proof of the Quantum Noiseless Coding Theorem*, R. Josza and B. Schumacher (J. Mod. Opt.; 41 23439; 1994)

[5] *The Capacity of Quantum Communication Channel with General Signal States*, A.S. Holevo (IEEE Trans. Inform. Theory; 44 26972; arXiv:quant-ph/9611023; 1998)

[6] *Sending Classical Information via Noisy Quantum Channels*, B. Schumacher and M.D. Westmoreland (Phys. Rev. A; 56 1318; 1997)

[7] *Quantum Channels with Memory*, D. Kretschmann and R.F. Werner, Phys. Rev. A, 72(6):62323, 2005 (arXiv:quant-ph/0502106)

# Chapter 3

# Classical Capacity of a Qubit Depolarizing Channel with Memory

Published:

*The Classical Capacity of a Qubit Depolarizing Channel with Memory*,

J. Wouters, I. Akhalwaya, M. Fannes, F. Petruccione (Phys. Rev. A 79, 042303, 2009)

## Abstract

The classical product state capacity of a noisy quantum channel with memory is investigated. A forgetful noise-memory channel is constructed by Markov switching between two depolarizing channels which introduces non-Markovian noise correlations between successive channel uses. The computation of the capacity is reduced to an entropy computation for a function of a Markov process. A reformulation in terms of algebraic measures then enables its calculation. The effects of the hidden-Markovian memory on the capacity are explored. An increase in noise-correlations is found to increase the capacity.

## 3.1 Introduction

Quantum mechanics brings strange and wonderful features to the field of information theory. It introduces new information resources such as qubits with the power of superposition but also teasing restrictions such as the no-cloning theorem. We are interested in the possibility of the boosted transmission of classical information through a quantum channel with memory and no prior entanglement.

Great strides have been made in understanding the capacity of quantum channels. For example, the celebrated Holevo-Schumacher-Westmoreland (HSW) theorem [1, 2] gives an expression for the classical capacity of a noisy memoryless quantum channel with product state inputs. The memoryless channel restriction has since been extended to, so called, *forgetful* memory channels [4]. The inclusion of memory is the next step in the attempt of accurately modelling the complicated noise-correlated real world. Now that these initial seeds of the theoretical framework are in place, it is enlightening to use these tools, in specific cases, to analytically study the new effects that noise with memory has on the capacity.

We construct a forgetful channel and incorporate memory effects by Markov switching between two sub-channels. In order to investigate the classical product state capacity of this channel we must look at the entropy of the classical output. The output sequence of qubits and their associated errors are correlated. To manage this complicated conditional dependence, we use the hidden Markov nature of the process to reformulate the problem using the algebraic measure construction [5]. The algebraic measure approach allows us to derive an expression for the asymptotic entropy rate. We then explore the effects that our non-Markovian memory has on the classical product state capacity.

This paper is structured as follows. In Section 3.2, we take a closer look at the quantity we are investigating, namely the product state classical capacity. In Section 3.3, we construct the forgetful channel with Markovian noise correlations. In Section 3.4, algebraic measures are introduced, which are used in Section 3.5 to reformulate the problem. Finally, in Section 3.6, we show how this allows us to easily calculate the capacity of the

channel numerically.

## 3.2 Classical Capacity of Quantum Channels

The information process we are studying is classical communication through a noisy quantum channel. The layout of this section largely follows that in [1].

With the classical information we want to send encoded using an input alphabet $A = \{1, \ldots, a\}$, we choose for every element $i \in A$ an encoding quantum state $\rho_i$ on a Hilbert space $\mathfrak{H}$. This input state is then transmitted using a quantum channel $\Lambda : \mathcal{B}(\mathfrak{H}) \to \mathcal{B}(\mathfrak{K})$. For the channel to be a valid quantum channel it must be a completely positive trace preserving map.

Transmitting the element $i \in A$ results in a quantum state $R_i = \Lambda(\rho_i)$ being received on the output side. On this side, the received quantum state is measured using a resolution of identity in $\mathfrak{K}$. This resolution of identity is a set of positive operators $X = \{X_i\}$ on $\mathfrak{K}$ such that $\sum_i X_i = \mathbb{1}$.

The conditional probability of the receiver measuring $j$, when the input $i$ was sent, is given by $p(j|i) = \operatorname{Tr} R_i X_j$. If at the input side the element $i$ is sent with a probability $\pi_i$, the amount of information that will be received is quantified by the classical Shannon information,

$$I_{\Lambda,1}(\pi, \rho, X) = \sum_{i,j \in A} \pi_i p(i|j) \log \left( \frac{p(j|i)}{\sum_{k \in A} \pi_k p(j|k)} \right) . \tag{3.1}$$

If the sender is allowed to use the channel $n$ times, the channel use can be described by the product channel $\Lambda_n = \otimes^n \Lambda$ on $\otimes^n \mathfrak{H} = \mathfrak{H} \otimes \ldots \otimes \mathfrak{H}$. The input alphabet is now $A^n$ and the probability distribution of a word $u = (i_1, \ldots, i_n) \in A^n$ being sent is again denoted by $\pi_u$. The codeword corresponding to the input $u$ is given by

$$\rho_u = \rho_{i_1} \otimes \ldots \otimes \rho_{i_n}$$

and results in $R_u = R_{i_1} \otimes \ldots \otimes R_{i_n}$ being received. The conditional probability and the Shannon information $I_{\Lambda,n}$ for the $n$-product of the channel can now be introduced completely analogously to Eq. (3.1), with the summations over $A^n$ instead of $A$.

The maximum amount of information that can be sent with $n$ channel uses is now given by

$$C_n(\Lambda) = \sup_{\pi,\rho,X} I_{\Lambda,n}(\pi, \rho, X) \; .$$

Due to the fact that $C_n + C_m \leq C_{m+n}$, the limit

$$C_{\text{class}}(\Lambda) = \lim_{n\to\infty} \frac{C_n(\Lambda)}{n}$$

exists. Using Shannon's coding theorem, we see that $C$ is the least upper bound of the rate of information that can be transmitted with asymptotically vanishing error.

The HSW theorem [1, 2] gives an expression for this classical product state capacity of noisy memoryless quantum channels,

$$C_{\text{class}}(\Lambda) = \chi^* = \sup_{\pi,\rho} \chi(\Lambda),$$

where $\chi$ is the Holevo $\chi$ quantity

$$\begin{aligned} &\chi(\{(\pi_i, \Lambda(\rho_i))\}) \\ &= \mathrm{S}(\sum_i \pi_i \Lambda(\rho_i)) - \sum_i \pi_i \, \mathrm{S}(\Lambda(\rho_i)) \; . \end{aligned}$$

Due to the convexity of the von Neumann entropy, the supremum can in fact be taken over pure states $\rho_i$.

The memoryless channel restriction has recently been weakened to include, so called, *forgetful* memory channels. For such channels, the classical product state capacity has been shown [4] to correspond to

$$C^* = \lim_{n\to\infty} \frac{C_{\text{class}}(\Lambda_n)}{n} \; , \tag{3.2}$$

where $\Lambda_n$ is a channel representing the transmission of $n$ states, with the noise on subsequent transmissions is correlated. See [4] for details or Section 3.3 for an example.

## 3.3 The Depolarizing Memory Channel

Treating information or noise sources as independent random variables is a successful but crude first approximation. To improve the modelling process and to achieve better performance in real world applications, the independence assumption needs to be removed.

The first step in this direction is to introduce forgetful noise memory. A forgetful noise process is one which after sufficiently long time, 'forgets' or is independent of previous noise. Thus, here the independence is pushed further away, allowing a space to study the effects of short-term memory. With the theoretical tools in place, it is instructive to study even very simple models to see the effects of memory on the classical capacity.

### 3.3.1 Construction of the Channel

The forgetful channel is constructed by combining two memoryless single qubit depolarizing channels ($\mathcal{E}_0$ and $\mathcal{E}_1$), switching between them using a two-state Markov chain ($Q = (q_{ij}), \quad i,j \in \{0,1\}$). Thus, $Q$ is the $2 \times 2$ Markovian channel selection matrix with $q_{ij}$ being the probability of switching from channel $i$ to channel $j$. Hence, $q_{ij} \geq 0$ and $q_{i0} + q_{i1} = 1$ for $i,j \in \{0,1\}$. It is forgetful, in the case when the Markov chain is *aperiodic* and *irreducible*.

The depolarizing channels can be written as: $\mathcal{E}_i(\rho) = x_i^0 \rho + x_i^1(\mathbf{1} - \rho)$. These single qubit channels can be thought of as probabilistically mixing the identity channel (with probability $x_i^0$) and 'flip' channel (with probability $x_i^1 = 1 - x_i^0$) acting on a single qubit density operator $\rho$. However this rewriting is only completely positive for $1/3 \leq x_i^0 \leq 1$. The built-up channel $\Lambda_n$, corresponding to $n$ successive uses is

$$\Lambda_n = \rho_1 \otimes \ldots \otimes \rho_n \mapsto$$
$$\sum_{i_1,\ldots,i_n} \gamma_{i_1} q_{i_1 i_2} \ldots q_{i_{n-1} i_n} \mathcal{E}_{i_1}(\rho_1) \otimes \ldots \otimes \mathcal{E}_{i_n}(\rho_n) \, .$$

The sum is over all possible paths $(i_1, \ldots, i_n) \in \{0,1\}^n$ and each term is a tensor product of the selected sub-channels weighted by the probability of occurrence ($\gamma_i$ is the initial probability of selection set to the stationary distribution of the Markov process: $Q^T \gamma = \gamma$).

### 3.3.2 Classical Capacity

We calculate the capacity with this $n$-use form of the channel and regularize by taking the limit $n \to \infty$ as in Eq. (3.2). Since we are looking at the product state capacity, we

choose

$$\begin{aligned}\rho_i = \Phi^{(n)}(\bar{l}) &= \Phi^{(n)}(l_1, \ldots, l_n) \\ &:= |l_1\rangle\langle l_1| \otimes \ldots \otimes |l_n\rangle\langle l_n| \ ,\end{aligned}$$

where the $l_i$ are arbitrary pure qubit states.

Applying the channel $\Lambda_n$, we get

$$\begin{aligned}\Lambda_n(\Phi^{(n)}(\bar{l})) = \sum_{i_1,\ldots,i_n} \gamma_{i_1} q_{i_1 i_2} \ldots q_{i_{n-1} i_n} \\ (x^0_{i_1}|l_1 \oplus 0\rangle\langle l_1 \oplus 0| + x^1_{i_1}|l_1 \oplus 1\rangle\langle l_1 \oplus 1|) \otimes \ldots \\ \otimes (x^0_{i_n}|l_n \oplus 0\rangle\langle l_n \oplus 0| + x^1_{i_n}|l_n \oplus 1\rangle\langle l_n \oplus 1|) \ ,\end{aligned}$$

where $(l_i \oplus 1)$ denotes the qubit state with a flipped Bloch vector with respect to $l_i = (l_i \oplus 0)$

$$|l_i \oplus 1\rangle\langle l_i \oplus 1| = \mathbb{1} - |l_i \oplus 0\rangle\langle l_i \oplus 0|$$

By expanding the product above we see that the eigenvalues of the output state are given by

$$\lambda_n(\bar{k}) = \sum_{i_1,\ldots,i_n} \gamma_{i_1} q_{i_1 i_2} \ldots q_{i_{n-1} i_n} x^{k_1}_{i_1} \ldots x^{k_n}_{i_n} \ . \tag{3.3}$$

Note that these eigenvalues are independent of the choice of the input state.

The channel output can now be written as

$$\Lambda_n\left(\Phi^{(n)}(\bar{l})\right) = \sum_{\bar{k}} \lambda_n(\bar{k}) \Phi^{(n)}(\bar{l} + \bar{k}) \ .$$

Hence, if we calculate the first term in the Holevo $\chi$ quantity for $\pi$, the uniform distribution ($\pi_i = 1/2^n$), and $\Phi_i$ going over all the $\rho^{(n)}(\bar{l})$, we see that

$$\begin{aligned}\Phi_{\text{out}} &:= \sum_{\bar{l}} \frac{1}{2^n} \Lambda_n\left(\Phi^{(n)}(\bar{l})\right) \\ &= \frac{1}{2^n} \sum_{\bar{k}} \lambda_n(\bar{k}) \sum_{\bar{l}} \Phi^{(n)}(\bar{l} + \bar{k}) \ .\end{aligned}$$

Since $\bar{l}$ goes over all possible combinations, so does $\bar{l}+\bar{k}$, so we can relabel them

$$\Phi_{\text{out}} = \frac{1}{2^n}\sum_{\bar{k}} \lambda_n(\bar{k}) \sum_{\bar{l}'} \Phi^{(n)}(\bar{l}') \ .$$

Since the eigenvalues in Eq. (3.3) sum to one, we see that $\Phi_{\text{out}}$ is the maximally mixed state

$$\Phi_{\text{out}} = \frac{1}{2^n}\sum_{\bar{l}'} \Phi^{(n)}(\bar{l}') \ .$$

Thus, $S(\Phi_{\text{out}})$ is maximal and is equal to $\log_2(2^n) = n$.

The second term in the Holevo $\chi$ quantity is

$$-\sum_i \pi_i S\left(\Lambda_n(\rho_i)\right) \ .$$

Since the eigenvalues $\lambda_n(\bar{k})$ of $\Lambda_n(\rho_i)$ do not depend on the choice of $\rho_i$, this term does not influence the maximization. Hence our choice of $\pi$ and $\rho$ maximizes the Holevo $\chi$ quantity.

Thus, the final expression for the regularized capacity Eq. (3.2) is

$$C^* = \lim_{n\to\infty} \frac{1}{n} C_{\text{class}}(\Lambda_n)) = 1 - \lim_{n\to\infty} \frac{1}{n} \mathrm{S}(\Lambda_n(\rho)) \ . \tag{3.4}$$

If we were to calculate the output entropy using the eigenvalues in Eq. (3.3), the calculation would be exponentially long in $n$. Therefore, other techniques are needed. The way we approach the problem is by reformulating it as a hidden Markov process. The eigenvalues of the output state correspond to the probabilities of such a process.

A hidden Markov process can be defined as follows. If we have a translation-invariant measure $\nu$ with the Markov property on $L^{\mathbb{Z}}$, where $L$ is a finite set, then a hidden Markov measure can be constructed on $K^{\mathbb{Z}}$ through a function $\Phi : L \to K$, with the following local densities

$$\mu((\omega_m, \ldots, \omega_n)) = \sum_{\substack{\epsilon_m,\ldots,\epsilon_n \\ \Phi(\epsilon_m)=\omega_m \ldots \Phi(\epsilon_n)=\omega_n}} \nu((\epsilon_m, \ldots, \epsilon_n)) \ , \tag{3.5}$$

where $\omega_m, \ldots, \omega_n \in K$ and $\epsilon_m, \ldots, \epsilon_n \in L$ . For obvious reasons, these processes are also called functions of Markov processes.

## 3.4 Algebraic Measures

An algebraic measure, $\mu$, is a translational-invariant measure on a set $\{0,\ldots,q-1\}^{\mathbb{Z}}$, with probabilities determined by matrices $E_a$ with positive entries, one for each of the $q$ states. The probability of a sequence is obtained by applying a positive linear functional $\sigma$ to a matrix product of the corresponding matrices of the states of the sequence: $\mu(i_1,\ldots,i_n) = \sigma(E_{i_1}\ldots E_{i_n})$. This matrix algebraic construction is the reason for the name *Algebraic Measure*, studied in detail in Ref. [5]. As we shall see, the hidden Markov processes correspond to a set of algebraic measures with a specific positivity structure and remarkably, the converse holds too.

### 3.4.1 Manifestly Positive Measures

In [5] it was shown that hidden Markov processes correspond to manifestly positive algebraic measures. The local densities of such a manifestly positive algebraic measure on an infinite chain $K^{\mathbb{Z}}$ of classical state spaces $K = \{0,\ldots,q-1\}$ are of the form

$$\mu((\omega_1,\ldots,\omega_n)) = \langle\tau|E_{\omega_1}\ldots E_{\omega_n}\sigma\rangle \ ,$$

where $\omega_i \in K$, $\tau$ and $\sigma$ are vectors in $\mathbb{R}^d$ with non-negative elements (denoted $(\mathbb{R}^d)^+$) and the $E_i$ are $d\times d$ real matrices with non-negative elements (denoted $M_d^+$).
As an example of these manifestly algebraic measures, let us look at a regular Markov chain $\mu((\omega_m,\ldots,\omega_n))$ on $\{0,\ldots,q-1\}^{\mathbb{Z}}$. If we choose $\tau$, $\sigma$ and the $E_i$ as

$$\begin{aligned}
&\sigma \in (\mathbb{R}^d)^+ : \quad \sigma_a = 1 \text{ for } a \in K \ , \\
&\tau \in (\mathbb{R}^d)^+ : \quad \tau_a = \mu((a)) \text{ for } a \in K \ , \\
&E_a \in M_d^+ : \quad (E_a)_{b,c} = \delta_{a,b}\frac{\mu((b,c))}{\mu((b))} \text{ for } a,b,c \in K \ ,
\end{aligned}$$

one can check that $\langle\tau|E_{\omega_m}\ldots E_{\omega_n}\sigma\rangle$ indeed gives the correct densities.
From this example it is easy to see that if we have a hidden Markov process on $L^{\mathbb{Z}}$ defined by a map $\Phi : K \to L$ and a Markov measure $\mu$ on $K$ with corresponding matrices $E_a$, the manifestly positive algebraic measure corresponding to the hidden Markov measure

is given by the same vectors $\sigma$ and $\tau$ as before and the following matrices:

$$F_a \in M_d^+ : F_a = \sum_{\epsilon, \Phi(\epsilon)=a} E_\epsilon \text{ for } a \in K \ . \tag{3.6}$$

For a proof of the converse, which is namely, that every manifestly positive algebraic measure corresponds to a hidden Markov measure, we refer to [5].

### 3.4.2 Mean Entropy

We will now briefly summarize how the algebraic measure approach allows for a simpler approach to finding the entropy density [5, 6].
The entropy of a state $\mu$ on $K^{\mathbb{Z}}$ restricted to a region $\Lambda$ is defined by

$$S_\Lambda(\mu) = -\sum_{\omega_\Lambda \in K^\Lambda} \mu(\omega_\Lambda) \log \mu(\omega_\Lambda) \ .$$

$S_\Lambda$ can be shown to be bounded by $\#\Lambda \log q$, monotonically increasing in $\Lambda$ and strongly subadditive, that is

$$S_{\Lambda_1 \cap \Lambda_2}(\mu) + S_{\Lambda_1 \cup \Lambda_2}(\mu) \leq S_{\Lambda_1}(\mu) + S_{\Lambda_2}(\mu) \ .$$

Using the strong subadditivity of the entropy and the translational invariance of the measure, one can show that [3, 9]

$$\mathrm{S}(\mu) = \lim_{n\to\infty} \frac{\mathrm{S}(\mu_n)}{n} = \lim_{n\to\infty} (\mathrm{S}(\mu_n) - \mathrm{S}(\mu_{n-1})) \ .$$

We can then use this relation together with the expression for the local densities of the manifestly positive measures to reformulate the convergence of the mean entropy into a dynamical system of converging measures on the set of $d$-dimensional probability measures $\mathcal{B}_\sigma$ as

$$\mathrm{S}(\mu) = \lim_{n\to\infty} \sum_{a\in K} \int_{\mathcal{B}_\sigma} \phi_n(d\nu) h_a(\nu) \ ,$$

where

$$\begin{aligned}
\mu((\epsilon_0,\ldots,\epsilon_n)) &= \langle\tau|E_{\epsilon_0}\ldots E_{\epsilon_n}\sigma\rangle \\
&\text{with } \sigma,\tau\in(\mathbb{R}^d)^+ \\
\mathcal{B}_\sigma &= \{\nu\in(\mathbb{R}^d)^+|\,\langle\nu|\sigma\rangle=1\} \\
h_a(\nu) &= -\langle\nu|E_a\sigma\rangle\log\langle\nu|E_a\sigma\rangle \\
\phi_n(d\nu) &= \sum_{\epsilon_0,\ldots,\epsilon_n\in K}\mu((\epsilon_0,\ldots,\epsilon_n))\delta_{\frac{E^*_{\epsilon_n}\ldots E^*_{\epsilon_0}}{\mu((\epsilon_0,\ldots,\epsilon_n))}}(d\nu)\ .
\end{aligned}$$

If we define the linear transformation $T_\mu$ on functions on $\mathcal{B}_\sigma$ : $(T_\mu f)(\nu)=\sum_{a\in K}\langle\nu|E_a\sigma\rangle f\left(\frac{E_a^*\nu}{\langle\nu|E_a\sigma\rangle}\right)$, one can show that $\phi_n(f)=\phi_0(T_\mu^n f)$. $T_\mu$ is a contraction map, so a fixed point argument can be used to show that $\phi_n$ converges to a unique measure $\phi$ that is invariant under $T_\mu$

$$\phi(T_\mu f)=\phi(f)\ .$$

This measure allows us then to calculate the mean entropy

$$\mathrm{S}(\mu)=\sum_{a\in K}\int_{\mathcal{B}}\phi(d\nu)h_a(\nu)\ . \tag{3.7}$$

Our goal in the remaining part of the article is to translate the switching depolarizing channel into the setting of algebraic measures and to try and find the invariant measure that allows us to calculate the mean entropy.

## 3.5 Algebraic Measure of the Channel

The relationship between the hidden Markov measure, say $\mu'$ on $K^{\mathbb{Z}}$, and the underlying Markov measure $\nu$ with the Markov property on $L^{\mathbb{Z}}$ is through a 'tracing' function $\Phi : L\to K$, as is shown in Eq. (3.5).

The underlying Markov process for the overall quantum channel has a four state configuration space corresponding to channel selection and error occurrence:

$K=\{(0,0),(0,1),(1,0),(1,1)\}$. The first index indicates which depolarizing channel

has been chosen and the second indicates whether a bit flip occurred. The elements of the transition matrix, $E$, for this process are then given by

$$(E)_{\{(i,j)(i',j')\}} = q_{ii'} x_{i'}^{j'} \ , \tag{3.8}$$

the probability of going from $(i,j)$ to $(i',j')$ is given by the switching probability $q_{ii'}$ from channel $i$ to $i'$, multiplied by the probability $x_{i'}^{j'}$ that channel $i'$ produces the error-occurrence $j'$.

The function that produces the correct hidden Markov process is then given by

$$\Phi((i,j)) = j \ .$$

This function reflects the fact that we are unaware of the choice of channel that has been made. The only effect that is visible from the outside is whether or not an input qubit has been flipped. Thus, $\Phi$ has to 'trace out' the choice of channel. $\Phi$ maps into the two-state error configuration space containing 'no flip' and 'flip': $L = \{0,1\}$ .

Using the fact that the matrices $E_{(i,j)}$ defining the algebraic measure of a Markov process ((Sec. 3.4.1, Pg. 26), $a = (i,j) \in K$) have only one non-zero row and Eq. (3.6), we get the matrices $F_0$ and $F_1$ that define the algebraic measure corresponding to $\mu'$. The matrix corresponding to 0, the first element of $L$ is given by

$$\begin{aligned} F_0 &= \sum_{(i,k),\Phi'((i,k))=0} E_{(i,k)} = \sum_i E_{(i,0)} \\ &= \begin{pmatrix} q_{00}x_0^0 & q_{00}x_0^1 & q_{01}x_1^0 & q_{01}x_1^1 \\ 0 & 0 & 0 & 0 \\ q_{10}x_0^0 & q_{10}x_0^1 & q_{11}x_1^0 & q_{11}x_1^1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \end{aligned}$$

and similarly for 1, the second element of $L$.

The hidden Markov process then gives us almost the same probabilities as the eigenvalues in Eq. (3.3)

$$\begin{aligned} p((k_1,\ldots,k_n)) &= \langle \tau | F_{k_1} \ldots F_{k_n} 1 \rangle \\ &= \sum_{i_1,\ldots,i_n} \tau_{i_1,k_1} q_{i_1 i_2} \ldots q_{i_{n-1} i_n} x_{i_2}^{k_2} \ldots x_{i_n}^{k_n} \ . \end{aligned}$$

Note that according to our discussion in Section 3.4, the vector $\tau$ is the stationary distribution of the full matrix $E$. Using Eq. (3.8), one can see that the invariant distribution $\tau$ is in fact $\tau_{(i,k)} = \gamma_i x_i^k$, so the probabilities of the hidden Markov process coincide with the eigenvalues in Eq. (3.3).

Having constructed the correct algebraic measure, we can determine $T_\mu$ explicitly and use it to greatly simplify the corresponding invariant measure $\phi$.

The expression for $T_\mu$, as can be found in [5], is

$$(T_\mu f)(\hat{\nu}) = \sum_{a \in L} \langle \hat{\nu} | F_a 1 \rangle f\left( \frac{F_a^* \hat{\nu}}{\langle \hat{\nu} | F_a 1 \rangle} \right) ,$$

where $\hat{\nu}$ is any 4-dimensional vector such that $\langle \hat{\nu} | 1 \rangle = 1$ and $f$ is a continuous real-valued function on the set of such vectors. For the case of our hidden Markov measure, the form of this transformation can be greatly simplified. Due to the stochasticity of the matrix $E$, we have the following:

$$F_0 |1\rangle = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \text{ and } F_1 |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} .$$

If we furthermore denote the four row vectors of $E$ by $\hat{\mu}_0$, $\hat{\mu}_1$, $\hat{\mu}_2$ and $\hat{\mu}_3$, we can write

$$F_0^* \hat{\nu} = \nu_0 \hat{\mu}_0 + \nu_2 \hat{\mu}_2 \text{ and } F_1^* \hat{\nu} = \nu_1 \hat{\mu}_1 + \nu_3 \hat{\mu}_3 .$$

On top of this, $\mu_0 = \mu_1$ and $\mu_2 = \mu_3$, so the total form of the transformation becomes

$$(T_\mu f)(\hat{\nu}) = (\nu_0 + \nu_2) f\left( \frac{\nu_0 \hat{\mu}_0 + \nu_2 \hat{\mu}_2}{\nu_0 + \nu_2} \right) + (\nu_1 + \nu_3) f\left( \frac{\nu_1 \hat{\mu}_0 + \nu_3 \hat{\mu}_2}{\nu_1 + \nu_3} \right) .$$

From this form of the transformation, we can already greatly restrict the support of $\phi$. Our claim is that the support of $\phi$ is restricted to the set of convex combinations of $\hat{\mu}_1$ and $\hat{\mu}_3$

$$\text{supp}(\phi) \subset \{ a\hat{\mu}_1 + (1-a)\hat{\mu}_3 \,|\, a \in [0,1] \} .$$

To show this, let's suppose that $\hat{\nu} \in \text{supp}(\phi)$ and $\hat{\nu} \notin S := \{ a\hat{\mu}_1 + (1-a)\hat{\mu}_3 \,|\, a \in [0,1] \}$.

Take $\zeta_{\hat{\nu}}$ a function on $\mathcal{B}_\sigma$ such that $\zeta_{\hat{\nu}}(\hat{s}) = 0$ for all $\hat{s} \in S$ and $\zeta_{\hat{\nu}}(\hat{\nu}) \neq 0$, then

$$\begin{aligned} 0 \neq &\phi(\zeta_{\hat{\nu}}) = \phi(T_\mu \zeta_{\hat{\nu}}) = \int \phi(d\nu) T_\mu(\zeta_{\hat{\nu}}(\nu)) \\ &= \int \phi(d\nu) \Big[ (\nu_1 + \nu_3) \zeta_{\hat{\nu}} \Big( \frac{\nu_1 \hat{\mu}_1 + \nu_3 \hat{\mu}_3}{\nu_1 + \nu_3} \Big) \\ &\quad + (\nu_2 + \nu_4) \zeta_{\hat{\nu}} \Big( \frac{\nu_2 \hat{\mu}_1 + \nu_4 \hat{\mu}_3}{\nu_2 + \nu_4} \Big) \Big] \ . \end{aligned}$$

However, this integral is equal to zero, since the arguments to $\zeta_{\hat{\nu}}$ run over the set $S$. Therefore, we have for $f \in \mathcal{C}(\mathcal{B})$,

$$\phi(f) = \int_0^1 d\lambda(a) f(a\hat{\mu}_1 + (1-a)\hat{\mu}_3) \ , \tag{3.9}$$

with $\lambda$ a measure on $[0, 1]$.

Now let us look at $\phi$ acting on the transformed $f$:

$$\begin{aligned} \phi(T_\mu f) = &\int \phi(d\nu) \Big[ (\nu_1 + \nu_3) f \Big( \frac{\nu_1 \hat{\mu}_1 + \nu_3 \hat{\mu}_3}{\nu_1 + \nu_3} \Big) \\ &+ (\nu_2 + \nu_4) f \Big( \frac{\nu_2 \hat{\mu}_1 + \nu_4 \hat{\mu}_3}{\nu_2 + \nu_4} \Big) \Big] \\ &= \int_0^1 d\lambda(a) \Big[ (\hat{\mu}_{a,1} + \hat{\mu}_{a,3}) f \Big( \frac{\hat{\mu}_{a,1} \hat{\mu}_1 + \hat{\mu}_{a,3} \hat{\mu}_3}{\hat{\mu}_{a,1} + \hat{\mu}_{a,3}} \Big) \\ &+ (\hat{\mu}_{a,2} + \hat{\mu}_{a,4}) f \Big( \frac{\hat{\mu}_{a,2} \hat{\mu}_1 + \hat{\mu}_{a,4} \hat{\mu}_3}{\hat{\mu}_{a,2} + \hat{\mu}_{a,4}} \Big) \Big] \ , \end{aligned} \tag{3.10}$$

where

$$\hat{\mu}_a = a\hat{\mu}_1 + (1-a)\hat{\mu}_3 \ .$$

By invariance (Sec. 3.4.2, Pg. 27), we can equate Eq. (3.9) and the above Eq. (3.10) to discover an invariance concerning $\lambda$. We thus arrive at the following symmetry of $\lambda$:

$$\lambda = T[\lambda] = a \mapsto c_1(a)\lambda[f_1(a)] + c_2(a)\lambda[f_2(a)] \ .$$

The two functions $f_1$ and $f_2$ are relatively simple shrink functions about two separate points in the domain $[0, 1]$, that shrink the $[0, 1]$ domain into two (possibly overlapping) sub-intervals of $[0, 1]$.

We can turn this analytic symmetry into a cyclic definition or iterative procedure to generate $\lambda$ up to some approximation $\lambda_n$.

$$\lambda_{n+1} = T(\lambda_n) \ .$$

We still have not defined $\lambda_0$, but taking a look at the iterative procedure, we see that there exist fixed points of the two shrink functions, call them $a_1$ and $a_2$,

$$a_1 = f_1(a_1) \quad a_2 = f_2(a_2) \qquad a_1, a_2 \in [0,1] \ .$$

With this observation the idea is to begin the iteration procedure with two Dirac delta's at these fixed points,

$$\lambda_0(a) = \frac{1}{2}\delta(a - a_1) + \frac{1}{2}\delta(a - a_2) \ .$$

Note that $\int \lambda_0(a)da = 1$, as a measure should be. Since there is unique convergence then the initial weightings should not matter [5].

To see that this is a good starting point and to get further insight into the support of $\lambda$, it can be seen that the support will grow, but most importantly, once a point is within the support of $\lambda_m$ it remains there for all $n \geq m$. So if the procedure is taken to infinity the support is fixed and countably infinite. Thus, we arrive at the following expression for the full support,

$$\begin{aligned} \text{supp}(\lambda) = \{a \in [0,1] : \exists n \in \mathbf{N}, \exists k_i \in \{0,1\} \forall i \in [1,n] \\ f_{k_n} \circ f_{k_{n-1}} \circ \ldots \circ f_{k_1}(a_1 \text{ or } a_2) = a\} \ . \end{aligned}$$

We use this iterative procedure to generate $\lambda_n$ and then use it in Eq. (3.9) to approximate the measure. The entropy in Eq. (3.7) can then be calculated and finally we use the entropy to calculate the capacity through Eq. (3.4). It is the capacity and its dependence on memory that we are interested in.

## 3.6 Results

In constructing our channel we defined certain parameters. It is useful to introduce a new set of suggestive parameters in terms of the old and also to reduce their number by

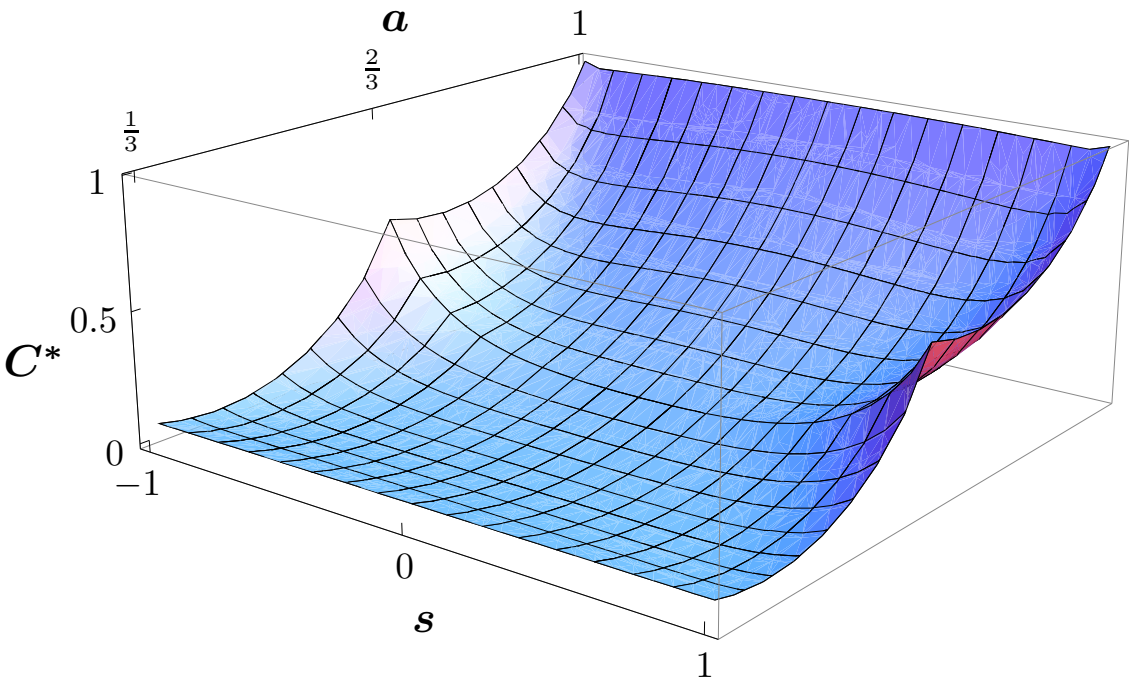



Figure 3.1: Capacity for maximally different sub-channels increases with memory.

making some assumptions. Firstly, we assume that the sub-channels switch symmetrically, that is, the probabilities of reuse are the same for both sub-channels. This makes the Markov matrix doubly stochastic and allows us to use its non-one eigenvalue as a useful characterizing parameter $s$. Thus, we set $q_{00} \to (1+s)/2$ and $q_{10} \to (1-s)/2$. The domain of $s$ is $(-1, 1)$, with $s = 0$ corresponding to no noise correlations. Secondly, we parametrize the error probabilities by their average and difference: $x_0^0 \to a + d$, $x_1^0 \to a - d$.

The main result is that the capacity increases with stronger noise-correlations. This manifests itself in two ways. Firstly, if we make the switching more correlated ($s$ away from 0) the capacity increases and secondly, if we increase the difference between the two sub-channels the capacity also increases. Similar results have been found for the quantum capacity of the dephasing channel with Markovian memory [13].

In Figure 3.1, $d$ is set to the maximum possible value while keeping an average of $a$ ($d = \min[a - 1/3, 1 - a]$). Remember that both $a - d$ and $a + d$ have to lie in the $[1/3, 1]$

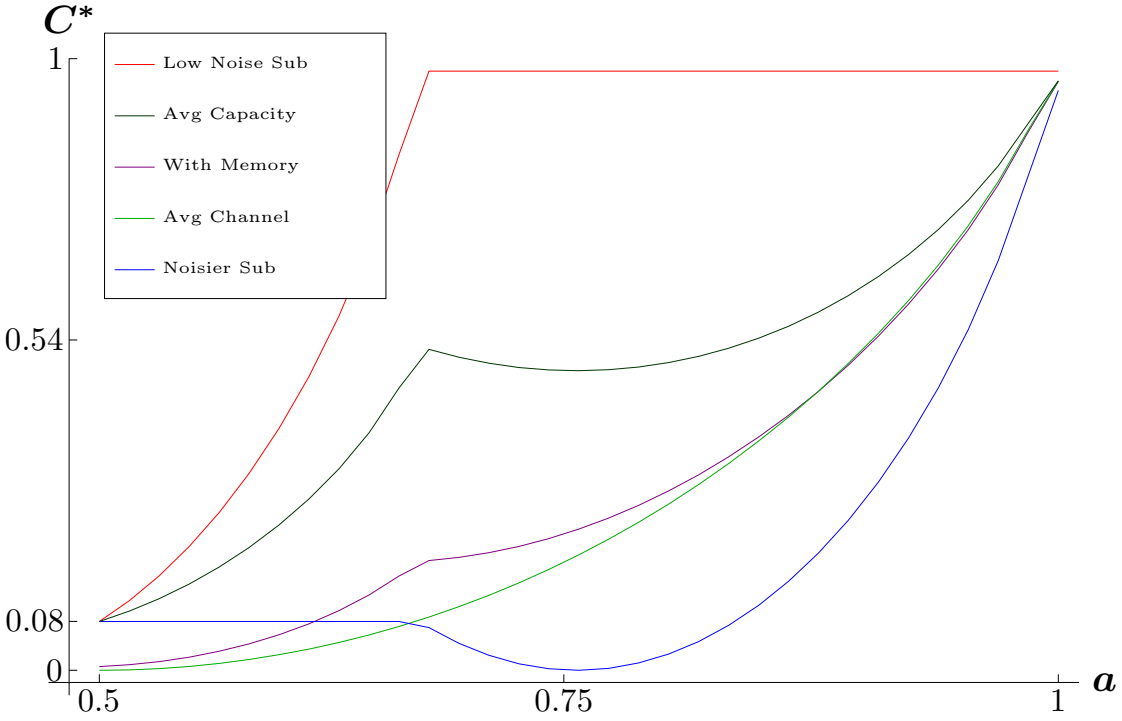



Figure 3.2: Capacity versus the average no-error probability $a$.

interval for the two sub-channels to be completely positive. The capacity is plotted against varying $a$ and $s$. We can see that the capacity increases as the noise-correlation ($s$) gets stronger. When $a = 2/3$, $d$ attains its maximum ($1/3$) and the effect of increasing $s$ on the capacity is greatest. Another interesting observation is the case when the two sub-channels average to the maximally mixing channel ($a = 1/2$, which ignoring memory, has zero capacity), taking into account memory effects there is a non-zero capacity.

To better illustrate the last point and to further explore the relationship between the capacity of the memory channel and its sub-channels, we plot in Figure 3.2, slices of Figure 3.1 of fixed $s$ together with plots of the underlying sub-channel capacities.

Thus, in the 'Avg Capacity' curve of Figure 3.2, we see the edge of Figure 3.1 (for fixed $s = 1$, equivalently $s = -1$, not actually attained), which corresponds to the average of the capacities of the sub-channels. The sub-channels' separate capacities are plotted in curves labelled 'Low Noise Sub' and 'Noisier Sub'. They are chosen to have maximum

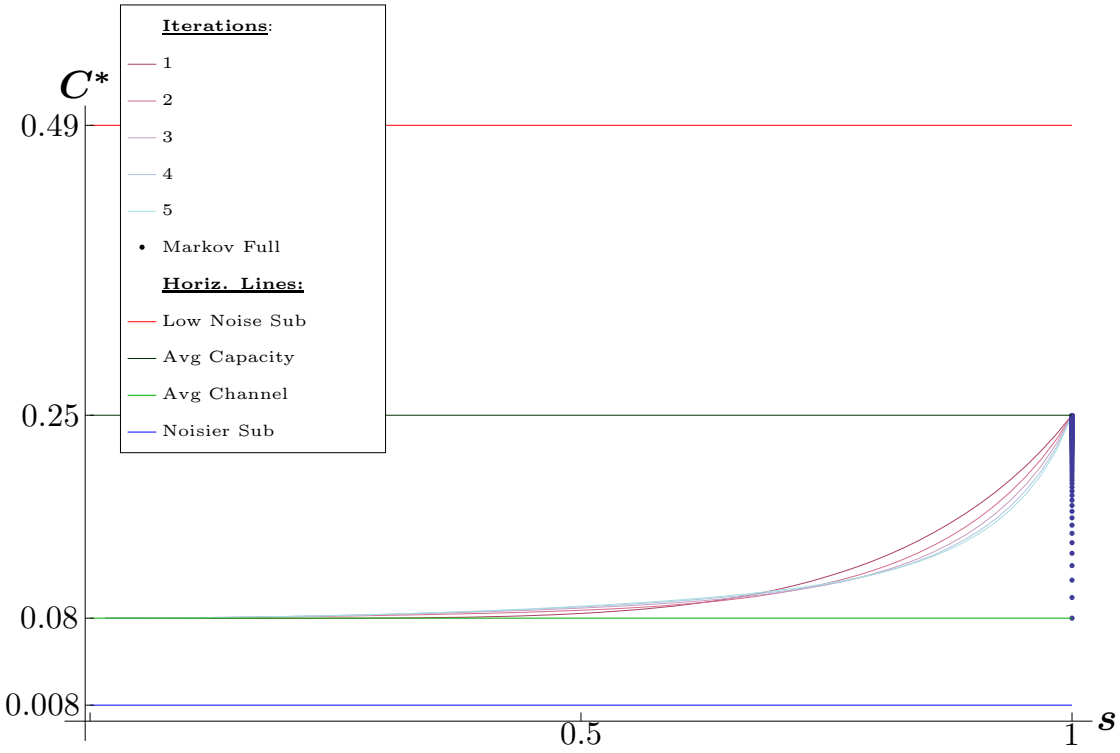



Figure 3.3: Capacity versus the memory parameter $s$ using many iterations and including full Markov calculation.

allowed separation for each point as $a$ varies (and thus the artificial discontinuities). In a real world example, this separation parameter is fixed by the channel and the sub-channels and their capacities would not be accessible. The capacity of the average channel, labelled 'Avg Channel', corresponds to a slice of fixed $s = 0$ (the center of Figure 3.1), since a no-memory/non-biased Markov walk factors into a tensor product of the average of the sub-channels, which is thus equivalent to just one depolarizing channel with the average error probability. The curve, 'With Memory', is a smooth intermediary between the 'Avg Channel' and 'Avg Capacity' and is an example slice of Figure 3.1 for $0 < s = \frac{2}{3} < 1$, which illustrates how taking memory into account improves the capacity. Of course, again, in a real world example this parameter is specified by the channel. The smooth transformation is not straightforward nor linear, which can be seen in the way Figure 3.1 curves for varying $s$.

To see the last point more clearly and also to indicate the convergence of the iteration procedure we next plot a slice of Figure 3.1 for fixed $a$. In Figure 3.3 we plot the regularized capacity against $s$ with the following fixed parameters: $a = \frac{2}{3}$, $d = \frac{1}{3}$.
We can see that the capacity increases as the noise-correlation gets stronger. The (blue) dots are calculated using a simplified ($s = 1$) full Markov walk calculation (1000 steps) which doesn't suffer from the usual exponential blow-up. The horizontal solid (green) line is the output entropy for $s = 0$, which is corresponds to no correlations and is equivalent to having only one depolarizing channel.

### 3.6.1 Non-Forgetful Limit

To complete the discussion concerning correlations we need to look at the two extreme cases: $s = 1$, corresponding to the case where a sub-channel is selected and used for every channel use afterwards, and $s = -1$, corresponding to the case where the choice of sub-channel is flipped with every channel use. Therefore, in constructing the overall channel and taking into account the initial random channel selection, we just have the mixing of two $n$-use channels. Specifically, in the $s = 1$ case, we have the mixing of the two $n$-fold tensor products of the two sub-channels separately, and in the $s = -1$ case, we have the mixing of two $n$-use channels where each deterministically alternates between the sub-channels but starting with a different sub-channel.
Both these extreme cases are non-forgetful since the initial sub-channel selection (the initial noise) is 'remembered' and the forgetful Holevo capacity theorem no longer applies (the Markov selection matrix is periodic in the $s = -1$ case and reducible in the $s = 1$ case). While our forgetful channel approach breaks down there are alternate theoretical frameworks that do actually capture these extreme cases. For $s = -1$ the capacity can be calculated using [10] and agrees with the limit of the forgetful approach, the capacity is the average capacity of the two sub-channels separately. However, for $s = 1$ case there is a discontinuity and the capacity suddenly drops to the minimum capacity of the sub-channels [11].
The intuition is that in the $s = -1$ case, the deterministic flip can be used to determine

'on-the-fly' which sub-channel is being used and then it is the same as using the two channels separately each half the time, so the capacity must be the average capacity. For the $s = 1$ case once you have the poorer channel you are stuck with it forever and so because of the mixture you can only guarantee the lower capacity.

## 3.7 Conclusion

We have constructed a simple forgetful noise-memory quantum channel. The noise-correlation is a function of the underlying hidden Markov process. This setup allowed us to construct a corresponding algebraic measure. We used the measure in an algebraic asymptotic entropy expression. Without this, the entropy would be very difficult to compute, involving exponentially many paths in configuration space.

We studied the effects that the noise correlations had on the classical capacity and discovered that the capacity increases with stronger correlations. This is sensible because the correlations can be used to combat the noise when coding information. We have arrived at the understanding that stronger correlations increases the capacity from that of the average channel to the average capacity of the sub-channels with very interesting limiting behaviour.

Further work includes using other approximation techniques, arriving at a full analytic expression of the capacity and looking at other similarly constructed channels. We are also confident and hopeful that the hidden Markov technique could be successfully employed in other contexts.

We would like to acknowledge N. Datta and T. Dorlas for the idea of the channel construction and valuable assistance. This work is based upon research supported by the South African Research Chair Initiative of the Department of Science and Technology and National Research Foundation.

# Bibliography


[1] *The Capacity of the Quantum Channel with General Signal States*, A.S. Holevo (IEEE Trans. Inform. Theory 44; 269; 1998)

[2] *Sending classical information via noisy quantum channels*, B. Schumacher and M.D. Westmoreland, (Phys. Rev. A; 56; 131; 1997)

[3] *Quantum Dynamical Systems*, R. Alicki and M. Fannes, (Oxford University Press; Oxford; 2001)

[4] *Quantum Channels with Memory*, D. Kretschmann and R.F. Werner, (Phys. Rev. A; 72(6):062323; 2005)

[5] *Functions of Markov processes and algebraic measures*, M. Fannes, B. Nachtergaele, and L. Slegers, (Rev. Math. Phys. 4; 39; 1992)

[6] *The Entropy of Functions of Finite State Markov Chains*, D. Blackwell, (Trans. First Prague Conference on Information Theory, Decision Functions, and Random Processes; Prague; 13-20; 1957)

[7] *The Capacity of the Quantum Depolarizing Channel*, C. King, (IEEE Transactions on Information Theory; 49(1); 2003)

[8] *Grundbegriffe der Warscheinlichkeitsrechnung*, A.N. Kolmogorov, (Springer Verlag; Berlin; 1933)

[9] *Monotonicity with volume of entropy and of mean entropy for translationally invariant systems as consequences of strong subadditivity*, A.R. Kay and B.S. Kay, (J. Phys. A: Math. Gen. 34; pg. 365-382; 2001)

[10] *Classical capacity of quantum channels with general Markovian correlated noise*, N. Datta and T.C. Dorlas, (arXiv:0712.0722v1 [quant-ph]; 2007)

[11] *The Coding Theorem for a Class of Quantum Channels with Long-Term Memory*, N. Datta and T.C. Dorlas, (J. Phys. A: Math. Theor.; 40; 8147-8164; 2007)

[12] *Entanglement Assisted Classical Capacity of a Class of Quantum Channels with Long-Term Memory*, N. Datta, Y. Suhov and T.C. Dorlas, (Quantum Information Processing; 7(6); 2008)

[13] *Quantum capacity of dephasing channels with memory*, A. D'Arrigo, G. Benenti and G. Falci, (New J. Phys. 9, 310, 2007)

# Chapter 4

# Monte Carlo Simulation Background

## 4.1 Monte Carlo Method

### 4.1.1 Introduction

"Monte Carlo Algorithms" is a broad class of computational algorithms that employ random sampling. The term derives from the Monte Carlo Casino in Monaco, where activities involve randomness and probability theory. The Metropolis-Hastings Markov Chain Monte Carlo (MCMC) algorithm is named after Nicholas Metropolis, who led the team that first invented and used the method [1] and W.K. Hastings [2] who generalized it.

The main idea is to sample from a difficult-to-compute probability distribution by setting up a specially designed Markov chain random walk through the support of the probability distribution. For a correctly set up Markov walk, the desired probability distribution is the unique, invariant (also called stationary) and attractive distribution under the random walk.* We then use these draws from the desired probability distribution to perform an Importance Sampling Monte Carlo Integration to calculate our statistic.

*This probability-distribution drawing Markov chain is unrelated to the Markov chain used to switch between sub-channels in our application.

### 4.1.2 Invariant Distribution

In our use of the MCMC algorithm, a desired probability distribution is given, $\Pi(\mathbf{x})$, and we must construct a Markov chain whose stationary distribution is the desired distribution.

We know the distribution that we want to draw from completely, but for reasons of tractability, we can not practically draw from this distribution. The MCMC approach designs a time-homogeneous Markov process to draw from the desired distribution. A Markov process is characterized by a transition matrix $T(\mathbf{k}_1, \mathbf{k}_2)$ that moves one state, $\mathbf{k}_1$, to another, $\mathbf{k}_2$, with some probability $T(\mathbf{k}_1, \mathbf{k}_2)$. The probability of transition depends only on the current state. The state space in our case is the space of eigenvalues. If we keep track of the probability of occupying a state after a transition we'll require a vector of probabilities of reaching all states. We may then apply the transition matrix again and calculate the new probabilities. After repeated application of the transition matrix we may expect that the probability vector converges. Indeed for a Markov process that is irreducible, aperiodic and positive recurrent we do have convergence to a unique eigenvector with eigenvalue one. This probability eigenvector is what we want to specify and the task is to work backwards to create a transition matrix that has it as its stationary distribution. Therefore we desire the following eigenvalue relationship,

$$\Pi(\mathbf{y}) = \int T(\mathbf{x}, \mathbf{y})\Pi(\mathbf{x})d\mathbf{x}.$$

In our case we require $\Pi(\mathbf{k}) = \lambda_N(\mathbf{k})$. The remaining task is to design $T$ to satisfy the above eigenvalue equation. As suggested by Hastings [2], a sufficient condition is the so called *detailed balance* or *time reversibility* condition,

$$\Pi(\mathbf{x})T(\mathbf{x}, \mathbf{y}) = \Pi(\mathbf{y})T(\mathbf{y}, \mathbf{x}).$$

This condition captures the idea that the transition matrix must balance any 'asymmetry' in the desired distribution for that distribution to remain unchanged under one transition. So that moving from a state of, say, high probability to a state of low probability happens with just the right probability that the reverse move (picking the

low probability site and then transitioning to the high probability site) has the same probability.

Once we construct a transition matrix that satisfies this condition, we don't actually keep track of a probability vector over all states, we just realise the probability distribution by taking concrete steps to specific states chosen randomly. Then, because of ergodicity[†], the statistics we collect from a single run (or multiple averaged runs) are equivalent to the expectation under the unique stationary distribution (which is often termed *ensemble statistic*).

### 4.1.3 Transition Matrix

It is useful to decompose the transition step into two steps. Firstly, a candidate generating step, $q(\mathbf{x}, \mathbf{y})$, and thereafter the candidate accepting step, $a(\mathbf{x}, \mathbf{y})$.

$$T(\mathbf{x}, \mathbf{y}) = q(\mathbf{x}, \mathbf{y})a(\mathbf{x}, \mathbf{y}).$$

In order for this separation to be especially useful we impose that the candidate accepting step be symmetric:

$$q(\mathbf{x}, \mathbf{y}) = q(\mathbf{y}, \mathbf{x}).$$

This split allows the algorithm to be tuned more easily. The candidate generating step is free to tune parameters such as step size away from the current position and the candidate accepting step focuses on satisfying detailed balance. To see this, notice how the detailed balance condition is now expressed only in terms of $a(\mathbf{x}, \mathbf{y})$,

$$\begin{aligned} \Pi(\mathbf{x})a(\mathbf{x}, \mathbf{y}) &= \Pi(\mathbf{y})a(\mathbf{y}, \mathbf{x}) \\ a(\mathbf{x}, \mathbf{y}) &= \frac{\Pi(\mathbf{y})}{\Pi(\mathbf{x})}a(\mathbf{y}, \mathbf{x}). \end{aligned}$$

The last line shows how the forward and backwards probabilities must relate to each other. To keep the algorithm moving along as fast as possible we would like $a(\mathbf{x}, \mathbf{y})$ and $a(\mathbf{y}, \mathbf{x})$ to be as high as possible. But, if $\frac{\Pi(\mathbf{y})}{\Pi(\mathbf{x})} > 1$, then $a(\mathbf{y}, \mathbf{x})$, must be sufficiently small

[†]Here we mean the mathematical definition of an ergodic chain, which, in the form of a theorem, implies the equivalence of 'time' and 'ensemble' statistics, see Section 4.1.4

to ensure that the product is less than unity, since $a(\mathbf{x}, \mathbf{y})$ is after all, a probability. Since we want to make $a(\mathbf{y}, \mathbf{x})$ as large as possible, we choose the largest value that makes the product unity, namely $\frac{\Pi(\mathbf{x})}{\Pi(\mathbf{y})}$. In order to keep both directions as valid probabilities and as large as possible the following must be satisfied,

$$\begin{aligned} a(x, y) &= \begin{cases} 1 & \frac{\Pi(\mathbf{y})}{\Pi(\mathbf{x})} >= 1 \\ \frac{\Pi(\mathbf{x})}{\Pi(\mathbf{y})} & \text{otherwise} \end{cases} \\ &= \min\left(1, \frac{\Pi(\mathbf{x})}{\Pi(\mathbf{y})}\right). \end{aligned}$$

All that remains is to define $q(\mathbf{x}, \mathbf{y})$, the candidate generating function. We choose to explore around the current location with a probability that falls off with increasing Hamming distance (the number of sites that the error strings differ at),

$$q(\mathbf{k}_1, \mathbf{k}_2) = \left(\frac{2}{N}\right)^{N - \mathbf{k}_1 \cdot \mathbf{k}_2} \left(\frac{N-2}{N}\right)^{\mathbf{k}_1 \cdot \mathbf{k}_2},$$

where the 'dot product', $\mathbf{k}_1 \cdot \mathbf{k}_2$, is the number of sites that agree and $2/N$ is the stipulated probability that the candidate state is flipped at a particular site. Since there are $N$ sites the expected number of differences between the candidate and current error strings has been set up to be 2. This number can actually be a parameter that can be varied. It captures the average step size away from the current position that is explored. Further on, we study the effects of varying this parameter on the performance of the algorithm. Thus the final transition matrix is:

$$T(\mathbf{x}, \mathbf{y}) = q(\mathbf{x}, \mathbf{y}) a(\mathbf{x}, \mathbf{y}) = \left(\frac{2}{N}\right)^{N - \mathbf{x} \cdot \mathbf{y}} \left(\frac{N-2}{N}\right)^{\mathbf{x} \cdot \mathbf{y}} \min\left(1, \frac{\Pi(\mathbf{x})}{\Pi(\mathbf{y})}\right).$$

It is easy to verify that the matrix is indeed a valid stochastic matrix, $\sum_{\mathbf{y}} T(\mathbf{x}, \mathbf{y}) = 1, \forall \mathbf{x}$.

### 4.1.4 Ergodicity

There are two common uses of the term *ergodic*[‡]. In the mathematics community, ergodic simply means a Markov chain that is irreducible (every state can be reached from every other) and positive recurrent (every state is revisited). In the physics community ergodic

[‡]The discussion on ergodicity is common knowledge, but the following references may be consulted: [3] and [4].

means any process (not only Markov), for which the 'time' averaged statistic is equal to the equivalent ensemble statistic. Here by ensemble statistic we mean the expectation with respect to the stationary distribution of the process.

We are working with Markov chains in the MCMC method where both senses of the word are important. We start with the mathematical sense of ergodic by requiring that our chain be ergodic. Then we use the theorem that an ergodic chain (one that is irreducible and positive recurrent) is automatically ergodic in the physic's sense (time averaged statistics are equal to ensemble statistics). Here we have also used mathematical ergodicity in proving that a unique stationary distribution even exists in order to define what we mean by taking the ensemble expectation.

To have good convergence properties it is also desirable that the Markov chain be *mixing*, that is after sufficient time two state's occurrences become independent. In a Markov chain this is guaranteed by ergodicity plus *aperiodicity*, that is the greatest common divisor of the periods of return is one, or put differently, after a certain integer all periods of return are possible.

In the case of a finite state space such as we have, mathematical ergodicity is easier to demonstrate. Only irreducibility is required because positive recurrence follows from the finiteness of the state space. We still however need aperiodicity for mixing.

So in our case, irreducibility follows from $T(\mathbf{x}, \mathbf{y}) > 0$ for all $\mathbf{x}$ and $\mathbf{y}$. Aperiodicity also follows from the same fact.

### 4.1.5 Estimator and its Variance

We are using the MCMC to calculate some statistic,

$$\chi = \langle f(X) \rangle = \sum_{\mathbf{k}} \Pi(\mathbf{k}) f(\mathbf{k}).$$

Since we cannot actually execute the sum, we need to design an estimator. As the Markov walk proceeds, draws are used to calculate an estimate of the statistic, $\overline{\chi} = \sum_{i=0}^{m} f(\mathbf{k_i})/m$, which involves each draw, $\mathbf{k_i}$, and a final averaging over the draws.

$\overline{\chi}$ is of course a random variable on its own, which by the law of large numbers is

approximately Gaussian (the underlying distribution has a finite variance since we are considering finite sample spaces). Since the mean is unbiased, the expectation of the estimator is equal to the quantity we are trying to estimate

$$\langle \overline{\chi} \rangle = \chi.$$

If we define the underlying variance as $\sigma_f^2 \equiv \left\langle (f(X) - \chi)^2 \right\rangle$, then the variance of the estimator is

$$\sigma_{\overline{\chi}}^2 = \sigma_f^2 / m.$$

This shows a decrease in the variance of the mean as the sample size increases. The proof follows simply from the theorem that the variance of a sum of independent random variables is the sum of the variances and the variance of a constant times a random variable is the constant squared times the variance.

We do not actually have access to $\sigma_f^2$ (due to intractability[§]) but it is some small constant to start off with. The very reason for the importance sampling approach is to make this specific number small. The crude Uniform Monte-Carlo method tackles the integral by sampling uniformly with the unfortunate limitation of a larger underlying variance. Most of the motivations for more advanced methods is precisely *variance reduction.* But once a scheme is chosen, the underlying variance is fixed, and the variance of the estimator typically has the same $1/m$ behaviour.

Since we don't have $\sigma_f^2$, we don't have the true $\sigma_{\overline{\chi}}^2$. However, we approximate $\sigma_f^2$ by the variance estimator

$$\overline{\sigma_\chi^2} = \frac{1}{m-1} \sum_{i=1}^{m} (f(\mathbf{k}_i) - \overline{\chi})^2.$$

The variance estimator is an unbiased estimator[#] of the underlying variance,

$$\left\langle \overline{\sigma_\chi^2} \right\rangle = \sigma_\chi^2.$$

[§] A striking thought is that in our implementation there are more paths than back-of-the-envelope estimates of particles in the observable universe!

[#] Note, though that square rooting the variance to arrive at the standard deviation is not unbiased. In this thesis, we ignore this complication because of sufficiently small error bars.

Since the variance estimator is itself a random variable, technically we should also worry about its variance[‖]. Fortunately, if we kept on going in this fashion there is a point where we could stop by virtue of the emerging normal behaviour (the mean estimator is approximately normal). This happens when we look at the standard deviation of the variance estimator of the variance estimator [5]

$$\sigma_{\overline{\sigma^2_{\overline{\sigma^2_\chi}}}} \approx \frac{\overline{\sigma^2_{\overline{\sigma^2_\chi}}}}{\sqrt{\frac{1}{2}m}}.$$

Notice how the right-hand side now only contains sample statistics. Any higher-order variances equate to zero because of the assumed normality (which emerges even for very low sample numbers).

In the end, instead of worrying about the variance of the variance of the variance, we content ourselves with repeating the entire Monte Carlo run and checking to see that the inter-run variance agrees with the intra-run variance. This would signal that the sample size is large enough and that we may trust the variance estimator.

## 4.2 Random Number Generator

All Monte Carlo algorithms make use of randomness in some way. It is, in general, difficult to satisfactorily define randomness, but for our purposes we desire a sequence of seemingly uncorrelated numbers, as judged by a bank of statistical tests. Most importantly, the way in which the random sequence is used must be uncorrelated to the way in which the sequence is generated. This allows the use of pseudo-random number generators, where the generation is actually deterministic but where the sequence passes the relevant statistical tests. These generators are fast and convenient, especially in the sense that they are run on the same CPU's that execute the Monte Carlo algorithm and so no external source of randomness is required[**].

There are of course some minor pitfalls to be wary of, such as with the *LCG* generators and higher-dimensional correlations. However, the field has matured over the years to

[‖] and so on, that is, "turtles all the way down".

[**] except the *seed*, which is based on the time of execution.

arrive at some guidelines that have proven their worth in use. For example, a good generator should combine at least two unrelated methods which share no state [6]. A requirement most relevant to the MCMC algorithm is that the generator should have a period of at least $2^{64}$.

For our implementation we use the recommended generator in Numerical Recipes [6], which has a period of $3 \times 10^{57}$. This period is more than enough for our purposes, where our longest run is $2^{30} < 10^{10}$.

## 4.3 Correlations

The concept of correlations features very strongly in the MCMC algorithm. Markov chains are at the heart of the algorithm, which contains the most basic kind of correlation, namely the current draw depends on the past only through the most recent previous draw.

These correlations assist in sampling the probability distribution correctly by favouring the more likely draws by the exact right amount. Built in, as well, is the correct low probability of visiting unlikely regions. As we saw in the Transition Matrix section (Sec. 4.1.3), drawing from the full probability distribution, is guaranteed by detailed balance. Detailed balance imposes some demands on the kind of correlations needed to realise the required invariant distribution. However, it does not dictate how much correlation. The amount of correlation has a bearing on the efficiency of the MCMC algorithm. In this thesis, we confirm how varying the correlation strength, as captured by the step size, affects the efficiency. Here we see that a balance is sought; not too little correlation, lest we waste time generating useless possible candidates and not too much correlation, otherwise we slow convergence by exploring the support too slowly.

In the way draws are generated, intermediate correlations are useful. But in the way draws are used, if the application requires independent draws, the resulting residual correlations are only a nuisance. Left uncorrected, convergence is slowed, but at least not destroyed. To distil away these correlations costs processing time, either by thin-

ning/skipping the use of draws or by some *blocking* technique.

Fortunately, the correlations between draws drop off exponentially fast. This is measured by calculating the correlation of a sequence of draws against a shifted version of the same sequence. For a general Markov process, the correlation drops exponentially fast with greater shift. The factor in the exponential factor is called the *correlation time* and it has the following meaning: it is the amount of time (or the number of steps) to wait after which the shifted correlation has dropped by a factor $1/e$ of the variance (zero-shifted autocorrelation). In this thesis we employ thinning and monitor the correlation time.

### 4.3.1 The Interplay of Noise Correlations Exhibit Fractal Behaviour

In closing this background chapter, we would like to comment on the effects of correlations in the noise on the desired probability distribution. These correlations have nothing to do with the artificial algorithmic correlations within the MCMC method. These are real world correlations in the noise that we are modelling.

As is shown in the next chapter we use the MCMC algorithm to approximate some statistic. This calculation involves the desired probability distribution $\Pi(\mathbf{k})$, which is derived from the correlations in the channel noise.

To see that this exponential in size probability distribution is indeed a complicated function that requires approximation, we plot the discrete probability mass function, $\Pi(\mathbf{k})$, with $N = 2^{20}$, in Fig. 4.1.

We first plot it as indexed by $\mathbf{k}$ interpreted as a binary number (for convenience it is converted to decimal). In this view we see some beautiful fractal-like patterns. This is partially unnatural because of the arbitrariness of the indexing[††]. To see this we sort the probability distribution by the probability value to arrive at Fig. 4.2.

However, this sorting could have been done for a legitimate fractal, hiding the fractal nature in the $x$ co-ordinate, so the argument could still be made that the probability distribution is in some sense fractal. Regardless, the form of this sorted distribution is unknown and actually doing the sorting, takes exponentially long. Therefore, since the

[††]Indeed the number of 1's in a binary number have a fractal like relationship to the value of the number.

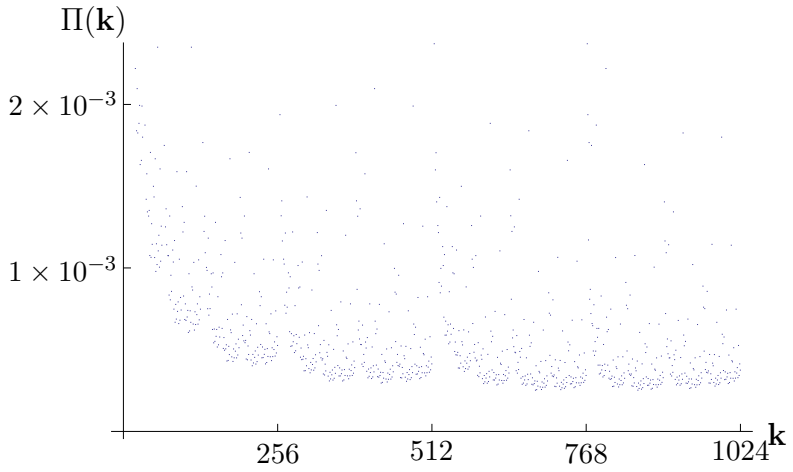



Figure 4.1: $\Pi(\mathbf{k})$ vs Binary$(\mathbf{k})$.

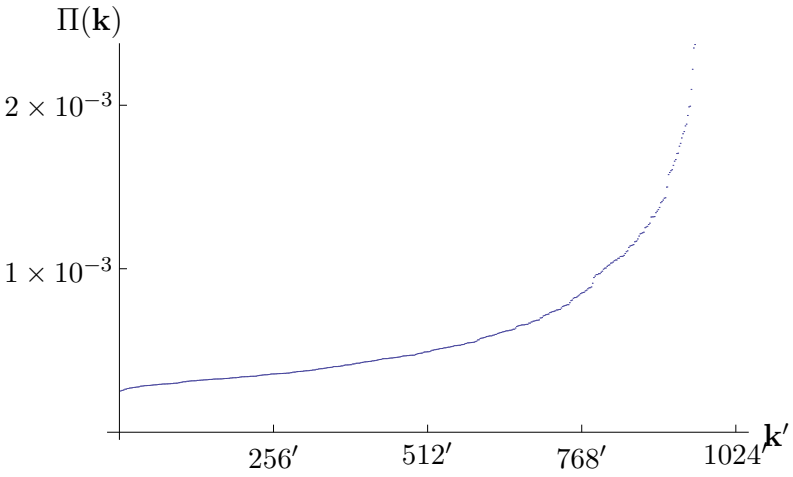



Figure 4.2: Sorted $\Pi(\mathbf{k})$ vs $\mathbf{k}'$.

probability distribution is so complicated, intricate and unknown, we have to resort to MCMC approximation techniques.

# Bibliography


[1] *Equation of State Calculations by Fast Computing Machines*, N. Metropolis, et al. (J. Chem. Phys. 21, 1087; 1953)

[2] *Monte Carlo Sampling Methods Using Markov Chains and Their Applications*, W.K. Hastings (Biometrika Vol. 57; No. 1; pg. 97-109; 1970)

[3] *Understanding the Metropolis-Hastings Algorithm*, S. Chib and E. Greenberg, (The American Statistician; Vol. 49; No. 4; pg. 327-335; 1995)

[4] *Lecture Notes on Ergodic Theory*, O. Sarig, (Penn State; 2008)

[5] *Monte Carlo Methods*, J.M. Hammersley and D.C. Handscomb (Metheun's Monographs on Applied Probability and Statistics, London: Metheun & Co Ltd; 1975)

[6] *Numerical Recipes: The Art of Scientific Computing*, W. H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery (Cambridge University Press; 2007)

[7] *Markov Chains and Stochastic Stability Second Edition*, S. Meyn and R.L. Tweedie, (Cambridge University Press; 2009)

# Chapter 5

# A Monte Carlo Simulation of a Noisy Quantum Channel with Memory


## Abstract

The classical capacity of quantum channels is well understood for channels with uncorrelated noise. For the case of correlated noise, however, there are still open questions. We calculate the classical capacity of a forgetful channel constructed by Markov switching between two depolarizing channels. Techniques have previously been applied to approximate the output entropy of this channel and thus its capacity. In this paper, we use a Metropolis-Hastings Monte Carlo approach to numerically calculate the entropy. The algorithm is implemented in parallel and its performance is studied and optimized. The effects of memory on the capacity is explored and previous results are confirmed to higher precision.


## 5.1 Introduction

Understanding noise-memory effects on capacity is an important step in modelling real world quantum channels. The first major theorem concerning the classical capacity of quantum channels was the celebrated Holevo-Schumacher-Westmoreland (HSW) theorem [1]. This theorem assumes the channel receives product state inputs and each

application of the channel is independent of subsequent applications. Progress has been made in generalizing the results to entangled inputs and correlated uses. We are interested in the extension to, so called, *forgetful* memory channels [2], where the channel's application depends on previous applications.

A simple example of noise correlations between successive channel uses, is a forgetful channel constructed by Markov switching between two depolarizing channels. Here the properties of the channel for any given application depends on the properties of the channel on the previous application in a Markovian way. The channel properties do not depend on the states that are acted upon, which would be the subject of a different generalization. Nevertheless, the output states become correlated in a non-Markovian way via the Markovian correlated channel uses.

This complicated non-Markovian state correlation evades an easy closed form analytic treatment. As we explore more complicated generalizations, it is helpful to build the numerical tools to deal with the increasingly intractable probability distributions that are involved. With previous partial analytic results [3], we are in a position to switch to non-trivial numerical methods and confirm their correct functioning, before moving into regimes where there are no analytic comparisons.

The numerical method that we employ is the Metropolis-Hastings Markov Chain Monte Carlo Method (MCMC). Beginning with its formulation [4] and generalization [5], MCMC has proven to be a powerful and versatile tool in tackling a variety of recent complicated and analytically intractable problems.

> "MCMC methods have revolutionized statistical computing ... [and has] enabled the development and use of intricate models in an astonishing array of disciplines..." [6].

By virtue of the ability to compare to partial analytic results, this paper could be an example of learning about the application of the Monte Carlo algorithm and monitoring its effectiveness. In Sec. 5.2. we recount the construction of the forgetful channel and derive an expression for the classical capacity in terms of the output entropy. In Sec. 5.3.

we express the capacity in a form that makes the application of the MCMC algorithm natural and we identify the parameters of the algorithm that need to be tuned. In Sec. 5.4. we analyse both the performance of the algorithm and the results of the simulation. Finally in Sec. 5.5. we conclude and mention further work.

## 5.2 Construction of the Channel

We construct a *forgetful* memory channel and incorporate memory effects by switching between two memoryless single qubit depolarizing channels ($\mathcal{E}_0$ and $\mathcal{E}_1$), using a two-state Markov chain. The $2 \times 2$ channel transition/selection matrix is $Q = (q_{ij}), \quad i, j \in \{0, 1\}$ with $q_{ij}$ being the probability of switching from channel $i$ to channel $j$.

The usual depolarizing channel, is normally viewed as uniformly shrinking the Bloch sphere. For its use in our noisy channel it is helpful to rewrite it as a mixture between the identity channel and the 'flip channel',

$$\mathcal{E}_i(\rho) = x_i^0 \rho + x_i^1 (\mathbb{1} - \rho),$$

where $x_i^0$ is the probability of returning the state unscathed (identity channel) and $x_i^1$ is the probability of reflecting $\rho$ about the centre of the Bloch sphere. Naturally, $x_i^1 = 1 - x_i^0$. To see the action of the flip, write $\rho$ as

$$\rho = \frac{1}{2}(\mathbb{1} + \vec{\sigma} \cdot \vec{r}),$$

where $\vec{\sigma}$ is a vector consisting of the Pauli matrices $(\sigma_x, \sigma_y, \sigma_z)$, $\mathbb{1}$ is the $2 \times 2$ identity matrix and $\vec{r}$ is the Bloch vector. The flip action is now

$$\mathbb{1} - \rho = \frac{1}{2}(\mathbb{1} + \vec{\sigma} \cdot -\vec{r}),$$

where the Bloch vector has been multiplied by $-1$. Surprisingly the channel is only completely positive for $1/3 \leq x_i^0 \leq 1$.

The built-up channel, $\Lambda_N$, corresponding to $N$ successive uses of the single qubit sub-channels, is constructed as follows

$$\Lambda_N = \rho_1 \otimes \ldots \otimes \rho_N \mapsto \sum_{i_1, \ldots, i_N} \gamma_{i_1} q_{i_1 i_2} \ldots q_{i_{N-1} i_N} \mathcal{E}_{i_1}(\rho_1) \otimes \ldots \otimes \mathcal{E}_{i_N}(\rho_N).$$

The sum is over all possible Markov paths $(i_1, \ldots, i_N) \in \{0,1\}^N$ and each term is a tensor product of the selected sub-channels weighted by the probability of occurrence ($\gamma_i$ is the initial probability of channel selection). We first need to calculate the capacity of this $N$-use form of the channel and then take the limit as $N \longrightarrow \infty$, which is termed *regularizing* the channel.

By expanding the above product and focusing on pure product state inputs, we see that the output is a sum over all possible combinations of flipping (an error occurs) and not flipping (no error occurs). Due to the symmetry of the depolarizing channel, these output states remain diagonal in the bases of flipped and not flipped, no matter what pure product states are used. Therefore, the eigenvalues of the output state as a matrix are

$$\lambda_N(\mathbf{k}) = \sum_{i_1,\ldots,i_N} \gamma_{i_1} q_{i_1 i_2} \ldots q_{i_{N-1} i_N} x_{i_1}^{k_1} \ldots x_{i_N}^{k_N},$$

where the eigenvalues are indexed by $\mathbf{k} = \{k_1, k_2, \ldots, k_N\} \in \{0,1\}^N$, which is a string recording the sequence of flips/errors and non-flips/non-errors.

The HSW theorem [1] built on by the forgetful channel extension [2] provides an expression for the capacity $C^1_{\text{Classical}}$ (classical product state)

$$C^1_{\text{Classical}}(\Lambda_N) = \chi^*(\Lambda_N),$$

where $\chi^*$ is the maximisation over input ensembles of the Holevo $\chi$ quantity.

For this channel, the maximum is obtained using the uniformly distributed computational basis states. By taking the asymptotic average of the $N$-product classical capacity, we arrive at the capacity of the memory channel in terms of the output entropy [2],

$$C^* = \lim_{N\to\infty} \frac{1}{N} C^1_{\text{Classical}}(\Lambda_N)) = 1 - \lim_{N\to\infty} \frac{1}{N} \mathrm{S}(\Lambda_N(\rho)) = 1 - \lim_{N\to\infty} \frac{1}{N} \mathrm{S}(\{\lambda_N(\mathbf{k})\}).$$

The difficulty in calculating the entropy, $\mathrm{S}(\{\lambda_N(\mathbf{k})\})$, is with the summation of exponentially-in-$N$ many terms, which quickly becomes intractable to perform computationally. Therefore powerful approximation techniques, such as the Metropolis-Hastings Markov Chain Monte Carlo (MCMC) algorithm, are required.

## 5.3 Monte Carlo Method

### 5.3.1 Entropy

Entropy features in our calculation of the capacity because it captures the average noise injected into the input state.

$$\langle -\log(P_X(\mathbf{k}))\rangle = \int -\log(P_X(\mathbf{k}))dF_X(\mathbf{k}) = \sum_x -\log(P_X(\mathbf{k}))P_X(\mathbf{k}) = \sum_{\mathbf{k}} -\lambda_N(\mathbf{k})\log(\lambda_N(\mathbf{k})).$$

It is this integral that we are approximating using Monte Carlo integration. In this way, drawing from the space of the eigenvalues' indexes, allows us to calculate the entropy by simply averaging the log's of the drawn eigenvalues. There are $2^N$ configurations of the index $\mathbf{k}$ corresponding to the different error strings of $N$ uses of the channel. Simply summing over all configurations is computationally intractable for large $N$ and we resort to the MCMC algorithm. Having identified the desired probability distribution we must now construct a Markov Chain in order to use the MCMC algorithm.

A Markov chain is defined by its state space and the transition matrix that has our desired probability distribution as its stationary distribution. Our state space is the $2^N$ configurations of $\mathbf{k}$. Our transition matrix as suggested by Hastings [5], satisfies the so called *Detailed Balance* or *Time Reversibility* condition and contains a candidate generating part (with a parameter controlling the step size) and a candidate accepting part,

$$T(\mathbf{x},\mathbf{y}) = q(\mathbf{x},\mathbf{y})a(\mathbf{x},\mathbf{y}) = \left(\frac{2}{N}\right)^{N-\mathbf{x}\cdot\mathbf{y}}\left(\frac{N-2}{N}\right)^{\mathbf{x}\cdot\mathbf{y}}\min\left(1,\frac{\Pi(\mathbf{x})}{\Pi(\mathbf{y})}\right).$$

It is easy to verify that the matrix is indeed a valid stochastic matrix, $\sum_y T(\mathbf{x},\mathbf{y}) = 1, \forall \mathbf{x}$. Now that we have constructed the channel and a Monte Carlo Markov Chain to sample from the output eigenvalue distribution to calculate the entropy, we are ready to implement it,

$$\mathrm{S}(\{\lambda_N(\mathbf{k})\}) \equiv \langle -\log(P_X(\mathbf{k}))\rangle \approx \frac{1}{m}\sum_{i=1}^{m}\log\lambda_N(\mathbf{k}) \equiv \overline{\mathrm{S}},$$

where $m$ is the number of samples and the $\mathbf{k}$'s are drawn from the probability mass function,

$$P_X(\mathbf{k}) = \lambda_N(\mathbf{k}).$$

$X$ is the random variable whose instances are the possible error strings $\mathbf{k}$.

### 5.3.2 Algorithm

Bringing it all together the algorithm proceeds as follows,

- Initialize the error string $\mathbf{k}$ (with a uniformly random starting point).
- Perform *burn-in*, which is to take a few Markov Chain steps without collecting statistics. This step tries to ensure that the Markov chain reaches its equilibrium distribution. We experiment with different burn-in lengths. While this process is non-rigorous, theoretically a poor burn-in length does not negate the validity of the final answer. All that it does do, is slow down convergence and produces larger error-variances in the final answer.
- The actual Markov step consists of generating a new candidate state based on the current state. As detailed above, we flip each site ($k'_i = k_i \oplus 1$) with some probability (`expectedflip`). The candidate state is accepted according to the *detailed balance* condition outlined above. The acceptance probability involves the calculation of $\lambda_N(\mathbf{k})$ for specific $\mathbf{k}$'s. There are of course in total, exponentially many $\mathbf{k}$'s and we are avoiding the need to calculate $\lambda_N$ for all these $\mathbf{k}$'s by taking a representative Markov walk through the space of $\mathbf{k}$'s. However, one remaining potential problem is that even for just one specific $\mathbf{k}$, the calculation of $\lambda_N(\mathbf{k})$ itself involves exponentially-in-$N$ many terms in its sum. Fortunately, a convenient and original rewriting, turns the sum of exponentially many terms into a recursive algorithm with polynomial-in-$N$ number of steps. A further programming optimization employed was the partial reuse of previous calculations of $\lambda_N(\mathbf{k})$ for different $\mathbf{k}$'s by virtue of the fact that only a few sites are flipped at a time and the sum has been rewritten recursively. Therefore it is only necessary to rewind the calculation to the first change in the flipped string.
- We experiment with different values for the above `expectedflip` probability while

monitoring the acceptance rate. The heuristic optimum is to maintain an acceptance rate of between $40\% - 50\%$. If the acceptance rate is too large (for example achieved by taking smaller steps), it results in the equilibrium distribution being explored too slowly. If the acceptance rate is too low, it results in a slowing of the convergence rate and an increase in the correlation between draws, because most candidates are rejected implying a reselection of the current state.

- The running sum of the $\log(\lambda_N(\mathbf{k}))$'s is updated after passing over a few steps as determined by the `skips` parameter. This process is called *thinning*. It is employed to reduce the correlation between draws so that each draw is independent. This is possible because correlations between Markov steps falls away exponentially fast. The number of steps to skip is experimented with. While thinning is non-rigorous (similar to burn-in), due to the Fundamental Theorem of Markov Chains this is not crucial. The theorem shows that even with no thinning the collected statistics for a *mixing* Markov chain is the same as for the equilibrium distribution. Thinning is employed only as a tactic to avoid complicated variance calculations (such as *blocking*) and to try and speed up convergence.

- Mean, variance and correlation statistics are collected.

- The algorithm loops from step 3 for as many sample points as desired.

Assuming that thinning sufficiently produces independent draws (which we monitor) from $P_{X_i}(\mathbf{k}) = \lambda_N(\mathbf{k})$ we are calculating the unbiased empirical entropy estimate of a known distribution, $P_X$ [7],

$$\overline{\mathrm{S}} = \frac{1}{m}\sum_{i=1}^{m} \log \lambda_N(X_i).$$

Defining the underlying variance as $\sigma_S^2 \equiv \left\langle (\log(P_X(\mathbf{k})) - S)^2 \right\rangle$, the variance of the estimator is,

$$\sigma_{\overline{\mathrm{S}}}^2 = \sigma_S^2/m.$$

Since we don't have $\sigma_S^2$, we don't have the true $\sigma_{\overline{S}}^2$. However we approximate $\sigma_S^2$ by the variance estimator,

$$\overline{\sigma_{\overline{S}}^2} = \frac{1}{m-1}\sum_{i=1}^{m}(\log \lambda_N(X_i) - \overline{S})^2.$$

The variance estimator is an unbiased estimator of the underlying variance,

$$\left\langle \overline{\sigma_{\overline{S}}^2} \right\rangle = \sigma_{\overline{S}}^2.$$

We check that the algorithm agrees to within error bars with the full calculation on two separate achievable occasions: low $N$ and high $N$ but with $s = 1$ (see later). Furthermore we repeat the entire Monte Carlo run and check to see that the inter-run variance agrees with the intra-run variance. This would signal that the sample size is large enough and that we may trust the variance estimator.

### 5.3.3 Parallel Programming

A very useful feature of the Metropolis Algorithm is that it can easily be parallelized. The different sample paths can be calculated on separate nodes and only at the end of the Markov sampling is it necessary to collect all the data and calculate averages and error estimates. This type of problem is often called *embarrassingly parallel*.

For this study, our own C++ code is written and run on the Centre for High Performance Computing's Sun Nehalem Cluster. The OpenMPI Message Passing Interface (MPI) library is used to manage the communication between nodes.

We can report success on the programming, compiling and running of tasks, with a marked speed-up from using the cluster. For example on one 128-node run that took 5 hours and 36 minutes, the total CPU time combined to 700 hours and 49 minutes, which equates to a 125× speed up.

We employed the following optimizations:

- more efficient running sums calculation of mean and variance
- a tailored load balancing algorithm

- only one MPI_Gather call at the end
- OpenMP for the full calculation

## 5.4 Analysis and Results

For easier analysis, let us introduce a relabelling of the parameters, $\{q_{00} \to q, q_{10} \to 1-q, x_{00} \to a+d, x_{10} \to a-d, q \to (s+1)/2\}$, where we have made the Markov switching matrix symmetric and parametrized it with its non-one eigenvalue, $(-1 < s < 1)$. An absence of correlations corresponds to $s = 0$. We have rewritten the channel error probabilities in terms of their average($a$) and difference($d$).

The main idea behind the construction of this channel is to explore the classical capacity as a function of the non-markovian memory. Thus plots of capacity versus the parameters are of primary interest. Our goal in this paper is to investigate how the MCMC algorithm performs in approximating the entropy and hence the capacity of the channel.

### 5.4.1 Capacity versus $s$

The main result is that the capacity increases as the noise correlations increase.

In Fig. 5.1, we see five iterations of the Algebraic Markov approach [3] and one run of the MCMC approach. The MCMC approach is further along in convergence and comes with error bars. As we can see, the capacity increases as the noise correlations increase from 0 to 1. The non-linear curve interpolates between the capacity of the average channel (lower) and the average capacity of the separate channels (higher). Here and throughout $a = 2/3$ and $d = 2/9$, with $s$, of course, varying. In subsequent sections where we explore the workings of the MCMC algorithm, $s$ is fixed to 0.7.

The dots correspond to a complete full entropy calculation with $s = 1$, where the entropy can be calculated explicitly for very high values of the channel length without exponential blow up. Ironically, it is this easier case that is hard for the MCMC method in that the error increases with $s$. This behaviour is because the underlying $\lambda_N(\mathbf{k})$ probability distribution becomes more peaked and with larger jumps as $s$ increases and so more

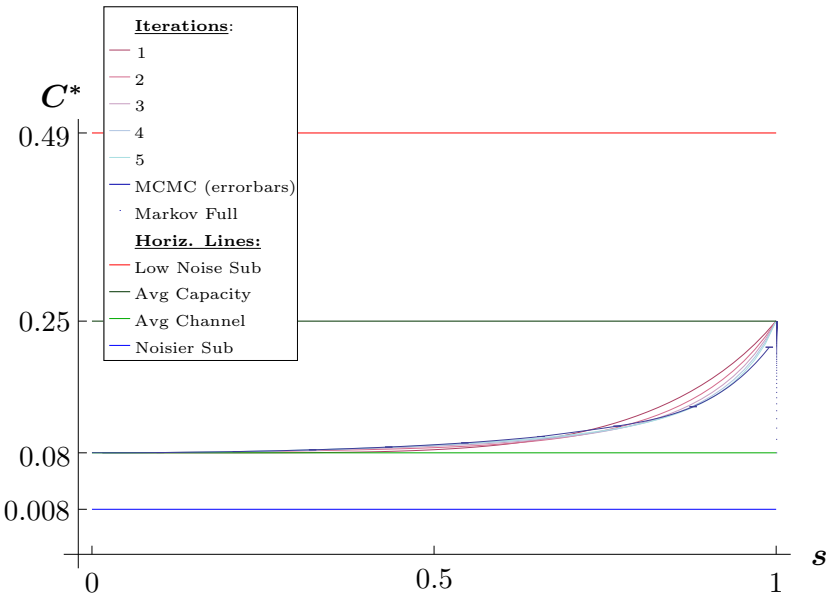



Figure 5.1: Capacity versus the memory parameter $s$ using many iterations and including full Markov calculation.

samples need to be drawn to explore this distribution to achieve the same accuracy. This is an example of how the parameters of the problem affect the performance of the MCMC algorithm. Of course for fixed problem parameters we still get the usual $1/m$ reduction in the estimate variance as the number of samples increase.

### 5.4.2 Expected $1/m$ behaviour for the Estimate Variance vs Sample size

In order to verify that the MCMC algorithm is working as expected we should observe a $1/m$ improvement in the variance of the estimate, $\sigma_S^2$, as the sample size, $m$, increases. Or equivalently a $1/\sqrt{m}$ improvement of the standard deviation, $\sigma$. If this is confirmed, we know that we can calculate the entropy to higher precision, if so desired, by increasing the sample size.

In collecting statistics we distinguish between inter-run and intra-run. Inter-run means the statistics we collect during one instantiation of a Markov Chain: one random starting point, one run of burn-in and one correlated sequence of Markov steps. For intra-run statistics we look at the final end estimates of different Markov runs and compare their

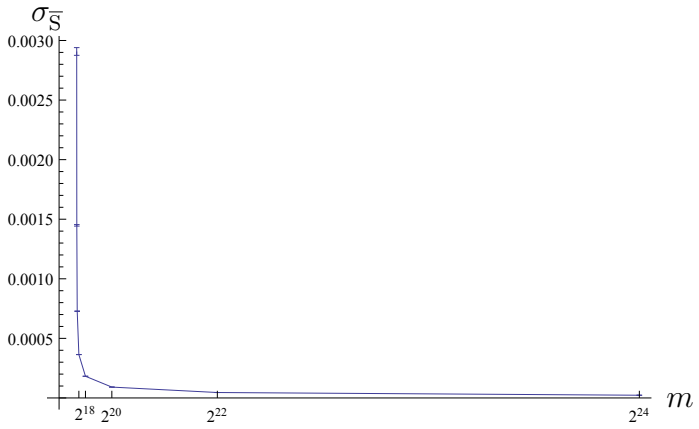



Figure 5.2: Inter-run $\sigma_{\overline{S}}$ vs $m$.

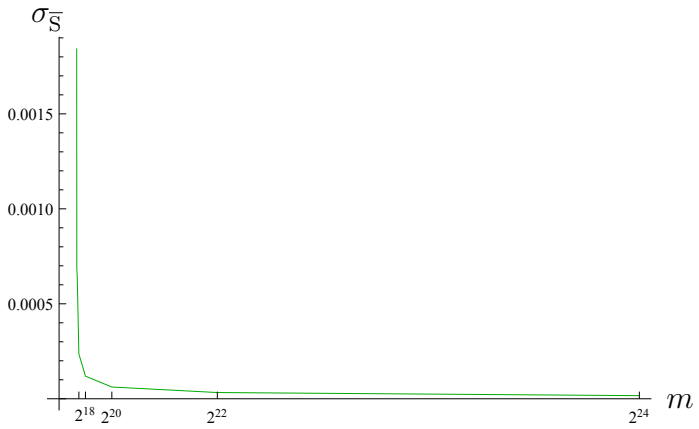



Figure 5.3: Intra-run $\sigma_{\overline{S}}$ vs $m$.

means and variances.

We plot both the average inter-run standard deviation of the estimates, Fig. 5.2 and the intra-run standard deviation of the estimates from the mean of the estimates, Fig. 5.3. The inter-run $\sigma$ and the intra-run $\sigma$ should behave the same and operate to within the same orders of magnitude. Indeed, we are using this as a test of whether enough samples have been taken. If too few samples are taken, the inter-run, $\sigma$ would in general be lower than the intra-run, $\sigma$, because of the correlations between steps within the Markov Chain and the deliberate lack of correlation between the random initial starting

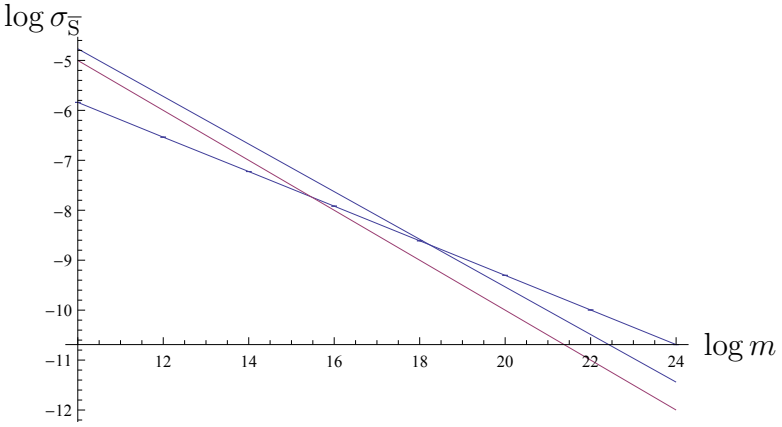



Figure 5.4: Inter-run $\log \sigma_{\overline{S}}$ vs $\log m$.

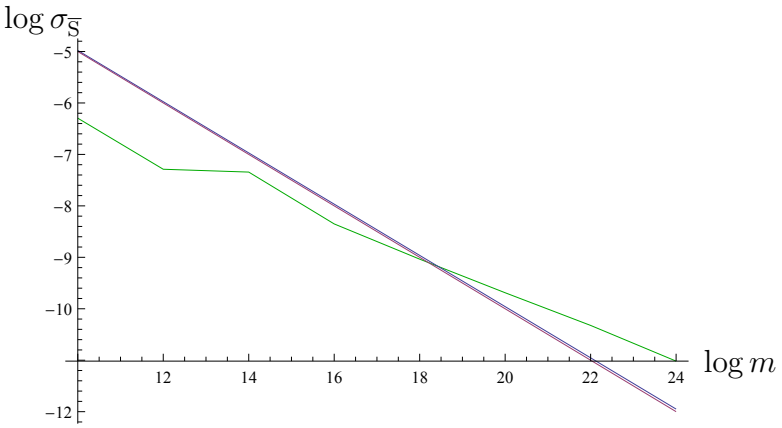



Figure 5.5: Intra-run $\log \sigma_{\overline{S}}$ vs $\log m$.

point between runs.

To more clearly see the $m^{-1/2}$ behaviour, we plot, in Fig. 5.4 and Fig. 5.5, the log of the inter-run and intra-run $\sigma_{\overline{S}}$ against $\log m$ and look for a slope of $-1/2$. The solid line is the joined data. The dashed line is the best fit straight line through the origin and for comparison we also plot the true $m = -1/2$ line (dot-dashed).

Overall we can see that the behaviour is the same and that the order of magnitude is roughly the same.

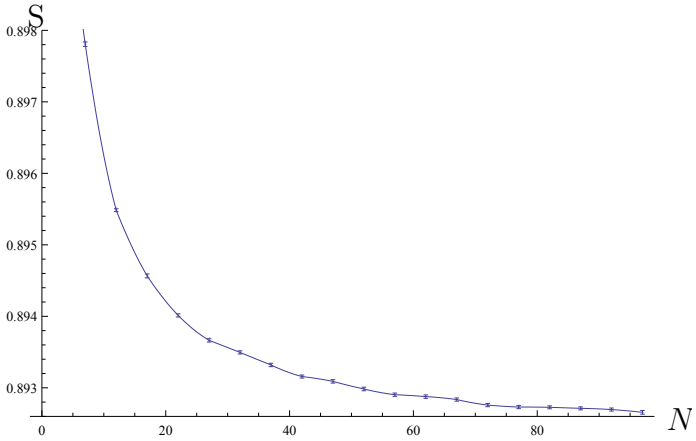



Figure 5.6: Average Entropy vs Chain Length (intra-run $\sigma$ error bars).

### 5.4.3 Correct Regularizing Behaviour

The next most important test of the MCMC algorithm is to take the algorithm into the regime where it truly outshines other numerical techniques. In this regime we also hope to confirm the theoretical idea that regularizing the channel does lead to a well defined limit. Here we take the channel to a length of 100. If we were to cover all possible Markov paths, we would need to enumerate $2^{100}$ paths, which is computationally intractable.
In Fig. 5.6 we plot the entropy versus the chain length and observe convergence. We also chart with dots the full true entropy up to chain length 30, to demonstrate that the true value lies closely within the error bars.

### 5.4.4 Tuning Parameters

In this section we discuss the tuning of the MCMC algorithm for improved performance. In the quest for optimal efficiency, we are also indirectly verifying that the algorithm is indeed working as expected.

#### Burn-in and reaching the invariant distribution

The burn-in runs differ from the normal MCMC algorithm in that after every burn-in procedure we collect a few samples representative of the distribution reached at the end

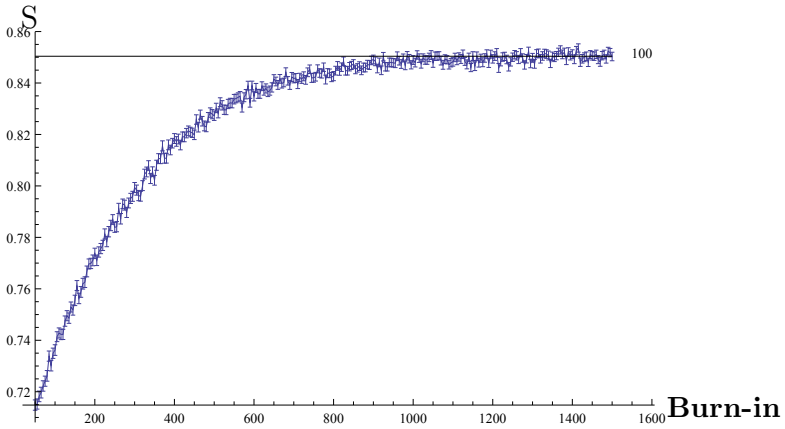



Figure 5.7: Average Entropy vs Burn-in Length (intra-run $\sigma$ error bars).

that specific burn-in (while doing this we set the skip parameter to zero). We then repeat this procedure many times to actually build up the statistics. In this way we are teasing out the effect of the initial burn-in procedure from the naturally present "self-burn-in" effect of a Markov walk. To further restrict other randomization effects and focus only on burn-in, we also switch off the initial randomization and always start at the same state vector, $\mathbf{k}_0 = \mathbf{0}$.

As we collect statistics at the tail of different burn-ins, the degree to which the statistics reproduce the invariant distribution is the degree to which "enough" burn-in has occurred. To see this, we plot in Fig. 5.7, the entropy versus the burn-in length. As we can see, the burn-in entropy asymptotically approaches a limiting value. For illustration purposes a line has been drawn at the expected limit, whose entropy value has been calculated using a normal run with a large number of samples*. Notice that the error-bars are not large enough to accurately include the limiting value. For low burn-in values this cannot even be remedied by doing more meta-runs, the reason is that for low number of sample values, the pocket of the invariant distribution explored does indeed have a different "true" value of the entropy.

This burn-in way of calculating entropy is highly inefficient because of the chain restart-

*Note that the approach is from below because of the correlations between draws.

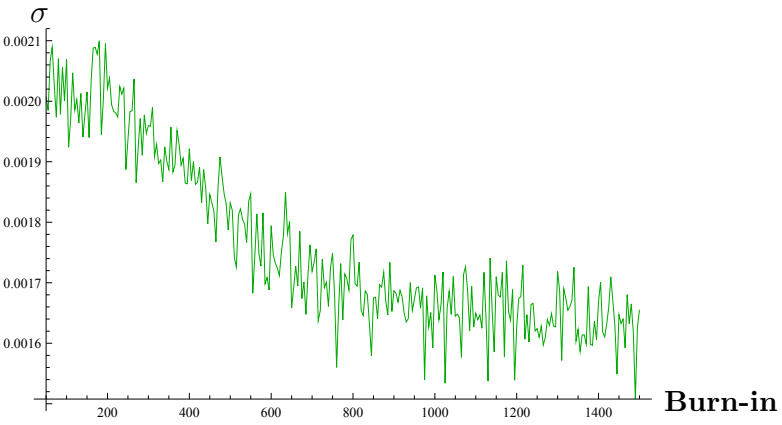



Figure 5.8: $\sigma$ vs Burn-in Length.

ing and the repeated discarding of initial draws. We are just seeking a point where the advantages of burn-in, namely the increase in convergence speed of the subsequent full MCMC algorithm [†], are outweighed by the inefficiency of throwing away close-enough useful draws. A good indicator of where this happens is to look for an inflection or threshold point in the intra-run standard error vs burn-in. This indicates that a kind of minimum distance has been crossed[‡] such that we are now drawing more closely from the invariant distribution. Thus in Fig. 5.8 we plot $\sigma$ versus the burn-in length. We can clearly see a kind of levelling off of the improvement in error as burn-in is increased. With this plot we have confidence to choose our general burn-in value to be 1000, which is only a small fraction of the typical number of samples used ($10^{25}$).

### Expected Flip/Step-size and Acceptance Ratio

As we vary the number of expected flips in generating a candidate state, we are in effect varying the step-size away from the current state. As discussed previously, this parameter needs to be tuned. The measure to optimize is the acceptance ratio. The

[†]because less points need to be taken to average away the initial atypical draws.

[‡]It would be interesting to study this minimum distance's dependence on the initial point (since in the full MCMC algorithm we randomize this). It would also be more rigorous to study the burn-in's obvious dependence on chain-length. To obviate both concerns, we just set burn-in to the highest required to conservatively reach the required distribution for the longest chain length under study.

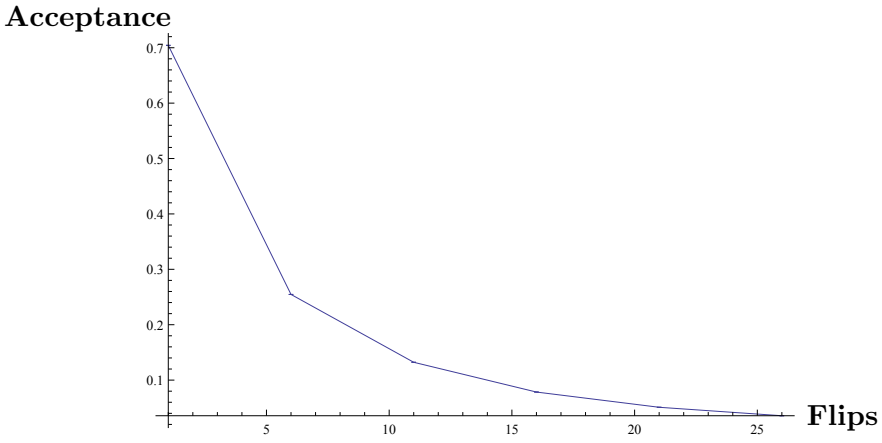



Figure 5.9: Acceptance Ratio vs Expected Flip.

heuristically preferred value of the acceptance ratio is between $40\% - 50\%$ [8].

In Fig. 5.9, we plot the acceptance ratio versus the expected number of flips. As expected, the acceptance ratio drops as the step size away from the current state increases.We chose an expected number of flips of 2 and independently monitored the acceptance ratio throughout all other runs in the rest of the study. We noticed that the acceptance ratios stayed satisfactorily steady and close to the preferred heuristic.

### Skipping/Thinning and Correlations

One method of dealing with correlations is just to skip a few draws before using the next draw, called *thinning*. Thinning is a quick-and-dirty method compared with more complicated methods such as *blocking*. The number of correlated draws to skip is a parameter of the MCMC algorithm that needs to be tuned [6].

We could plot the correlation as a function of shifted time and extract from the data the correlation time. In this case we would then set the skip parameter to equal the correlation time. However we prefer to vary the skipping parameter and take actual data after skipping has been applied. We then observe when the correlation time drops to one and use this value of the skipping parameter. This method has the advantage of being sure that the correlation is satisfactorily low and the code to produce it is

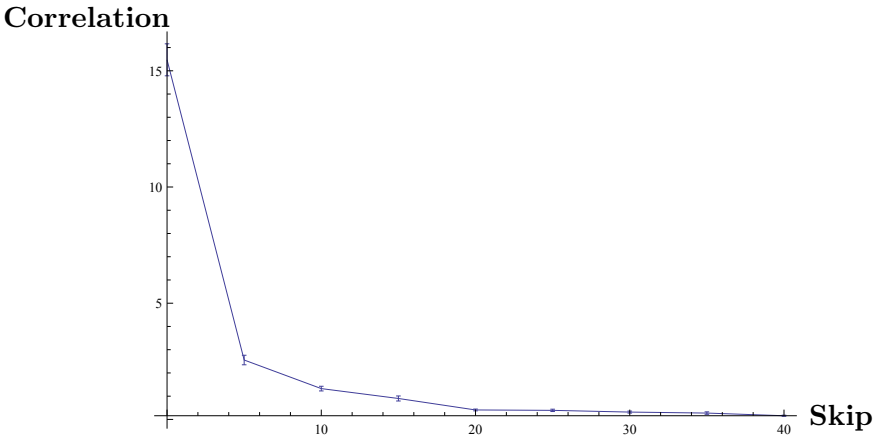



Figure 5.10: Correlation time vs # Skip.

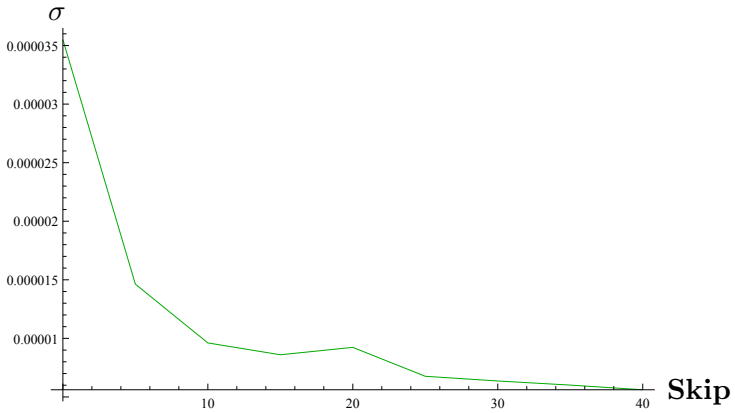



Figure 5.11: $\sigma$ vs # Skip.

used throughout the study to monitor the correlation levels. The computational method of calculating the correlation time that we use, is called the running sums integrated correlation time method [9], which has the advantage that there is no curve fitting to extract the exponential factor.

In Fig. 5.10, we plot the correlation time versus the skip parameter. We see an exponential drop off, which is related to but not the same thing as the exponential drop in the correlation as a function of shift. We see that the correlation reaches 1 at around a skip value of 14, which justifies uses of skip values equal to or greater than this.

Finally, to see that indeed skipping improves the performance of the algorithm we plot, in Fig. 5.11 the intra-run error in the entropy versus the skip parameter and observe a general decrease in error.

## 5.5 Conclusion

We have constructed a simple forgetful noise-memory quantum channel and implemented the MCMC algorithm in parallel, to manage the exponential intractability.

We studied the effects that the noise correlations had on the classical capacity and discovered that the capacity increases with stronger correlations. This is sensible because the correlations can be used to combat the noise when coding information.

The capacity varied smoothly as the correlation parameter was varied. When there was no correlation, the uncorrelated switching is equivalent to a single depolarizing channel with a flip error that is the average of the sub-channels' flip errors. The capacity is thus that of the equally mixed or averaged sub-channels. When the correlation parameter is close to the maximum, the capacity of the deterministic switching channel is the average of the capacity of the sub-channels treated separately.

We learnt that in order to implement the MCMC algorithm, entropy had to be rewritten as an expectation over a probability distribution with an unavoidable, exponentially-many terms. Furthermore, the probability of a single path, had to be carefully rewritten into an expression with polynomially-in-$N$ many terms to avoid the exponentially many terms in the naive writing.

In the use of the algorithm we discovered that the inter-run variance is actually misleading because of correlations, even after attempting to correct for the correlation time. A much better metric of variance turned out to be the intra-run variance because of the randomization of the starting point. We also found a problem-specific and effective method of determining when sufficient burn-in had been undertaken, namely we looked for convergence in the statistic under study as burn-in was varied.

Further work concerning the MCMC algorithm includes a comparison of the general

MCMC method used with the more specialised Gibbs Sampler Method. On a mathematical note, more exploration could be spent on understanding the complicated structure of the fractal-like probability distribution. On the quantum information channel-capacity front, the analysis and code can be applied to more complicated channels, including studying the capacity on channels whose correlations are affected by the state as well as previous noise.

# Bibliography


[1] *The Capacity of the Quantum Channel with General Signal States*, A.S. Holevo (IEEE Trans. Inform. Theory 44, 269; 1998); *Sending classical information via noisy quantum channels*, B. Schumacher and M.D. Westmoreland (Phys. Rev. A 56, 131; 1997)

[2] *Quantum Channels with Memory*, D. Kretschmann and R.F. Werner, Phys. Rev. A, 72(6):62323, 2005 (arXiv:quant-ph/0502106)

[3] *The Classical Capacity of a Qubit Depolarizing Channel with Memory*, J. Wouters, I. Akhalwaya, M. Fannes, F. Petruccione (Phys. Rev. A 79, 1, 2009)

[4] *Equation of State Calculations by Fast Computing Machines*, N. Metropolis, et al. (J. Chem. Phys. 21, 1087; 1953)

[5] *Monte Carlo Sampling Methods Using Markov Chains and Their Applications*, W.K. Hastings (Biometrika Vol. 57, No. 1 (Apr., 1970) pp. 97-109)

[6] *Handbook of Markov Chain Monte Carlo*, S. Brooks, A. Gelman, G. Jones, X. Meng (Taylor & Francis US, 2011)

[7] *Elements of Information Theory 2nd Edition* T. Cover and J. Thomas, (Wiley-Interscience, 2006)

[8] *Understanding the Metropolis-Hastings Algorithm*, S. Chib and E. Greenberg, (The American Statistician, Vol. 49, No. 4 (Nov., 1995), pp. 327-335)

[9] *Lecture Notes*, R.J. Gonsalves (http://www.physics.buffalo.edu/phy410-505/topic7/index.html retrieved: 30/10/2013)‘

# Chapter 6

# Atomic Clock Background

## 6.1 Introduction

The science of time-keeping has a fascinating history that weaves itself into the very story of human civilization and progress. The latest chapter has seen dramatic improvement in precision time-keeping using the marvels of quantum mechanics at the atomic scale. Time is now kept with more precision than almost any other man-made measurement and this precision is being used technologically to communicate faster and locate better than ever before.

This part of the thesis hopes to study noise in the atomic clock setting. An attempt is made to use correlations inherent in the noise to combat the noise.

Executing periodic corrections to a noise signal has the surprise effect of mis-correcting noise that happens to be oscillating at the same periodicity. This is called the Dick Effect and is a special case of the aliasing effect of sampling where high frequency noise appears as low frequency noise.

In order to build up to suggesting other techniques, we begin, in this background chapter, by trying to understand sampling and reconstruction via the Fourier transform. We then study methods of quantifying noise. In these methods, we find a convenient mathematical tool for describing the correction process. Finally, we introduce the basic concepts of an atomic clock, including the underlying quantum physics and the protocol of correcting a slaved quartz crystal clock.

## 6.2 Sampling and Reconstruction

### 6.2.1 Fourier Analysis

Fourier analysis is one the most powerful mathematical tools for analysing periodic phenomenon. It involves the expansion of a continuous function, $g(t)$, in the bases of oscillating functions. The overlap of $g(t)$ with a particular basis function is called the Fourier transform,

$$G(f) = \mathcal{F}(g)(f) \equiv \int_{-\infty}^{\infty} g(t)e^{2\pi itf}\, dt.$$

For a given $f_0$, $G(f_0)$ is the coefficient of $e^{-2\pi itf_0}$ in the expansion of $g(t)$, which is called the inverse Fourier transform,

$$g(t) = \int_{-\infty}^{\infty} \mathcal{F}(g)(f)e^{-2\pi itf}\, df.$$

$G(f)$ as a function, has $f$ vary over a domain called the Fourier domain, which is the space of frequencies of the basis functions. Hence, Fourier analysis sheds light on the rate of variations, or the frequency content, of the time domain.

Not only does the Fourier transform rewrite continuous functions and in so doing make explicit periodicity information, it also makes certain operations in the time domain very convenient in the Fourier domain. One example, that we make extensive use of, is the Fourier transform of the convolution.

The convolution is a sweeping overlap integral of one function, $g_1(t)$ by another, $g_2(t)$, in the time domain and is defined as

$$(g_1 * g_2)(t) \equiv \int_{-\infty}^{\infty} g_1(t - t')g_2(t')\, dt'.$$

In the Fourier domain, convolution is transformed into simple multiplication,

$$\mathcal{F}(g_1 * g_2)(f) = G_1(f)G_2(f).$$

### 6.2.2 Using the Fourier Transform

Our main use of the Fourier transform is to analyse the effects of *sampling* a continuous function.

The following succinct formula summarizes the periodic sampling of a continuous function $g(t)$ at period $T$ (that is, sampling at a frequency of $f = 1/T$ or angular frequency, $\omega = 2\pi/T$) and then attempting to reconstruct it, $g_r(t)$, by summing up weighted $\text{sinc}_T$ functions.

$$g_r(t) = (g(t) \times \text{Ш}_T(t)) * \text{sinc}_T(t). \tag{6.1}$$

The *Shah function/Dirac comb*, $\text{Ш}_T$, is a sum of Dirac delta's shifted by multiples of $T$,

$$\text{Ш}_T(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT).$$

The *sinc* function, $\text{sinc}_T(t)$, is an oscillating sin function (of period $2T$) with diminishing amplitude, defined as

$$\text{sinc}_T(t) = \frac{\sin(\pi t/T)}{\pi t/T}.$$

The purpose of the sinc function is firstly to reproduce the sample points unchanged and secondly to smoothly interpolate between them. This continuous function is symmetrically centred on zero with a large central peak, $\text{sinc}_T(0) = 1$ and shortening *side-lobes*. The zeroes of $\text{sinc}_T$ occur at integer multiples of $T$.

One way to view the reconstruction convolution (written as $*$ above), is as a sum of $g(t_s)$-weighted $\text{sinc}_T$ functions centred on the $t_s$-coordinate of the sample points. For a specific sample point, all the other $nT$-shifted sinc functions have one of their zeroes at that sample point and only the centred sinc contributes the perfectly weighted sampled amount. Thus guaranteeing that $g_r(nT) = g(nT)$, even before we look at whether or not $g(t)$ is successfully reconstructed between points.

Taking the Fourier transform of Eq. (6.1) yields,

$$\mathcal{F}(g_r)(f) = (G(f) * \tfrac{1}{T}\text{Ш}_{1/T}(f)) \times T\Pi_{1/T}(f). \tag{6.2}$$

Multiplication has been transformed into convolution and convolution into multiplication. The Fourier transform of the Dirac comb, in the time domain, is a Dirac comb, in the frequency domain. However, in the time domain, the deltas were placed $T$ apart, that is occur with frequency $f = 1/T$. Now, in the frequency domain, the deltas appear

$1/T$ apart*, which makes sense because the Fourier transform is an expression of the frequencies present in the time domain.

The Fourier transform of the $\text{sinc}_T$ function is a scaled rectangular function, $T\Pi_{1/T}$, centred on zero with total width $1/T$,

$$\Pi_P(x) = \begin{cases} 1 & |x| \leq P/2 \\ 0 & |x| > P/2 \end{cases}.$$

The Fourier transform, shows that convolution with $\text{sinc}_T$ in the time domain is nothing other than a rectangular low pass filter in the frequency domain (keeping low frequencies and setting to zero high absolute frequencies above a cut-off).

We are now ready, not only to state the Sampling theorem, but to prove it.

**Sampling Theorem. 1** *$g_r(t) = g(t)$ if $G(f) = 0$ when $|f| \geq B \equiv 1/2T$.*

PROOF: From Eq. (6.2),

$$\begin{aligned} \mathcal{F}(g_r)(f) &= (G(f) * \text{Ш}_{1/T}(f)) \times \Pi_{1/T}(f) \\ &= \sum_{n=-\infty}^{\infty} G\left(f + \frac{n}{T}\right) \times \Pi_{1/T}(f) \\ &= \sum_{n=-\infty}^{\infty} G\left(f + n2B\right) \times \Pi_{2B}(f) \end{aligned}$$

Since $G(f) = 0$ for $|f| \geq B$, the shifted $G(f)$'s do not overlap in the (*Poisson*) summation and thus the first unshifted term ($n = 0$), is not added to, on the interval $(-B, B)$, by any other terms. Thus the rectangular function, $\Pi_{2B}$, which is 1 on $[-B, B]$ and 0 outside the interval truncates/filters this infinite sum to give only the $n = 0$ term.

$$\mathcal{F}(g_r)(f) = G(f) \times \Pi_{2B}(f) = G(f)$$

Since the Fourier transforms are identical, by inversion† we conclude that the functions are identical **Q.E.D.**

*The frequency of occurrence in the Fourier domain is $T$. Thus the Fourier transform has 'swopped' the roles of period and frequency.

†The uniqueness of the Inverse Fourier Transform, of course, comes with conditions.

## 6.3 Characterizing Precise Oscillators: Time Domain

To quantify the error in the frequency of a highly precise oscillator, we need to make assumptions about the noise and introduce a measure of deviation from perfect oscillation. We follow Rutman [1] in introducing and using the *Allan variance*, a time domain quantity, for this purpose.

## 6.4 Modelling and Randomness

A perfect simple oscillator would produce the following output:

$$g(t) = A\sin(2\pi\nu_0 t).$$

Taking this as our starting point and only slightly modifying it by introducing random phase noise, $\phi(t)$, our simple noise model is,

$$g(t) = A\sin\left(2\pi\nu_0 t + \phi(t)\right).$$

It is sufficient to only introduce a general phase noise term, because instantaneous frequency noise can be written in terms of $\phi$,

$$\nu(t) = \frac{1}{2\pi}\frac{d}{dt}(2\pi\nu_0 t + \phi(t)) = \nu_0 + \frac{1}{2\pi}\frac{d}{dt}\phi(t).$$

Focusing on frequency noise, we introduce a new variable $\Delta\nu$ related by $\Delta\nu = \frac{1}{2\pi}\frac{d}{dt}\phi(t)$ so that

$$\nu = \nu_0 + \Delta\nu.$$

We are now able to model $\Delta\nu$ as we wish, and only later if desired reconnect it to the phase noise via integration. In fact, it is this relationship that brings us to another important question of modelling randomness. One vital and simplifying assumption when dealing with random processes is *stationarity*. So where should stationarity be, in a sense, artificially introduced? Should it be introduced at the level of phase noise or instantaneous frequency noise? It is their inter-relationship that forces us to remember that sometimes we can't have both and that the manner in which we set-up our models

has an influence on how we make sense of the world. Of course, the ultimate test is success in predicting and manipulating the real world, but we must remember that there is a degree of arbitrariness.

From empirical usage, we prefer to treat the frequency error, $\Delta\nu$, as a random stationary process, such that for each $t$, $\Delta\nu(t)$ is a random variable whose joint distribution across different times depends only on the time difference, $\Delta t$.

Lastly, to aid comparisons, it is useful to define the fractional/relative (or normalized) frequency noise as,

$$y(t) \equiv \frac{\Delta\nu}{\nu_0}.$$

It is this quantity that is treated as our fundamental random variable and becomes the basis of the Allan variance [1].

### 6.4.1 Statistics

Now that we have decided to focus on $y(t)$, let us discuss how to quantify the frequency noise of our model oscillator and how to collect statistics from real oscillators.

#### Interval Averaging

We begin by looking at the average frequency over some interval $(t_k, t_k + \tau)$. The averaging process begins at time $t_k$ and lasts $\tau$ long. Experimentally this is implemented by counting the number of oscillations within the interval and dividing by the length of the interval. Hence in terms of the model,

$$\overline{\nu(t)}_{t_k,\tau} = \nu_0 + \frac{1}{\tau}\int_{t_k}^{t_k+\tau} \Delta\nu(t)\, dt.$$

We can write the average frequency in terms of the fractional frequency noise.

$$\overline{\nu(t)}_{t_k,\tau} = \nu_0 \left(1 + \frac{1}{\tau}\int_{t_k}^{t_k+\tau} \frac{\Delta\nu(t)}{\nu_0}\, dt\right) \tag{6.3}$$

$$= \nu_0 \left(1 + \frac{1}{\tau}\int_{t_k}^{t_k+\tau} y(t)\, dt\right) \tag{6.4}$$

$$= \nu_0 \left(1 + \overline{y}_{t_k,\tau}\right), \tag{6.5}$$

where we have introduced $\overline{y}_{t_k,\tau} = \frac{1}{\tau}\int_{t_k}^{t_k+\tau} y(t)\,dt$, the average fractional frequency noise over the interval.

### Ensemble Averaging and Variance

With $y(t)$ as our random variable we may impose assumptions, our first being a zero mean[‡], for each $t$,

$$\langle y(t)\rangle = 0.$$

Here the average is an ensemble average, over infinitely many runs of the experiment. The $\tau$ averaging of the random variable $y(t)$ still leaves $\overline{y}_{t_k,\tau}$ a random variable. But the zero mean carries through, regardless of the interval length $\tau$, because of the ensemble average,

$$\left\langle \overline{y}_{t_k,\tau} \right\rangle = 0.$$

With the mean of the noise eliminated, we next introduce the variance of the noise. It is the variance that turns out to be our main window into the workings of noise. Indeed from signal processing theory, the variance of a signal is very important and is called by analogy to many physical situations, the expected *power* of the signal.
Thus, the variance of $\overline{y}_{t_k,\tau}$ is

$$\sigma^2[\overline{y}_{t_k,\tau}] = \left\langle \overline{y}^2_{t_k,\tau} \right\rangle,$$

where we have used the zero mean to simplify the standard definition of the variance. For a given $t_k$ and $\tau$ and having taken the ensemble average, the variance is a single number. It is the true mean of the square of the average of the fractional frequency noise over $(t_k, t_k + \tau)$.
For a stationary process, as we are assuming, $\sigma^2[\overline{y}_{t_k,\tau}]$ is actually independent of the starting time $t_k$. Hence the convention to define a new function as follows,

$$I^2(\tau) \equiv \sigma^2[\overline{y}_{t_0,\tau}].$$

[‡]In general, if there is some simple systematic drift, it could still be handled by first removing the drift before analysis.

Taking the square-root yields

$$I(\tau) = \sqrt{\sigma^2[\overline{y}_{t_0,\tau}]},$$

which, in words, is the true root mean square of the $\tau$-averaged fractional noise.
If we take the limit $\tau \to 0$, we get $I(\tau) \to \sqrt{\langle y \rangle}$, the instantaneous root mean square of $y(t)$. If we take the other extreme $\tau \to \infty$, we get $I(\tau) \to 0$, which is just another way of saying (assuming ergodicity) that the $y(t)$ has zero mean. That is the $\tau$-averaging is over a long enough interval that the noise fluctuations wash away to zero before being squared and ensemble averaged.

### Allan Variance

In trying to measure $I(\tau)$ for real world oscillators we have to contend with finite sample sizes. Since variance estimators can be biased or unbiased[§] and the variance of the variance can behave differently for different sample sizes; and since it is important to be able to compare different frequency sources, a convention needs to be established.
We have already assumed stationarity and if we add the ergodicity assumption, then we are able to take $\tau$-long averages, one after another, without "restarting" the experiment to $t_0$[#]. During one $\tau$-long run we are feverishly counting the number of oscillations after which we produce only one sample of $\overline{y}_{t_k,\tau}$, so we need to specify how many $\tau$ runs, the gap between runs and the manner of combining them to arrive at a variance estimate.
The IEEE [6] has recommended the following variance estimator:

$$\widehat{\sigma}_y^2(\tau) \equiv \frac{1}{2}(\overline{y}_{t_1,\tau} - \overline{y}_{t_0,\tau})^2, \quad t_1 = t_0 + \tau.$$

Notice that the second interval starts immediately after the first interval with zero, so called, dead time. This estimator is itself a random variable, and its expectation is what completes our quest for a time-domain measure of frequency stability, called the Allan

[§]Sometimes, whether an estimator is biased even depends on the underlying noise distribution.

[#]Restarting is in some sense not even theoretically possible because it would be restarting the universe to the same conditions. Then again, the scientific method presupposes the ability to prepare an identical closed system afresh. However, when we are dealing with noise we are in general assuming that we are working with an open system where we have no control over the environment. The environment cannot be reset, so we rather assume ergodicity and see how far it takes us.

variance,

$$\sigma_y^2(\tau) \equiv \left\langle \widehat{\sigma}_y^2(\tau) \right\rangle .$$

Notice that in terms of notation[‖], the expectation of the estimator "removes" the hat and tells us something about the underlying variance. Indeed, this estimator was chosen so that its expectation *is* unbiased, with respect to white noise,

$$\sigma_y^2(\tau)_{\text{white}} = I^2(\tau).$$

For other types of noise the Allan variance is in general biased,

$$\sigma_y^2(\tau) = 2(I^2(\tau) - I^2(2\tau)) \neq I^2(\tau).$$

Ironically, this bias is actually helpful for some noise models. For example, it is useful when the true variance $I^2$ is unbounded and yet the Allan variance (with the infinities 'cancelling') is finite! In this way the Allan variance can still be used to help identify and characterize these cases.

Ultimately, because we can't take an infinite expectation in practice, we are forced to approximate, as best as possible, the ensemble average in the Allan variance. So even though one estimate calls for only two $\tau$-averaged runs, we still require many estimates in order that we may build up enough statistics to accurately approximate the ensemble average,

$$\widehat{\sigma_y^2(\tau)}(m) = \widehat{\left\langle \widehat{\sigma}_y^2(\tau) \right\rangle}(m) = \frac{1}{2(m-1)} \sum_{i=0}^{m-2} (\overline{y}_{t_{i+1},\tau} - \overline{y}_{t_i,\tau})^2, \quad t_{i+1} = t_i + \tau.$$

The $m-2$ term is there, because we are counting from zero, and the $m-1$ term appears because we can only get $m-1$ estimates of the Allan variance from $m$ $\tau$-long runs[**]. This mean is *also* its own separate estimator (and a random variable) whose (empirical) variance may be used to plot the error bars in our estimate of the Allan variance. Thus

[‖] The bar over the $y$ is absent from this widely used notation. We use it to remind us of the $\tau$-averaging of $y$. The bar is left out, since, with $\tau$ being specified, the averaging process is implied anyway and our bar notation is redundant (though still a useful visual aid).

[**] It has nothing to do with making the mean unbiased as in the variance case. The natural definition of the mean is automatically unbiased.

at this level, we are calculating a realization of an estimator of the Allan variance (which is the true/full expectation of a two-sample estimator of the variance of the $\tau$-averaged fractional frequency noise).

## 6.5 Characterizing Precise Oscillators: Frequency Domain

Stationary random processes are widely used in modelling a diverse set of phenomena. The theory is well-developed and the tools are powerful. One such tool is the spectral density of a random process. The spectral density is an abstract concept that involves the Fourier transform and hence occupies the frequency domain.

We are using a stationary random process to model the noise fluctuations of atomic clocks. Thus the question of frequencies are already paramount and the tools of the Fourier domain are particularly appropriate and bring much to our understanding of noise. Ultimately, of course, we must connect the time domain measures introduced above to the frequency domain concepts introduced below.

### 6.5.1 Autocorrelation

First we introduce the autocorrelation of a stationary random process, $y(t)$,

$$R_y(\tau) \equiv \langle y(t_0)y(t_0 - \tau)\rangle \,.$$

This function's domain is still time, but now it has the interpretation of a time delay/lag/shift. The autocorrelation is measuring the ensemble averaged correlation of the same process at different times. The autocorrelation at zero lag is the instantaneous variance of the random variable at $t_0$, which is the first crucial part of the connection to the time domain measure of noise. This indeed is one of the main reasons we are able to use the autocorrelation,

$$R_y(0) = \left\langle y^2 \right\rangle .$$

If we normalize by this variance, which is sometimes part of the definition, we get a normalized measure of correlation such that the self-correlation at zero lag is 100%, in other words complete correlation.

If the random process is stationary, as we are assuming, then there is no dependence of the autocorrelation on $t_0$. If the process is ergodic, as we also assume, then the ensemble average may be replaced by the time average.

Finally, there are two very important properties of the autocorrelation that are relevant to our study,

$$R_y(0) \geq |R_y(\tau)|.$$

This is a general property of the autocorrelation and it applies for random or deterministic processes. For random processes this property is a key aspect of exploring how long, if at all, it takes for correlations to disappear. The second property that concerns us, is the fact that the autocorrelation of a periodic function is itself a periodic function with the same period. This property enables the exploration of the periodicities in the original function via the autocorrelation. This becomes important when the Fourier transform of the original function cannot be taken because the original function is not for example square integrable.

### 6.5.2 Power Spectral Density

We have seen in the time domain section how the square of $y(t)$ plays a crucial role in studying noise. Indeed from signal processing theory, the square of a signal $y(t)$, is called the instantaneous power $P(t)$,

$$P(t) = y^2(t).$$

Signals that continue indefinitely but have finite variance are called power signals[††]. This name comes from the analogy with the definition of power in electric circuits ($P = I^2/R$) or irradiance of the electric field ($P \propto |E|^2$).

The expectation of the instantaneous power is the ensemble averaged power, which for stationary processes, is independent of time,

$$P = \langle y^2 \rangle .$$

[††]The total energy of power signals, which is the integral of the power with respect to all time, is infinite. Signals with finite total energy are transient and are called energy signals.

We saw above how the power is equal to the autocorrelation at lag 0, by definition,

$$P = \langle y^2 \rangle = R_y(0).$$

The next step in extracting information from the autocorrelation function is to use the Wiener-Khinchin theorem to write the autocorrelation as a inverse Fourier-like transform,

$$R_y(\tau) = \int_{-\infty}^{\infty} S_y(f) e^{i2\pi f\tau}\, df,$$

where $S_y(f)$ is called the Power Spectral Density. We say Fourier-like because the Wiener-Khinchin theorem carries through even when the Fourier transform of the autocorrelation function does not exist.

It is called the Power Spectral Density because for $\tau = 0$,

$$P = \langle y^2 \rangle = R_y(0) = \int_{-\infty}^{\infty} S_y(f)\, df.$$

This writes the average power, $P$, as an integral over frequencies, such that $S_y(f)df$ is the infinitesimal power[‡‡] in the frequency interval $(f, f+df)$. That is, $S_y(f)$ is the power density with respect to frequency[§§].

If it would be possible to take the Fourier transform of the autocorrelation function (for example with extra convergence constraints, such as $R_y(\tau) \to 0$ exponentially fast) then,

$$S_y(f) = \mathcal{F}(R_y)(f) = \int_{-\infty}^{\infty} R_y(\tau) e^{-2\pi i f\tau}\, d\tau.$$

Finally, the connection to the time domain statistics can now be stated as

$$\lim_{\tau \to 0} I^2(\tau) = \langle y^2 \rangle = \int_{-\infty}^{\infty} S_y(f)\, df.$$

### 6.5.3 $I^2$ Power Spectral Density

When making the connection between the time domain measure of noise $I^2$ and the frequency domain, power spectral density, we had to take the limit as $\tau \to 0$ of $I^2(\tau)$. In

[‡‡] This is made possible by $S_y(f) \geq 0 \forall f$. Another useful property is that $S_y(f) = S_y(-f)$, so that, as is often done, we may focus on positive $f$ only, by defining the one-sided density.

[§§] With spectrum meaning range of frequencies, as used by Newton when referring to the 'ghostly apparition' (*spectre*) of colours splitting from white light through a prism [2].

practice this is not possible since there is a minimum non-zero $\tau$ that can not practically be made smaller. It would be helpful if the power spectral density would take the $\tau$ averaging into account. Indeed, it can as we now describe.

We start with the variance, or power in the $\tau_a$-averaged noise signal,

$$I^2(\tau_a) = \left\langle \overline{y}^2_{t_k,\tau_a} \right\rangle .$$

Here we have replaced the original $\tau$ with $\tau_a$ ($a$ for 'average' over) to distinguish it from autocorrelation $\tau$, which we now label $\tau_s$ ($s$ for 'shift').

We follow the same power spectral density construction for $\left\langle \overline{y}^2_{t_k,\tau_a} \right\rangle$ as we did for the ensemble averaged instantaneous power, $\left\langle y^2 \right\rangle$.

We first define the autocorrelation function,

$$R_{\overline{y}_{\tau_a}}(\tau_s) \equiv \left\langle \overline{y}_{t_0,\tau_a} \overline{y}_{t_0-\tau_s,\tau_a} \right\rangle .$$

We then define the power spectral density (in terms of the Fourier transform, but if necessary by the Wiener-Khinchin theorem we do not need the Fourier transform),

$$S_{\overline{y}_{\tau_a}}(f) \equiv \mathcal{F}(R_{\overline{y}_{\tau_a}})(f).$$

We write the autocorrelation as the inverse Fourier(-like) transform,

$$R_{\overline{y}_{\tau_a}}(\tau_s) = \int_{-\infty}^{\infty} S_{\overline{y}_{\tau_a}}(f) e^{i2\pi f \tau_s} \, df.$$

We make the connection to $I^2$ by evaluating the autocorrelation at $\tau_s = 0$; giving us the power spectral density of the ensemble average of the $\tau_a$-averaged noise,

$$I^2(\tau_a) = \left\langle \overline{y}^2_{t_0,\tau_a} \right\rangle = R_{\overline{y}_{\tau_a}}(0) = \int_{-\infty}^{\infty} S_{\overline{y}_{\tau_a}}(f) \, df.$$

Now, since $y(t)$ is our fundamental building block, about which we make basic assumptions and build our noise model around, we should connect $S_{\overline{y}_{\tau_a}}(f)$ to $S_y(f)$.

This is done by explicitly writing out $\overline{y}_{\tau_a}$ in terms of $y$ and a moving[##] averaging window, $h_{\tau_a}$, and then following the detailed calculations of taking the Fourier transform of the autocorrelation function.

[##] The window is a 'continuously moving' rectangular function. This may seem incompatible with adjacent window averaging. It is however not a problem. We could, if we wished to model the process

To take advantage of some of the properties of the Fourier transform, we rewrite the process of averaging as a convolution. In our case, the $\tau_a$-averaging process is convolution with a $1/\tau_a$-weighted and shifted[‖‖] rectangular window, $h_{\tau_a}(t)$, of length $\tau_a$,

$$h_{\tau_a}(t) = \tfrac{1}{\tau_a}\Pi_{\tau_a}(t + \tfrac{\tau_a}{2}).$$

The Fourier transform of the window is proportional to the $\mathrm{sinc}_{1/\tau_a}$ function,

$$|H_{\tau_a}(f)| = |\mathcal{F}(h_{\tau_a})(f)| = |\mathrm{sinc}_{1/\tau_a}(f)|.$$

The window, $h_{\tau_a}$, is called the *impulse response*[***] and its Fourier transform, $H_{\tau_a}$ is called the *transfer function*[†††].

Using $h_{\tau_a}$ we have,

$$\begin{aligned}\overline{y}_{t_0,\tau} &= \frac{1}{\tau}\int_{t_0}^{t_0+\tau} y(t)\,dt \\ &= \int_{-\infty}^{\infty} h_{\tau_a}(t_0 - t)y(t)\,dt \\ &= (h_{\tau_a} * y)(t_0).\end{aligned}$$

of using only adjacent windows, still first take the continuous moving average convolution, and then sample the result by multiplying with a Dirac Comb. Here it is not necessary, because we are only using the autocorrelation of the moving average at zero lag and then ensemble averaging by collecting statistics. That is, for now we are not interested in how neighbouring windows correlate, we are rather interested in how one ensemble averaged window's noise is distributed. We do use the valid, but practically inaccessible, theory of true continuous moving averages and its autocorrelation to introduce the idea of the power spectral density. Later on in the thesis we actually do make use of sampling the moving average to gain insight into aliasing.

[‖‖] The flip and shift to the beginning ($t = t_0$) of the noise integration is taken care of by the convolution. The $\frac{\tau_a}{2}$ shift to the left concerns which $t$-value is assigned the value of the integration. Normally the convolution with the rectangular function assigns the integral's value to the center ($t_0 + \tau_a/2$) of the rectangular function. In the Allan variance, it has been arbitrarily chosen to be the beginning, $t_0$, which is not a problem, a mere exponential factor, $e^{2\pi i f \tau_a/2}$, does the translation in the frequency domain. The convolution is 'a-causal' in either case because future $y(t)$ values are used in their pasts. Later in the thesis we insist on causal windows.

[***] The reason for the name is that, if the input were an *impulse* (the Dirac Delta), the output, after convolution with the window, is the window itself.

[†††] The reason for this name is that, in the frequency domain, the Fourier transform of the window called the transfer function, 'takes' or '*transfers*' the input to the output by simple multiplication (convolution has been transformed into multiplication).

Now we can write out the previously defined autocorrelation of the noise in full,

$$\begin{aligned}
R_{\overline{y}_{\tau_a}}(\tau_s) &\equiv \left\langle \overline{y}_{t_0,\tau_a} \overline{y}_{t_0-\tau_s,\tau_a} \right\rangle \\
&= \left\langle \int_{-\infty}^{\infty} h_{\tau_a}(t_0 - t_1) y(t_1)\, dt_1 \int_{-\infty}^{\infty} h_{\tau_a}(t_0 - \tau_s - t_2) y(t_2)\, dt_2 \right\rangle \\
&= \left\langle \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h_{\tau_a}(w) h_{\tau_a}(z) y(t_0 - w) y(t_0 - \tau_s - z)\, dw\, dz \right\rangle \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h_{\tau_a}(w) h_{\tau_a}(z) \left\langle y(t_0 - w) y(t_0 - \tau_s - z) \right\rangle dw\, dz \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h_{\tau_a}(w) h_{\tau_a}(z) R_y(\tau_s + z - w)\, dw\, dz\,,
\end{aligned}$$

where we have introduced new variables ($w = t_0 - t_1$ and $z = t_0 - \tau_s - t_2$) and changed the variables of integration. By linearity of the integration we took the expectation in. Furthermore by stationarity of the random process we rewrote the product of the $y$'s as an autocorrelation at the corresponding lag.

We are now ready to take the Fourier transform of the autocorrelation with respect to $\tau_s$, as in the definition. We also introduce another new variable, to record the lag ($x = \tau_s + z - w = t_1 - t_2$), and change the relevant variable of integration,

$$\begin{aligned}
S_{\overline{y}_{\tau_a}}(f) &\equiv \mathcal{F}(R_{\overline{y}_{\tau_a}})(f) \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h_{\tau_a}(w) h_{\tau_a}(z) R_y(\tau_s + z - w) e^{-2\pi i f \tau_s}\, dw\, dz\, d\tau_s \\
&= \iiint h_{\tau_a}(w) h_{\tau_a}(z) R_y(x) e^{-2\pi i f (x - z + w)}\, dw\, dz\, dx \\
&= \int h_{\tau_a}(w) e^{-2\pi i f w}\, dw \int h_{\tau_a}(z) e^{2\pi i f z}\, dz \int R_y(x) e^{-2\pi i f x}\, dx \\
&= |H_{\tau_a}(f)|^2 S_y(f) \\
&= \operatorname{sinc}^2_{1/\tau_a}(f) S_y(f) \\
&= \tfrac{\sin^2(\pi \tau_a f)}{(\pi \tau_a f)^2} S_y(f).
\end{aligned}$$

Thus we have related the averaged noise's power spectral density directly to the noise's power spectral density. The effect of the averaging window is to attenuate the true noise's high frequency components, acting as an oscillating low pass filter.

### 6.5.4 Allan Variance Power Spectral Density

Since we concluded the time domain section by stating that the Allan variance is the recommended measure of noise, this power spectral density section would not be complete without deriving the Allan variance's power spectral density. The derivation is exactly the same as above except with a different window, $h_{A_{\tau_a}}$,

$$h_{A_{\tau_a}}(t) = -\tfrac{1}{\sqrt{2}} h_\tau(t) + \tfrac{1}{\sqrt{2}} h_\tau(t - \tau_a).$$

The window is simply the sum of two scaled adjacent windows (the dead time being zero) with the first window being multiplied by negative one to capture the difference of the two $\tau_a$-averages in the two-sample variance estimator.
The absolute value of the Fourier transform of this window is

$$|H_{A_{\tau_a}}(f)| = \frac{\sqrt{2}\sin^2(\pi\tau_a f)}{\pi\tau_a f}.$$

Therefore the Allan variance's power spectral density is

$$\begin{aligned}\sigma_y^2(\tau_a) &= \int_{-\infty}^{\infty} |H_{A_{\tau_a}}(f)|^2 S_y(f)\, df \\ &= \int_{-\infty}^{\infty} \frac{2\sin^4(\pi\tau_a f)}{(\pi\tau_a f)^2} S_y(f)\, df\end{aligned}$$

### 6.5.5 Power Law Noise Models

In the process of covering the Allan variance and the power spectral density we introduced constraints and assumptions on our model of noise[‡‡‡]. We are now ready to introduce a description of the noise that governs its character. Assumptions such as stationarity were mainly introduced for convenience (and so limit the generality of the models in favour of manageability). However the following stipulations of the noise are informed by real world, largely universal, phenomenological behaviour.
The tool we use to specify the noise model that we work with is the power spectral density. Of course, this tool can also be used for analysing real world data. But for

[‡‡‡] In a sense, imposing limits of what can be modelled with these set of tools.

the rest of this study, we assume a certain power spectral density and work towards our results.

In many different systems it empirically seems that natural systems including noise have a *power law* power spectral density. There have been many attempts made at explaining this and for many settings there is no consensus on the final explanation. For us instead of providing a mechanism for the noise we simply impose the power law behaviour as an assumption.

The specification of the spectral density achieves two things with regard to the random character of our fundamental random process $y(t)$. Firstly, it obviously specifies the frequency components of the noise[§§§]. In this regard it is important to note that sometimes cut-offs are needed to remain physically realistic. Curiously, the analysis sometimes carries through even without cut-offs. For example for white noise, no high frequency cut-off is needed for the Allan variance to converge because the windowing process, effectively averages away very high frequency contributions.

Secondly, specifying the spectral density specifies the kinds of correlations in the noise we are considering. Power law spectral densities can model from the case of completely independent noise, called white noise (constant spectral density), through Markovian noise, called Brown Noise ($1/f^2$) towards increasingly correlated noise. The faster the power at higher frequencies drop, the more correlated the noise becomes.

In Table 6.1 we draw up a list of commonly used power law spectral densities. To see the connection between the Allan variance column and the one-sided* $S_y^{(1)}$, power spectral density column, we demonstrate the derivation of the Allan variance from the chosen

[§§§] More precisely, the frequency components of the autocorrelation of the noise, which is of course related directly to the noise. Indeed for deterministic energy signals, the Fourier transforms are defined and we have: $\mathcal{F}(R_y) = |Y(f)|^2$, where $Y(f) = \mathcal{F}(y)$.

* It is just a convenience to work with the one-sided density instead of worrying about identical power levels at frequencies, $f$ and $-f$.

| **Power-law Noise** | $S_y^{(1)}$ $(S_y^{(1)}(f) \equiv 2S_y(f))$ | $S_\phi$ **slope** | **Allan variance** $\sigma_y^2(\tau)$ |
|---|---|---|---|
| White phase ($f < f_H$) | $h_2 f^2$ | $f^0$ | $\left(\frac{3 f_H h_2}{4\pi^2}\right) \tau^{-2}$ |
| Flicker/Pink phase ($f < f_H$) | $h_1 f^1$ | $f^{-1}$ | $\left(\frac{(3(\gamma+\ln(2\pi f_H \tau))-\ln 2) h_1}{4\pi^2}\right) \tau^{-2}$ |
| White frequency (Brown phase) | $h_0 f^0$ | $f^{-2}$ | $\left(\frac{h_0}{2}\right) \tau^{-1}$ |
| Flicker/Pink frequency | $h_{-1} f^{-1}$ | $f^{-3}$ | $(2 \ln 2 h_{-1})\, \tau^0$ |
| Random walk/Brown frequency | $h_{-2} f^{-2}$ | $f^{-4}$ | $\left(\frac{2\pi^2 h_{-2}}{3}\right) \tau^1$ |

Table 6.1: Power law Noise ($h_i$ are constants, $f_H$ are cut-offs, $\gamma$ is the Euler-Mascheroni constant) .

power law noise for the case of White frequency noise,

$$\begin{aligned}
\sigma_y^2(\tau)_{\text{white}} = I^2(\tau) &= \int_{-\infty}^{\infty} \frac{\sin^2(\pi\tau f)}{(\pi\tau f)^2} \frac{h_0}{2}\, df \\
&= \frac{h_0}{2\pi\tau} \int_{-\infty}^{\infty} \frac{\sin^2\theta}{\theta^2}\, d\theta \\
&= \frac{h_0}{2\tau}.
\end{aligned}$$

Finally, it should be noted that in order to generate the power spectral density we took the expectation of the autocorrelation. So that the actual erratic behaviour of a specific realization of $y(t)$ is ensemble-averaged away to arrive at deterministic properties of the random process, such as the autocorrelation and the power spectral density. Thus just because two processes have the same power spectral density does not mean that they themselves are correlated. This information would be in the conditional power spectral density. However, we do not make use of this advanced tool directly and in the case of a process that is written as a deterministic function of another random process, we content ourselves with a comparison of the two processes power spectral densities as a measure of actual correlation because the one is a deterministic function of the other.

## 6.6 Atomic Clock Closed Loop Correction

### 6.6.1 Quantum Physics of Atomic Clocks

In the first half of the Twentieth century, physics underwent a major and surprising upheaval. Atomic beam experiments played a substantial role in bringing to the fore some of the new strange quantum effects, such as the quantization of energy levels and magnetic moments. These same experiments also paved the way for one of the first technological applications of quantum mechanics: the atomic clock.

Three highlights stand out both for their theoretical and practical contributions: the 1921 Stern-Gerlach quantized magnetic moment experiment [3], the 1938 Rabi resonance experiment [4] and the 1950 Ramsey double pulse experiment [5]. By the 1950's a few laboratories around the world, based on these experiments, had built the first working atomic clocks. This background section draws from Blair et al's introduction [6].

#### Cesium Atoms and the Hyperfine transition

One atomic clock scheme involves Cesium atoms as the source of precise oscillations. Cesium is chosen because it fulfils a list of criteria including, by virtue of having one valence electron, its ground state is well separated from its first excited state making state selection easier. The specific transition that is used is the hyperfine transition from $(F = 3, m_F = 0)$ to $(F = 4,\ m_F = 0)$.

*Hyperfine* refers to the tiny energy differences between configuration states that are due only to the differing alignments of the electron's magnetic moment in relation to the nucleus's moment in the presence of an external magnetic field. Since the valence electron and the nucleus are relatively far apart the coupling is very weak and the energy difference is very small.

$F$ is the quantum number that labels the eigenvalues of the total angular momentum operator acting on the combined valence-electron-nucleus wave-function. Therefore $F = I + J$, where $J = L + S$, the total electron angular momentum quantum number and $I$ is the nucleus' intrinsic spin angular momentum quantum number. The intrinsic

nuclear spin is fixed depending on the isotype of Cesium. For $Cs^{133}$, $I = 7/2$, thus $F = 3 = 7/2 + (0 - 1/2)$ and $F = 4 = 7/2 + (0 + 1/2)$, both correspond to the ground state $L = 0$ of the valence electron, the only difference being the alignment or anti-alignment of the electron spin, relative to the nuclear spin.

$m_F$ refers to the quantized projection ($-F \leq m_F \leq F$) of the total angular momentum along the direction of the external magnetic field. $m_F = 0$ is selected for the atomic clock protocol because it is the least susceptible to magnetic field variations.

The energy difference between ($F = 3, m_F = 0$) and ($F = 4$, $m_F = 0$) corresponds to a transition frequency of about 9192 MHz[†], which is in the microwave range and thus easily manageable by electronic circuits.

### Atomic Clock Ramsey Protocol

The original scheme used a beam of atoms, with more recent techniques using a 'fountain'. However there is little difference in the actual protocol and we describe the original beam approach.

A beam of heated collimated Cesium atoms is first passed through a magnet that selects those atoms in the desired initial state. The 'prepared' atoms are then passed through a region of a uniform magnetic field. At the beginning of the region, the atoms are subjected to a pulse of microwaves close to the transition frequency for a duration $t = \frac{\pi}{2\Omega_R}$, where $\Omega_R$ is the coupling strength of the interaction between the two energy levels. This, so called $\frac{\pi}{2}$-pulse, places the atoms in a superposition of the two states. Towards the end of the uniform magnetic field another $\frac{\pi}{2}$-pulse is shone on the atoms moving some of the population to the final state. The atoms leave the magnetic region and after passing through a second selecting magnet, impinge on an ionizing detector, to determine the final populations.

The two microwave pulses are produced by circuits that use the frequency of a 'classical' quartz crystal clock. The circuit up-converts the ideal expected quartz crystal frequency

[†]In fact, the 1964 declaration of the International Committee of Weights and Measures *defined* this transition to be 9192.631770 MHz in order to define the second.

to a predetermined microwave frequency close to the transition frequency (sometimes intentionally not exactly equal). The two microwave pulses lead to an expected final population (sometimes intentionally not a complete population inversion, in order to improve sensitivity). The actual population is measured and compared to the expected population and the difference is in most part, due to the error of the actual quartz crystal frequency away from the expected ideal frequency. This error is then compensated for in a closed-loop frequency adjustment and the process continues.

In the preparation and measurement phases of the protocol, the feedback loop is obviously not correcting any errors. This so called *dead time* occurs periodically as the protocol repeats itself. Here-in lies a problem called *aliasing* and in particular, the Dick Effect. The periodic lack of sensitivity allows noise at that correcting frequency and its higher harmonics to, not only go uncorrected, but to also resurface as low frequency noise.

To gain a deeper understanding, a detailed mathematical description of the correction process is required. The aliasing effect should be explicitly derived using the tools that have been introduced above before we can attempt to develop new anti-aliasing techniques.

### The Necessity of Correction

As we saw for some noise models with Allan variance, if we average for longer, $\tau_a \to \infty$, the average fractional frequency error goes to zero, $\overline{y}_{t_0,\tau_a} \to 0$. So it seems that 'if we wait long enough', no correction needs to be made at all! This is ultimately because we assumed a driftless (zero mean) error. Even if this were realistic, the problem is that we couldn't reliably use our clock before the 'end of time'. But we require that our clock is as accurate as possible at all times.

With other noise models the variance was infinite, that is $I^2$ unbounded, the departure of a single realization from the correct frequency can grow indefinitely, even though the average error is zero and even though we have assumed ergodicity[‡].

[‡]The result that $\lim_{T_r\to\infty} I^2(T_r) = 0$ is for models where $I^2(T_r) < \infty$. For unbounded models it would

But even for finite fractional frequency variance models, the accumulated error in time is proportional to the integral of the instantaneous frequency error or using the average frequency error: $T_r \times \overline{y}_{t_0,T_r}$. So it is also possible that the frequency noise is bounded but the error in time is unbounded. Thus, the 'wait long enough' argument above is even less of a consolation because agreement may come and go and in between, departures of time error may be unbounded.

For all the above reasons, we need to perform ongoing corrections to the noisy quartz clock.

be interesting to take the ensemble average 'at the same time' as increasing $T_r$ to see if we can recover a finite result.

# Bibliography


[1] *Characterization of Phase and Frequency instabilities in Precision Frequency Sources: Fifteen Years of Progress*, J Rutman (Proc IEEE Vol 66 pg 1048-1075; 1978)

[2] *Opticks or, a treatise of the reflexions, refractions, inflexions and colours of light: also two treatises of the species and magnitude of curvilinear figures*, I. Newton, (Royal Society; 1704)

[3] *On the quantization of direction in a magnetic field*, W. Gerlach and O. Stern (Ann. Physik; Vol 74; pg. 673-699; 1924)

[4] *A new method of measuring nuclear magnetic moment*, I.I. Rabi, J.R. Zacharias, S. Millman and P. Kusch; (Phys. Rev.; Vol 53, pg. 318; 1938)

[5] *A molecular beam resonance method with separated oscillating fields*, N.F. Ramsey (Phys.Rev.; Vol 78; pg. 695-699; 1950)

[6] *Time and Frequency: Theory and Fundamentals*, B.E. Blair Editor (Nat. Bur. Stand. (U.S.); Monogr. 140; 1974)

# Chapter 7

# Ameliorating Aliasing (and the Dick Effect) in Atomic Clocks


## Abstract

Atomic clocks employ the periodic calibration of a 'classical' quartz crystal against an atomic quantum reference in the form of a well-defined, precise hyperfine transition frequency. This periodic calibration has a surprising noise-inducing effect called aliasing. In fact, this sampling effect is mathematically identical to visual digital aliasing. Visual anti-aliasing techniques such as blurring trade accuracy for smoothness, which would be unacceptable for atomic clocks. A new technique is proposed where previous samples are incorporated to boost correction on certain portions of the noise spectral density and dampen sensitivity on the aliased portion. Incorporating previous samples relies heavily on there being correlations in the noise.


## 7.1 Introduction

The Dick Effect [1, 2, 3] is the down-conversion and persistence of time-keeping noise at frequencies that are an integer multiple of the sampling frequency of atomic clocks. It is actually a very similar effect to visual aliasing. Visual aliasing, an artefact of the digitization of images containing high frequency components, is jarring to the human eye but can be combated by relaxing precision, for example by blurring pixels. This *looks* acceptable because the goal is aesthetic and accuracy. Any anti-aliasing like technique

that reduces accuracy or even introduces noise, exactly defeats the purpose of atomic clock correction and so the Dick Effect cannot be beaten by these anti-aliasing techniques. The standard atomic clock scheme is based on a single averaging window and correction that repeats. We suggest an extension to the single window correcting process involving multiple but overlapping windows.

## 7.2 Noise Correction

### 7.2.1 Sensitivity Function

We start by defining the single cycle sensitivity function, $g(t)$, as the time profile of sensitivity to the fractional frequency noise $y(t)$ during one run. We define, $g(t)$ in a form ready for convolution, that is, it is flipped with respect to time. We require $g(t)$ to be a causal filter and hence must only be non-zero before $t = 0^*$. Since it models a finite measurement it must only be non-zero on a finite interval. Putting the last two conditions together, $g(t)$ is thus only allowed to be non-zero on the interval $(-T_c, 0)$, where $T_c$ is the period of one full cycle. Of course, for some times within this interval $g(t)$ may also be zero within a cycle. Initially, we leave the exact form of $g(t)$ open and general. Further along we do assume a certain structure. For a single idealised Ramsey protocol run, $g(t)$, has the form of a shifted rectangular function, that reflects 100% sensitivity for some time ($T_r$), with zero sensitivity on either side, $T_p$ for preparation and $T_m$ for measurement. $T_p$ and $T_m$ sum to give us what we referred to before as dead time: $T_d = T_p + T_m$. Therefore $T_c = T_r + T_d$. See Fig. 7.1, for a pictorial representation of the various intervals.

Averaging in the time domain measure of noise is integration of the frequency noise over some interval and divided by the length of the interval to arrive at the average frequency error. Integration over the interval can be viewed as multiplication by a rectangular window on that interval followed by integration over the whole real line. This window view of integration is very useful when it comes to rewriting a sequence of measurements

*The time axis is flipped so that when the convolution translates and flips $g(t)$ it becomes the window positioned where the integration takes place (before $t_0 + nT_c$).

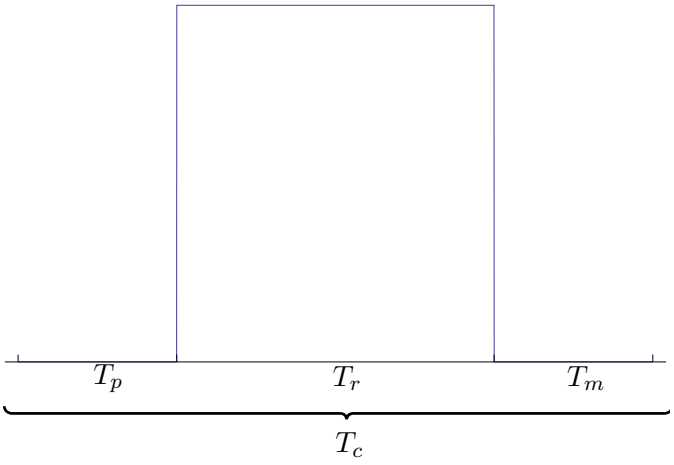



Figure 7.1: A typical $g(t)$ sensitivity function. $T_p$ is the preparation time, $T_r$ is the length of time that the system is sensitive to noise ($r$ for Ramsey), $T_m$ is the measurement and processing time and $T_c$ is the total time ($c$ for cycle).

as a sampled convolution. Experimentally this integration is achieved by counting the number of oscillations and dividing by the length of elapsed time.

In the atomic clock case, the sensitivity function plays the same role as an integration window. The difference is that the form of the window is capturing experimental limitations with, for example, compulsory dead time and its experimental implementation does not involve the direct counting of oscillations. In the Ramsey protocol, the final population of Cesium atoms is a single number that is the result of integrating the product of the sensitivity function by the frequency noise [4],

$$P(t_0) \propto \overline{y}_g(t_0) \equiv \int_{\infty}^{\infty} g(t - t_0) y(t)\, dt.$$

In the Allan variance measure, the window consists of two pieces and the single number after integration was used towards a measure of the noise as a function of the length of the windows. In the atomic clock correction scheme, that single number is used to actually make a correction. The method of correction is open and we consider two ways.

### 7.2.2 Phase Correction

Since the integral of the frequency error is proportional to the phase error, the single number produced after one Ramsey cycle actually contains time error information. Thus one method of correction is to carry out the Ramsey protocol for as long as possible ($T_r >> 1$) and use the final measured error for a *phase* correction of the slaved quartz clock. Phase correction could be implemented by temporarily and as quickly as possible speeding up or slowing down the quartz frequency via voltage control of the frequency until the phase is re-matched. This way the quartz crystal should have very good long term stability.

#### Constraints on Large $T_r$

A large $T_r$ would render the effects of hard minimum dead-time limits, less problematic. However, $T_r$ can only be lengthened by a finite amount. The first constraint is called *phase wrap*. If the error in phase, accumulates beyond $|\pi|$, the protocol cannot distinguish it from a phase error which is less by an integer multiple of $2\pi$ bringing it into the $(-\pi, \pi)$ interval.

Secondly, even if the time to phase wrap is large, some application may want to interrogate the quartz clock at a high frequency. If we keep corrections far apart, the short time stability is not being improved on[†].

Finally, with a finite $T_r$, the inevitable dead time still occurs. If the frequency is allowed to wander uncorrected, that uncorrected dead-time may start to cause long term degradation. Therefore we suggest a combination of phase and frequency correction. The same Ramsey protocol measurement contains both phase and frequency information. Once the phase has been corrected, it would be a good strategy to reset the frequency taking into account the frequency information. If this is employed then the following section on frequency correction has a direct bearing. Especially the way, in general, aliasing is made worse by longer averaging and leaving the frequency uncorrected for

[†]Though quartz clocks are known to have very good short time stability.

longer leads to higher frequency variation. But we have discussed above how longer averaging is better for phase correction. Thus these two goals of frequency and phase correction are in a surprising antagonism.
Further research would therefore include studying this trade-off between phase and frequency correction. In the next section however, we consider frequency correction independently from phase correction.

### 7.2.3 Frequency Correction

With the goal of correcting the frequency only, the ideal would be an instantaneous measurement of the noise and a same-time correction to the clock. By the constraints of the physical set-up we end up integrating the instantaneous fractional frequency to arrive at the average fractional frequency, $\overline{y}_g$. This average must then be used to predict the instantaneous fractional frequency $y(t)$ at the time of correction. It seems we should thus integrate for as short as possible but long enough to get a meaningful result, that is at least longer than the error associated with the actual preparation and measurement processes and its finite resolution. This integration length is called $T_r$. The measurement result should be used for correction as soon as possible. However, there is a minimum time it takes to measure and adjust the system $T_m$. There is also minimum time of preparation before integration, $T_p$. These times [4] all add up to the total cycle time,

$$T_c \equiv T_p + T_r + T_m.$$

We can wonder if it would help to lengthen the cycle or introduce a delay before the next cycle to see its effect on aliasing. As we show below, a longer $T_c$ means a lower sampling cut-off frequency $1/2T_c$, which means aliasing of a larger band of frequencies. Therefore for frequency-only correction it seems it is always better to have as high a sampling cut-off as possible, which means as low a cycle period as possible. However, if $T_d = T_p + T_m$ is large and cannot be made smaller, so that aliasing is pronounced, perhaps a better strategy is to increase $T_r$ to reduce aliasing. All these questions and more are explored below for the case of a single window and then again later with multiple windows.

In the standard scheme, the average is used directly as the corrective amount to make, $C(T_r)$. In essence the assumption is that the average over a small interval is close enough to the instantaneous fractional frequency noise at the later moment of correction, in order to cancel it. While running, the atomic clock is directly sensitive to $y(t)$. This sensitivity is captured by the single cycle $g(t)$, which is then periodically repeated with period $T_c$. A finite difference treatment of a closed-loop model of the sensitivity followed by correction is possible in the time domain as was explored in [5]. We attempt a Fourier domain treatment.

Extending [6], the atomic clock corrected fractional frequency noise, for single window correction, can be written as

$$y^{\mathrm{C}}(t) = y(t) - \overline{y}_g(t_0 + nT_c).$$

The $C$ superscript stands for 'corrected'; $n$ is the largest integer such that $t_0 + nT_c < t$, that is, $n = \lfloor \frac{t-t_0}{T_c} \rfloor$. Notice that $y^{\mathrm{C}}(t)$ depends on the *uncorrected* noise and *only* the last implemented correction. Thus the required corrections to a free running clock are

$$C(n) = \overline{y}_g(t_0 + nT_c).$$

Each correction amount is the sensitivity modulated moving-average of $y(t)$ sampled at multiples of period $T_c$ after the starting time $t_0$.

$C(n)$ is a discrete sequence of corrections to the uncorrected noise. The corrections for a running corrected/locked clock is the difference between two weighted neighbouring corrections for the free-running clock,

$$C_L(n) \equiv C(n) - \tfrac{1}{T_r} \int g(t)\, dt\, C(n-1). \tag{7.1}$$

This is proved in [6] but has the following intuitive/alternative 'proof'. Say at some $t = t_0 + (n_1 - 1)T_c$, the unlocked correction, $C(n_1 - 1)$ is subtracted from the noise. If it had not been made, the next step's unlocked correction $C(n_1)$ would have been the required correction. What is the new required correction, which is built on the same

sensitivity modulated average process of the assumed same noise? Since each step's sensitivity integration process is linear, the last implemented instantaneous constant correction passes through the integration as a constant contribution, yielding $C_L(n_1)$ as the adjusted required correction. Now, if we solve for $C(n_1)$,

$$C(n_1) = C_L(n_1) + \tfrac{1}{T_r} \int g(t)\, dt\, C(n_1 - 1),$$

we see that it 'contains' a $C(n_1 - 1)$ correction on top of a locked correction. Thus implementing $C_L(n_1)$ on the $C(n_1 - 1)$-corrected noise is equivalent to implementing $C(n_1)$ on the uncorrected noise.

Now, the analysis can continue exactly the same for $n_1 + 1$ on the uncorrected noise, showing that the locked corrections can be written in terms of only two unlocked correction terms[‡].

Turning our attention to the power spectral density of the unlocked *corrections*, we would like it to be as close as possible to the power spectral density of the noise, because then at least the correction is modelling the noise correctly. This is obviously not necessarily enough to cancel the noise, because one could imagine subtracting a highly delayed version of the noise from the current noise. The corrections would have the same power density but would be poorly correlated to the current noise and therefore unfortunately be unable to cancel the noise. The power spectral density discards phase information, which in the time domain is translation information. Naturally, *when* the noise occurs is as important as what kind of noise occurs. Therefore a better metric would be the cross-correlation between the correction and the noise and not just the comparison of the separate auto-correlations. Nevertheless arranging for the auto-correlations to be similar in shape is a necessary first step to getting the cross-correlation high.

[‡]This dependence, though, shows how the locked clock's correction is in general not a stationary process. It therefore wouldn't be rigorous to define its power spectrum. Therefore we content ourselves with looking at the corrections to be applied to an unlocked clock, knowing that it can easily be translated into corrections for a locked clocked.

### Averaging as a Predictor

In the above formulation, it is seen that the causal moving average $\overline{y}_g(t_0 + nT_c)$ is being used as a predictor of the noise at $y(t_0 + nT_c)$. The reason we have to average the noise and not simply measure the noise is because we do not have access to the instantaneous noise. Naturally, the shorter the integral, the better the prediction. But in practice there is a minimum length that cannot be shortened. In subsequent sections we experiment with the length of $T_d$ and $T_r$ subject to realistic constraints. We also propose the idea of using previous weighted corrections to help better predict $y(t_0 + nT_c)$.

### Power Spectral Density of the Correction

Perfect correction would be $S_C(f) = S_y(f)$[§]. However the sensitivity function and periodic correction introduce distortions. To calculate this effect we need to calculate the power spectral density of the corrections. Since the corrections are discrete, in order to calculate the power spectral density we need to take the Discrete-time Fourier Transform (DTFT). However, there is an equivalent Fourier transform that takes into account the sampling.

We begin with the discrete autocorrelation of the correction,

$$R_C(n) \equiv \langle C(n_0)C(n_0 + n)\rangle \,.$$

But instead of working with $R_C(n)$ directly we rather work with the continuous autocorrelation and later sample at $nT_c$,

$$\begin{aligned} R_{\overline{y}_g}(\tau) &\equiv \left\langle \overline{y}_g(t_0)\overline{y}_g(t_0 - \tau)\right\rangle \\ &= \left\langle \int_{-\infty}^{\infty} g(t_0 - t_1)y(t_1)\,dt_1 \int_{-\infty}^{\infty} g(t_0 - \tau - t_2)y(t_2)\,dt_2 \right\rangle . \end{aligned}$$

Therefore, following the usual derivation of the power spectral density, but now with

[§]As a caveat, see previous comments about comparing power spectral densities.

sampling, yields,

$$\begin{aligned} S_C(f) &= \mathcal{F}_{\text{DTFT},n}\left(R_C\right), \qquad |f| < \tfrac{1}{2T_c} \\ &= \mathcal{F}_\tau\left(T_c R_{\overline{y}_g} \cdot \text{Ш}_{nT_c}\right) \\ &= T_c \mathcal{F}\left(R_{\overline{y}_g}\right) * \mathcal{F}\left(\text{Ш}_{T_c}\right) \\ &= \left(T_c |G(f)|^2 S_y(f)\right) * \left(\tfrac{1}{T_c}\text{Ш}_{1/T_c}\right) \\ &= \sum_n |G(f+\tfrac{n}{T_c})|^2 S_y(f+\tfrac{n}{T_c}) \\ &= \sum_n S_g(f+\tfrac{n}{T_c}) S_y(f+\tfrac{n}{T_c}). \end{aligned}$$

$S_g(f) \equiv |G(f)|^2$, is the Fourier transform, absolute value squared, of the sensitivity function. Since $g(t)$ is deterministic and of finite support, we do not need to go through the Fourier transform of the autocorrelation to define it, we may take the Fourier transform directly.

In the case of the box sensitivity function of Fig. 7.1, this reduces to

$$S_C(f) = \begin{cases} \sum_n \text{sinc}^2_{1/T_r}(f+\tfrac{n}{T_c}) S_y(f+\tfrac{n}{T_c}) & , \quad \text{for } |f| < 1/2T_c \\ 0 & , \quad \text{otherwise} \end{cases}. \tag{7.2}$$

As we can see, the power spectral density of the corrections does not look like the power spectral density of the noise. We have to contend with the modulation of the transfer function (the Fourier transform of the finite averaging window) and the aliasing of sampling. However, at least, the averaging based predictor, does very well in capturing noise with low frequencies that don't cancel out in the averaging window.

### Sampling and Reconstruction

The sampling of the moving average is identical to the sampling phenomenon we introduced in the background chapter. All that is missing is the reconstruction with convolution of the $\text{sinc}_{T_c}$. We could very well introduce the $\text{sinc}_{T_c}$, which reproduces an approximation to the continuous moving average, which we could then sample at $nT_c$ to reproduce the discrete points. Convolution with $\text{sinc}_{T_c}$ is multiplication of the power

spectral density by a convenient rectangular function giving us a noise spectrum that is only non-zero for $|f| < B = 1/2T_c$.

Since we are performing the DTFT, we get the same restriction on $f$ by virtue of the fact that the highest frequency present in the discrete points is one whose period is twice the gap between points.

Yet another way to see this restriction, is to think of the discrete points as the *Fourier series* of the periodically $1/T_c$-repeating, power spectral density[#]. We may choose any periodic interval for the Fourier Series and for convenience we select the zero centred $1/T_c$ interval (splitting it into two equal pieces), showing that all the frequency information for $|f| < 1/2T_c$ is enough to produce a discrete set of numbers which has the same information as the repeated function.

### Uncorrected Noise

Now we are ready to identify the noise that is not corrected. There are two ways that the noise is left unrepresented. Firstly there is the obvious sampling induced cut-off,

$$|f| \geq \frac{1}{2T_c}.$$

This means that the continuous moving average no longer has frequency content faster than half the sampling frequency. The fastest that finite samples separated by $T_c$ can change sign and back is over a period of $2T_c$ which is at a frequency of $1/2T_c$ which is half the sample frequency. Faster underlying variations are there, but resurface as lower frequencies, and no longer appear as fast variations.

Secondly, the $\text{sinc}^2_{1/T_r}$ in the transfer function has a diminishing envelope. The $\text{sinc}^2_{1/T_r}$ function does 'bounce' with diminishing amplitude on the zero line, but the first zero does not occur before the cut-off. $1/T_r$ is the location of the first zero, however for that

[#]Note, we are taking the Fourier series from the Fourier domain, bringing us back into the time domain. A kind of inverse procedure, except that we are specifically using the Fourier Series to take us from a continuous repeating domain to a discrete domain.

zero to appear below the sample-induced cut-off frequency

$$1/T_r < 1/2T_c$$

$$T_r > 2T_c,$$

which is not possible because $T_r \leq T_c$, by definition. Even though the sinc envelope does not go to zero in the sampled frequency range, it does diminish so that the spectral density of the correction is always less than the spectral density of the raw noise[‖]. This diminishing effect is reduced for shorter $T_r$, because then the sinc's first zero is pushed to infinity and higher frequencies are brought under the main flattening lobe.

#### Over-corrected/Mis-corrected Noise

Besides not correcting parts of the noise power spectral density, the atomic clock set-up also introduces its own noise due to out-of-sync correction, loss of correlation and sampling down-conversion. Out-of-sync correction is due to the delay inherent in using a past average to predict the future. This error is not reflected in the power spectral representation because of the absolute values. This leads to frequencies that *are* picked up by the averaging, e.g. $f = 1/(4T_c)$, but whose contribution to the average does not match their contribution to the point value at the end of the cycle ($nT_c$). The most extreme example of this effect is for frequencies just slower than half the sample frequency $f \approx 1/(2T_c)$. Here the fluctuation typically does contribute to the average but which then becomes a completely unnecessary correction by the time that fluctuation reaches the point of correction. That is, the zeroes of the noise at this frequency occur at the end of the cycle, at the moment when their average contribution is being made to count.

It is proposed that this error could perhaps be partially corrected for by using a model of the noise and predicting future noise. This would entail not only estimating the

[‖] As long as the window's spectral density is normalized to one at its maximum, which we always do. In fact, as a future possible technique we could experiment with scaling the window to accentuate sensitivity for certain parts of the spectrum in exchange for over-correcting and thus introducing noise at other parts.

amplitude of the noise but also the phase. The phase of the noise at a particular frequency is only meaningful within the correlation time of the noise for that specific frequency. To try and compensate for this effect in the Fourier domain a single window needs to be phase shifted by $e^{2\pi i f(T_c/2)}$ to move the center of the casual window to the moment of correction. If the single window is viewed as a sum of dividing smaller windows (each centred on $-T_c < t_c < 0$) then each window's transfer function needs to be translated by $e^{2\pi i f t_c}$. Unfortunately all of this does not help because we do not have the Fourier transform of the noise, which is not even defined. Thus rather than explicitly calculating forward predictions perhaps a feedback scheme on a parameter that captures this delay could be implemented. This idea is very similar to ideas already well developed in the field of active 1-D audio noise cancellation techniques [7]. It may work well for noise spectral densities where there is one dominant noise frequency. Closely related to the above error is the problem of accurately correlating with the noise. Even if we compensate for delay synchronization, since the noise is random we have to be concerned about correlation. In using past information to predict future random error, even information about low frequency noise may be less valuable due to the loss of correlation after some time. Noise operating at low frequency with a random phase kick for example would scupper our attempts to correct for that low frequency noise once the correlation time has elapsed.

### 7.2.4 Aliasing and the Dick Effect

The most important case of the introduction of errors associated with periodic frequency correction is aliasing. By setting $f = 0$ in Eq. (7.2) we arrive at the famous Dick Effect [1], namely, frequencies of the noise power spectral density corresponding to $\frac{n}{T_c}$ are down-converted into $S_C(0)$. That is the correction's power spectral density's DC term, which corresponds to a permanent offset, is

$$S_C(0) = \sum_n S_g(\tfrac{n}{T_c}) S_y(\tfrac{n}{T_c}).$$

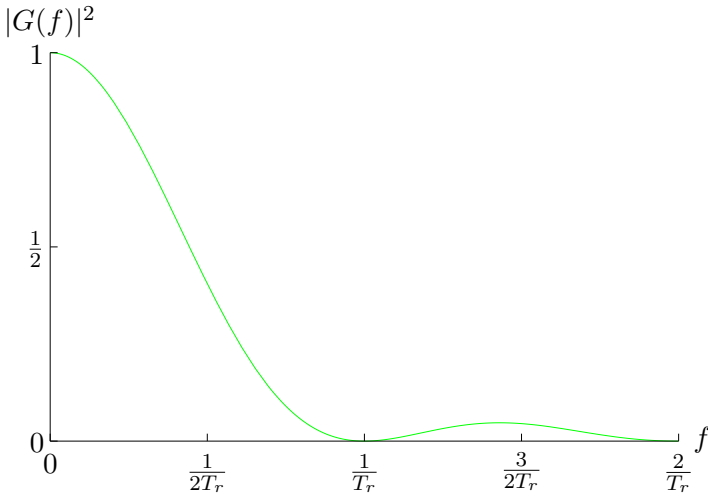



Figure 7.2: The transfer function squared of the single window sensitivity function, showing the regions of sensitivity to noise in the Fourier domain (with no sensitivity at $n/T_r$).

This demonstrates that noise at multiples of the sampling frequency (the sampling frequency's faster harmonics) look like constant fixed shifts to the averaging and sampling process.

We now try to better understand aliasing in general and the Dick Effect specifically for the box sensitivity function in Fig. 7.1. The transfer function squared of the single window box function which appears in Eq. (7.2) is

$$|G(f)|^2 = \text{sinc}^2_{1/T_r}(f).$$

A plot of this sinc function with zeros at $n/T_r$ appears as Fig. 7.2. In the Fourier domain, this *filter*, multiplies the noise, to give the power spectral density of the continuous moving average of the noise. Of course, in practice we can not measure the *continuous* moving average. The effects of taking snapshots of the continuous moving average at $nT_c$ in the time domain, is to translate by $n/T_c$ and sum multiple copies of the continuous moving average's Fourier transform in the Fourier domain. Fig. 7.3 shows these translated copies before summation. Of course, $S_C(f)$ is zero for $|f| > 1/2T_c$, however the full translated versions are shown to aid understanding. Fig. 7.4 is a zoomed in version of the same plot, on the region where the DTFT is non-zero, with the translated copies'


Figure 7.3: Aliasing due to Overlapping Translated Copies.

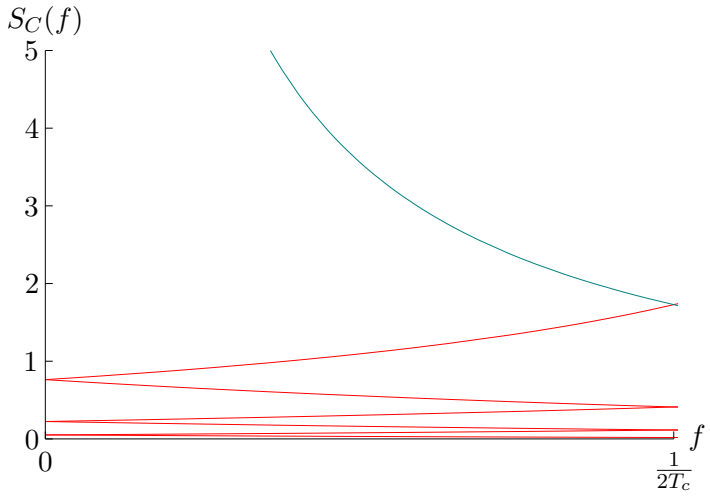



Figure 7.4: Aliasing (on the non-zero region).

contribution coloured red. In Eq. (7.2) the aliasing parts of the window are multiplied only by the corresponding high frequency parts of the noise. That is, while we show the sinc window aliasing, it actually first needs to be multiplied by the noise at the higher frequencies of the translated copies, before aliasing, to calculate the impact of aliasing. We do as such in Fig. 7.12 – Fig. 7.16. In Fig. 7.2 – Fig. 7.5, $T_d = 0.7, \quad T_r = 0.2$.

In Fig. 7.5 we show the aliasing again, but with a suggestive dotted curve, which makes the translated copies' contribution look like reflections of the main untranslated transfer

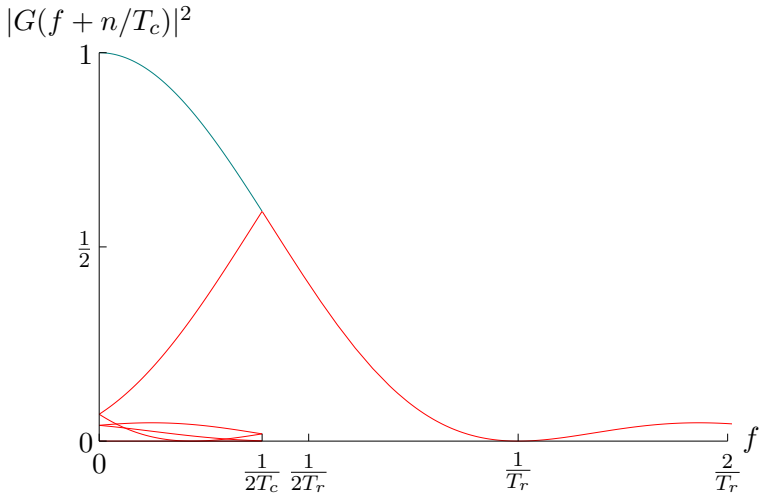



Figure 7.5: The transfer function squared Aliased.

function. That is, notice how the transfer function 'folds' back on itself as it reaches $1/2T_c$, and then 'bounces' back again at zero. The accumulation of these contributions at $f = 0$ is the Dick Effect.

To make precise and to actually see that 'folding' and 'bouncing'** are good descriptions of what is occurring mathematically, we prove the following simple, original mathematical lemma.

**Reflection Lemma. 1** *The translation, by $A$, of a symmetric function, is equivalent to reflection about $A/2$.*

PROOF: Given $f(x) = f(-x)$. Translation by $A$: $f(x - A)$. Reflection about $A/2$, can be thought of as a translation of the point $A/2$ to zero, followed by reflection about the $y$-axis, followed by a translation back,

$$f(x) \to f(x + A/2) \to f(-x + A/2) \to f(-(x - A/2) + A/2).$$

Finally the two formulae are equivalent after simplification and the use of symmetry,

$$f(-(x - A/2) + A/2) = f(-x + A) = f(-(-x + A)) = f(x - A). \qquad \textbf{Q.E.D.}$$

Since $S_y(f)$ is symmetric about the $y$-axis, i.e. $S_y(f) = S_y(-f)$. The first fold corresponds to the translated term $S_y(f - 1/T_c) = S_y(-(-f + 1/T_c)) = S_y(-f + 1/T_c) =$

**On the interval $(0, 1/2T_c)$. Of course, the analysis occurs symmetrically about the $y$-axis.

$S_y(-(f-1/2T_c)+1/2T_c)$, which is reflection about $1/2T_c$. When $f=1/2T_c$, this term is equal to $S_y(-1/2T_c+T_c)=S_y(1/2T_c)$, which is the main non-aliased term evaluated at the same point. So the translated power spectral density (centred on $T_c$) meets the original term at $1/2T_c$ and looking at the first-fold term with $f$ running backwards from $1/2T_c$ to 0, we see that the folding term is a copy of $S_y(f)$ running forwards from $1/2T_c$ to $T_c$. That is the fold term is a reflection of the main term about $1/2T_c$ exactly as an application of the above lemma would arrive at.

The 'bounce back' contribution comes from the term that has been translated by $-1/T_c$, $S_y(f+1/T_c)$. When $f=0$, it meets the first fold-over term: $S_y(+1/T_c)=S_y(-1/T_c)$, by symmetry. For this term, as $f$ goes from 0 to $1/2T_c$, the contribution is the same as the main term from $1/T_c$ to $3/2T_c$. This procedure continues indefinitely folding at $1/2T_c$ and bouncing back at zero, involving further and further translated copies of the main $S_y(f)$. The net effect looks like the main $S_y(f)$ is bouncing back and forth between the two 'walls'. This makes the calculation of the total aliased error ($A_{1/f}(T_d,T_r)$) particularly easy, it is simply the integral of the main $S_y(f)$ from $1/2T_c$ to infinity. Doing the calculation analytically for $S_y(f)=1/f$[††] noise yields,

$$A_{1/f}(T_d,T_r)=\int_{1/(T_d+T_r)}^{\infty}\frac{\mathrm{sinc}^2_{1/T_r}(f)}{f}\,df$$
$$=\frac{\pi T_r\left((T_d+T_r)\sin\left(\frac{\pi T_r}{T_d+T_r}\right)-\pi T_r\mathrm{Ci}\left(\frac{\pi T_r}{T_d+T_r}\right)\right)-(T_d+T_r)^2\left(\cos\left(\frac{\pi T_r}{T_d+T_r}\right)-1\right)}{\pi^2T_r^2},$$

with $\mathrm{Ci}(z)=-\int_z^\infty\frac{\cos(t)}{t}\,dt$. To visualise the total aliasing error, we make a 3D plot of $A_{1/f}(T_d,T_r)$ in Fig. 7.6. We firstly confirm the obvious, that a shorter $T_d$ is always preferable (pushing the bouncing-back wall further away). However we see that, even though a shorter $T_r$ would also push the bouncing-back wall further away, a shorter $T_r$ also pushes the zero of the sinc further away and thus increasing the aliasing effect. The final behaviour is thus, that for a fixed $T_d$, a *longer* $T_r$ produces lower aliasing by

[††]It may seem that we have set $h_{-1}=1$. However, since we are operating theoretically and in general, perhaps a better way to interpret this is that we are defining the time scale. Our $1/f=1/(f'/h_{-1})$, where the units of our $f$, corresponds to $1/h_(-1)$ times the units of the experiment's frequency, $f'$ (e.g. Hz). Since our fractional noise is dimensionless the dimensions of $S_y(f)$ is simply 'time' or 'per frequency'.

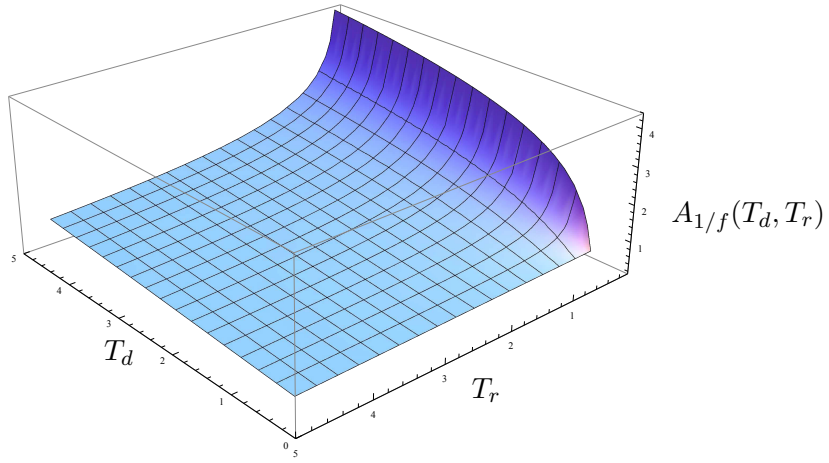



Figure 7.6: Total Aliasing Error ($1/f$ noise).

suppressing higher frequency sensitivity. This property is not useful when in comes to error correction. Indeed we see later that a smaller $T_r$ is still preferred for overall noise reduction. Incidently, we also observe and can calculate analytically that for $T_d > 0$, as $T_r \to 0$ the total aliasing error diverges for $1/f$ noise. Perhaps this is a strong theoretical justification for why there has to be a high frequency cut-off for $1/f$ noise. In the field, sometimes a $1/f$ cut-off is ignored because the sinc goes to zero, here by taking $T_r \to 0$ we are effectively removing the sinc and running into the usual problem that the area under the $1/f$ curve is infinite.

A surprising non-physical but mathematical result is that $T_d = 0, T_r \to 0$, does seem to produce a finite non-zero net total aliasing effect,

$$\lim_{T_r \to 0} A_{1/f}(0, T_r) = \frac{2}{\pi^2} - \mathrm{Ci}(\pi).$$

This is surprising because a zero dead-time, infinitesimal window cycle would be expected to reproduce the noise exactly, indeed at $T_r = 0$, the aliasing integral is undefined (the interval of integration is $(\infty, \infty)$, which we can then define to be zero). On further investigation, we realise that the limit depends on the direction of approach and so the 2D limit is also undefined and might as well be defined to be zero.

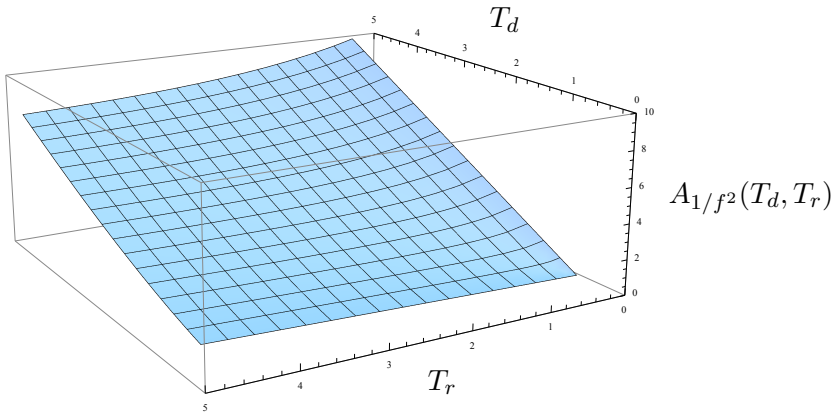



Figure 7.7: Total Aliasing Error ($1/f^2$ noise).

We repeat the calculation for $1/f^2$ noise and arrive at,

$$A_{1/f^2}(T_d, T_r) = \int_{1/(T_d+T_r)}^{\infty} \frac{\mathrm{sinc}^2_{1/T_r}(f)}{f^2}\, df$$
$$= \left(2\left(\pi^3 T_r^3 \mathrm{Si}\left(\frac{\pi T_r}{T_d+T_r}\right) + (T_d+T_r)\left((T_d+T_r)\left(2\,(T_d+T_r) + \pi T_r \sin\left(\frac{\pi T_r}{T_d+T_r}\right)\right)\right.\right.\right.$$
$$\left.\left.\left. + \left(-4T_dT_r - 2T_d^2 + \left(\pi^2-2\right)T_r^2\right)\cos\left(\frac{\pi T_r}{T_d+T_r}\right)\right)\right) - \pi^4 T_r^3\right)/(3\pi^2 T_r^2),$$

with $\mathrm{Si}(z) = \int_0^z \frac{\sin t}{t}\, dt$. We plot the total aliasing error for $1/f^2$ noise in Fig. 7.7 and observe similar behaviour to the $1/f$ plot, except that there is no divergence for $T_r = 0$. We are now ready to replot Fig. 7.4, with the aliasing terms summed up, but keeping their $f$ dependence, as well as with $A_{1/f}$ represented as a line at height $A_{1/f}/(1/2T_c)$. Thus the area under the line is the total aliasing error and the height is the average total aliasing error over the region aliased to. In Fig. 7.8, we see two dashed (red) curves replacing the zig-zag separate aliasing terms. The flat dashed line is the average total aliasing, and the curved dashed line is the sum of all the aliasing terms. We can see how the $f$ dependent sum is very close to the average, leading to the common notion that aliasing resurfaces as flat white noise. The solid lines correspond to the $1/f$ noise and the slightly lower non-translated sinc modulated $1/f$ noise. With the specific choices of parameters ($T_c = 0.9$, $T_r = 0.2$), we see that up to $1/2T_c$ the averaging process captures the noise quite well. It is just unfortunate that the aliasing noise supersedes the natural

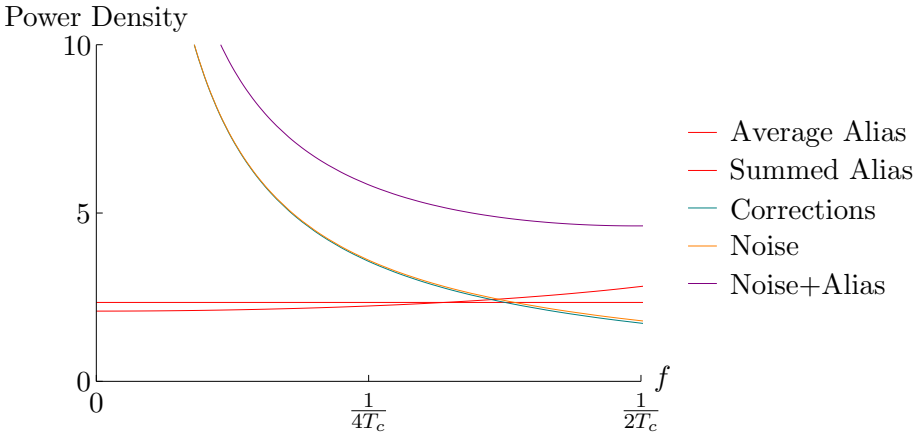



Figure 7.8: Summed and Averaged Aliasing Error ($1/f$ noise).

noise even before $1/2T_c$. The dot-dashed (purple) line is the $1/f$ noise plus the summed aliasing. This gives a good visual cue as to the noise that is not corrected plus introduced, that is the space between the dot-dashed line and the solid sinc line is noise left after the correction.

Bringing together all these noise contributions, the overall performance of the scheme to detect and capture the noise can be measured by the final total noise, $N(T_d, T_r)$, which includes the noise left uncorrected and the noise introduced due to aliasing. This total noise is thus the total noise minus the corrected amount on the interval $(0, 1/2T_c)$ plus the aliased noise. The corrected noise on the interval actually does manage to handle the infinite area for both $1/f$ and $1/f^2$ because the sinc goes to one as $f \to 0$. We do have to carefully handle the limits since $1/f^{(1,2)}$ and $\frac{\text{sinc}_{1/T_r}(f)^2}{f^{1,2}}$ both diverge as $f \to 0$, yet the difference is finite. The difficulty comes in for the uncorrected noise on the interval $(1/2T_c, \infty)$ for $1/f$ noise, because this uncorrected noise diverges. As for $1/f^2$ noise, the uncorrected noise on the infinite interval does indeed converge and to a straightforward value of $2T_c$. Therefore for $1/f^2$ we can arrive at a complete analytic result without

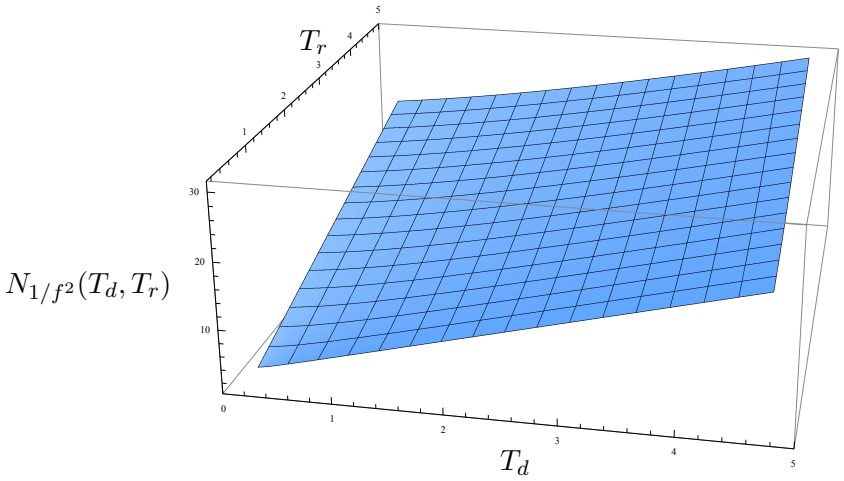



Figure 7.9: Final Noise $(1/f^2)$.

introducing any cut-offs,

$$N_{1/f^2}(T_d,T_r)=\left(4\pi^3T_r^3\mathrm{Si}\left(\frac{\pi T_r}{T_c}\right)+24T_d^2T_r+24T_dT_r^2+4T_c\left(\pi T_rT_c\sin\left(\frac{\pi T_r}{T_c}\right)\right.\right.$$
$$\left.\left.+\left(-4T_dT_r-2T_d^2+\left(\pi^2-2\right)T_r^2\right)\cos\left(\frac{\pi T_r}{T_c}\right)\right)+8T_d^3-\left(\pi^4-8\right)T_r^3\right)\left(3\pi^2T_r^2\right).$$

We plot $N_{1/f^2}$ in Fig. 7.9 and witness that for $1/f^2$ noise, smaller $T_d$ and $T_r$ is always preferred. For $1/f$ noise we have to introduce a high frequency cut-off, $f_{\mathrm{hi}}$, to measure the overall performance in the final noise. We also assume that the alias wall occurs before the high frequency cut-off, $1/2T_c < f_{\mathrm{hi}}$, otherwise there would be no aliasing.

$$N_{1/f}(T_d,T_r,f_{\mathrm{hi}})=\left(2\pi f_{\mathrm{hi}}T_r\left(2\pi f_{\mathrm{hi}}T_r\left(-2\mathrm{Ci}\left(\frac{\pi T_r}{T_c}\right)+\mathrm{Ci}\left(2\pi f_{\mathrm{hi}}T_r\right)+\log\left(2\pi f_{\mathrm{hi}}T_r\right)\right)\right.\right.$$
$$+4f_{\mathrm{hi}}T_c\sin\left(\frac{\pi T_r}{T_c}\right)-\sin\left(2\pi f_{\mathrm{hi}}T_r\right)\Big)+2f_{\mathrm{hi}}^2\left(8T_dT_r+4T_d^2+\left(\pi^2(2\gamma-3)+4\right)T_r^2\right)$$
$$\left.-8f_{\mathrm{hi}}^2T_c^2\cos\left(\frac{\pi T_r}{T_c}\right)+\cos\left(2\pi f_{\mathrm{hi}}T_r\right)-1\right)/(4\pi^2f_{\mathrm{hi}}^2T_r^2).$$

Ci is the same as above and $\gamma$ is the Euler-Mascheroni constant. We plot this final noise in Fig. 7.10 for $f_{\mathrm{hi}}=10$. If we replotted this graph for higher $f_{\mathrm{hi}}$ we would notice that the shape does not change much, the main difference is that the whole graph is shifted higher due to the increased uncaptured noise brought on by the higher noise frequency cut-off.

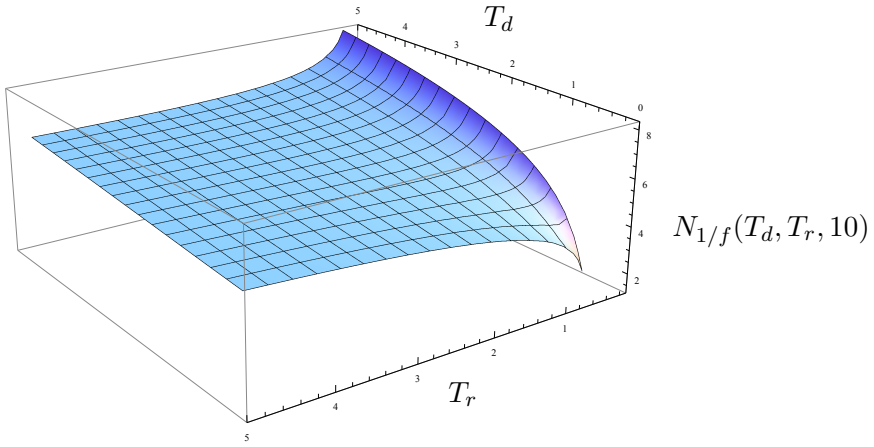



Figure 7.10: Final Noise $(1/f)$.

The most interesting feature of the $1/f$ final noise plot is that there is a lip in the graph for low $T_r$. For $T_r$ greater than values near the lip, the graph behaves the same as the $1/f^2$ graph, that is, lower $T_d$ and $T_r$ leads to lower final noise. However the lip comes into effect for very low $T_r$. This is the contribution towards the noise from aliasing. We saw in Fig. 7.6 that aliasing increased with lower $T_r$ eventually diverging. Here we see the increase dominating for small $T_r$ and moderate $T_d$. There is however no divergence because of the cut-off.

If $T_d$ is not sufficiently small to miss the lip, then smaller $T_r$ does not help and in fact makes the noise worse. Therefore for practical atomic clock implementations the priority is to make the dead time as small possible and at least below the lip region, before making $T_r$ smaller.

### Trying to Alleviate the Dick Effect

Eq. (7.2) suggests a way to eliminate the constant shift aliasing $S_C(0)$. Namely aliasing of the zero frequency, is eliminated if $S_g(0) = 1$ and $S_g(\frac{n}{T_c}) = 0$, $n > 0$. But $g(t)$ is by definition of finite duration and so must have frequency components for $f > 0$. How do we construct a $g(t) = 0, t < 0$ and $t > T_c$ such that $G(f) > 0$ for some $f > 0$ but $G(\frac{n}{T_c}) = 0$ ?

Let $g(t) = \frac{1}{T_c}\Pi_{T_c}(t - \frac{T_c}{2})$, that is 'always on' over the allowed interval, $(0, T_c = T_r)$.

That is, there is no dead time. Now $|\mathcal{F}(g_{T_r=T_c})(f)| = |\mathrm{sinc}_{1/T_c}(f)|$. The sinc function is exactly what we were looking for, namely,

$$\mathrm{sinc}_{1/T_c}(0) = 1 \text{ and}$$
$$\mathrm{sinc}_{1/T_c}(nT_c) = 0,\ n > 0.$$

Thus in terms of correcting errors, at least we get $S_C(0) = S_f(0) = 0$. The other cases of error, aliasing for $f > 0$, diminished amplitude, high frequency non-correction and delayed correction still apply but at least the errors in correction oscillate around the desired frequency and do not come in the form of an incorrect permanent offset.

Eliminating the Dick Effect by choosing $T_r = T_c$ ('always on') does not eliminate all aliasing it just positions the zeros of the folded back aliases at $f = 0$, so that DC aliasing is zero.

Of course, the assumption about 'always on' is not practically realisable, because of minimum preparation and measurement times. So we next look at ways of tackling aliasing, given the practical limitations in a trade-off fashion.

## 7.3 Multiple Window Attack on Aliasing Noise

In the single window case we found that aliasing was unavoidable. The best we could do was to try and make $T_d$ as small as possible. The Dick Effect, DC aliasing, would disappear for $T_c = T_r$, that is $T_d = 0$, but nevertheless aliasing at other frequencies still occurred, due to the discrete sampling.

The ideal transfer function to make aliasing completely disappear would be a box function of total width $1/T_c$. The inverse Fourier transform of a box is related to a $\mathrm{sinc}_{T_c}$ function in the time domain. If we could sample the frequency noise weighted by a sinc function, we could then perfectly average out high frequency noise. There are two major problems with this idea, firstly the sinc function is not causal and requires noise values from the future. Secondly, we still need to sample the hypothetical continuous moving sinc average. If we wait to sample multiple $T_c$'s to gather a representative weighting

under the sinc, we end up sampling at a longer period, thereby changing the aliasing fold-over point to an early position, making matters automatically worse. We need to somehow store noise data so that every $T_c$ we could reuse some of the previous data under a new shifted sinc.

Another way to motivate this usage of previous noise measurements is to consider a longer multiple window sensitivity function. Again the problem with this is that, if we sample at this larger period, for example say for three practicable windows each with the same $T_d$ and $T_r$, then the longer period is now $3\times(T_d+T_r)$, all the negative properties of sampling are made worse. That is aliasing comes into force even sooner, from $f = 1/6(T_d + T_r)$ and more noise is left uncorrected $f > |1/6(T_d + T_r)|$. Multiple windows would seem a completely bad idea compared to a single window, simply because of a lower sampling cut-off, even if there would be some desirable properties of the transfer function of the multiple windows $g(t)$.

The solution is related to a visual anti-aliasing technique that is ordinarily not available to us. It is called *super-sampling*. Super-sampling is merely sampling at a higher frequency. We already know that a smaller $T_c$ is better for frequency correction and we know that we are constrained by a minimum dead time which can not be made smaller. Thus it seems the idea of super-sampling is not applicable. However, if we use multiple windows with a larger period, perhaps it is possible to sample at a period less than the period of the full cycle. The trick is to let the $g(t)$'s overlap. That is, each of the multiple windows must be of the same form, so that when we sample at say, a period of a third of the total period, we reuse previous measured window results in a new shifted position. This third of the total period is not shorter than the previous minimum period and may be the same but it is shorter relative to the total period.

The constraint that the total $g(t)$ cycle must now be split into *identical* sub-regions, does not mean we cannot scale previous window results. Thus, there is still some flexibility in designing the new multiple window, $g_3(t)$.

One extra point that must be remembered when implementing the scheme, is that these

corrections are for an unlocked clock. To correct a locked clock, the corrections need to take into account previous corrections. This is not too difficult and proceeds along the lines of Eq. (7.1), but for multiple windows, we need to take into account previous corrections passing through subsequent sub-windows.

Experimenting with different numbers of windows and different multiple window configurations are all combined in Fig. 7.12. Three of the tested multiple window configurations are inspired by the above discussion of the sinc in the time domain. One of the sinc windows corresponds to three windows scaled to fit exactly under a $\mathrm{sinc}_{T_c}$ time domain function. Indeed the transfer functions of these windows comes closest to a repeated box pattern with a flattish top compared to the other windows.

### 7.3.1 Implementation

The idea of using overlapping multiple window sensitivity functions does not require any changes to the experimental set up. The actual sensitivity during the smallest cycle possible (now called the sub-window) is still the same. The only difference occurs after the usual measurement but before correcting the quartz clock. The stored data from previous short cycles are combined (computationally) according to the selected multiple window profile. For example, in the case of three sub-windows, the output of two sub-cycles ago would be scaled to become the middle window in the multiple window calculation. Corrections still take place at the end of every sub-cycle in order to keep aliasing down.

This data crunching step before affecting correction does raise the point that the calculation must be executed fast enough to keep up with the sub-cycles. To speed up performance, perhaps FPGA or analogue implementations would have to be considered.

### 7.3.2 Window Design

A general three window, $g_3(t)$, layout that is ready for overlap, consists simply of three of the single windows, $g(t)$, that we have been dealing with previously, lined up before

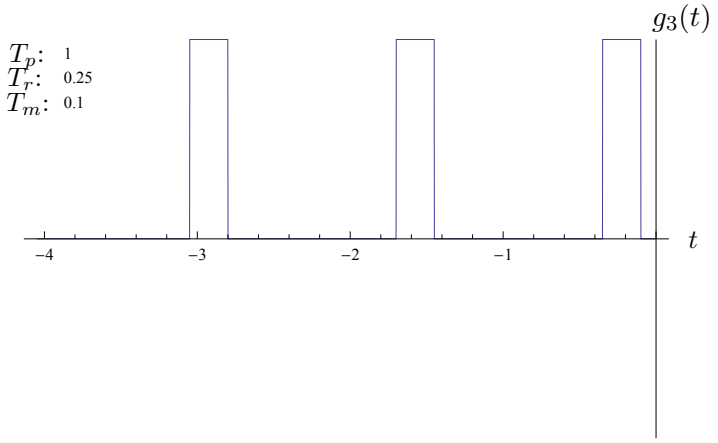



Figure 7.11: Example Three Window Sensitivity.

$t = 0$,

$$g_3(t) = a_2\Pi\left(\frac{t + 2T_m + T_p + \frac{3T_r}{2}}{T_r}\right) + a_3\Pi\left(\frac{t + 3T_m + 2T_p + \frac{5T_r}{2}}{T_r}\right) + a_1\Pi\left(\frac{t + T_m + \frac{T_r}{2}}{T_r}\right).$$

Here the scaling and shifting has been done in the arguments of the unit box $\Pi_1(t)$. Fig. 7.11 gives one example of a configuration. The Fourier transform squared of $g_3(t)$, has a particularly amenable and recognizable form,

$$|G_3(f)|^2 = M(f)\frac{1}{T_r^2}\mathrm{sinc}^2_{1/T_r}(f), \tag{7.3}$$

with

$$M(f) = (2a_3a_1\cos(4\pi f T_c) + 2a_2(a_1 + a_3)\cos(2\pi f T_c) + a_1^2 + a_2^2 + a_3^2).$$

We immediately recognise the sinc function, which is exactly the transfer function of a single window of width $T_r$. The co-efficient in front, $M(f)$, is a result of the 'interference' of the three windows' phase factors that implement the time domain translation, in the Fourier domain. Conveniently, this modulation of the single window sinc function depends only on the total cycle time. So henceforth we need only consider $T_r$ and $T_d$ (the total dead time $T_d = T_p + T_m$).

Next the choice of the amplitudes of the windows are completely open. An actual measurement in practice corresponds to $a_i = 1$, but because we have access to the

stored separate measurements we are free to scale the integral by simple computation (namely multiplication).

In what follows we show that indeed non-trivial windows are helpful in combating aliasing. In practice the actual $a_i$'s could be optimized relative to real life power spectral densities or even experimentally optimized. Many real life power spectral densities have spikes that protrude out of the normal $1/f$ or $1/f^2$ noise profiles. By varying the $a_i's$ through a numerical optimization procedure it may very well be possible to do better than the windows we study next.

We choose three of the configurations based on the previous sinc discussion and the rest are chosen according to symmetry considerations to try and cover the typical behaviours. Firstly, we choose a single window case and two representative double windows. For the three non-trivial windows scenario, we start by considering the signs of the amplitudes. There are $2 \times 3 \times 2$ possibilities (allowing for the middle window to be zero, which is a valid separate three window configuration, different from the two window cases). Since the modulation expression, Eq. (7.3), is invariant under multiplication of $-1$ and/or swopping the first and the last amplitudes, we first remove the 'multiply-by-minus-1' duplications (which halves the number of possibilities). Thereafter, we remove the one remaining swop and multiply-by-minus-1 invariance duplication, which leaves us with five unique sign allocations. Using these sign allocations, we then symmetrically assign amplitudes which conveniently also leads to the simplest trigonometric expressions. It turns out, by virtue of the symmetry imposition that the produced expressions are extreme cases of the modulation patterns, perfect for testing a finite number of cases. For one of the sign allocations where horizontal symmetry is difficult to achieve $(-++)$, we permit two representatives. All in all we have chosen eleven non-trivial window configurations, to see if at least one of them can beat the single window case.

In Table 7.1 we list twelve window configurations. The second column is the modulation expression, $M(f)$ and the third column is the assignment of amplitudes to achieve that configuration.

| # | $M(f)$Modulation | $\{a_1, a_2, a_3\}$ |
|---|---|---|
| 1 | $1$ | $\{0, 0, 1\}$ |
| 2 | $4\cos^2(\pi cf)$ | $\{0, 1, 1\}$ |
| 3 | $4\sin^2(\pi cf)$ | $\{0, -1, 1\}$ |
| 4 | $4\sin^2(2\pi cf)$ | $\{-1, 0, 1\}$ |
| 5 | $4\sin^4(\pi cf)$ | $\{\frac{1}{2}, -1, \frac{1}{2}\}$ |
| 6 | $4\cos^2(2\pi cf)$ | $\{1, 0, 1\}$ |
| 7 | $4\cos^4(\pi cf)$ | $\{\frac{1}{2}, 1, \frac{1}{2}\}$ |
| 8 | $2 - \cos(4\pi cf)$ | $\left\{-\frac{1}{\sqrt{2}}, 1, \frac{1}{\sqrt{2}}\right\}$ |
| 9 | $3 - 2\cos(4\pi cf)$ | $\{-1, 1, 1\}$ |
| 10 | $-1.56\cos(2\pi cf) - 0.44\cos(4\pi cf) + 2.0484$ | $\{-0.22, -1, 1\}$ |
| 11 | $1.56\cos(2\pi cf) - 0.44\cos(4\pi cf) + 2.0484$ | $\{-0.22, 1, 1\}$ |
| 12 | $0.377\cos(2\pi cf) - 0.1168\cos(4\pi cf) + 0.6234$ | $\{-0.08, 0.29, 0.73\}$ |

Table 7.1: A Listing of Different Window Designs and their Modulations.

### 7.3.3 Window Performance

In the first column of Fig. 7.12 we plot a pictorial representation of the time profile of the window configuration. In the second column we plot the green sinc function superimposed by the blue modulation curve. The third column contains plots of the modulated sinc (the blue modulation curve times the green sinc) in a bluish-green colour. This plot also shows the aliasing that kicks in at $1/2T_c$. Beyond that interval, the multiplied modulated sinc is depicted in a dashed red curve, which is also folded in to the $(0, 1/2T_c)$ interval, in solid red, representing aliasing. These window aliasing terms still need to be multiplied by the aliased noise before representing the final alias noise contribution.

In the fourth column we plot the $1/f$ noise in orange and the multiplication of the modulated sinc and the noise in a light pea-green (the colour coming from mixing blue, green and orange). This multiplication represents the noise correction on the interval $(0, 1/2T_c)$. In red we depict the sum of all the noise alias terms. When the red curve reaches a non-zero value on the $y$-axis it constitutes DC aliasing, which is called the Dick Effect.

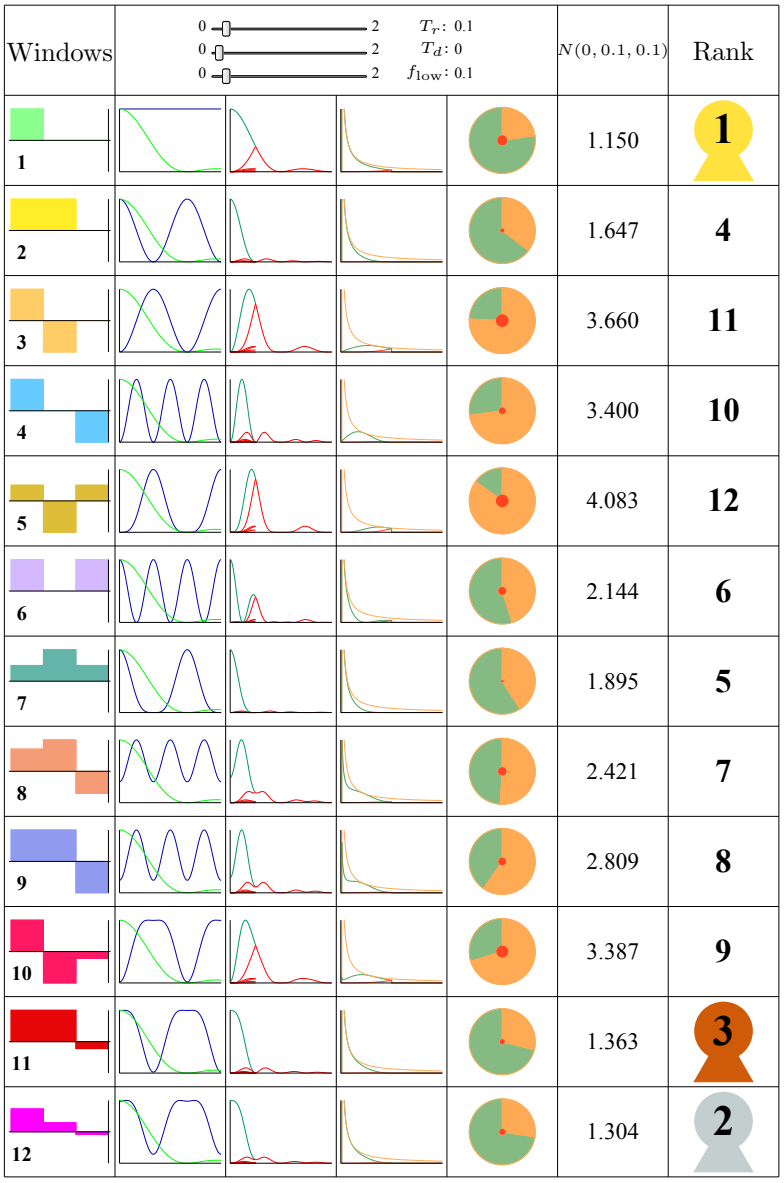



Figure 7.12: Comparison Run 1.

In the fifth column the integral of the correction is presented in a pea-green slice of the pie chart, while the full orange pie is the total noise. The exposed orange sector is the uncorrected noise. Superimposed, at the center of the pie, the aliased noise is depicted as a filled in red circle that is added to the total noise. All the areas of the respective parts are true to the ratio of represented quantities. In the fifth column we list the

value of the remaining noise $N(T_d, T_r, f_{\text{low}})$, which was introduced earlier except that we are now more interested in the low frequency cut-off $f_{\text{low}}$ than the high frequency cut-off (which was fixed to $f_{\text{hi}} = 10$ since the relative behaviour does not change much for higher $f_{\text{hi}}$, in fact there is just a common increase in uncorrected noise because the sinc quickly modulates all windows to zero). Finally, in the last column we list the rank of the different windows according to their performance as measured by the remaining noise where naturally, lower is better.

To re-iterate, the performance is measured by $N(T_d, T_r, f_{\text{low}})$ which is the final noise after correction. This includes the noise left uncorrected and the noise introduced due to aliasing calculated as the total noise minus the corrected amount, plus the aliased noise.

The low frequency cut-off is introduced so that transfer functions that do not go to one as $f \to 0$, can still compete with those windows that do. This is not too unrealistic, as at worst we are not too interested in fluctuations with periods longer than the age of the universe. In fact, it can also be argued that extremely low frequency fluctuations at extremely high amplitudes are ultimately unphysical. Besides giving certain windows a fair comparison, there can be cases of noise models where there are spikes at intermediate frequencies. For these cases, transfer functions that peak at later intermediate frequencies may perform better than the usual windows. Finally, on this point, from a cynical point of view, since the industry measure of the atomic clock performance is the Allan variance, certain window configurations that resemble the Allan variance window (specifically, for example, 3 and 4), focus their efforts of correction in the regions where the Allan variance is most sensitive. These window configurations may actually outperform the other window configurations as measured by the Allan variance with or without the low frequency cut-off. Since, this is an 'unfair' advantage, we do not choose the Allan variance as our measure of performance, but rather the total final noise power, $N(T_d, T_r, f_{\text{low}})$[‡‡].

[‡‡] An 'industry', where artificial optimization is tolerated, is the super-computing world and the Linpack test. Here, while such optimizations favour certain architectures unfairly, its simplicity, wide adoption

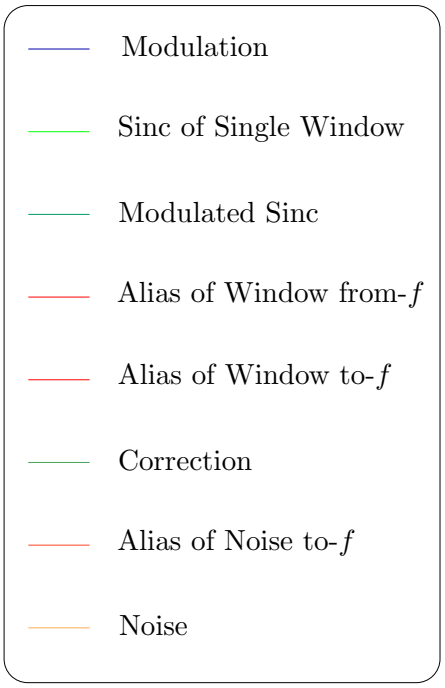



Table 7.2: Legend for all the Performance Comparison Figures.

Note, in column three that the transfer function is normalized. This gives windows of initially low amplitude a better chance of capturing the noise. This normalization in the Fourier domain corresponds to a simple global scaling of the measurement results in the time domain by linearity of the Fourier transform.

To summarize the items in the plot, we refer to the legend in Table 7.2.

### 7.3.4 $T_d$, $T_r$ and $f_{\text{low}}$ Dependence

Now that we have set up the different configurations, we are ready to see which windows fair the best in the $(T_d, T_r)$ plane and also as $f_{\text{low}}$ is changed.

In Fig. 7.13, we calculate $N(T_d, T_r, f_{\text{low}})$ for each window and select the best window. We then colour that $(T_d, T_r)$ point with the colour of the winning window. The colour and numbering of the winning window aligns with Fig. 7.12. We also repeat the 2D plot for different $f_{\text{low}}$.

and 'arm's race'-inducing reference point has perhaps been 'good' for the industry.

The first observation is that indeed different windows are better suited for different settings of $T_d$ and $T_r$, this essentially means that the multiple window construction and effort has been worth it. Of course, as mentioned previously questions of correlation time and synchronicity need to be experimentally checked in the laboratory to confirm that past measurements do have a bearing on future measurements.

We also observe that there is a strong dependence on $f_{\text{low}}$. This is because for higher $f_{\text{low}}$, the intermediate $T_d$'s and $T_r$'s in the range of the plots, start to cause the sampling cut-off to approach the low frequency cut-off. As this happens, the different profiles, that happen to have the largest overlap with the non-zero region, win the competition for best correction. This does not seem to show any merit in superior performance. However, the result can be viewed as showing that different profiles better target different portions of the noise spectrum, while suppressing aliasing better. Therefore the prospect is still there in a practical setting, that by searching through the different windows for a certain given noise profile and minimum dead and Ramsey times, improvements could be had over the regular simple single window.

Notice for $f_{\text{low}} = 0.01$ most of the area is still won by the single window. This does indeed make sense since the single window has a modulation of constant one. Therefore for cases where overall sensitivity is desirable, the single window should win. Using Fig. 7.13 we can select specific $T_d$'s and $T_r$'s to replot versions of Fig. 7.12 to try and better understand how certain windows beat the single window. In Fig. 7.14 - Fig. 7.16 we choose points where different windows beat the standard. In all the cases it seems that the other windows do better by aliasing less. Indeed the effect is quite marked, especially in Fig. 7.14 and Fig. 7.16, where the alias suppression is so successful that the Dick Effect is almost perfectly eliminated.

### 7.3.5 Given $T_d$, Finding an Optimal $T_r$

While the above analysis tells us which window is best for each $(T_d, T_r)$ pair, it does not compare the performance between pairs. We need to make plots similar to Fig. 7.6 and Fig. 7.9. In Fig. 7.17, we plot the total aliasing for all the windows in one plot.

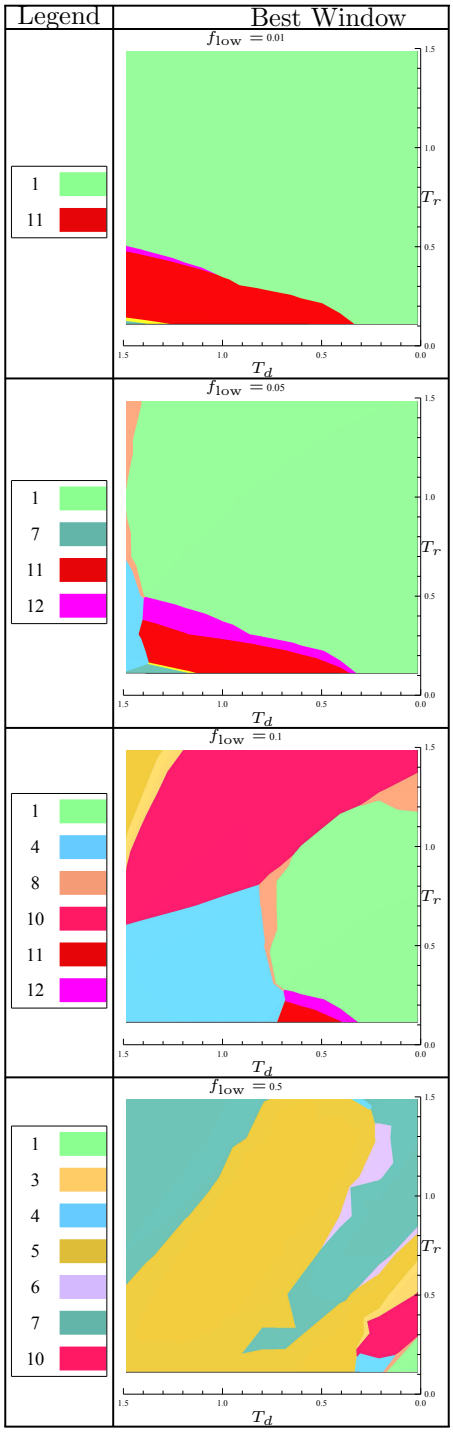



Figure 7.13: Selecting the Best Window over $(T_r, T_d)$.

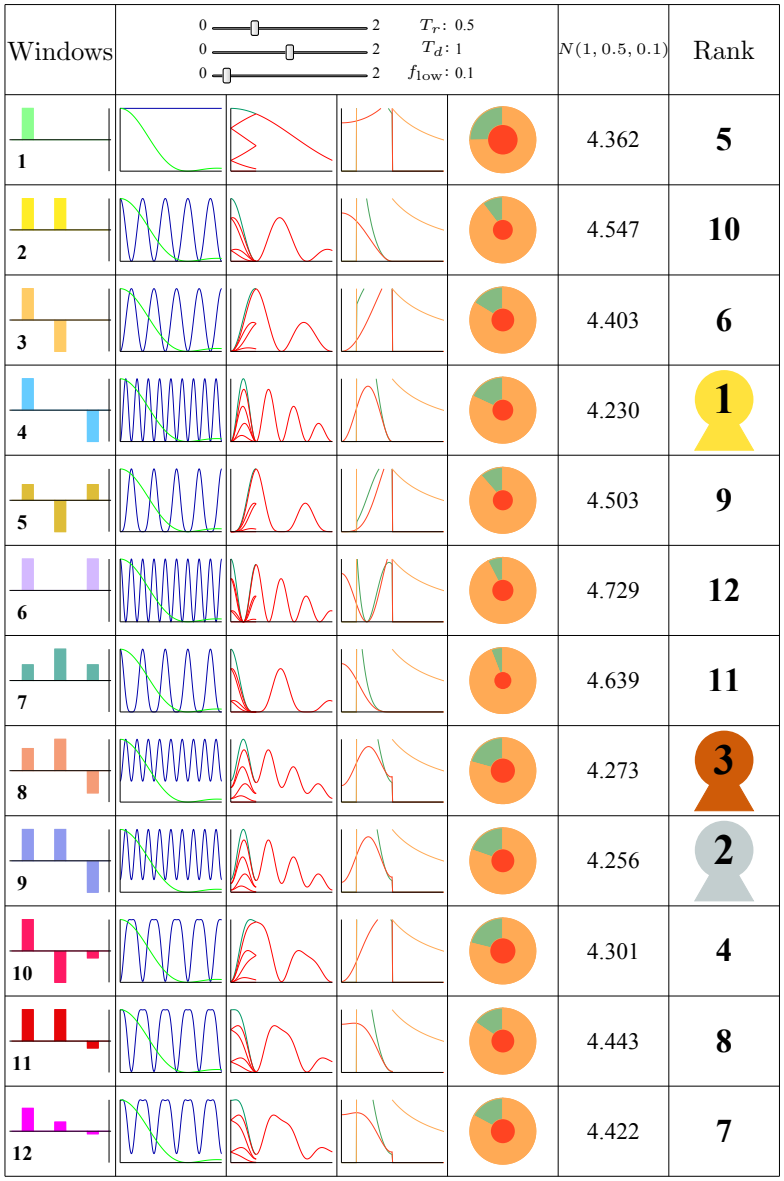



Figure 7.14: Comparison Run 2.

The height, $A_{1/f}$, of each point is the total aliasing power, calculated by integrating the aliased mis-correction from $1/2T_c$ to infinity. We colour the different surfaces according to the same window labelling scheme used throughout this section. Not surprisingly, we see that the single window (light green) has the worst aliasing because it has no modulation. It is reassuring to see that this plot is very similar to Fig. 7.6. In Fig.

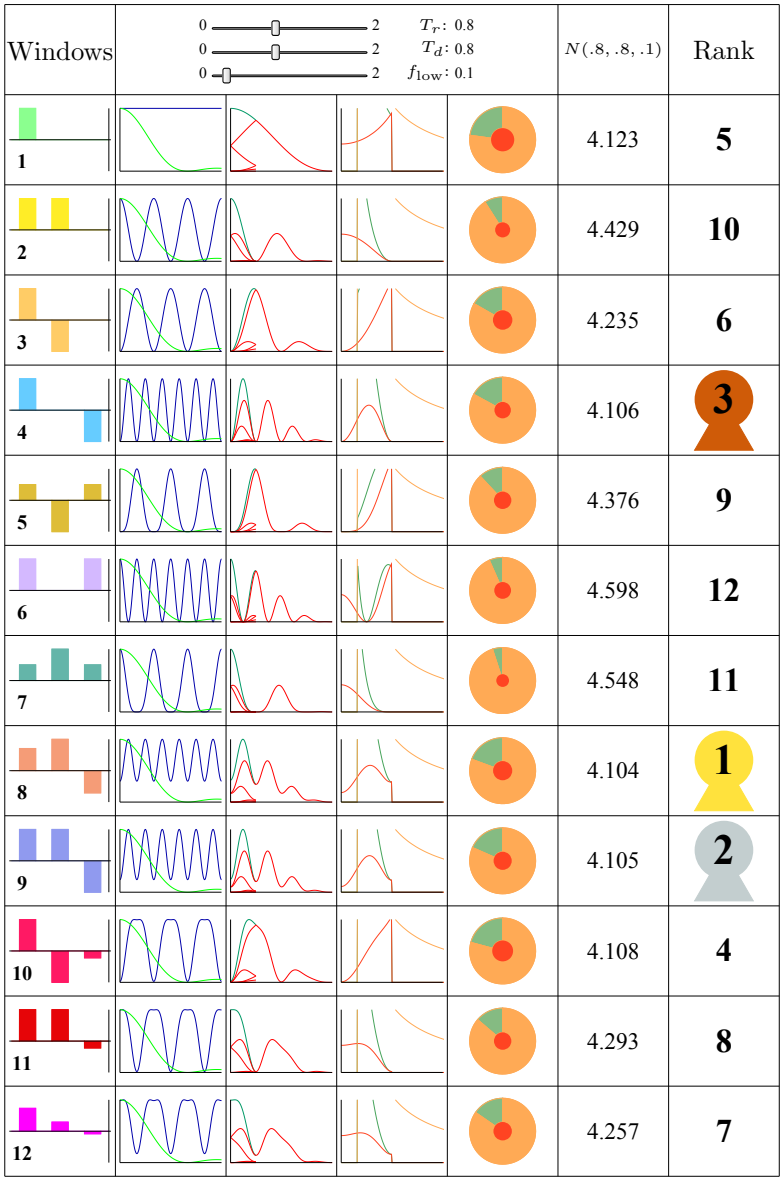



Figure 7.15: Comparison Run 3.

7.18 we plot the performance $N(T_r, T_d)$ at each point and for each window, allowing the surfaces to overlap. To reiterate, the performance is measured by the final noise which is calculated by subtracting the correction from the total noise and adding aliasing. Since smaller final noise is better, we flipped over the graph such that $z$-axis runs from high to low in the upward direction. The best performing window now colours the upward

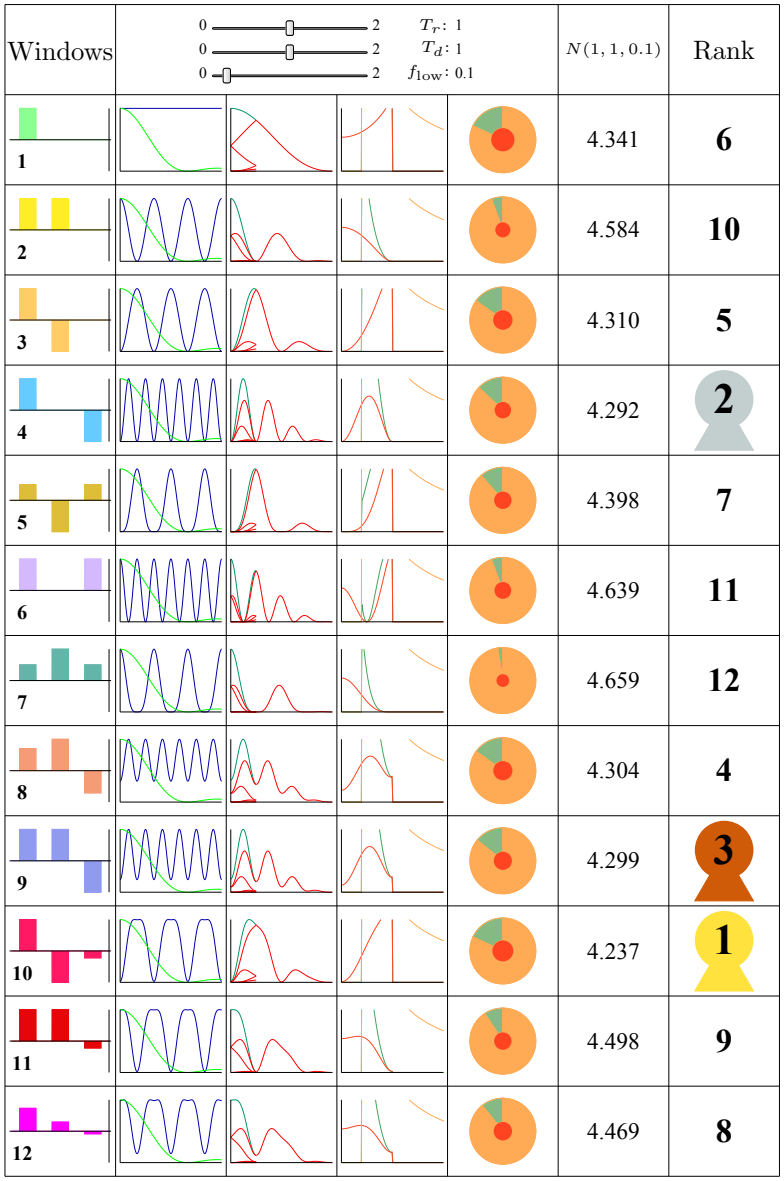



Figure 7.16: Comparison Run 4.

facing surface. Fig. 7.18 is very similar to Fig. 7.10. We see the same lip behaviour as for the $1/f$ noise. We see that for a certain low range of $T_d$, smaller $T_r$ is better. But beyond a minimum $T_d$ it turns out that a higher $T_r$ is actually better. In a practical setup, the region of acceptable parameters must first be identified on the graph. The direction of the gradient can thus be followed to find an optimum for the parameters.

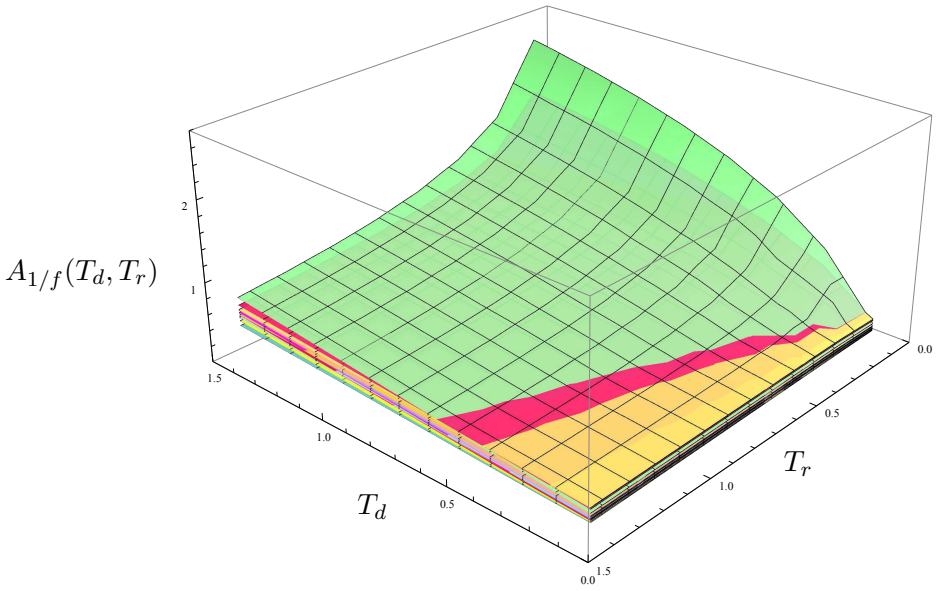



Figure 7.17: Total Aliasing Comparison.

Once a local maximum is found, the colour of that spot is noted and where possible the corresponding window can be used to correct for the error. Fig. 7.18 does, of course, take aliasing into. However, the Dick Effect is very different from general aliasing in that it surfaces as a *constant DC shift* of the ideal frequency. $S_C(f)$ is the power spectral density of the corrections and $S_C(0)$ is the Dick Effect. However, the final noise measure integrates the power spectral density, which implies that any single point 'makes no contribution' or if we count a small region around zero as part of the Dick Effect then $S_C(0)$ does not make any special contribution to the final noise measure. As a result of its extra undesirability it is perhaps helpful to multiply $S_C(0)$ by some constant and add this as an extra penalty to the measure of performance. This can be viewed as counting the power around the zero frequency 'twice' to emphasize our dislike for it. In Fig. 7.19, we plot the best performers as measured by the total noise left in the clock plus an extra penalty for the Dick Effect. By the substantial change of the colouring, compared to Fig. 7.13, we see that the best performer heavily relies upon the criteria of the best, that is how it is defined. The actual practical determination of the criteria is likely to be made on a case-by-case basis.

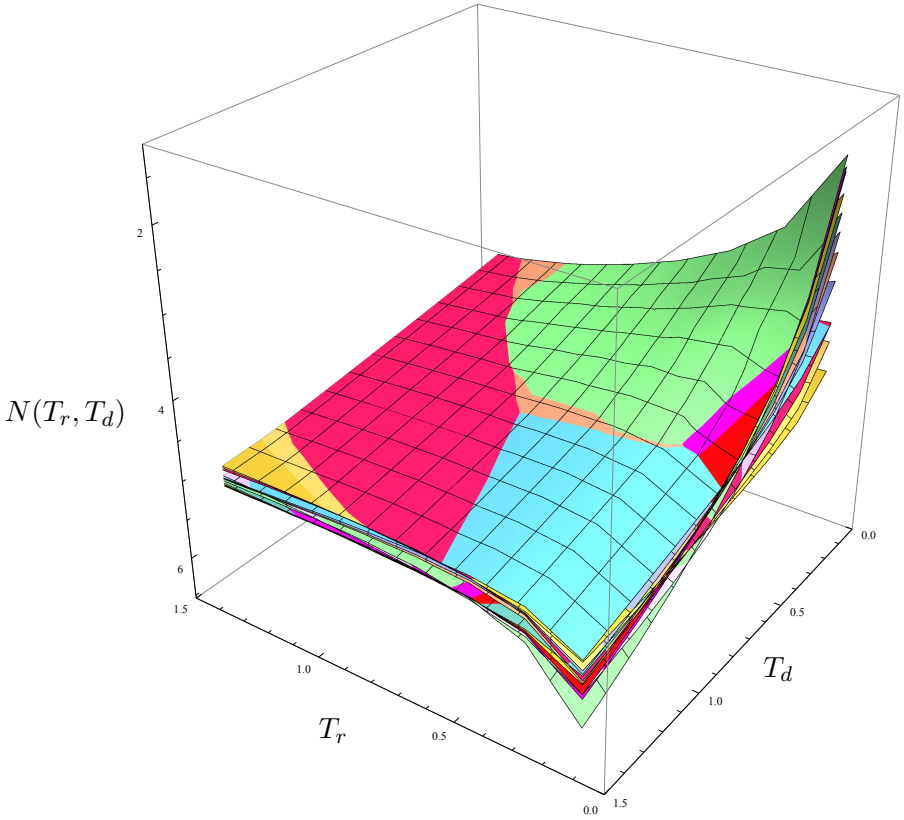



Figure 7.18: Overall performance (z axis rotated).

## 7.4 Conclusion

We have explored in detail the noise processes of an atomic clock slaved to a quartz crystal. We have equipped ourselves with powerful mathematical tools to model the noise as well as quantify errors in the noise correction protocol.

To combat some of the errors, we have proposed an extension to the standard single window protocol. We successfully demonstrated that the multiple window extension can be advantageous over the standard window in certain cases.

Further work would include extending the number of windows to more than three, to perhaps more closely follow a sinc function in the time domain. Numerical simulations should also be undertaken to confirm that the many window protocol does indeed improve the instantaneous correction. Finally a more general scheme of combined frequency

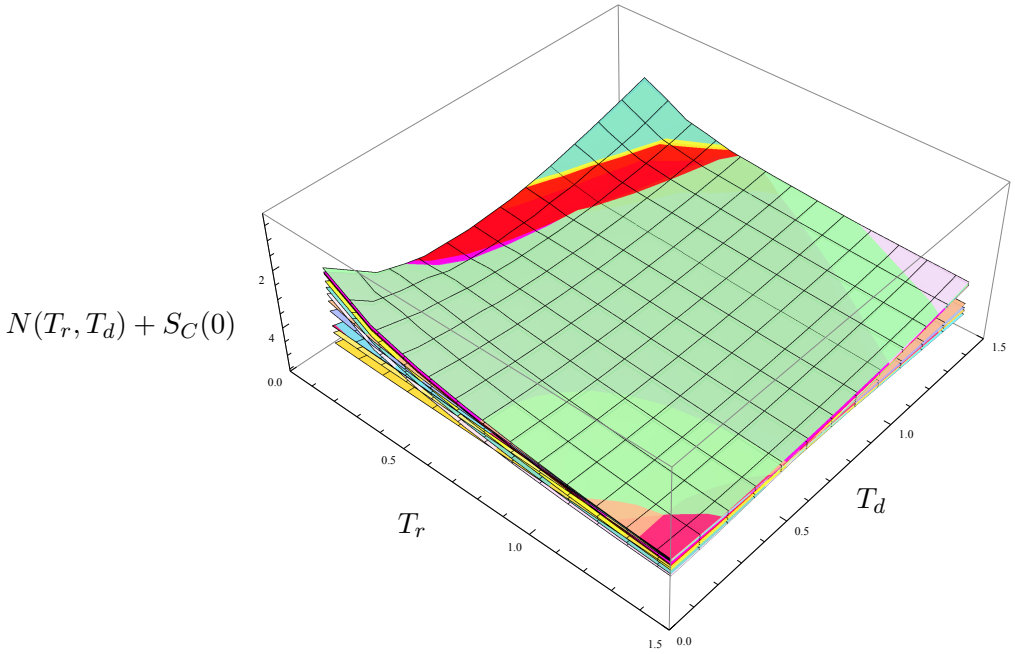



Figure 7.19: Overall performance with Dick Penalty (z axis rotated).

and phase correction needs to be devised, tested and shown to be superior to current methods.

# Bibliography


[1] *Local oscillator induced instabilities in trapped ion frequency standards*, G. Dick (Proc. Precise Time and Time Interval, pg. 133-147; 1987)

[2] *Frequency stability degradation of an oscillator slaved to a periodically interrogated atomic resonator*, G. Santarelli, C. Audoin, A. Makdissi, P. Laurent, G.J. Dick and A. Clairon (IEEE Trans. Ultra-son. Ferroelectr. Freq. Contro; Vol. 45; pg. 887-894; 1998)

[3] *The Dick effect for an optical frequency standard*, A. Quessada, R.P. Kovacich, I. Courtillot, A. Clairon, G. Santarelli, and P. Lemonde (J. Opt. B; Vol. 5; pg. S150-S154; 2003)

[4] *Minimizing the Dick Effect in an Optical Lattice Clock*, P.G. Westergaard, J. Lodewyck, and P. Lemonde (IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, Vol. 57; No. 3; 2010)

[5] *A Derivation of the Long-Term Degradation of a Pulsed Atomic Frequency Standard from a Control-Loop Model*, C.A. Greenhall (IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control; Vol 45; No 4; 1998)

[6] *Predictive Feedforward Protocols for Passive Frequency Standards*, J Sastrawan (The University of Sydney)

[7] *Active Control of Sound and Vibration*, C.R. Fuller and A.H. von Flotow (IEEE Control Systems; 1995)

# Chapter 8

# Conclusion

## 8.1 Different Mechanisms to Model and Tools to Measure Randomness

### 8.1.1 Channel Capacity Noise Correlations

In the chapters concerning channel capacity (Ch. 3, Ch. 5), we used the mechanism of functions of Markov processes to model our non-trivial noise with memory. The random channel selection was Markovian, but in combination with the indistinguishability of the actual sub-channel used, made the memory or correlations in the noise, non-Markovian. Thus not only did we introduce correlations in the noise, but the correlations were distinctly more complicated than the usual Markovian or random walk noise. However, the noise was not intractably complicated and indeed the channel was *still* forgetful. Forgetfulness is related to the property of stationarity, an assumption we made for the atomic clock noise which allowed us to use the extended HSW theorem.

The tool to measure the adverse effects of correlations in the random noise was the capacity (Ch. 2). This 'measure' of noise obviously has in mind only a specific use, namely communication, but it does suggest the future possibility of bringing capacity measures to other noise scenarios. In calculating the capacity we made use of the powerful tool of algebraic measure theory. This framework specifically concerns functions of hidden Markov processes and so cannot be used for more general noise modelling without some form of generalization. As it stands though, the process of calculating the entropy was

already quite involved, which indicates that capturing the effects of correlations is a complicated task. Indeed, in the Monte Carlo chapter (Ch. 4), when we plotted the raw eigenvalues before calculating the entropy, we saw fractal like patterns brought about by the correlations.

Overall, we showed that the channel capacity increases with stronger correlations.

### 8.1.2 Atomic Clock Noise Correlations

In the atomic clock background chapter (Ch. 6) we used the mechanism of a continuous random process, its autocorrelation and the Fourier transform squared of the autocorrelation to arrive at the *power spectral density*. In this framework, specifying the power spectral density in the Fourier domain is related to specifying the correlations of the noise at different times. This tool is very general and it may prove profitable in future work to introduce channels whose noise is given by a power spectral density.

We can already make a connection between the noise introduced in the channel capacity setting and the power spectral density framework. We would easily be able to calculate the autocorrelation of the channel switching depolarizing noise and then take the Fourier transform squared. We know what not to expect. Markovian or Brownian noise corresponds to a $1/f^2$ noise power spectral density, while a constant power spectral density corresponds to uncorrelated noise. Thus $1/f$ noise is an example of correlated noise that is distinct from Markovian noise and for the switching depolarizing noise, we would expect a profile similarly non-Markovian.

In the atomic clock setting (Ch. 7), we went on to measure the effects of the noise, by considering sampling aliasing while correcting the frequency of a slaved quartz crystal clock. The measure of total noise was then taken to be the area under the integral of the original noise profile minus the correction's profile plus the aliasing profile. Thus the tool of measurement was integration and subtraction of the relevant noise power spectral densities, yielding the total power of the noise after correction.

We used this tool to characterize the performance of novel correcting strategies based on overlapping windows. We found parameter regions where the performance of our

non-standard windows improved on the standard single pulse performance, essentially by taking into account correlations between windows in order to dampen high frequency fluctuation sensitivity and their associated aliasing contributions.

This ability to assess strategies could potentially be used in the channel capacity setting to help design the book of code-words. The HSW theorem and Shannon's original coding theorem's are infamous for their teasingly non-constructive proof of the existence of good codes, without providing a clue on how to go about finding them.

### 8.1.3 Monte Carlo Algorithmic Use of Correlations

Finally, in the Monte Carlo Simulation chapter (Ch. 5), which has the main purpose of confirming the findings of the algebraic measure approach and once done, indirectly confirming the validity of the use of the MCMC algorithm. We were fortunately able to achieve both goals and are now in a position to apply Monte Carlo methods to more complicated correlation scenarios. Besides studying correlations in client systems, the MCMC algorithm is itself a fascinating example of the algorithmic use of manufactured correlation. We constructed a Markov chain to have a stationary distribution matching the probability distribution for which we wanted to calculate the entropy. Hence the mechanism of generating correlations was the simple Markov process. We then drew samples from this distribution by taking a random walk, thereby efficiently calculating the relevant statistic we were after. We studied the effects of correlations between the steps of the random walk. Our measure of performance was the variance of the statistic and we found that too low a correlation or too high a correlation degraded the efficiency of sampling. Hence, here too, in this abstract algorithmic space, correlation became a very important parameter that affected the workings of the desired task. Even though the entire algorithm hinged on the use of intermediate strength correlations, when using draws, we had to remove traces of the correlations. Thus, here final correlations were undesirable.

## 8.2 Concluding Thoughts

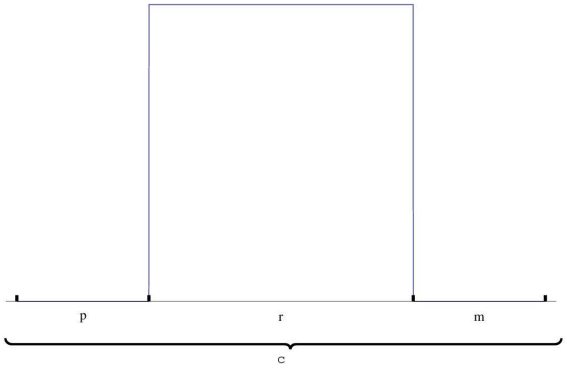
This thesis explored different theoretical frameworks of modelling and managing randomness and correlations in randomness. It has become clear that correlations in randomness plays a very important role in how that randomness effects different systems.
Considering the sophistication of the mathematical tools used, as a microscopic on correlations, we can reflect on how it seems that correlations are involved in very complicated ways. Future analysis could well benefit from the invention of new mathematical tools. Nevertheless, we have learnt that correlations are important practically and we wonder if it can shed light on deeper fundamental questions as well.

# Appendix

# Single Window Aliasing

## Sensitivity

```
sensitivity =
 Underoverscript[Underoverscript[Show[Plot[UnitBox[t + 1], {t, -2, 0}, Axes → {True, False},
     Ticks → {{{-2, ""}, {-1.5, ""}, {-0.5, ""}, {0, ""}}}, TicksStyle → Thick],
    Graphics[{Text["p", {-1.75, -0.05}] , Text["r", {-1, -0.05}], Text["m", {-0.25, -0.05}]}],
    PlotRange → {-0.1, 1}], Style[⎵, 30], ""], "c", ""]
```

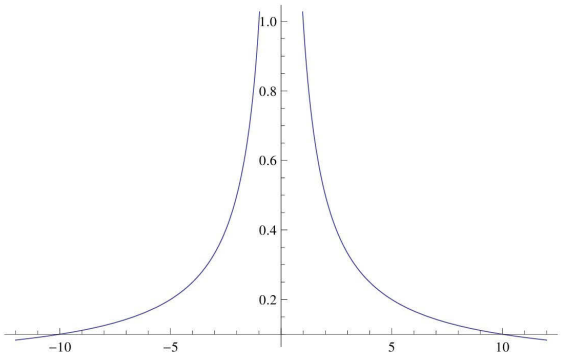



```
Export["/home/researcher/Work/Research/Topics_Collaborators/Atomic/sensitivity.eps", sensitivity]
```

```
"/home/researcher/Work/Research/Topics_Collaborators/Atomic/sensitivity.eps"
```

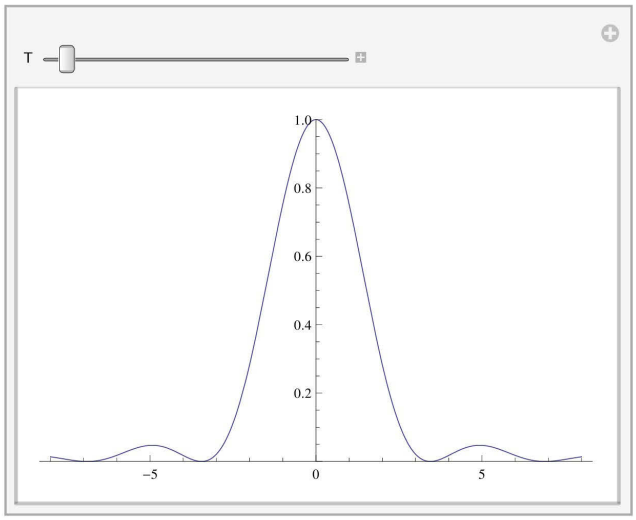
## 1/f Noise

```
Plot[1/Abs[f], {f, -12., 12.}]
```

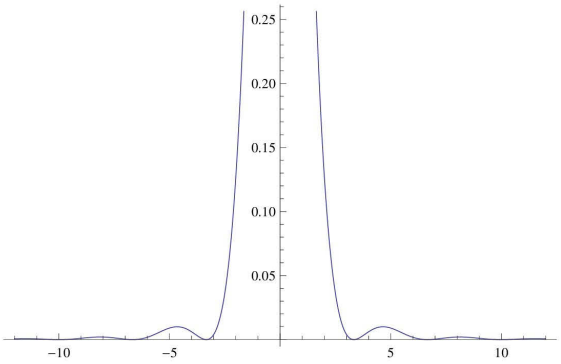


## Sinc^2 in Frequency Domain (T period in time domain)

```
Manipulate[Plot[Sinc[π f / (1 / T)]^2, {f, -8, 8}, PlotRange → {0, 1}], {T, 0.01, 8}]
```



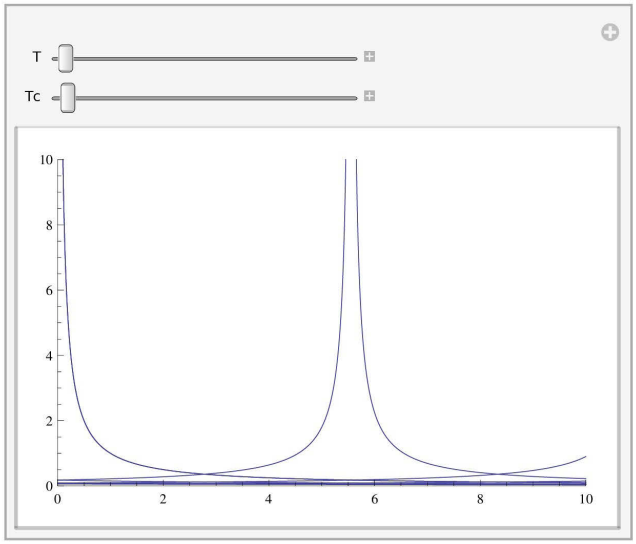

```
Plot[Sinc[f π 0.3]^2 / Abs[f], {f, -12., 12.}]
```



# Aliasing due to Sampling (Tc is the Cycle Time, T is the Ramsey Sensitive time)

## The Sinc Multiplies the 1/f Noise

```
Manipulate[
 Plot[Join[{1 / Abs[f]}, Table[Sinc[Pi (f + n / Tc) T]^2 h1 / Abs[f + n / Tc], {n, -3, 3}]] /. {h1 → 1},
  {f, 0, 10}, PlotRange → {{0, 10}, {0, 10}}], {T, 0, Tc}, {Tc, 0.1, 10}]
```



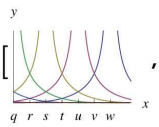

```
grphs = Table[Sinc[Pi (f + n / Tc) / (1 / T)]^2 1 / Abs[f + n / Tc], {n, -3, 3}]
```

$$\left\{\frac{\text{Sinc}\left[\pi T \left(f-\frac{3}{Tc}\right)\right]^2}{\text{Abs}\left[f-\frac{3}{Tc}\right]}, \frac{\text{Sinc}\left[\pi T \left(f-\frac{2}{Tc}\right)\right]^2}{\text{Abs}\left[f-\frac{2}{Tc}\right]}, \frac{\text{Sinc}\left[\pi T \left(f-\frac{1}{Tc}\right)\right]^2}{\text{Abs}\left[f-\frac{1}{Tc}\right]},\right.$$

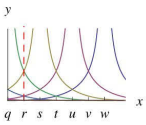
$$\left.\frac{\text{Sinc}[f \pi T]^2}{\text{Abs}[f]}, \frac{\text{Sinc}\left[\pi T \left(f+\frac{1}{Tc}\right)\right]^2}{\text{Abs}\left[f+\frac{1}{Tc}\right]}, \frac{\text{Sinc}\left[\pi T \left(f+\frac{2}{Tc}\right)\right]^2}{\text{Abs}\left[f+\frac{2}{Tc}\right]}, \frac{\text{Sinc}\left[\pi T \left(f+\frac{3}{Tc}\right)\right]^2}{\text{Abs}\left[f+\frac{3}{Tc}\right]}\right\}$$

```
Manipulate[
 Plot[{Sinc[π T (f - 3/Tc)]^2 / Abs[f - 3/Tc], Sinc[π T (f - 2/Tc)]^2 / Abs[f - 2/Tc],
   Sinc[π T (f - 1/Tc)]^2 / Abs[f - 1/Tc], Sinc[f π T]^2 / Abs[f], Sinc[π T (f + 1/Tc)]^2 / Abs[f + 1/Tc],
   Sinc[π T (f + 2/Tc)]^2 / Abs[f + 2/Tc], Sinc[π T (f + 3/Tc)]^2 / Abs[f + 3/Tc]}, {f, 0, 10}, PlotRange → {{0, 4}, {0, 4}},
  AxesLabel → {x, y}, Ticks → {{{0, q}, {1 / (2 Tc), r}, {1 / (Tc), s}, {3 / (2 Tc), t},
     {2 / (Tc), u}, {5 / (2 Tc), v}, {3 / (Tc), w}}, None}], {T, 0.2, Tc}, {Tc, 0.9, 10}]
```

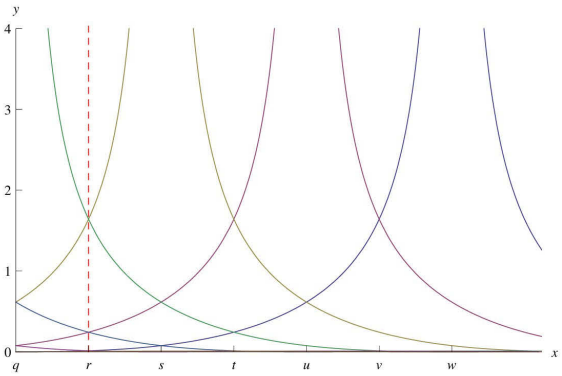
```
Show[
```



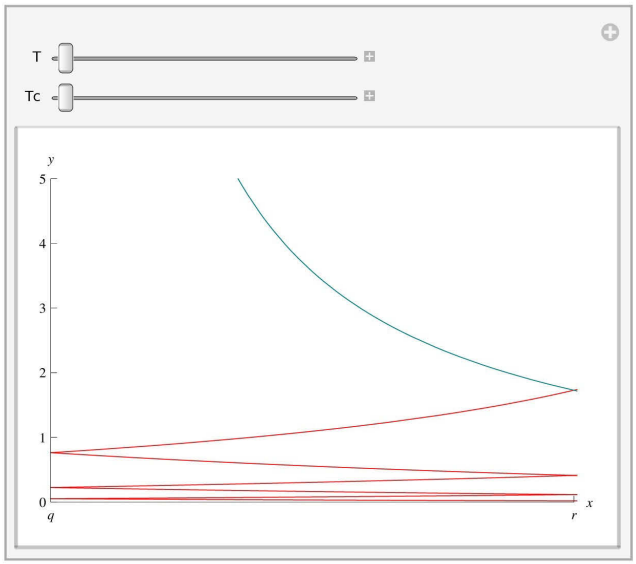

```
, Graphics[{Red, Dashed, Line[{{1 / 1.8, 0}, {1 / 1.8, 10}}]}]]
```



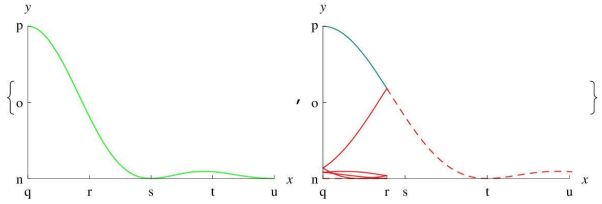

```
/. (Ticks → {a_, b_}) → Ticks → {a, {0, 1, 2, 3, 4, 5}}
```



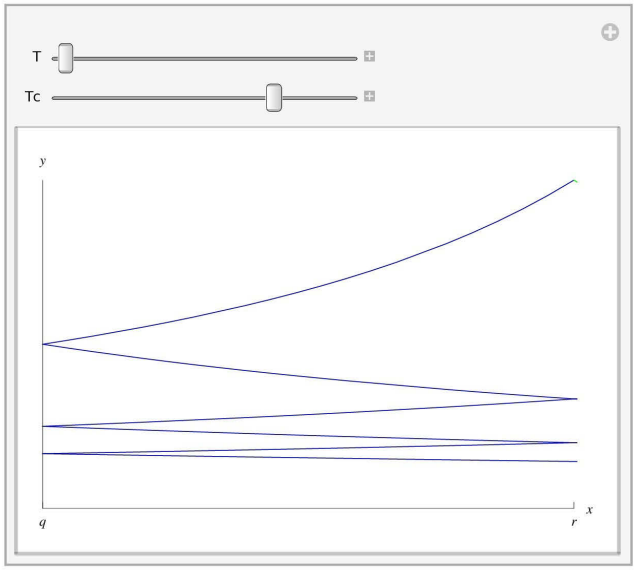

```
Manipulate[Plot[{Sinc[π T (f - 3/Tc)]^2/Abs[f - 3/Tc], Sinc[π T (f - 2/Tc)]^2/Abs[f - 2/Tc],
   Sinc[π T (f - 1/Tc)]^2/Abs[f - 1/Tc], Sinc[f π T]^2/Abs[f],
   Sinc[π T (f + 1/Tc)]^2/Abs[f + 1/Tc], Sinc[π T (f + 2/Tc)]^2/Abs[f + 2/Tc],
   Sinc[π T (f + 3/Tc)]^2/Abs[f + 3/Tc]},
  {f, 0, 10}, PlotStyle → {Red, Red, Red, Blend[{Green, Blue}], Red, Red, Red},
  PlotRange → {{0, 1 / (2 × 0.9)}, {0, 5}}, AxesLabel → {x, y},
  Ticks → {{{0, q}, {1 / (2 Tc), r}, {1 / (Tc), s}, {3 / (2 Tc), t}, {2 / (Tc), u},
     {5 / (2 Tc), v}, {3 / (Tc), w}}, {0, 1, 2, 3, 4, 5}}], {T, 0.2, Tc}, {Tc, 0.9, 10}]
```



```
ReleaseHold[
 Hold[{Plot[#, {f, 0, 2 / r}, PlotRange → {{0, 2 / r}, {0, 1}}, PlotStyle → {Darker[Blue], Green},
       AxesLabel → {x, y}, AxesOrigin → {0, 0}, Ticks → {{{0, "q"}, {1 / (2 r), "r"},
          {1 / (r), "s"}, {3 / (2 r), "t"}, {2 / (r), "u"}, {5 / (2 r), "v"}, {3 / (r), "w"}},
         {{0, "n"}, {0.5, "o"}, {1, "p"}}}] &[NormTransSincedFwindows[d, r][[1, 1]]],
    (Show[Plot[#, {f, 0, 1 / (2 (d + r))}, PlotStyle → ({Blend[{Green, Blue}]} ~Join~ Table[Red,
           {n, -numTrans, numTrans - 1}]), PlotRange → {{0, 3 / (2 r)}, {0, 1}}, Ticks → None], Plot[
        #[[1]] HeavisideTheta[f - 1 / (2 (d + r))], {f, 0, 2 / r}, PlotStyle → Directive[Dashed, Red],
        PlotRange → {{0, 3 / (2 r)}, {0, 1}}, Ticks → None], AxesLabel → {x, y},
       Ticks → {{{0, "q"}, {1 / (2 Tc), "r"}, {1 / (2 r), "s"}, {1 / (r), "t"}, {3 / (2 r), "u"},
          {2 / (r), "v"}, {5 / (2 r), "w"}, {3 / (r), "z"}}, {{0, "n"}, {0.5, "o"}, {1, "p"}}}]) &[
     NormTransSincedFwindows[d, r][[1]]]}] //. {Tc → (d + r), d → 0.2, r → 0.7}]
```




```
Manipulate[Plot[{Sinc[π T (f - 3/Tc)]^2 / Abs[f - 3/Tc], Sinc[π T (f - 2/Tc)]^2 / Abs[f - 2/Tc],
   Sinc[π T (f - 1/Tc)]^2 / Abs[f - 1/Tc], Sinc[f π T]^2 / Abs[f],
   Sinc[π T (f + 1/Tc)]^2 / Abs[f + 1/Tc], Sinc[π T (f + 2/Tc)]^2 / Abs[f + 2/Tc],
   Sinc[π T (f + 3/Tc)]^2 / Abs[f + 3/Tc]}, {f, 0, 10}, PlotStyle →
  {Darker[Blue], Darker[Blue], Darker[Blue], Green, Darker[Blue], Darker[Blue], Darker[Blue]},
  PlotRange → {{0, 1 / (2 Tc)}, {0, 2 Tc}}, AxesLabel → {x, y},
  Ticks → {{{0, q}, {1 / (2 Tc), r}, {1 / (Tc), s}, {3 / (2 Tc), t},
     {2 / (Tc), u}, {5 / (2 Tc), v}, {3 / (Tc), w}}, None}], {T, 0, Tc}, {Tc, 0.9, 10}]
```

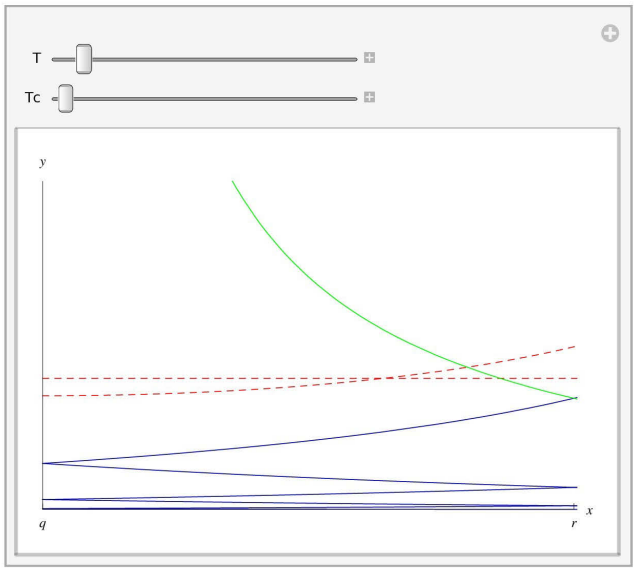



```
Remove[alias]
```

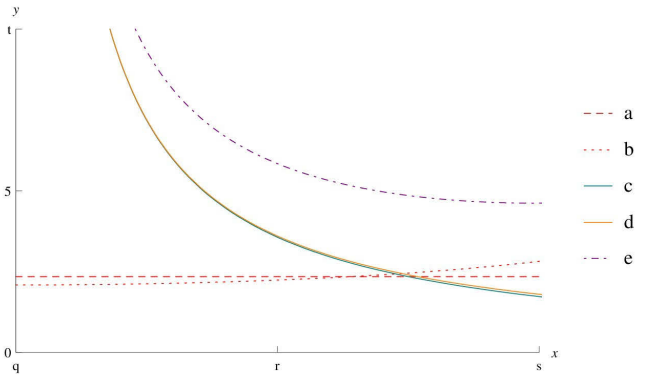
```
alias[T_, Tc_] =
 Plus @@ (Sinc[Pi (f + n / Tc) / (1 / T)]^2 h1 / Abs[f + n / Tc] /. n → {-5, -4, -3, -2, -1, 1, 2, 3, 4, 5}) /.
  h1 → 1
```

$$\frac{\text{Sinc}\left[\pi T \left(f - \frac{5}{Tc}\right)\right]^2}{\text{Abs}\left[f - \frac{5}{Tc}\right]} + \frac{\text{Sinc}\left[\pi T \left(f - \frac{4}{Tc}\right)\right]^2}{\text{Abs}\left[f - \frac{4}{Tc}\right]} + \frac{\text{Sinc}\left[\pi T \left(f - \frac{3}{Tc}\right)\right]^2}{\text{Abs}\left[f - \frac{3}{Tc}\right]} + \frac{\text{Sinc}\left[\pi T \left(f - \frac{2}{Tc}\right)\right]^2}{\text{Abs}\left[f - \frac{2}{Tc}\right]} + \frac{\text{Sinc}\left[\pi T \left(f - \frac{1}{Tc}\right)\right]^2}{\text{Abs}\left[f - \frac{1}{Tc}\right]} +$$
$$\frac{\text{Sinc}\left[\pi T \left(f + \frac{1}{Tc}\right)\right]^2}{\text{Abs}\left[f + \frac{1}{Tc}\right]} + \frac{\text{Sinc}\left[\pi T \left(f + \frac{2}{Tc}\right)\right]^2}{\text{Abs}\left[f + \frac{2}{Tc}\right]} + \frac{\text{Sinc}\left[\pi T \left(f + \frac{3}{Tc}\right)\right]^2}{\text{Abs}\left[f + \frac{3}{Tc}\right]} + \frac{\text{Sinc}\left[\pi T \left(f + \frac{4}{Tc}\right)\right]^2}{\text{Abs}\left[f + \frac{4}{Tc}\right]} + \frac{\text{Sinc}\left[\pi T \left(f + \frac{5}{Tc}\right)\right]^2}{\text{Abs}\left[f + \frac{5}{Tc}\right]}$$

```
Manipulate[Plot[
 {1/(2 π^2 T^2) (-2 Tc^2 (-1 + Cos[π T / Tc]) + π T (π T (-2 CosIntegral[π T / Tc] + Log[1/T^2] + 2 Log[T]) + 2 Tc Sin[π T / Tc]))/
   (1 / (2 Tc)), Sinc[π T (f - 3/Tc)]^2 / Abs[f - 3/Tc] + Sinc[π T (f - 2/Tc)]^2 / Abs[f - 2/Tc] + Sinc[π T (f - 1/Tc)]^2 / Abs[f - 1/Tc] + Sinc[π T (f + 1/Tc)]^2 / Abs[f + 1/Tc] +
  Sinc[π T (f + 2/Tc)]^2 / Abs[f + 2/Tc] + Sinc[π T (f + 3/Tc)]^2 / Abs[f + 3/Tc], Sinc[π T (f - 3/Tc)]^2 / Abs[f - 3/Tc], Sinc[π T (f - 2/Tc)]^2 / Abs[f - 2/Tc],
  Sinc[π T (f - 1/Tc)]^2 / Abs[f - 1/Tc], Sinc[f π T]^2 / Abs[f], Sinc[π T (f + 1/Tc)]^2 / Abs[f + 1/Tc], Sinc[π T (f + 2/Tc)]^2 / Abs[f + 2/Tc], Sinc[π T (f + 3/Tc)]^2 / Abs[f + 3/Tc]},
 {f, 0, 10}, PlotStyle → {{Dashed, Red}, {Dashed, Red}, Darker[Blue], Darker[Blue],
   Darker[Blue], Green, Darker[Blue], Darker[Blue], Darker[Blue]},
 PlotRange → {{0, 1 / (2 × 0.9)}, {0, 5}}, AxesLabel → {x, y},
 Ticks → {{{0, q}, {1 / (2 Tc), r}, {1 / (Tc), s}, {3 / (2 Tc), t}, {2 / (Tc), u},
    {5 / (2 Tc), v}, {3 / (Tc), w}}, None}], {T, 0.2, Tc}, {Tc, 0.9, 10}]
```



```
Manipulate[Plot[
 {1/(2 π^2 T^2) (-2 Tc^2 (-1 + Cos[π T/Tc]) + π T (π T (-2 CosIntegral[π T/Tc] + Log[1/T^2] + 2 Log[T]) + 2 Tc Sin[π T/Tc]))/
    (1 / (2 Tc)), alias[T, Tc], Sinc[f π T]^2/Abs[f], 1 / Abs[f], 1 / Abs[f] + alias[T, Tc]}, {f, 0, 10},
  PlotStyle → {{Dashed, Red}, {Dotted, Red}, Blend[{Green, Blue}], Orange, {DotDashed, Purple}},
  PlotRange → {{0, 1 / (2 × 0.9)}, {0, 10}}, AxesLabel → {x, y}, Ticks →
   {{{0, q}, {1 / (2 Tc), r}, {1 / (Tc), s}, {3 / (2 Tc), t}, {2 / (Tc), u}, {5 / (2 Tc), v}, {3 / (Tc), w}},
    None}, PlotLegends → {"a", "b", "c", "d", "e"}], {T, 0.2, Tc}, {Tc, 0.9, 10}]

Module[{Tc = 0.9`, T = 0.2`},
 Plot[{1/(2 π^2 T^2/(2 Tc)) (-2 Tc^2 (-1 + Cos[π T/Tc]) + π T (π T (-2 CosIntegral[π T/Tc] + Log[1/T^2] + 2 Log[T]) + 2 Tc Sin[π T/Tc])),
   alias[T, Tc], Sinc[f π T]^2/Abs[f], 1/Abs[f], 1/Abs[f] + alias[T, Tc]}, {f, 0, 10},
  PlotStyle → {{Dashed, Red}, {Dotted, Red}, Blend[{Green, Blue}], Orange, {DotDashed, Purple}},
  PlotRange → {{0, 1/(2 × 0.9`)}, {0, 10}}, AxesLabel → {x, y},
  Ticks → {{{0, "q"}, {1 / (4 Tc), "r"}, {1/(2 Tc), "s"}}, {0, 5, {10, "t"}}},
  PlotLegends → {"a", "b", "c", "d", "e"}]]
```




```
Export[
 "/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/sum_alias.eps", %]
/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/sum_alias.eps
```

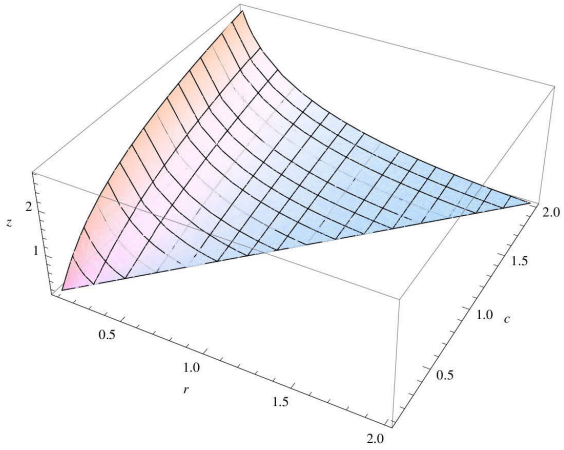
## Error in Aliasing 1/f (for fixed Tc, bigger Tr is better, for fixed Tr smaller Tc is better)

```
Integrate[(Sinc[Pi (f) / (1 / T)]^2 h1 / Abs[f]),
 {f, 1 / (2 Tc), Infinity}, Assumptions :→ Element[{T, h1, Tc}, Reals]]
ConditionalExpression[
 1/(2 π^2 T^2) h1 (-2 Tc^2 (-1 + Cos[π T/Tc]) + π T (π T (-2 CosIntegral[π T/Tc] + Log[1/T^2] + 2 Log[T]) + 2 Tc Sin[π T/Tc])),
 Tc == 0 || (Tc ≥ 0 && T ≥ 0)]
```

```
FullSimplify[1/(2 π^2 T^2) h1
  (-2 Tc^2 (-1 + Cos[π T/Tc]) + π T (π T (-2 CosIntegral[π T/Tc] + Log[1/T^2] + 2 Log[T]) + 2 Tc Sin[π T/Tc])) /. h1 → 1]
```

$$\frac{1}{2\pi^2 T^2}\left(-2\,Tc^2\left(-1+\cos\left[\frac{\pi T}{Tc}\right]\right)+\pi T\left(\pi T\left(-2\,\text{CosIntegral}\left[\frac{\pi T}{Tc}\right]+\log\left[\frac{1}{T^2}\right]+2\log[T]\right)+2\,Tc\sin\left[\frac{\pi T}{Tc}\right]\right)\right)$$

```
totalAliasing = Plot3D[
  1/(2 π^2 T^2) (-2 Tc^2 (-1 + Cos[π T/Tc]) + π T (π T (-2 CosIntegral[π T/Tc] + Log[1/T^2] + 2 Log[T]) + 2 Tc Sin[π T/Tc])),
  {T, 0.1, 2}, {Tc, 0.1, 2}, AxesLabel → {r, c, z}, Axes → {True, True, True},
  ClippingStyle → None, PlotRange → All, RegionFunction → Function[{x, y, z}, x < y]]
```

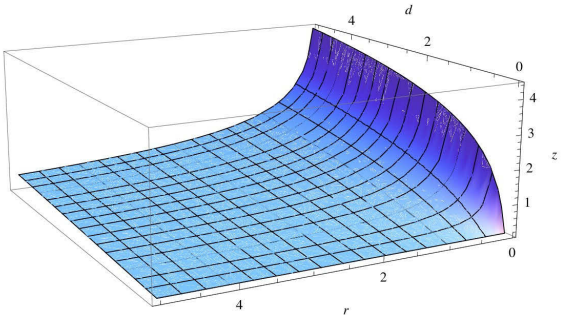



```
1/π^2 r^2 (-(d + r)^2 (-1 + Cos[π r/(d + r)]) + π r (-π r CosIntegral[π r/(d + r)] + (d + r) Sin[π r/(d + r)])) /. {d → T_d, r → T_r}
```

$$\frac{1}{\pi^2 T_r^2}\left(-\left(-1+\cos\left[\frac{\pi T_r}{T_d+T_r}\right]\right)(T_d+T_r)^2+\pi T_r\left(-\pi\,\text{CosIntegral}\left[\frac{\pi T_r}{T_d+T_r}\right]T_r+\sin\left[\frac{\pi T_r}{T_d+T_r}\right](T_d+T_r)\right)\right)$$

```
Plot3D[1/π^2 r^2 (-(d + r)^2 (-1 + Cos[π r/(d + r)]) + π r (-π r CosIntegral[π r/(d + r)] + (d + r) Sin[π r/(d + r)])), {d, 0, 5},
  {r, 0.05, 5}, AxesLabel → {d, r, z}, Axes → {True, True, True}, ClippingStyle → None, PlotRange → All]
```

aliastotalerror =



```
Limit[
 1/(2 π^2 T^2) (-2 Tc^2 (-1 + Cos[π T/Tc]) + π T (π T (-2 CosIntegral[π T/Tc] + Log[1/T^2] + 2 Log[T]) + 2 Tc Sin[π T/Tc])), T → 0]
```

$$\infty$$

```
Limit[
 1/(2 π^2 T^2) (-2 Tc^2 (-1 + Cos[π T/Tc]) + π T (π T (-2 CosIntegral[π T/Tc] + Log[1/T^2] + 2 Log[T]) + 2 Tc Sin[π T/Tc])) /.
  Tc → T, T → 0]
```

$$\frac{2}{\pi^2} - \text{CosIntegral}[\pi]$$

## Error in 1/f for f < 1/2 Tc (Failure to capture noise on correcting interval)

```
Integrate[(Sinc[Pi f r]^2 / Abs[f]), {f, 1 / (2 (d + r)), Infinity}, Assumptions :→ Element[{d, r}, Reals]]
```

$$\begin{cases} \frac{1}{\pi^2 r^2} i \left(-i d^2 - 2 i d r - i r^2 + \pi^3 r^2 + i d^2 \text{Cos}\left[\frac{\pi r}{d+r}\right] + 2 i d r \text{Cos}\left[\frac{\pi r}{d+r}\right] + \right. & d + r > 0 \&\& r < 0 \\ \quad \left. i r^2 \text{Cos}\left[\frac{\pi r}{d+r}\right] + i \pi^2 r^2 \text{CosIntegral}\left[\frac{\pi r}{d+r}\right] - i d \pi r \text{Sin}\left[\frac{\pi r}{d+r}\right] - i \pi r^2 \text{Sin}\left[\frac{\pi r}{d+r}\right]\right) & \\ \frac{1}{\pi^2 r^2}\left(d^2 + 2 d r + r^2 - d^2 \text{Cos}\left[\frac{\pi r}{d+r}\right] - 2 d r \text{Cos}\left[\frac{\pi r}{d+r}\right] - \right. & d + r > 0 \&\& r \geq 0 \\ \quad \left. r^2 \text{Cos}\left[\frac{\pi r}{d+r}\right] - \pi^2 r^2 \text{CosIntegral}\left[\frac{\pi r}{d+r}\right] + d \pi r \text{Sin}\left[\frac{\pi r}{d+r}\right] + \pi r^2 \text{Sin}\left[\frac{\pi r}{d+r}\right]\right) & \\ 0 & d + r == 0 \\ \text{Integrate}\left[\frac{\text{Sinc}[f \pi r]^2}{\text{Abs}[f]}, \left\{f, \frac{1}{2 (d+r)}, \infty\right\}, \text{Assumptions} \to (d \mid r) \in \text{Reals} \&\& d + r < 0\right] & \text{True} \end{cases}$$

```
Limit[1/(π^2 r^2) (d^2 + 2 d r + r^2 - d^2 Cos[π r/(d + r)] - 2 d r Cos[π r/(d + r)] - r^2 Cos[π r/(d + r)] -
      π^2 r^2 CosIntegral[π r/(d + r)] + d π r Sin[π r/(d + r)] + π r^2 Sin[π r/(d + r)]) /. d → r, {r → 0}] // N
```

$$\{0.569904\}$$

```
Integrate[1 / f - Sinc[Pi (f) r]^2 / f, {f, l, 1 / (2 (d + r))},
 Assumptions :→ And[r > 0, r > 0, 0 < l < 1 / (2 (d + r))]]
```

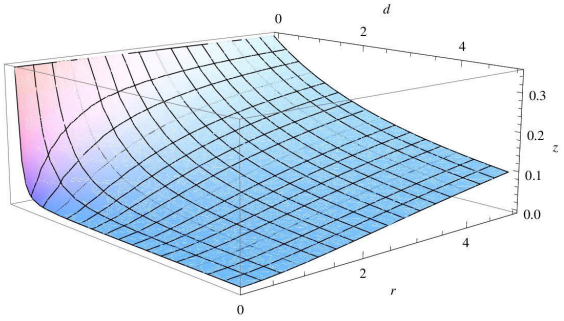
$$-\text{Log}[2\, l\, (d + r)] - \frac{1}{4 \pi^2 r^2}\left(-4 d^2 + \frac{1}{l^2} - 8 d r - 4 r^2 - \frac{\text{Cos}[2 l \pi r]}{l^2} + 4 d^2 \text{Cos}\left[\frac{\pi r}{d + r}\right] + 8 d r \text{Cos}\left[\frac{\pi r}{d + r}\right] + 4 r^2 \text{Cos}\left[\frac{\pi r}{d + r}\right] - 4 \pi^2 r^2 \text{CosIntegral}[2 l \pi r] + 4 \pi^2 r^2 \text{CosIntegral}\left[\frac{\pi r}{d + r}\right] + \frac{2 \pi r \text{Sin}[2 l \pi r]}{l} - 4 d \pi r \text{Sin}\left[\frac{\pi r}{d + r}\right] - 4 \pi r^2 \text{Sin}\left[\frac{\pi r}{d + r}\right]\right)$$

```
FullSimplify[%81]
```

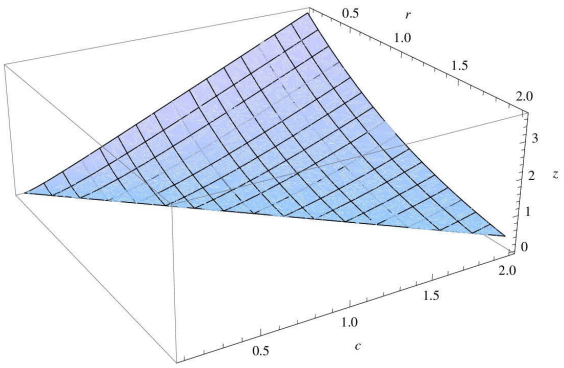
$$\frac{1}{4 l^2 \pi^2 r^2}\left((-1 + 2 l (d + r))(1 + 2 l (d + r)) + \text{Cos}[2 l \pi r] - 4 l^2 (d + r)^2 \text{Cos}\left[\frac{\pi r}{d + r}\right] + 2 l \pi r \left(-\text{Sin}[2 l \pi r] + 2 l \left(\pi r \text{CosIntegral}[2 l \pi r] - \pi r \left(\text{CosIntegral}\left[\frac{\pi r}{d + r}\right] + \text{Log}[2 l (d + r)]\right) + (d + r) \text{Sin}\left[\frac{\pi r}{d + r}\right]\right)\right)\right)$$

```
Limit[%, l → 0]
```

```
Plot3D[
 1/(2 π^2 r^2) (2 d^2 + 4 d r + 2 r^2 - 3 π^2 r^2 + 2 EulerGamma π^2 r^2 - 2 (d + r)^2 Cos[π r/(d + r)] - 2 π^2 r^2 CosIntegral[π r/(d + r)] +
    2 π^2 r^2 Log[π] + 2 π^2 r^2 Log[r] - 2 π^2 r^2 Log[d + r] + 2 d π r Sin[π r/(d + r)] + 2 π r^2 Sin[π r/(d + r)]),
 {d, 0, 5}, {r, 0.1, 5}, AxesLabel → {d, r, z}, Axes → {True, True, True},
 ClippingStyle → None, PlotRange → All]
```



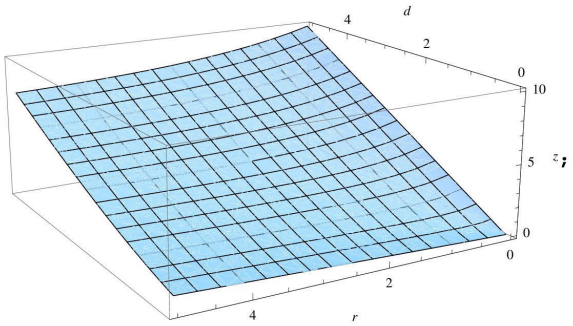

## Error in Aliasing 1/f^2 (for fixed Tc bigger Tr is better, for fixed Tr smaller Tc is better, seems best when Tc=Tr)




```
Integrate[(Sinc[Pi f r]^2 / f^2), {f, 1 / (2 (d + r)), Infinity}, Assumptions :→ Element[{d, r}, Reals]]
ConditionalExpression[1/(3 π^2 r^2)
 (-π^4 r^2 Abs[r] + 2 (-(d + r) ((2 d^2 + 4 d r - (-2 + π^2) r^2) Cos[π r/(d + r)] - (d + r) (2 (d + r) + π r Sin[π r/(d + r)])) +
    π^3 r^3 SinIntegral[π r/(d + r)])), d + r ≥ 0]
```

```
FullSimplify[1/(3 π^2 r^2)
  (-π^4 r^2 Abs[r] + 2 (-(d + r) ((2 d^2 + 4 d r - (-2 + π^2) r^2) Cos[π r/(d + r)] - (d + r) (2 (d + r) + π r Sin[π r/(d + r)])) +
      π^3 r^3 SinIntegral[π r/(d + r)])), Assumptions :> {r > 0}] /. {d → T_d, r → T_r}
1/(3 π^2 T_r^2) (-π^4 T_r^3 + 2 (π^3 SinIntegral[π T_r/(T_d + T_r)] T_r^3 +
    (T_d + T_r) (Cos[π T_r/(T_d + T_r)] (-2 T_d^2 - 4 T_d T_r + (-2 + π^2) T_r^2) + (T_d + T_r) (π Sin[π T_r/(T_d + T_r)] T_r + 2 (T_d + T_r)))))

Plot3D[1/(3 π^2 r^2)
  (-π^4 r^2 Abs[r] + 2 (-(d + r) ((2 d^2 + 4 d r - (-2 + π^2) r^2) Cos[π r/(d + r)] - (d + r) (2 (d + r) + π r Sin[π r/(d + r)])) +
      π^3 r^3 SinIntegral[π r/(d + r)])), {d, 0, 5}, {r, 0.001, 5},
 AxesLabel → {d, r, z}, Axes → {True, True, True}, ClippingStyle → None, PlotRange → All]
```

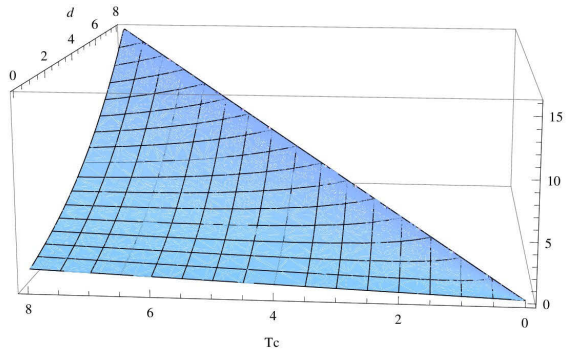
aliastotalerrorf2 =



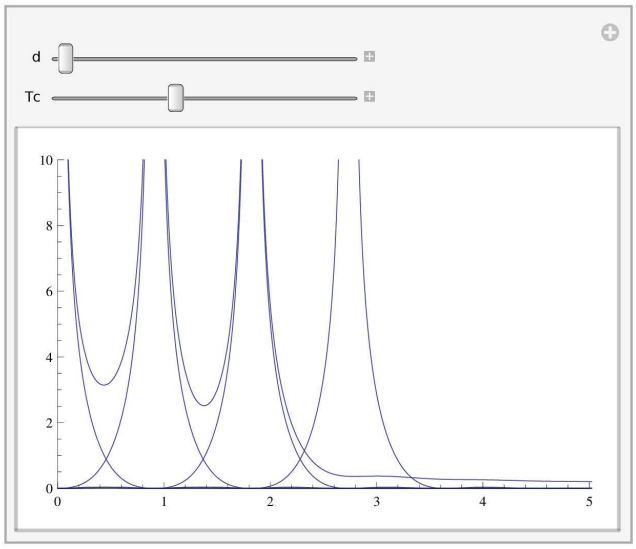

;

```
Limit[1/(3 π^2 r^2)
  (-π^4 r^2 Abs[r] + 2 (-(d + r) ((2 d^2 + 4 d r - (-2 + π^2) r^2) Cos[π r/(d + r)] - (d + r) (2 (d + r) + π r Sin[π r/(d + r)])) +
      π^3 r^3 SinIntegral[π r/(d + r)])), r → 0]
2 d
```

## Error in Aliasing 1/f^2 in terms of Tc and d (for fixed Tc, smaller d=bigger Tr is better, for fixed d larger Tc is slightly better, seems best when Tr=Tc)




```
Manipulate[Plot[
  Join[{1 / Abs[f] + Plus @@ (Sinc[Pi (f + n / Tc) / (1 / T)]^2 h1 / Abs[f + n / Tc] /. n → {-2, -1, 1, 2})},
    Table[Sinc[Pi (f + n / Tc) / {1 / T}]^2 h1 / Abs[f + n / Tc], {n, -3, 3}]] /. {h1 → 1, T → Tc - d},
  {f, 0, 10}, PlotRange → {{0, 5}, {0, 10}}], {d, 0, Tc}, {Tc, 0.5, 2}]
```



## Error in failing to capture 1/f^2 for f<1/2Tc (for fixed Tc smaller Tr is better, for fixed Tr bigger Tc is slightly better, here the trend is to keep Tc and Tr apart)

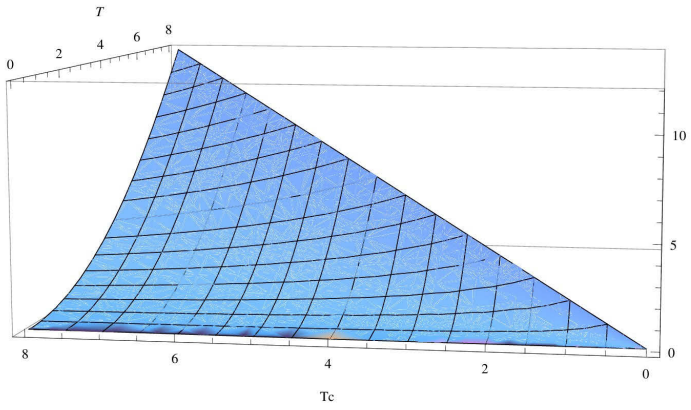
```
Integrate[1 / f^2 - (Sinc[Pi (f) / {1 / T}]^2 ) / f^2,
 {f, 1, 1 / (2 Tc)}, Assumptions :> And[T > 0, Tc > 0, 0 < 1 < 1 / (2 Tc)]]
```

$$\left\{\frac{1}{1}-2\,Tc-\frac{1}{6\,1^3\,\pi^2\,T^2}\left(1-8\,1^3\,Tc^3+\left(-1+2\,1^2\,\pi^2\,T^2\right)\,\text{Cos}\left[2\,1\,\pi\,T\right]+4\,1^3\,Tc\,\left(-\pi^2\,T^2+2\,Tc^2\right)\,\text{Cos}\left[\frac{\pi\,T}{Tc}\right]+\right.\right.$$
$$\left.\left.1\,\pi\,T\left(\text{Sin}\left[2\,1\,\pi\,T\right]-4\,1^2\,Tc^2\,\text{Sin}\left[\frac{\pi\,T}{Tc}\right]\right)+4\,1^3\,\pi^3\,T^3\left(\text{SinIntegral}\left[2\,1\,\pi\,T\right]-\text{SinIntegral}\left[\frac{\pi\,T}{Tc}\right]\right)\right)\right\}$$

```
Limit[Out[5], 1 → 0][[1]]
```

$$\frac{1}{3\,\pi^2\,T^2}\,2\left(Tc\left(-3\,\pi^2\,T^2+2\,Tc^2+\left(\pi^2\,T^2-2\,Tc^2\right)\,\text{Cos}\left[\frac{\pi\,T}{Tc}\right]+\pi\,T\,Tc\,\text{Sin}\left[\frac{\pi\,T}{Tc}\right]\right)+\pi^3\,T^3\,\text{SinIntegral}\left[\frac{\pi\,T}{Tc}\right]\right)$$

```
Plot3D[1 / (3 π^2 T^2) 2 (Tc (-3 π^2 T^2 + 2 Tc^2 + (π^2 T^2 - 2 Tc^2) Cos[π T / Tc] + π T Tc Sin[π T / Tc]) + π^3 T^3 SinIntegral[π T / Tc]),
 {T, 0, 8}, {Tc, 0, 8}, AxesLabel → {T, Tc}, RegionFunction → Function[{x, y, z}, x < y]]
```



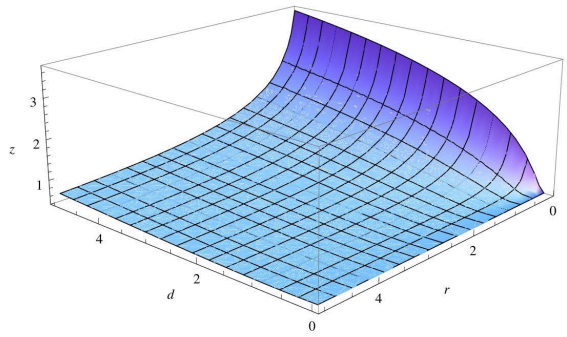

```
Integrate[1 / f - Sinc[Pi (f) r]^2 / f,
  {f, 1, 1 / (2 (d + r))}, Assumptions :> And[r > 0, r > 0, 0 < 1 < 1 / (2 (d + r))]] +
 1 / (π^2 r^2) (d^2 + 2 d r + r^2 - d^2 Cos[π r / (d + r)] - 2 d r Cos[π r / (d + r)] - r^2 Cos[π r / (d + r)] -
   π^2 r^2 CosIntegral[π r / (d + r)] + d π r Sin[π r / (d + r)] + π r^2 Sin[π r / (d + r)])
- Log[2 1 (d + r)] -
 1 / (4 π^2 r^2) (-4 d^2 + 1 / 1^2 - 8 d r - 4 r^2 - Cos[2 1 π r] / 1^2 + 4 d^2 Cos[π r / (d + r)] + 8 d r Cos[π r / (d + r)] + 4 r^2 Cos[π r / (d + r)] -
   4 π^2 r^2 CosIntegral[2 1 π r] + 4 π^2 r^2 CosIntegral[π r / (d + r)] +
   2 π r Sin[2 1 π r] / 1 - 4 d π r Sin[π r / (d + r)] - 4 π r^2 Sin[π r / (d + r)]) +
 1 / (π^2 r^2) (d^2 + 2 d r + r^2 - d^2 Cos[π r / (d + r)] - 2 d r Cos[π r / (d + r)] - r^2 Cos[π r / (d + r)] -
   π^2 r^2 CosIntegral[π r / (d + r)] + d π r Sin[π r / (d + r)] + π r^2 Sin[π r / (d + r)])
```

```
FullSimplify[%76]
```

$$\frac{1}{4\,l^2\,\pi^2\,r^2}\left(-1+8\,l^2\,(d+r)^2+\text{Cos}[2\,l\,\pi\,r]-8\,l^2\,(d+r)^2\,\text{Cos}\left[\frac{\pi\,r}{d+r}\right]+2\,l\,\pi\,r\left(2\,l\,\pi\,r\left(\text{CosIntegral}[2\,l\,\pi\,r]-2\,\text{CosIntegral}\left[\frac{\pi\,r}{d+r}\right]-\text{Log}[2\,l\,(d+r)]\right)-\text{Sin}[2\,l\,\pi\,r]+4\,l\,(d+r)\,\text{Sin}\left[\frac{\pi\,r}{d+r}\right]\right)\right)$$

```
Limit[%, l → 0]
```

```
Plot3D[
 1/(2 π^2 r^2) (4 d^2 + 8 d r + 4 r^2 - 3 π^2 r^2 + 2 EulerGamma π^2 r^2 - 4 (d + r)^2 Cos[π r/(d + r)] - 4 π^2 r^2 CosIntegral[π r/(d + r)] +
    2 π^2 r^2 Log[2 π r] - 2 π^2 r^2 Log[2 (d + r)] + 4 d π r Sin[π r/(d + r)] + 4 π r^2 Sin[π r/(d + r)]),
 {d, 0, 5}, {r, 0.1, 5}, AxesLabel → {d, r, z}, Axes → {True, True, True},
 ClippingStyle → None, PlotRange → All]
```

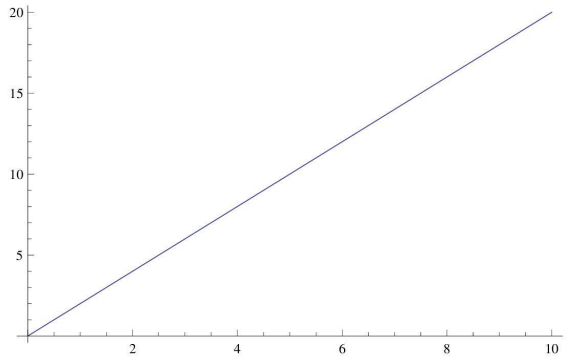



## Error in failing to capture 1/f^2 for f > 1/2 Tc (independent of Tr, always better to have as small a Tc as possible)

```
Integrate[1 / f^2 , {f, 1 / (2 Tc), Infinity}, Assumptions :→ Tc > 0]
```

2 Tc

```
Plot[Integrate[1 / f^2 , {f, 1 / (2 Tc), Infinity}, Assumptions :→ Tc > 0], {Tc, 0, 10}]
```




## Error for 1/f for f > 1/2 Tc with cutoff

```
Integrate[1 / f, {f, 1 / (2 (d + r)), hhi}, Assumptions :→ {0 < 1 / (2 (d + r)) < hhi}]
```

Log[2 hhi (d + r)]

```
1 / (2 (d + r)) < hhi
```

## Total Remaining Error for 1/f

```
Integrate[(Sinc[Pi (f) r]^2 1 / Abs[f]), {f, 1 / (2 (d + r)), hhi},
 Assumptions :> And[d > 0, r > 0, 0 < 1 / (2 (d + r)) < hhi]]
```

$$\frac{1}{4\,\text{hhi}^2\,\pi^2\,r^2}\left(-1+4\,d^2\,\text{hhi}^2+8\,d\,\text{hhi}^2\,r+4\,\text{hhi}^2\,r^2+\text{Cos}[2\,\text{hhi}\,\pi\,r]-4\,d^2\,\text{hhi}^2\,\text{Cos}\left[\frac{\pi\,r}{d+r}\right]-\right.$$
$$8\,d\,\text{hhi}^2\,r\,\text{Cos}\left[\frac{\pi\,r}{d+r}\right]-4\,\text{hhi}^2\,r^2\,\text{Cos}\left[\frac{\pi\,r}{d+r}\right]+4\,\text{hhi}^2\,\pi^2\,r^2\,\text{CosIntegral}[2\,\text{hhi}\,\pi\,r]-$$
$$\left.4\,\text{hhi}^2\,\pi^2\,r^2\,\text{CosIntegral}\left[\frac{\pi\,r}{d+r}\right]-2\,\text{hhi}\,\pi\,r\,\text{Sin}[2\,\text{hhi}\,\pi\,r]+4\,d\,\text{hhi}^2\,\pi\,r\,\text{Sin}\left[\frac{\pi\,r}{d+r}\right]+4\,\text{hhi}^2\,\pi\,r^2\,\text{Sin}\left[\frac{\pi\,r}{d+r}\right]\right)$$

```
FullSimplify[%21]
```

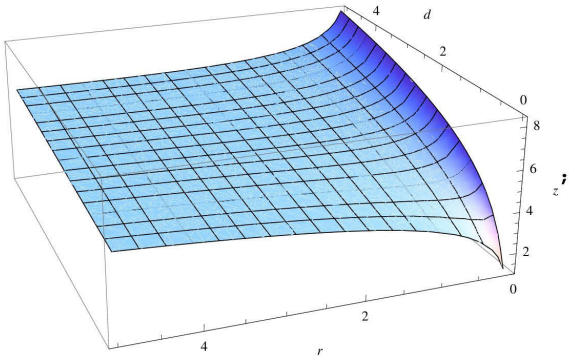
$$\frac{1}{4\,\text{hhi}^2\,\pi^2\,r^2}\left((-1+2\,\text{hhi}\,(d+r))\,(1+2\,\text{hhi}\,(d+r))+\text{Cos}[2\,\text{hhi}\,\pi\,r]-4\,\text{hhi}^2\,(d+r)^2\,\text{Cos}\left[\frac{\pi\,r}{d+r}\right]+\right.$$
$$\left.2\,\text{hhi}\,\pi\,r\left(-\text{Sin}[2\,\text{hhi}\,\pi\,r]+2\,\text{hhi}\left(\pi\,r\left(\text{CosIntegral}[2\,\text{hhi}\,\pi\,r]-\text{CosIntegral}\left[\frac{\pi\,r}{d+r}\right]\right)+(d+r)\,\text{Sin}\left[\frac{\pi\,r}{d+r}\right]\right)\right)\right)$$

```
Integrate[1 / f - Sinc[Pi (f) r]^2 / f, {f, l, 1 / (2 (d + r))},
  Assumptions :> And[d > 0, r > 0, 0 < l < 1 / (2 (d + r))]] + 1 / (4 hhi^2 π^2 r^2)
 ((-1 + 2 hhi (d + r)) (1 + 2 hhi (d + r)) + Cos[2 hhi π r] - 4 hhi^2 (d + r)^2 Cos[π r / (d + r)] + 2 hhi π r (-Sin[2 hhi π r] +
     2 hhi (π r (CosIntegral[2 hhi π r] - CosIntegral[π r / (d + r)]) + (d + r) Sin[π r / (d + r)]))) + Log[2 hhi (d + r)]
```

```
Limit[Log[2 hhi (d + r)] - Log[2 l (d + r)] -
  1/(4 π^2 r^2) (-4 d^2 + 1/l^2 - 8 d r - 4 r^2 - Cos[2 l π r]/l^2 + 4 d^2 Cos[π r/(d + r)] + 8 d r Cos[π r/(d + r)] +
     4 r^2 Cos[π r/(d + r)] - 4 π^2 r^2 CosIntegral[2 l π r] + 4 π^2 r^2 CosIntegral[π r/(d + r)] +
     (2 π r Sin[2 l π r])/l - 4 d π r Sin[π r/(d + r)] - 4 π r^2 Sin[π r/(d + r)]) +
  1/(4 hhi^2 π^2 r^2) ((-1 + 2 hhi (d + r)) (1 + 2 hhi (d + r)) + Cos[2 hhi π r] -
     4 hhi^2 (d + r)^2 Cos[π r/(d + r)] + 2 hhi π r (-Sin[2 hhi π r] +
        2 hhi (π r (CosIntegral[2 hhi π r] - CosIntegral[π r/(d + r)]) + (d + r) Sin[π r/(d + r)]))), l -> 0]
```

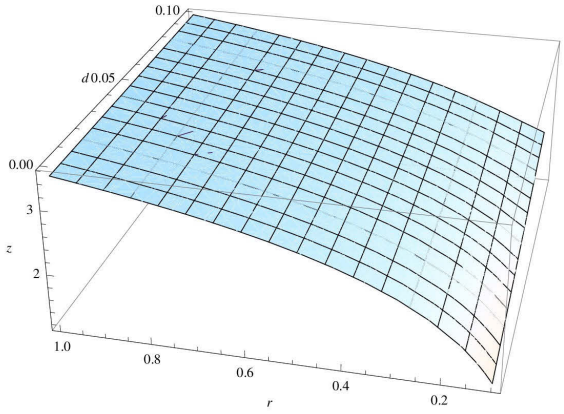
```
ReleaseHold[
 Hold[Plot3D[-3/2 + EulerGamma + 2/π^2 + 2 d^2/(π^2 r^2) - 1/(4 hhi^2 π^2 r^2) + 4 d/(π^2 r) + Cos[2 hhi π r]/(4 hhi^2 π^2 r^2) - 2 Cos[π r/(d+r)]/π^2 - 2 d^2 Cos[π r/(d+r)]/(π^2 r^2) -
    4 d Cos[π r/(d+r)]/(π^2 r) + CosIntegral[2 hhi π r] - 2 CosIntegral[π r/(d + r)] + Log[2 π r] - Log[d + r] +
    Log[hhi (d + r)] - Sin[2 hhi π r]/(2 hhi π r) + 2 Sin[π r/(d+r)]/π + 2 d Sin[π r/(d+r)]/(π r), {d, 0, 5}, {r, 0.1, 5},
   AxesLabel -> {d, r, z}, Axes -> {True, True, True}, ClippingStyle -> None, PlotRange -> All,
   RegionFunction -> Function[{d, r, z}, 1 / (2 (d + r)) < hhi]] ] /. hhi -> 10]
```

performancef =




;

```
ReleaseHold[
 Hold[Plot3D[-3/2 + EulerGamma + 2/π^2 + 2 d^2/(π^2 r^2) - 1/(4 hhi^2 π^2 r^2) + 4 d/(π^2 r) + Cos[2 hhi π r]/(4 hhi^2 π^2 r^2) - 2 Cos[π r/(d+r)]/π^2 - 2 d^2 Cos[π r/(d+r)]/(π^2 r^2) -
   4 d Cos[π r/(d+r)]/(π^2 r) + CosIntegral[2 hhi π r] - 2 CosIntegral[π r/(d+r)] + Log[2 π r] - Log[d+r] +
   Log[hhi (d+r)] - Sin[2 hhi π r]/(2 hhi π r) + 2 Sin[π r/(d+r)]/π + 2 d Sin[π r/(d+r)]/(π r), {d, 0, 0.1}, {r, 0.1, 1},
  AxesLabel → {d, r, z}, Axes → {True, True, True}, ClippingStyle → None, PlotRange → All,
  RegionFunction → Function[{d, r, z}, 1 / (2 (d+r)) < hhi]] ] /. hhi → 10]
```



```
FullSimplify[-3/2 + EulerGamma + 2/π^2 + (2 d^2)/(π^2 r^2) - 1/(4 hhi^2 π^2 r^2) + (4 d)/(π^2 r) + Cos[2 hhi π r]/(4 hhi^2 π^2 r^2) -
  (2 Cos[π r/(d+r)])/π^2 - (2 d^2 Cos[π r/(d+r)])/(π^2 r^2) - (4 d Cos[π r/(d+r)])/(π^2 r) + CosIntegral[2 hhi π r] - 2 CosIntegral[π r/(d+r)] +
  Log[2 π r] - Log[d + r] + Log[hhi (d + r)] - Sin[2 hhi π r]/(2 hhi π r) + (2 Sin[π r/(d+r)])/π + (2 d Sin[π r/(d+r)])/(π r),
 Assumptions :> {d > 0, r > 0, hhi > 0}] /. {d → T_d, r → T_r, hhi → f_hi}
```

$$\frac{1}{4\pi^2 f_{hi}^2 T_r^2}\left(-1+\text{Cos}[2\pi f_{hi} T_r]-8\,\text{Cos}\left[\frac{\pi T_r}{T_d+T_r}\right] f_{hi}^2 (T_d+T_r)^2+2 f_{hi}^2\left(4T_d^2+8T_d T_r+\left(4+(-3+2\,\text{EulerGamma})\pi^2\right)T_r^2\right)+2\pi f_{hi} T_r\left(-\text{Sin}[2\pi f_{hi} T_r]+2\pi\left(\text{CosIntegral}[2\pi f_{hi} T_r]-2\,\text{CosIntegral}\left[\frac{\pi T_r}{T_d+T_r}\right]+\text{Log}[2\pi f_{hi} T_r]\right) f_{hi} T_r+4\,\text{Sin}\left[\frac{\pi T_r}{T_d+T_r}\right] f_{hi}(T_d+T_r)\right)\right)$$

```
1/(4 hhi^2 π^2 r^2) (-1 + 2 hhi^2 (4 d^2 + 8 d r + (4 + (-3 + 2 EulerGamma) π^2) r^2) + Cos[2 hhi π r] - 8 hhi^2 (d + r)^2 Cos[π r/(d + r)] + 2 hhi π
   r (2 hhi π r (CosIntegral[2 hhi π r] - 2 CosIntegral[π r/(d + r)] + Log[2 π r] - Log[d + r] + Log[hhi (d + r)]) -
     Sin[2 hhi π r] + 4 hhi (d + r) Sin[π r/(d + r)])) /. {d → T_d, r → T_r, hhi → f_hi}
```

$$\frac{1}{4\pi^2 f_{hi}^2 T_r^2}\left(-1+\text{Cos}[2\pi f_{hi} T_r]-8\,\text{Cos}\left[\frac{\pi T_r}{T_d+T_r}\right] f_{hi}^2 (T_d+T_r)^2+2 f_{hi}^2\left(4T_d^2+8T_d T_r+\left(4+(-3+2\,\text{EulerGamma})\pi^2\right)T_r^2\right)+2\pi f_{hi} T_r\left(-\text{Sin}[2\pi f_{hi} T_r]+2\pi\left(\text{CosIntegral}[2\pi f_{hi} T_r]-2\,\text{CosIntegral}\left[\frac{\pi T_r}{T_d+T_r}\right]+\text{Log}[2\pi T_r]-\text{Log}[T_d+T_r]+\text{Log}[f_{hi}(T_d+T_r)]\right) f_{hi} T_r+4\,\text{Sin}\left[\frac{\pi T_r}{T_d+T_r}\right] f_{hi}(T_d+T_r)\right)\right)$$

```
FullSimplify[%18, Assumptions :> {d > 0, r > 0. f_hi > 0}]
```

$$-\frac{3}{2}+\text{EulerGamma}+\text{CosIntegral}[2\pi f_{hi} T_r]-2\,\text{CosIntegral}\left[\frac{\pi T_r}{T_d+T_r}\right]+\text{Log}[2\pi f_{hi} T_r]$$

```
FullSimplify[Log[2 π r] - Log[d + r] + Log[f_hi (d + r)], Assumptions :> {d > 0, r > 0. f_hi > 0}]
```

$$\text{Log}[2\pi r f_{hi}]$$

## Total Remaining Error for 1/f

```
Integrate[1 / f^2 - Sinc[Pi (f) r]^2 / f^2,
 {f, l, 1 / (2 (d + r))}, Assumptions :> And[r > 0, r > 0, 0 < l < 1 / (2 (d + r))]]
```

$$-2d+\frac{1}{l}-2r-\frac{1}{6\pi^2 r^2}\left(-8d^3+\frac{1}{l^3}-24d^2 r-24 d r^2-8r^3-\frac{\text{Cos}[2l\pi r]}{l^3}+\frac{2\pi^2 r^2\,\text{Cos}[2l\pi r]}{l}+8d^3\,\text{Cos}\left[\frac{\pi r}{d+r}\right]+24d^2 r\,\text{Cos}\left[\frac{\pi r}{d+r}\right]+24 d r^2\,\text{Cos}\left[\frac{\pi r}{d+r}\right]-4d\pi^2 r^2\,\text{Cos}\left[\frac{\pi r}{d+r}\right]+8r^3\,\text{Cos}\left[\frac{\pi r}{d+r}\right]-4\pi^2 r^3\,\text{Cos}\left[\frac{\pi r}{d+r}\right]+\frac{\pi r\,\text{Sin}[2l\pi r]}{l^2}-4d^2\pi r\,\text{Sin}\left[\frac{\pi r}{d+r}\right]-8d\pi r^2\,\text{Sin}\left[\frac{\pi r}{d+r}\right]-4\pi r^3\,\text{Sin}\left[\frac{\pi r}{d+r}\right]+4\pi^3 r^3\,\text{SinIntegral}[2l\pi r]-4\pi^3 r^3\,\text{SinIntegral}\left[\frac{\pi r}{d+r}\right]\right)$$

```
FullSimplify[%85]
```

```
Limit[1/(6 l^3 π^2 r^2) ((-1 + 2 l (d + r)) (1 + 2 l (d + 2 d^2 l + r + 4 d l r + l (2 - 3 π^2) r^2)) + (1 - 2 l^2 π^2 r^2) Cos[2 l π r] -
    4 l^3 (d + r) (2 d^2 + 4 d r - (-2 + π^2) r^2) Cos[π r/(d + r)] + l π r (-Sin[2 l π r] + 4 l^2 (d + r)^2 Sin[π r/(d + r)]) +
    4 l^3 π^3 r^3 (-SinIntegral[2 l π r] + SinIntegral[π r/(d + r)])), l → 0]
```

$$\frac{1}{3\pi^2 r^2} 2\left(-(d+r)\left(-2d^2-4dr-2r^2+3\pi^2 r^2+\left(2d^2+4dr-\left(-2+\pi^2\right)r^2\right)\cos\left[\frac{\pi r}{d+r}\right]-\pi r(d+r)\sin\left[\frac{\pi r}{d+r}\right]\right)+\pi^3 r^3\,\text{SinIntegral}\left[\frac{\pi r}{d+r}\right]\right)$$

```
Integrate[1 / f^2, {f, 1 / (2 (d + r)), Infinity}, Assumptions :→ {Element[{d, r}, Reals]}]
```

```
ConditionalExpression[2 (d + r), d + r ≥ 0]
```

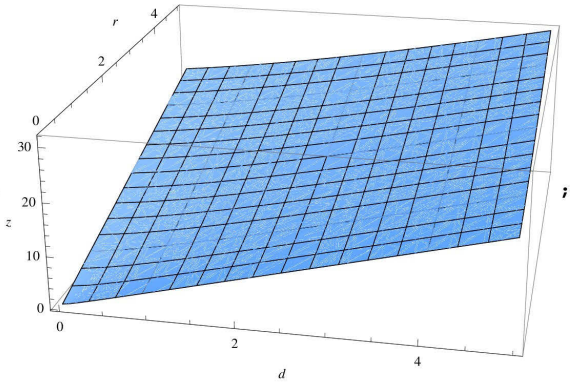
```
FullSimplify[
  1/(3 π^2 r^2) 2 (-(d + r) (-2 d^2 - 4 d r - 2 r^2 + 3 π^2 r^2 + (2 d^2 + 4 d r - (-2 + π^2) r^2) Cos[π r/(d + r)] - π r (d + r) Sin[π r/(d + r)]) +
      π^3 r^3 SinIntegral[π r/(d + r)]) + 1/(3 π^2 r^2)
   (-π^4 r^2 Abs[r] + 2 (-(d + r) ((2 d^2 + 4 d r - (-2 + π^2) r^2) Cos[π r/(d + r)] - (d + r) (2 (d + r) + π r Sin[π r/(d + r)])) +
        π^3 r^3 SinIntegral[π r/(d + r)])) + 2 (d + r), Assumptions :→ {r > 0, d > 0}]
```

$$\frac{1}{3\pi^2 r^2}\left(8d^3+24d^2 r+24dr^2-\left(-8+\pi^4\right)r^3+4(d+r)\left(\left(-2d^2-4dr+\left(-2+\pi^2\right)r^2\right)\cos\left[\frac{\pi r}{d+r}\right]+\pi r(d+r)\sin\left[\frac{\pi r}{d+r}\right]\right)+4\pi^3 r^3\,\text{SinIntegral}\left[\frac{\pi r}{d+r}\right]\right)$$

$$\frac{1}{3\pi^2 r^2}\left(8d^3+24d^2 r+24dr^2-\left(-8+\pi^4\right)r^3+4(d+r)\left(\left(-2d^2-4dr+\left(-2+\pi^2\right)r^2\right)\cos\left[\frac{\pi r}{d+r}\right]+\pi r(d+r)\sin\left[\frac{\pi r}{d+r}\right]\right)+4\pi^3 r^3\,\text{SinIntegral}\left[\frac{\pi r}{d+r}\right]\right)/.\ \{d\to T_d,\ r\to T_r\}$$

$$\frac{1}{3\pi^2 T_r^2}\left(8T_d^3+24T_d^2 T_r+24T_d T_r^2-\left(-8+\pi^4\right)T_r^3+4\pi^3\,\text{SinIntegral}\left[\frac{\pi T_r}{T_d+T_r}\right]T_r^3+4(T_d+T_r)\left(\pi\sin\left[\frac{\pi T_r}{T_d+T_r}\right]T_r(T_d+T_r)+\cos\left[\frac{\pi T_r}{T_d+T_r}\right]\left(-2T_d^2-4T_d T_r+\left(-2+\pi^2\right)T_r^2\right)\right)\right)$$

```
Plot3D[1/(3 π^2 r^2) (8 d^3 + 24 d^2 r + 24 d r^2 - (-8 + π^4) r^3 +
    4 (d + r) ((-2 d^2 - 4 d r + (-2 + π^2) r^2) Cos[π r/(d + r)] + π r (d + r) Sin[π r/(d + r)]) + 4 π^3 r^3 SinIntegral[π r/(d + r)]),
  {d, 0, 5}, {r, 0.1, 5}, AxesLabel → {d, r, z}, Axes → {True, True, True},
  ClippingStyle → None, PlotRange → All]
```

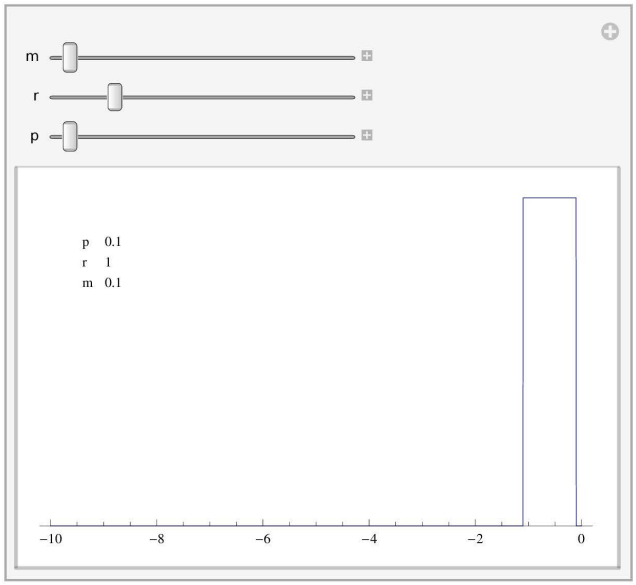
performancef2 = 

;

# Understanding Multiple Windows

In[1]:=
```
oneWindow[p_, r_, m_] := Show[Plot[UnitBox[(r / 2 + m + t) / r], {t, -10, 0}, Axes → {True, False}],
  Graphics[Text[Grid[{{"p", p}, {"r", r}, {"m", m}}, Alignment → Left], {-9, 0.8}]]]
```

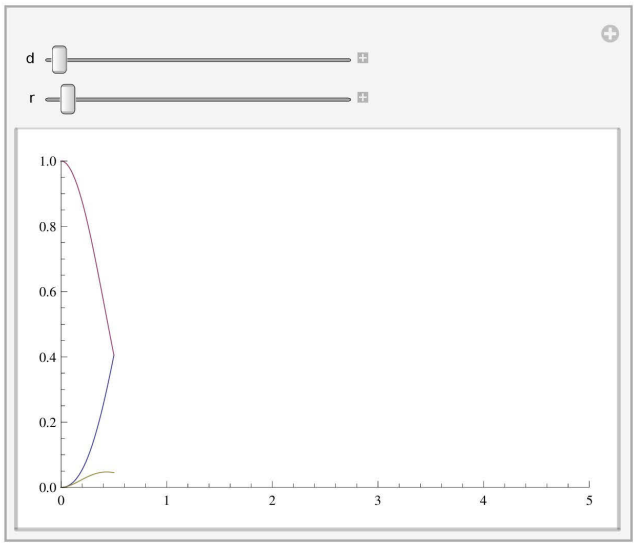
```
Manipulate[oneWindow[p, r, m], {m, 0, 5}, {r, 0.1, 5}, {p, 0, 5}]
```




```
FourierTransform[UnitBox[(r / 2 + m + t) / r], t, f, FourierParameters → {0, -2 Pi}]
```

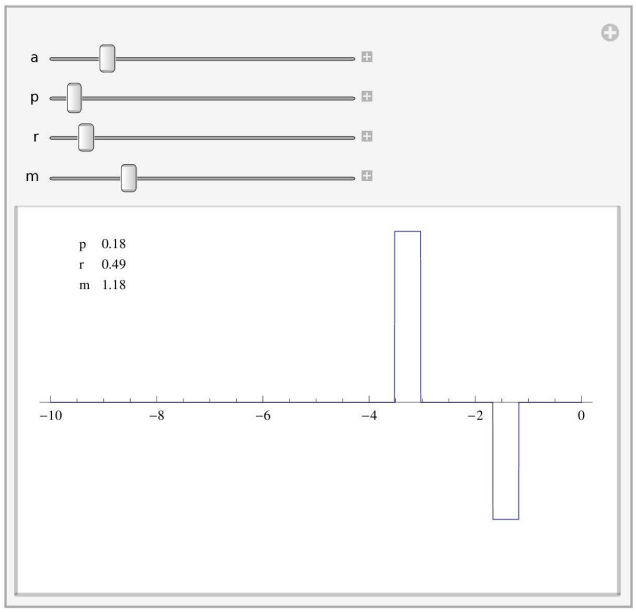
$$-\frac{1}{f\,\pi}e^{i\,f\,\pi\,(2\,m+r)}\,\text{Sin}[f\,\pi\,r]\,(\text{UnitStep}[-r]-\text{UnitStep}[r])$$

```
Manipulate[Plot[Evaluate[Table[ Sinc[(f + n / (d + r)) Pi r]^2, {n, -1, 1}]],
  {f, -1 / (2 (d + r)), 1 / (2 (d + r))}, PlotRange → {{0, 5}, {0, 1}}], {d, 0, 10}, {r, 0.1, 30}]
```




In[2]:=
```
twoWindows[a_, p_, r_, m_] := Show[Plot[a UnitBox[(r / 2 + m + t) / r] + UnitBox[(r / 2 + 2 m + p + r + t) / r],
    {t, -10, 0}, Axes → {True, False}, PlotRange → {Automatic, {-1, 1}}],
  Graphics[Text[Grid[{{"p", p}, {"r", r}, {"m", m}}, Alignment → Left], {-9, 0.8}]]]
```

```
Manipulate[twoWindows[a, p, r, m], {a, -1, 1}, {p, 0, 5}, {r, 0.1, 5}, {m, 0, 5}]
```




```
FourierTransform[UnitBox[(r / 2 + m + t) / r] + UnitBox[(r / 2 + 2 m + p + r + t) / r],
 t, f, FourierParameters → {0, -2 Pi}]
```

$$-\frac{1}{f\,\pi}\left(e^{i\,f\,\pi\,(2\,m+r)}+e^{i\,f\,\pi\,(4\,m+2\,p+3\,r)}\right)\,\text{Sin}[f\,\pi\,r]\;(\text{UnitStep}[-r]-\text{UnitStep}[r])$$

```
FullSimplify[ExpToTrig[Abs[(e^(i f π (2 m+r)) + e^(i f π (4 m+2 p+3 r)))]], Assumptions :> {Element[{m, p, r, f}, Reals]}]
```

$$2\,\text{Abs}[\text{Cos}[f\,\pi\,(m+p+r)]]$$

```
FourierTransform[a UnitBox[(r / 2 + m + t) / r] + UnitBox[(r / 2 + 2 m + p + r + t) / r],
 t, f, FourierParameters → {0, -2 Pi}]
```

$$-\frac{1}{f\,\pi}\left(a\,e^{i\,f\,\pi\,(2\,m+r)}+e^{i\,f\,\pi\,(4\,m+2\,p+3\,r)}\right)\,\text{Sin}[f\,\pi\,r]\;(\text{UnitStep}[-r]-\text{UnitStep}[r])$$

```
TrigReduce[(ComplexExpand[Abs[(a e^(i f π (2 m+r)) + e^(i f π (4 m+2 p+3 r)))]])^2] /.
 Cos[arg_] :> Cos[FullSimplify[arg]]
```

$$1+a^2+2\,a\,\text{Cos}[2\,f\,\pi\,(m+p+r)]$$

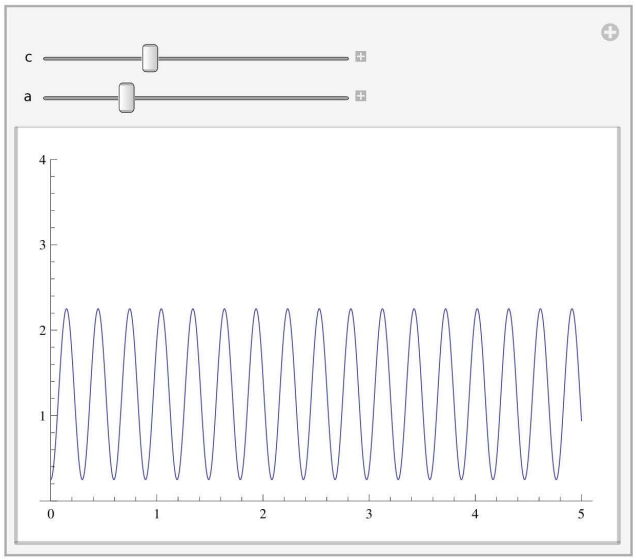
```
FullSimplify[1 + a^2 + 2 a Cos[2 f π (m + p + r)] /. a → 1]
```

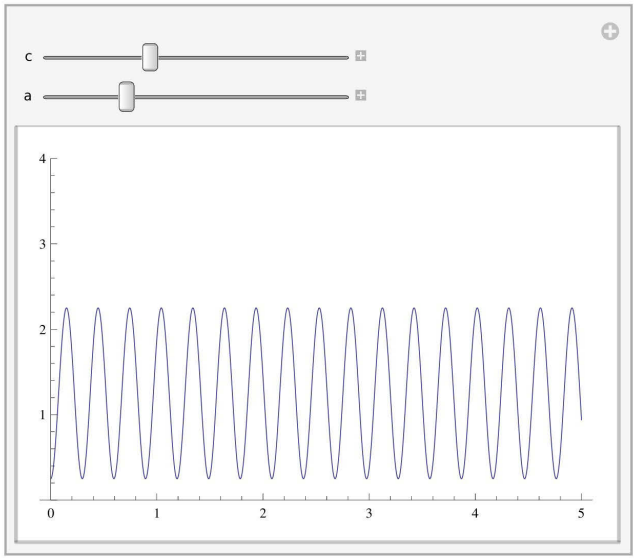
$$4\,\text{Cos}[f\,\pi\,(m+p+r)]^2$$

```
FullSimplify[1 + a^2 + 2 a Cos[2 f π (m + p + r)] /. a → -1]
```

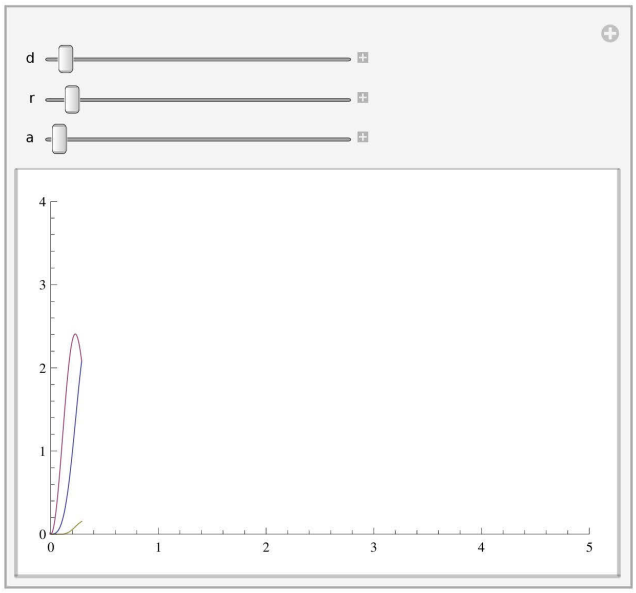
$$4\,\text{Sin}[f\,\pi\,(m+p+r)]^2$$

```
Manipulate[Plot[1 + a^2 + 2 a Cos[2 f π (c)], {f, 0, 5}, PlotRange → {0, 4}], {c, 0, 10}, {a, -1, 1}]
```


c
a
4
3
2
1
0
1
2
3
4
5


c
a
4
3
2
1
0
1
2
3
4
5

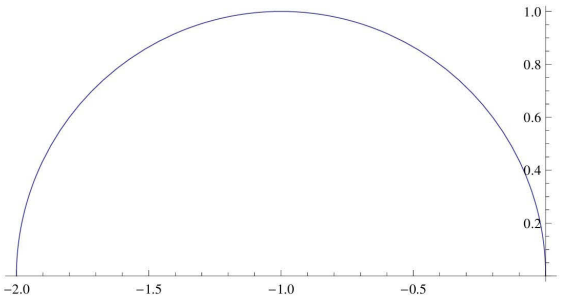
```
Manipulate[Plot[
  Evaluate[Table[(1 + a^2 + 2 a Cos[2 (f + n / (d + r)) π (d + r)]) Sinc[(f + n / (d + r)) Pi r]^2, {n, -1, 1}]],
  {f, -1 / (2 (d + r)), 1 / (2 (d + r))}, PlotRange → {{0, 5}, {0, 4}}], {d, 0, 10}, {r, 0.1, 30}, {a, -1, 1}]
```




```
Abs[FourierTransform[UnitBox[1.5 + t] - UnitBox[4.5 + t], t, f, FourierParameters → {0, -2 Pi}]]
```

```
Abs[(ℯ^((0.+9.42478 i) f) - ℯ^((0.+28.2743 i) f)) Sinc[f π]]
```

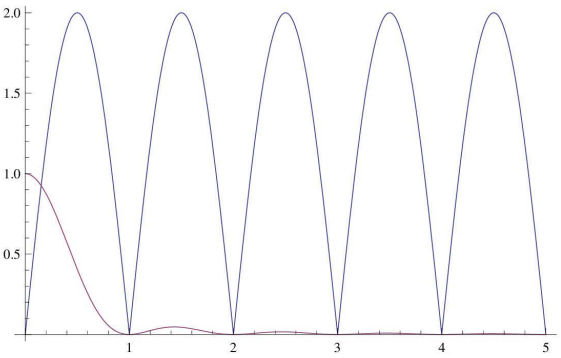
```
ParametricPlot[{Re[-1 + ℯ^(2 i f π)], Im[-1 + ℯ^(2 i f π)]}, {f, 0, 0.5}]
```



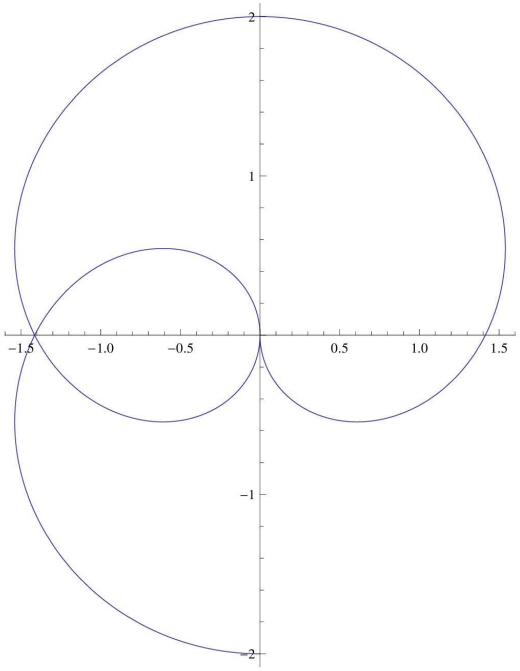

```
ExpToTrig[Abs[(-1 + ℯ^(2 i f π))]]
```

```
Abs[-1 + Cos[2 f π] + i Sin[2 f π]]
```

```
Plot[{Abs[(-1 + ℯ^(2 i f π))], Sinc[π f]^2}, {f, 0, 5}]
```



```
ParametricPlot[{Re[(e^(0.`+9.42477796076938` i) f - e^(0.`+28.274333882308138` i) f)],
  Im[(e^(0.`+9.42477796076938` i) f - e^(0.`+28.274333882308138` i) f)]}, {f, 0, 0.5}]
```

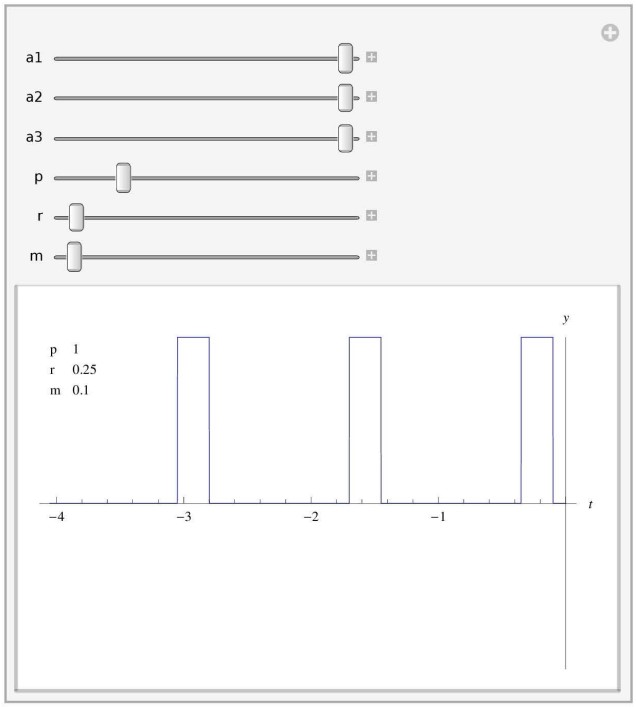



```
a1 UnitBox[(r / 2 + m + t) / r] + a2 UnitBox[(r / 2 + 2 m + p + r + t) / r] +
  a3 UnitBox[(r / 2 + 3 m + 2 p + 2 r + t) / r] /. {r → T_r, m → T_m, p → T_p, a1 → a_1, a3 → a_3, a2 → a_2}
```

$$a_1\ \text{UnitBox}\left[\frac{t + T_m + \frac{T_r}{2}}{T_r}\right] + a_2\ \text{UnitBox}\left[\frac{t + 2\,T_m + T_p + \frac{3\,T_r}{2}}{T_r}\right] + a_3\ \text{UnitBox}\left[\frac{t + 3\,T_m + 2\,T_p + \frac{5\,T_r}{2}}{T_r}\right]$$

In[3]:=
```
threeWindows[a1_, a2_, a3_, p_, r_, m_] :=
 Show[Plot[a1 UnitBox[(r / 2 + m + t) / r] + a2 UnitBox[(r / 2 + 2 m + p + r + t) / r] +
    a3 UnitBox[(r / 2 + 3 m + 2 p + 2 r + t) / r], {t, -3 (p + r + m), 0}, Axes → {True, True},
   PlotRange → {Automatic, {-1, 1}}, AxesLabel → {t, y}, Ticks → {Automatic, None}],
  Graphics[Text[Grid[{{"p", p}, {"r", r}, {"m", m}}, Alignment → Left], {-3 (p + r + m), 0.8}, {-1, 0}]]]
```

```
Manipulate[threeWindows[a1, a2, a3, p, r, m],
 {a1, -1, 1}, {a2, -1, 1}, {a3, -1, 1}, {p, 0, 5}, {r, 0.1, 5}, {m, 0, 5}]
```




```
FourierTransform[a1 UnitBox[(r / 2 + m + t) / r] + a2 UnitBox[(r / 2 + 2 m + p + r + t) / r] +
   a3 UnitBox[(r / 2 + 3 m + 2 p + 2 r + t) / r], t, f, FourierParameters → {0, -2 Pi}]
```

$$-\frac{1}{f\,\pi}\left(a1\, e^{i\,f\,\pi\,(2\,m+r)} + a2\, e^{i\,f\,\pi\,(4\,m+2\,p+3\,r)} + a3\, e^{i\,f\,\pi\,(6\,m+4\,p+5\,r)}\right) \text{Sin}[f\,\pi\,r]\ (\text{UnitStep}[-r] - \text{UnitStep}[r])$$

```
FullSimplify[TrigReduce[FullSimplify[
     ComplexExpand[Abs[(a1 e^(i f π (2 m+r)) + a2 e^(i f π (4 m+2 p+3 r)) + a3 e^(i f π (6 m+4 p+5 r)))]]^2]] /. m + p + r → c]
```

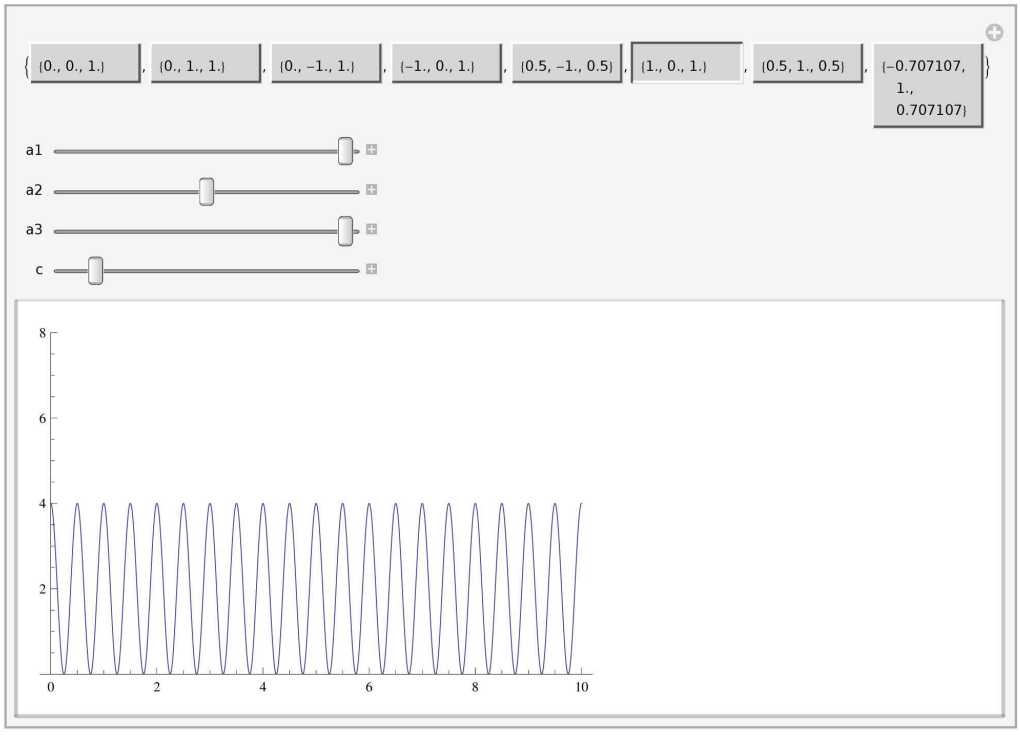
$$a1^2 + a2^2 + a3^2 + 2\,a2\,(a1 + a3)\,\text{Cos}[2\,c\,f\,\pi] + 2\,a1\,a3\,\text{Cos}[4\,c\,f\,\pi] \ /.$$
$$\{c \to T_c,\ r \to T_r,\ m \to T_m,\ p \to T_p,\ a1 \to a_1,\ a3 \to a_3,\ a2 \to a_2\}$$

$$a_1^2 + a_2^2 + 2\,\text{Cos}[4\,f\,\pi\,T_c]\,a_1\,a_3 + a_3^2 + 2\,\text{Cos}[2\,f\,\pi\,T_c]\,a_2\,(a_1 + a_3)$$

```
ReleaseHold[
 Grid[MapThread[Hold[{#2, TraditionalForm[FullSimplify[a1^2 + a2^2 + a3^2 + 2 a2 (a1 + a3) Cos[2 c f π] +
          2 a1 a3 Cos[4 c f π] /. #1]], TraditionalForm[{a1, a2, a3}] /. #1
     }] &, {
    {{a1 → 0, a2 → 0, a3 → 1}, {a1 → 0, a2 → 1, a3 → 1}, {a1 → 0, a2 → -1, a3 → 1},
     {a1 → -1, a2 → 0, a3 → 1}, {a1 → 1 / 2, a2 → -1, a3 → 1 / 2}, {a1 → 1, a2 → 0, a3 → 1},
     {a1 → 1 / 2, a2 → 1, a3 → 1 / 2}, {a1 → -1 / Sqrt[2], a2 → 1, a3 → 1 / Sqrt[2]},
     {a1 → -1, a2 → 1, a3 → 1}, {a1 → -0.22, a2 → -1, a3 -> 1}, {a1 → -0.22, a2 → 1, a3 → 1},
     {a1 -> -0.08, a2 -> 0.29, a3 -> 0.73}}, Range[numWind]}], Frame → All, Alignment → Left]]
```

| | | |
|---|---|---|
| 1 | $1$ | $\{0, 0, 1\}$ |
| 2 | $4\cos^2(\pi c f)$ | $\{0, 1, 1\}$ |
| 3 | $4\sin^2(\pi c f)$ | $\{0, -1, 1\}$ |
| 4 | $4\sin^2(2\pi c f)$ | $\{-1, 0, 1\}$ |
| 5 | $4\sin^4(\pi c f)$ | $\left\{\frac{1}{2}, -1, \frac{1}{2}\right\}$ |
| 6 | $4\cos^2(2\pi c f)$ | $\{1, 0, 1\}$ |
| 7 | $4\cos^4(\pi c f)$ | $\left\{\frac{1}{2}, 1, \frac{1}{2}\right\}$ |
| 8 | $2-\cos(4\pi c f)$ | $\left\{-\frac{1}{\sqrt{2}}, 1, \frac{1}{\sqrt{2}}\right\}$ |
| 9 | $3-2\cos(4\pi c f)$ | $\{-1, 1, 1\}$ |
| 10 | $-1.56\cos(2\pi c f)-0.44\cos(4\pi c f)+2.0484$ | $\{-0.22, -1, 1\}$ |
| 11 | $1.56\cos(2\pi c f)-0.44\cos(4\pi c f)+2.0484$ | $\{-0.22, 1, 1\}$ |
| 12 | $0.377\cos(2\pi c f)-0.1168\cos(4\pi c f)+0.6234$ | $\{-0.08, 0.29, 0.73\}$ |

```
buttons = ReleaseHold[
    Item[Setter[{Dynamic[a1], Dynamic[a2], Dynamic[a3]}, Hold[({a1, a2, a3} /. #)], ImageSize →
          Scaled[0.1]]] & /@ {{a1 → 0, a2 → 0, a3 → 1}, {a1 → 0, a2 → 1, a3 → 1}, {a1 → 0, a2 → -1, a3 → 1},
       {a1 → -1, a2 → 0, a3 → 1}, {a1 → 1 / 2, a2 → -1, a3 → 1 / 2}, {a1 → 1, a2 → 0, a3 → 1},
       {a1 → 1 / 2, a2 → 1, a3 → 1 / 2}, {a1 → -1 / Sqrt[2], a2 → 1, a3 → 1 / Sqrt[2]}} // N];

(*Can use Sequence @@ to pass buttons individually*)

Manipulate[Plot[a1^2 + a2^2 + a3^2 + 2 a2 (a1 + a3) Cos[2 c f π] + 2 a1 a3 Cos[4 c f π], {f, 0, 10},
  PlotRange → {0, 8}], Item[buttons], {a1, -1, 1}, {a2, -1, 1}, {a3, -1, 1}, {c, 0, 10}]
```



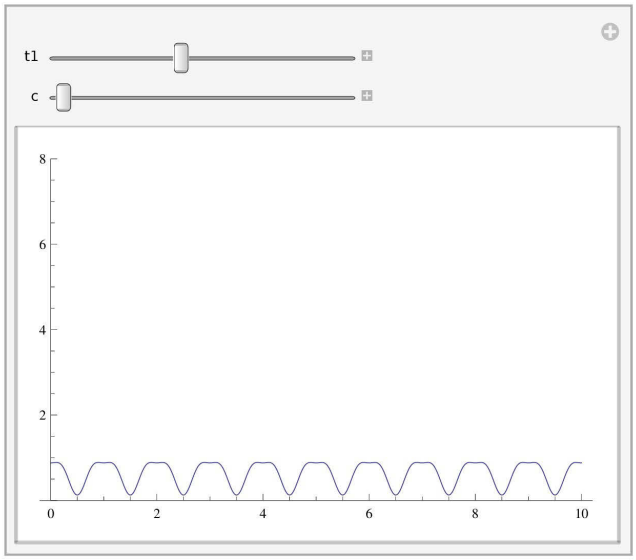
```
{t1, t1 - 1 / 3, t1 - 2 / 3} /. t1 → -0.432` // N
```

```
{-0.432, -0.765333, -1.09867}
```

```
Sinc[Pi #] & /@ {t1 - 2 / 3, t1 - 1 / 3, t1} /. t1 → -0.424` // N
```

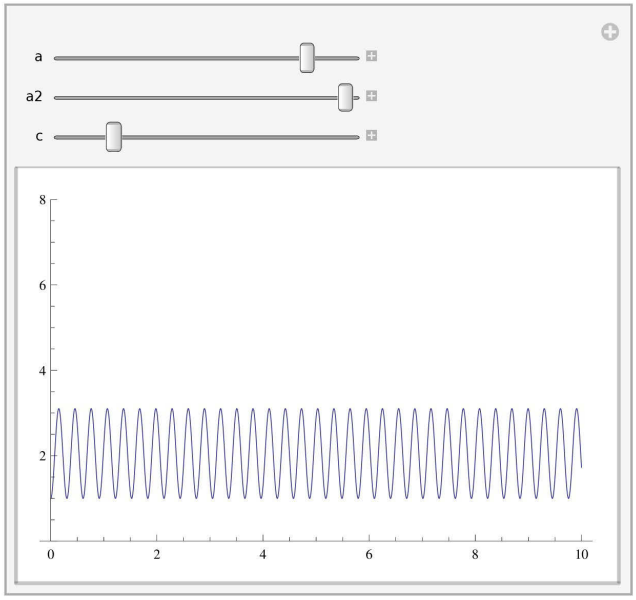
```
{-0.08201, 0.290274, 0.729434}
```

```
Manipulate[Plot[Sinc[π (-2/3 + t1)]^2 + Sinc[π (-1/3 + t1)]^2 +
    Sinc[π t1]^2 + 2 Sinc[π (-1/3 + t1)] (Sinc[π (-2/3 + t1)] + Sinc[π t1]) Cos[2 c f π] +
    2 Sinc[π (-2/3 + t1)] Sinc[π t1] Cos[4 c f π], {f, 0, 10}, PlotRange → {0, 8}], {t1, 0, -1}, {c, 1, 10}]
```




```
DynamicModule[{a1 = -1, a2 = 1.`, a3 = 1, c = 2.46`},
 Plot[a1^2 + a2^2 + a3^2 + 2 a2 (a1 + a3) Cos[2 c f π] + 2 a1 a3 Cos[4 c f π],
  {f, 0, 10}, PlotRange → {0, 8}]] (*Optical Illusion*)
```

```
Manipulate[Plot[a1^2 + a2^2 + a3^2 + 2 a2 (a1 + a3) Cos[2 c f π] + 2 a1 a3 Cos[4 c f π] /. {a1 → -a, a3 → a},
  {f, 0, 10}, PlotRange → {0, 8}], {a, -1, 1}, {a2, -1, 1}, {c, 0, 10}]
```



```
(*Interesting, if you allow a1, a2, a3 to be complex:*)
FullSimplify[TrigReduce[FullSimplify[ComplexExpand[
      Abs[(a1 e^(i f π (2 m+r)) + a2 e^(i f π (4 m+2 p+3 r)) + a3 e^(i f π (6 m+4 p+5 r)))], {a1, a2, a3}]^2]] /. m+p+r → c]
```

```
((a1 + a3) Conjugate[a2] + a2 Conjugate[a1 + a3]) Cos[2 c f π] +
  (a3 Conjugate[a1] + a1 Conjugate[a3]) Cos[4 c f π] + Im[a1]^2 + Im[a2]^2 + Im[a3]^2 +
  Re[a1]^2 + Re[a2]^2 + Re[a3]^2 + 2 Re[(-a1 + a3) Im[a2] + a2 Im[a1 - a3]] Sin[2 c f π] +
  2 Re[a3 Im[a1] - a1 Im[a3]] Sin[4 c f π] /. {a1 → -i, a2 → Sqrt[2], a3 → i}
```

```
4 - 2 Cos[4 c f π] - 4 √2 Sin[2 c f π]
```

```
((a1 + a3) Conjugate[a2] + a2 Conjugate[a1 + a3]) Cos[2 c f π] +
  (a3 Conjugate[a1] + a1 Conjugate[a3]) Cos[4 c f π] + Im[a1]^2 + Im[a2]^2 + Im[a3]^2 +
  Re[a1]^2 + Re[a2]^2 + Re[a3]^2 + 2 Re[(-a1 + a3) Im[a2] + a2 Im[a1 - a3]] Sin[2 c f π] +
  2 Re[a3 Im[a1] - a1 Im[a3]] Sin[4 c f π] /. {a1 → i, a2 → 0, a3 → i}
```

```
2 + 2 Cos[4 c f π]
```

```
(*Wait... meaningless in time domain!*)
```

# Correction and Aliasing of Windows

In[4]:=

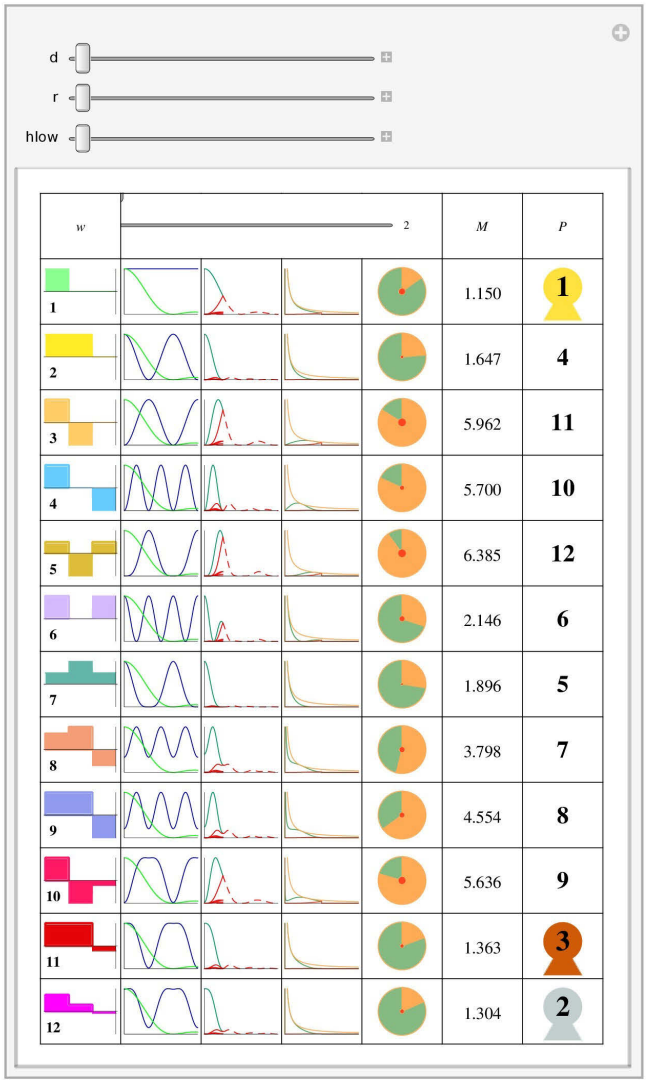
```
Remove[a1, a2, a3, f, d, r, n, hlow, hhi, maxSincedFwindows,
  correctVSalias, NormFwindowsVSsinc, NormTransSincedFwindows];
windowSettings = {{a1 → 0, a2 → 0, a3 → 1}, {a1 → 0, a2 → 1, a3 → 1}, {a1 → 0, a2 → -1, a3 → 1},
   {a1 → -1, a2 → 0, a3 → 1}, {a1 → 1/2, a2 → -1, a3 → 1/2}, {a1 → 1, a2 → 0, a3 → 1},
   {a1 → 1/2, a2 → 1, a3 → 1/2}, {a1 → -1/Sqrt[2], a2 → 1, a3 → 1/Sqrt[2]},
   {a1 → -1, a2 → 1, a3 → 1}, {a1 → -0.22, a2 → -1, a3 -> 1}, {a1 → -0.22, a2 → 1, a3 → 1},
   {a1 -> -0.08201004990182333`, a2 -> 0.2902742410524303`, a3 -> 0.7294339299920499`}};
numWind = Length[windowSettings];
Twindow = a1 UnitBox[(r/2 + m + t)/r] + a2 UnitBox[(r/2 + 2 m + p + r + t)/r] +
    a3 UnitBox[(r/2 + 3 m + 2 p + 2 r + t)/r] /. {m → d/2, p -> d/2};
Fwindow = (a1^2 + a2^2 + a3^2 + 2 a2 (a1 + a3) Cos[2 (d + r) f π] + 2 a1 a3 Cos[4 (d + r) f π]); (*Abs squared*)
SincedFwindow = Sinc[Pi r f]^2 Fwindow; (*Notice Sinc^2*)
(*hlow = 0.2;*)
hhi = 10;
NoiseBox[hlow_] = UnitBox[(f - (hhi - hlow)/2 - hlow)/(hhi - hlow)] +
   UnitBox[(-f - (hhi - hlow)/2 - hlow)/(hhi - hlow)];
(*AliasBox = UnitBox[(f-(1/(2(d+r))- (-1/(2(d+r))))/2 -(-1/(2(d+r))))/
    (1/(2(d+r))-(-1/(2(d+r))))];*)
AliasBox[d_, r_] = UnitBox[f (d + r)];

UnclippedNoise = 1/Abs[f];
totalNoise[hlow_] :=
 Evaluate[FullSimplify[Integrate[UnclippedNoise, {f, low, hi}, Assumptions :→ {0 < low < hi}]] /.
   {low → hlow, hi -> hhi}]
Noise[hlow_] = NoiseBox[hlow] UnclippedNoise;
NoiseSincedFwindow[hlow_] = Noise[hlow] SincedFwindow;
TransSincedFwindow = SincedFwindow /. f → (f + n/(d + r));
TransNoiseSincedFwindow[hlow_] = NoiseSincedFwindow[hlow] /. f → (f + n/(d + r));
numTrans = 2;

Twindows[d_, r_] = Twindow /. windowSettings;
Fwindows[d_, r_] = Fwindow /. windowSettings;
SincedFwindows[d_, r_] = SincedFwindow /. windowSettings;
TransSincedFwindows[d_, r_] =
  (TransSincedFwindow /. n → {0} ~Join~ Cases[Range[-numTrans, numTrans], Except[0]]) /.
   windowSettings;
NoiseSincedFwindows[d_, r_, hlow_] = NoiseSincedFwindow[hlow] /. windowSettings;
AliasNoiseSincedFwindows[d_, r_, hlow_] =
  AliasBox[d, r] (Plus @@ (TransNoiseSincedFwindow[hlow] /.
       n → Cases[Range[-numTrans, numTrans], Except[0]])) /. windowSettings;
```

```
NormFwindowsVSsinc[d_, r_]  := NormFwindowsVSsinc[d, r] =
    {1 / Maximize[{#, {0 ≤ f ≤ 1 / (d + r)}}, f][[1]] #, Sinc[Pi r f]^2} & /@ Fwindows[d, r];
maxSincedFwindows[d_, r_] := maxSincedFwindows[d, r] =
    FindMaximum[#, {f, 0, 1 / (2 (d + r))}][[1]] & /@ SincedFwindows[d, r];
NormTransSincedFwindows[d_, r_] := NormTransSincedFwindows[d, r] =
    MapThread[(1 / #1 #2 &), {maxSincedFwindows[d, r], TransSincedFwindows[d, r]}];
(*NormNoiseSincedFwindowsVSaliasVSNoise[d_,r_]:=MapThread[({1/#1#2,1/#1#3,Noise}&),
    {maxSincedFwindows[d,r],NoiseSincedFwindows[d,r],AliasNoiseSincedFwindows[d,r]}];*)
NormNoiseSincedFwindowsVSalias[d_, r_, hlow_] :=
  MapThread[({#2 / #1 , #3 / #1, Noise[hlow]} &), {maxSincedFwindows[d, r],
    (AliasBox[d, r] NoiseSincedFwindows[d, r, hlow]), AliasNoiseSincedFwindows[d, r, hlow]}];


NormNoiseSincedFwindows[d_, r_, hlow_] :=
  MapThread[(#2 / #1 &), {maxSincedFwindows[d, r], NoiseSincedFwindows[d, r, hlow]}];
correctVSalias[d_, r_, hlow_] := correctVSalias[d, r, hlow] =
    ({NIntegrate[#, {f, hlow, 1 / (2 (d + r))}], NIntegrate[#, {f, 1 / (2 (d + r)), hhi}]}) & /@
     NormNoiseSincedFwindows[d, r, hlow];
DickEffect[d_, r_, hlow_] := Sum[NormNoiseSincedFwindows[d, r, hlow],
    {f, 1 / (d + r), Infinity, 1 / (2 (d + r))}, Assumptions :→ {Element[f, Reals]}];
DickEffect2[d_, r_, hlow_] := Sum[NormNoiseSincedFwindows[d, r, hlow] /. f → n / (d + r),
    {n, 1, Infinity}];
DickEffect3[d_, r_, hlow_] := Sum[NormNoiseSincedFwindows[d, r, hlow] /. f → n / (d + r), {n, 1, 10}];

colours11 = Sort[({#, ColorData[60, "ColorList"][[#]]} & /@ {1, 2, 5, 8, 9, 12, 15, 16, 17, 19, 20, 21}),
     #1[[2, 2]] > #2[[2, 2]] &][[All, 2]];
badge[place_] := Graphics[{{RGBColor[1, 0.886275, 0.239216], RGBColor[0.764706, 0.807843, 0.811765],
       RGBColor[0.811765, 0.356863, 0.0313725]}[[place]], Disk[{0, Sqrt[3]}],
     Polygon[{{1, 0}, {0, Sqrt[3]}, {-1, 0}}], Black, (*,Text[Style[place,Bold,65],{0,Sqrt[3]}],
     colours11[[place]],*)Text[Style[place, Bold, FontSize → Scaled[0.7]], {0, Sqrt[3]}]}];
prizepics = ((# -> badge[#] &) /@ Range[3]) ~Join~
    ((# -> Graphics[{(*colours11[[#]],*)Text[Style[#, Bold, FontSize → Scaled[0.4]]]}] &) /@
      Range[4, numWind]);

FullSimplify[Integrate[UnclippedNoise, {f, hlow, hhi}, Assumptions :→ {0 < hlow < hhi}]]

Log[hhi / hlow]

Export["/home/researcher/Work/Research/Topics_Collaborators/Atomic/Figures/comparisonLegend.eps",
 LineLegend[{Darker[Blue], Green, Blend[{Green, Blue}, {0.6, 0.4}],
    Directive[Dashed, Red], Red, Blend[{Green, Blue, Lighter[Orange]}, {0.7^2, 0.4 × 0.7, 0.3}],
    Blend[{Red, Lighter[Orange]}, {0.6, 0.4}], Lighter[Orange]},
  Table[ToString[i] <> "                          ", {i, 1, 8}], LegendFunction → "Frame"]]

"/home/researcher/Work/Research/Topics_Collaborators/Atomic/Figures/comparisonLegend.eps"
```

In[38]:=

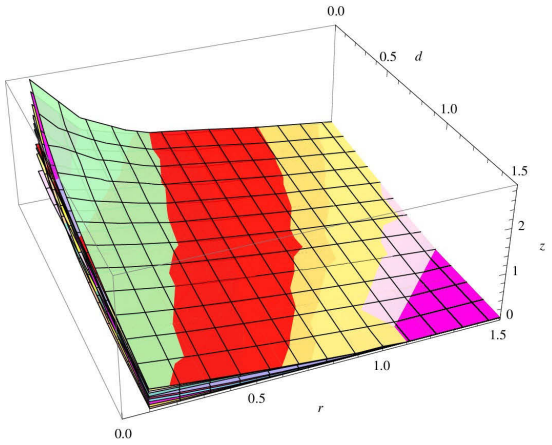
```
scaleText = 0.02;
DrawGrid[d_, r_, hlow_] := GraphicsGrid[
   {{"w", Grid[{{Text[Style["0", FontSize → Scaled[scaleText]]], Slider[Dynamic[r], {0, 2}],
         Text[Style["2                r", FontSize → Scaled[scaleText]]],
         Text[Style[r, FontSize → Scaled[scaleText]]]},
        {Text[Style["0", FontSize → Scaled[scaleText]]], Slider[Dynamic[d], {0, 2}],
         Text[Style["2                d", FontSize → Scaled[scaleText]]],
         Text[Style[d, FontSize → Scaled[scaleText]]]},
        {Text[Style["0", FontSize → Scaled[scaleText]]], Slider[Dynamic[hlow], {0, 2}],
         Text[Style["2                h", FontSize → Scaled[scaleText]]],
         Text[Style[hlow, FontSize → Scaled[scaleText]]]}}, Alignment → Left, ItemSize → Automatic],
      SpanFromLeft, SpanFromLeft, SpanFromLeft, "M", "P"}} ~Join~ Transpose[
     {MapThread[Show[Plot[#1, {t, -3 (d + r), 0}, Axes → True, Ticks → None,
          PlotRange → {Automatic, {-1, 1}}, Filling → Axis, FillingStyle → #2, PlotStyle → #2],
        Graphics[Text[Style["" <> ToString[#3], Bold, FontSize → Scaled[0.2]],
          {-8 / 3 (d + r), -0.7}]]] &, {Twindows[d, r], colours11, Range[numWind]}],
      Plot[#, {f, 0, 3 / (2 r)}, PlotRange → {{0, 3 / (2 r)}, {0, 1}}, Ticks → None,
        PlotStyle → {Darker[Blue], Green}] &/@ NormFwindowsVSsinc[d, r],
      (Show[Plot[#, {f, 0, 1 / (2 (d + r))}, PlotStyle → ({Blend[{Green, Blue}, {0.6, 0.4}]} ~Join~
            Table[Red, {n, -numTrans, numTrans - 1}]), PlotRange → {{0, 2 / (d + r)}, {0, 1}},
          Ticks → None], Plot[#[[1]] HeavisideTheta[f - 1 / (2 (d + r))], {f, 0, 2 / (d + r)},
          PlotStyle → Directive[Dashed, Red], PlotRange → {{0, 2 / (d + r)}, {0, 1}},
          Ticks → None]]) &/@ NormTransSincedFwindows[d, r],
      Plot[#, {f, 0, hhi}, PlotStyle → {Blend[{Green, Blue, Lighter[Orange]}, {0.7^2, 0.4 × 0.7, 0.3}],
          Blend[{Red, Lighter[Orange]}, {0.6, 0.4}], Lighter[Orange]}, PlotRange →
         {{0, 1 / (d + r)}, {0, 3}}, Ticks → None] &/@ NormNoiseSincedFwindowsVSalias[d, r, hlow],
      (*Min[2,Max[0.5,10#[[2]]/.f→ 1/(2(d+r)) - 0.001]]*)
      radInner = Sqrt[#[[2]] / totalNoise[hlow]] / (1 - Sqrt[#[[2]] / totalNoise[hlow]]) & /@
        correctVSalias[d, r, hlow];
      MapThread[Show[PieChart[{#1[[1]], totalNoise[hlow] - #1[[1]]}, SectorOrigin → {{Pi / 2}, #2},
          ChartLabels → None, ChartStyle → {Directive[EdgeForm[{Lighter[Orange]}],
            Lighter[Blend[{Green, Blue, Orange}, {0.7^2, 0.4 × 0.7, 0.3}]]],
           Directive[Lighter[Orange], EdgeForm[{Lighter[Orange]}]]}],
        Graphics[{Blend[{Red, Lighter[Orange]}, {0.6, 0.4}], Disk[{0, 0}, #2]}]] &,
       {correctVSalias[d, r, hlow], radInner}],
      Text[Style[SetPrecision[totalNoise[hlow] - (#[[1]] - #[[2]]), 4], FontSize → Scaled[0.03]]] & /@
       correctVSalias[d, r, hlow],
      Sort[MapThread[({#1[[2]], #2}) &, {Sort[MapThread[{#1[[1]] - #1[[2]], #2} &,
           {correctVSalias[d, r, hlow], Range[numWind]}], (#1[[1]] > #2[[1]] &)],
         Range[numWind]}], (#1[[1]] < #2[[1]] &)][[All, 2]] /. prizepics
     }], Frame → All, ImageSize → Scaled[0.515]];
```

```
Manipulate[DrawGrid[d, r, hlow], {d, 0, 10}, {r, 0.1, 10}, {hlow, 0.01, 0.2}]
```



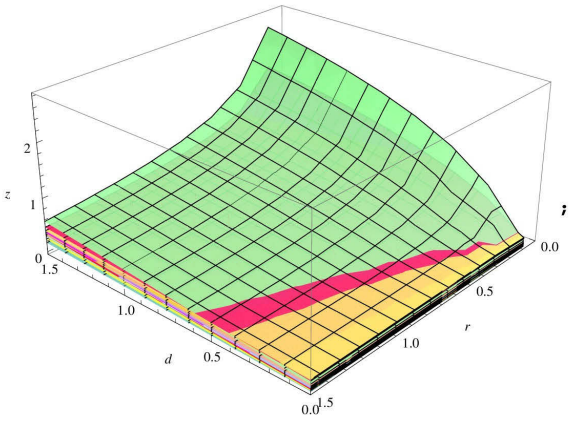
## Best Performing Windows for Correction (noise uncorrected, lower is better)

```
ReleaseHold[
 Hold[Show[MapThread[(ListPlot3D[Flatten[#1, 1], PlotStyle → {#2, Opacity[0.9]}, PlotRange →
          {{0, 1.5}, {0, 1.5}, All}, AxesLabel → {d, r, z}, Lighting → "Neutral"]) &,
      {Transpose[Table[{d, r, totalNoise[hlow] - #[[1]]} & /@ correctVSalias[d, r, hlow] ,
         {d, 0, 2, 0.2}, {r, 0.1, 2, 0.2}], {2, 3, 1}], colours11}]]] /. hlow → 0.1]
```




## Worst Performing Windows for Aliasing

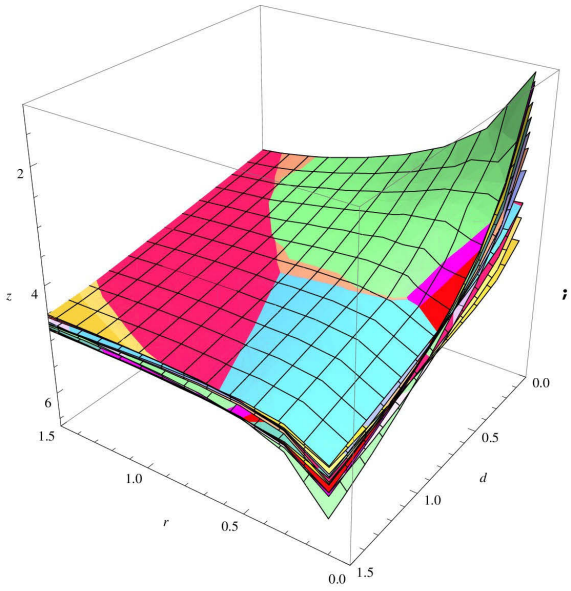
```
Show[MapThread[
   (ListPlot3D[Flatten[#1, 1], PlotStyle → {#2, Opacity[0.9]}, PlotRange → {{0, 1.5}, {0, 1.5}, All},
      AxesLabel → {d, r, z}, Lighting → "Neutral", BoxRatios → {1, 1, 0.5}]) &,
   {Transpose[Table[{d, r, #[[2]]} & /@ correctVSalias[d, r, 0.1], {d, 0, 2, 0.2}, {r, 0.1, 2, 0.2}],
     {2, 3, 1}], colours11}]]
```

worstalias =  ;



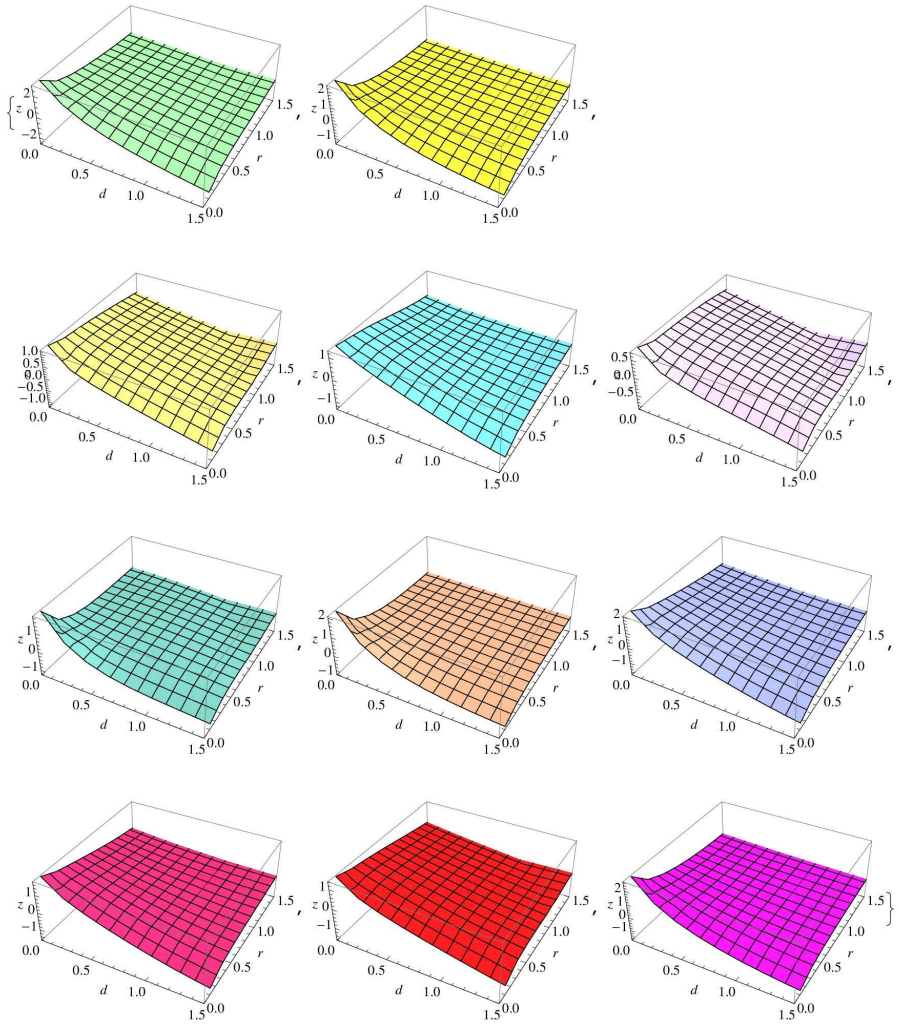
## Best Overall Performance of Windows (final noise, lower is better)

```
ReleaseHold[
 Hold[Show[MapThread[(ListPlot3D[Flatten[#1, 1], PlotStyle → {#2, Opacity[0.99]}, PlotRange →
          {{0, 1.5}, {0, 1.5}, All}, AxesLabel → {d, r, z},
        Lighting → "Neutral", Mesh → Automatic, BoxRatios → {1, 1, 1}]) &,
      {Transpose[Table[{d, r, totalNoise[hlow] - #[[1]] + #[[2]]} & /@ correctVSalias[d, r, hlow],
         {d, 0, 2, 0.2}, {r, 0.1, 2, 0.2}], {2, 3, 1}], colours11}]]] /. {hlow → 0.1}]
```

overallperf = 

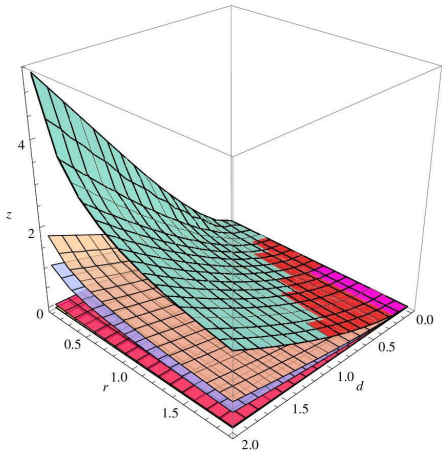
;

```
Flatten[MapThread[(ListPlot3D[Flatten[#1, 1], PlotStyle → {#2, Opacity[0.9]},
      PlotRange → {{0, 1.5}, {0, 1.5}, All}, AxesLabel → {d, r, z}, Lighting → "Neutral"]) &,
  {Transpose[Table[{d, r, totalNoise[hlow] - #[[1]] - #[[2]]} & /@ correctVSalias[d, r, 0.2],
      {d, 0, 2, 0.2}, {r, 0.1, 2, 0.2}], {2, 3, 1}], colours11}]]
```



## Worst Dick Effect Windows

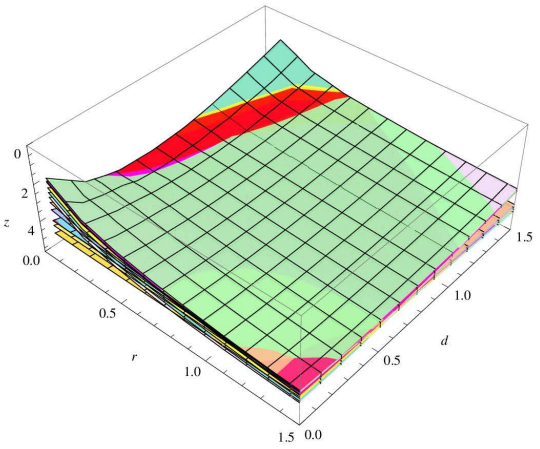
```
Show[MapThread[(ListPlot3D[Flatten[#1, 1], PlotStyle → {#2, Opacity[0.8]},
      PlotRange → All, AxesLabel → {d, r, z}, Lighting → "Neutral", BoxRatios → {1, 1, 1}]) &,
  {Transpose[Table[{d, r, #} & /@ DickEffect3[d, r, 0.1], {d, 0, 2, 0.2}, {r, 0.1, 2, 0.2}], {2, 3, 1}],
   colours11}]]
```




## Best Overall Performance of Windows including penalty for Dick Effect

```
ReleaseHold[
 Hold[Show[MapThread[(ListPlot3D[Flatten[#1, 1], PlotStyle → {#2, Opacity[0.9]}, PlotRange →
          {{0, 1.5}, {0, 1.5}, All}, AxesLabel → {d, r, z}, Lighting → "Neutral"]) &,
      {Transpose[Table[{d, r, #} & /@ MapThread[(totalNoise[hlow] - #[[1]] + #[[2]] - #2) &,
           {correctVSalias[d, r, hlow], DickEffect3[d, r, 0.1]}],
        {d, 0, 2, 0.2}, {r, 0.1, 2, 0.2}], {2, 3, 1}], colours11}]]] /. hlow → 0.1]
```

dickeffectpenalized =

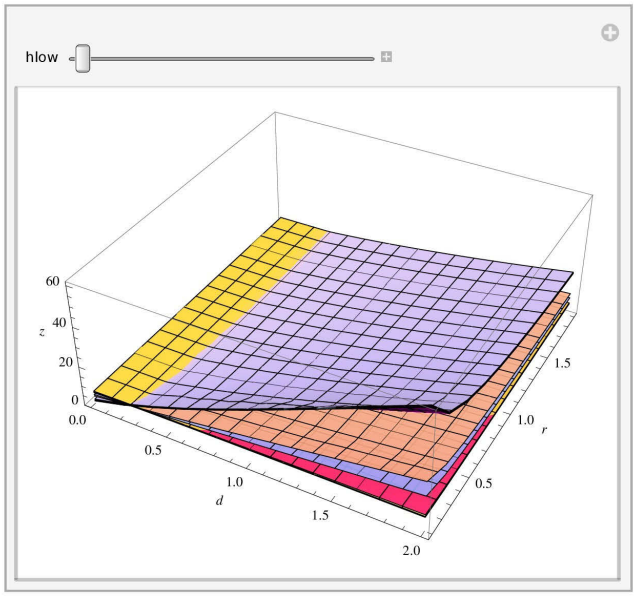



;

## Best Overall Performance of Windows with low hlow cutoff including high penalty for Dick Effect

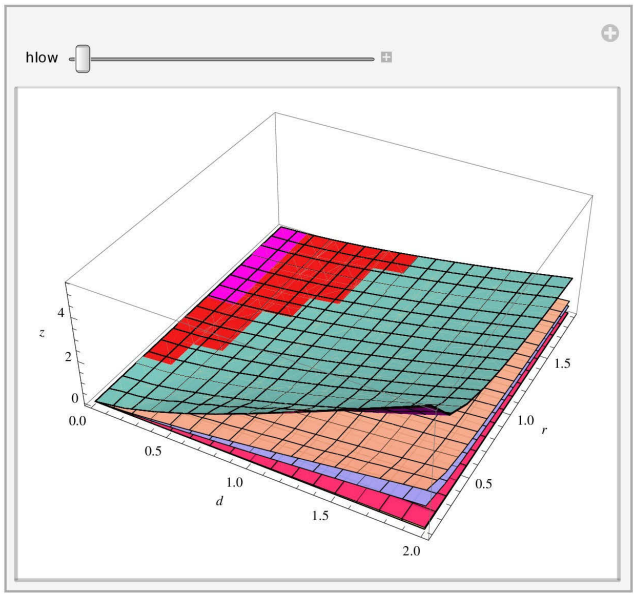
In[40]:=
```
Manipulate[Show[MapThread[(ListPlot3D[Flatten[#1, 1], PlotStyle → {#2, Opacity[0.9]},
      PlotRange → {All, All, All}, AxesLabel → {d, r, z}, Lighting → "Neutral"]) &,
   {Transpose[Table[{d, r, #} & /@ MapThread[(totalNoise[hlow] - #[[1]] + #[[2]] + #2) &,
       {correctVSalias[d, r, hlow], DickEffect3[d, r, hlow] 10}],
     {d, 0, 2, 0.2}, {r, 0.1, 2, 0.2}], {2, 3, 1}], colours11}]], {hlow, 0.01, 0.2}]
```

Out[40]=

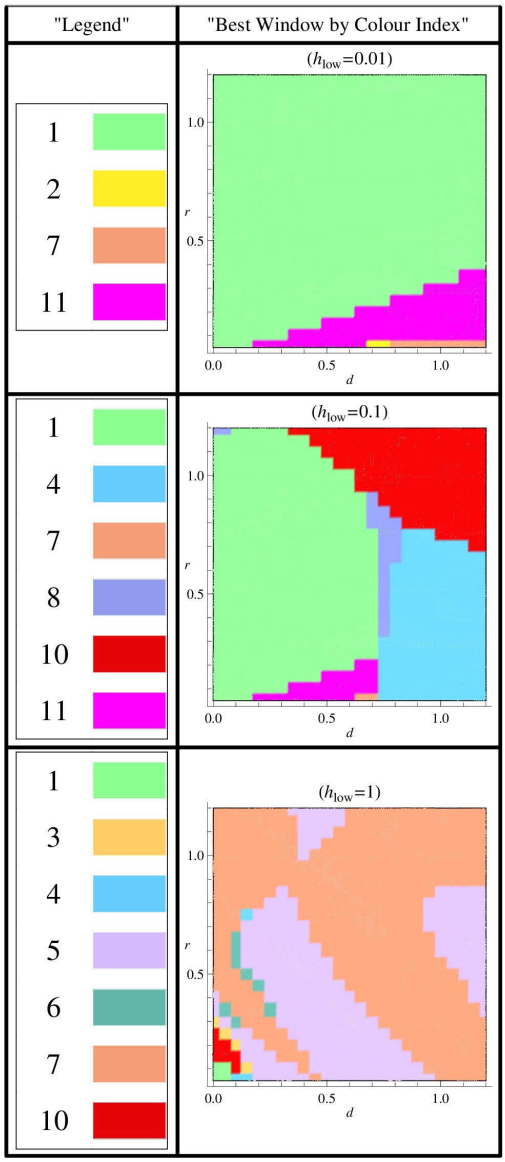


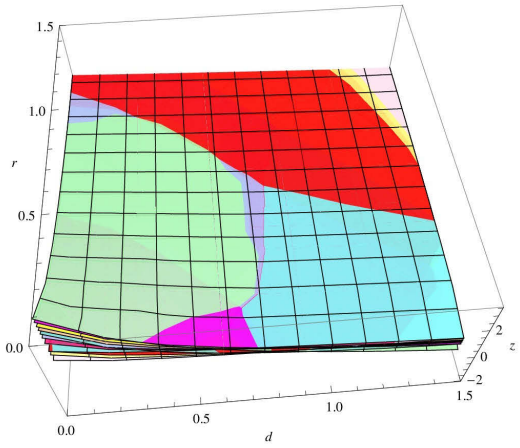

## Worst Dick Effect Windows for low hlow

```
Manipulate[
 Show[MapThread[(ListPlot3D[Flatten[#1, 1], PlotStyle → {#2, Opacity[0.9]}, PlotRange → All,
      AxesLabel → {d, r, z}, Lighting → "Neutral"]) &,
   {Transpose[Table[{d, r, #} & /@ DickEffect3[d, r, hlow], {d, 0, 2, 0.2}, {r, 0.1, 2, 0.2}], {2, 3, 1}],
    colours11}]], {hlow, 0.01, 0.2}]
```



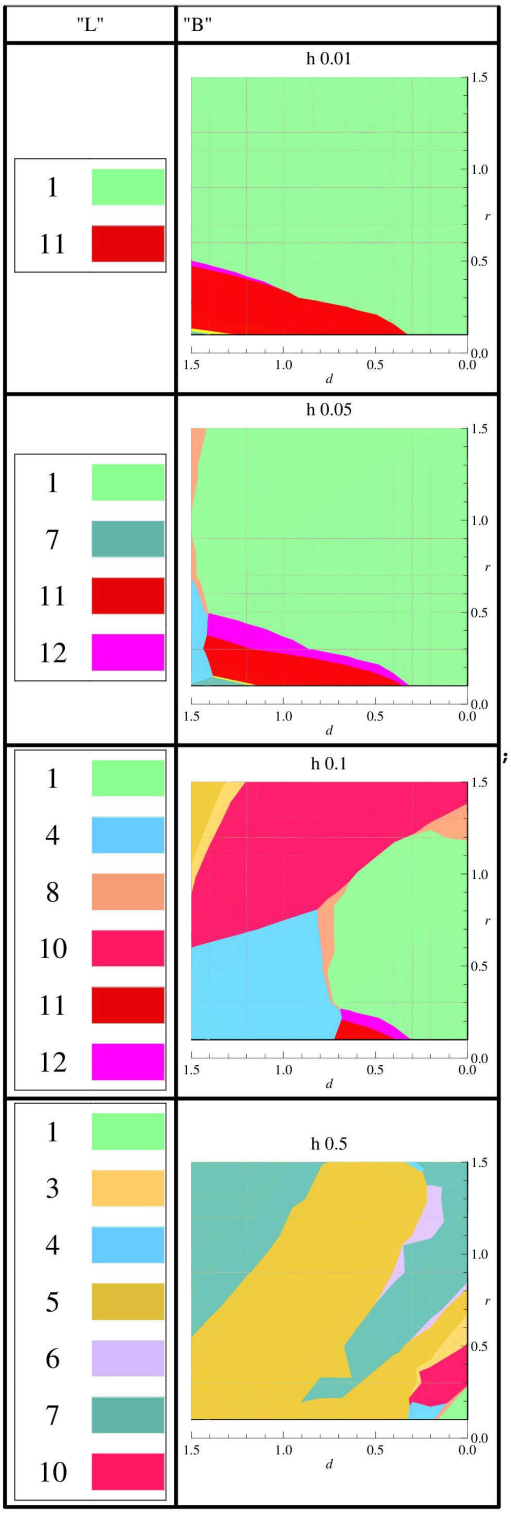
```
Remove[drBestWindowPerformance];
drStepSize = 0.1;
rStart = drStepSize;
dStart = 0;
drEnd = 1.2;
(*bestWindowhlow = 0.1;*)

drBestWindowPerformance[bestWindowhlow_] := drBestWindowPerformance[bestWindowhlow] =
   Table[Sort[MapThread[({d, r, totalNoise[bestWindowhlow] - #[[1]] + #[[2]], #2}) &,
       {correctVSalias[d, r, bestWindowhlow], Range[numWind]}], #1[[3]] < #2[[3]] &][[1]],
    {d, dStart, drEnd, drStepSize}, {r, rStart, drEnd, drStepSize}];


hlowruns = {0.01, 0.05, 0.1, 0.5};
drBestWindowPerformance[#] & /@ hlowruns;

(*We get rounded edges because of the Floor +1.5 :)*)
(*bestWindowhlow=0.2;*)
Grid[{{Text["Legend"], Text["Best Window by Colour Index"]}}~Join~
  Transpose[
   {(GraphicsGrid[({Graphics[Text[Style[#, 19]]], Graphics[{colours11[[#]], Rectangle[{0.2, 0.1},
               {0.4, 0.2}]}]}) & /@ Union[Flatten[drBestWindowPerformance[#][[All, All, 4]]]],
        ImageSize → 100, Frame → True, Spacings → {0, 0}] &) /@ hlowruns,
    (ListPlot3D[drBestWindowPerformance[#][[All, All, 3]],
        (*VertexColors→Map[colours11[[#]]&,pnts[[All,All,4]],{2}],*)
        DataRange → {{dStart, drEnd}, {rStart, drEnd}}, AxesLabel → {d, r, None},
        ColorFunction -> Function[{d, r, z}, colours11[[drBestWindowPerformance[#][[
            Floor[(d - dStart) / drStepSize + 1.5], Floor[(r - rStart) / drStepSize + 1.5], 4]]]]],
        Axes → {True, True, False}, PlotRange → All, BoxRatios → {1, 1, 0.001},
        ClippingStyle → None, Mesh → None, InterpolationOrder → 2,
        Boxed → False, ViewPoint → Top, ColorFunctionScaling → False,
        PlotLabel → "(h_low=" <> ToString[#] <> ")", ImageSize → 200]) & /@ hlowruns}]
 ,
 Frame →
  All]
```




```
(*We get rounded edges because of the Floor +1.5 :)*)
(*bestWindowhlow=0.2;*)
Grid[{{Text["L"], Item[Text["B"], Alignment → Left]}}~Join~
  Transpose[
   {(GraphicsGrid[({Graphics[Text[Style[#, 19]]], Graphics[{colours11[[#]], Rectangle[{0.2, 0.1},
                {0.4, 0.2}]}]}) & /@ Union[Flatten[drBestWindowPerformance[#][[All, All, 4]]]],
         ImageSize → 100, Frame → True, Spacings → {0, 0}] &) /@ hlowruns,
    (Function[{hlow}, Show[MapThread[(ListPlot3D[Flatten[#1, 1], PlotStyle → {#2, Opacity[0.99]},
               PlotRange → {{0, 1.5}, {0, 1.5}, All}, AxesLabel → {d, r, z},
               Lighting → "Neutral", ViewPoint → Bottom, BoxRatios → {1, 1, 0.001},
               ClippingStyle → None, Mesh → None, Boxed → False, ViewPoint → Top,
               PlotLabel → "h " <> ToString[hlow], ImageSize → 200, Axes → {True, True, False}]) &,
           {Transpose[Table[{d, r, totalNoise[hlow] - #[[1]] + #[[2]]} & /@ correctVSalias[d, r, hlow],
               {d, 0, 2, 0.2}, {r, 0.1, 2, 0.2}], {2, 3, 1}], colours11}]][#]) & /@ hlowruns}]
 ,
 Frame →
  All]
```

bestoverdr =

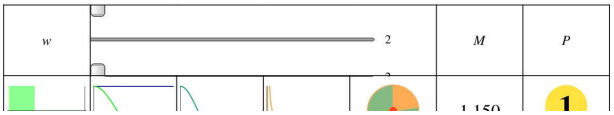


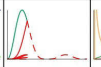
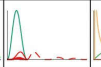
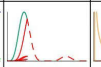
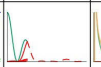
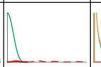
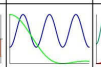
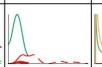
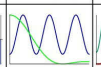
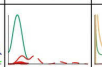
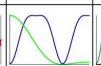
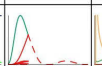

;

```
Export["/home/researcher/Work/Research/Topics_Collaborators/Atomic/bestoverdr.eps", Out[335]]
```

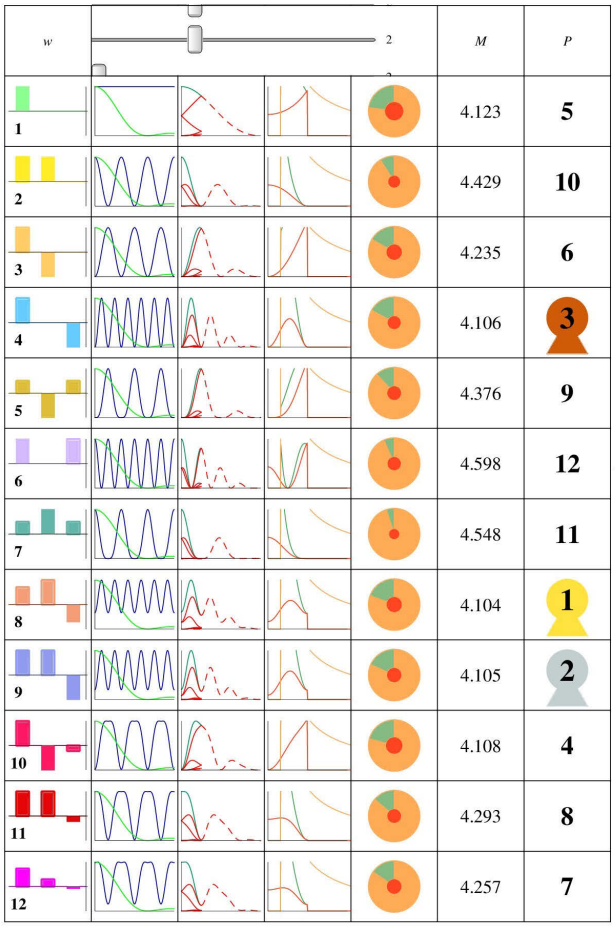
```
"/home/researcher/Work/Research/Topics_Collaborators/Atomic/bestoverdr.eps"
```

```
{g1, g2, g3, g4} = ReleaseHold[Hold[DrawGrid[d, r, hlow]] /.
    ( {d → #[[1]], r → #[[2]], hlow → 0.1} & /@ {{0, 0.1}, {1, 0.5}, {0.8, 0.8}, {1, 1}})]
```

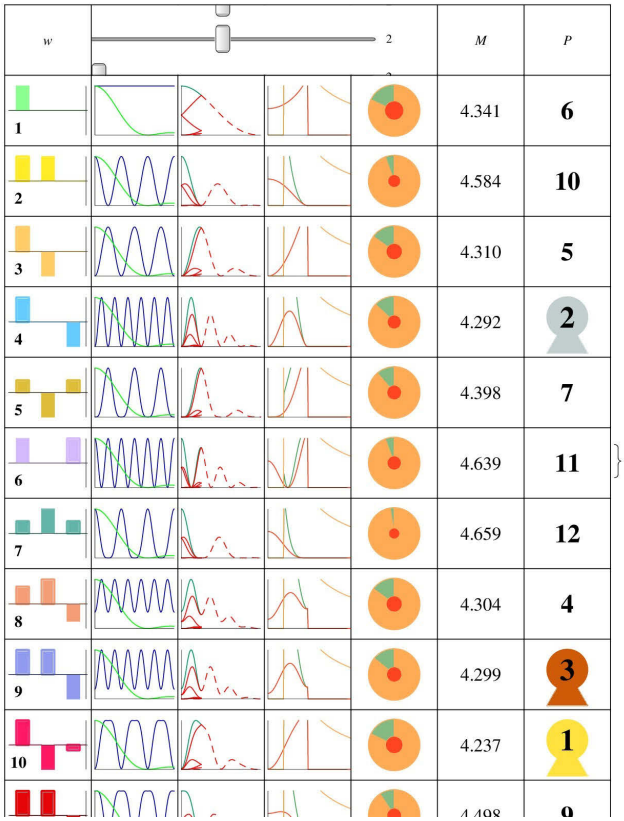


| | | | | | | |
|---|---|---|---|---|---|---|
| 1 |  |  |  |  | 1.150 | **1** |
| 2 |  |  |  |  | 1.647 | **4** |
| 3 |  |  |  |  | 3.660 | **11** |
| 4 |  |  |  |  | 3.400 | **10** |
| 5 |  |  |  |  | 4.083 | **12** |
| 6 |  |  |  |  | 2.144 | **6** |
| 7 |  |  |  |  | 1.895 | **5** |
| 8 |  |  |  |  | 2.421 | **7** |
| 9 |  |  |  |  | 2.809 | **8** |
| 10 |  |  |  |  | 3.387 | **9** |
| 11 |  |  |  |  | 1.363 | **3** |
| 12 |  |  |  |  | 1.304 | **2** |

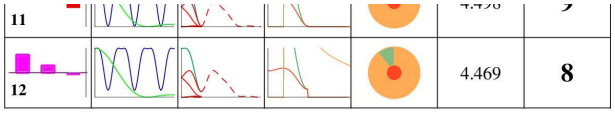
| *w* |  | | | | *M* | *P* |
|---|---|---|---|---|---|---|
| 1 |  |  |  |  | 4.362 | **5** |
| 2 |  |  |  |  | 4.547 | **10** |
| 3 |  |  |  |  | 4.403 | **6** |
| 4 |  |  |  |  | 4.230 | **1** |
| 5 |  |  |  |  | 4.503 | **9** |
| 6 |  |  |  |  | 4.729 | **12** |
| 7 |  |  |  |  | 4.639 | **11** |
| 8 |  |  |  |  | 4.273 | **3** |
| 9 |  |  |  |  | 4.256 | **2** |
| 10 |  |  |  |  | 4.301 | **4** |
| 11 |  |  |  |  | 4.443 | **8** |
| 12 |  |  |  |  | 4.422 | **7** |



| w | | M | P |
|---|---|---|---|
| 1 | | 4.123 | 5 |
| 2 | | 4.429 | 10 |
| 3 | | 4.235 | 6 |
| 4 | | 4.106 | 3 |
| 5 | | 4.376 | 9 |
| 6 | | 4.598 | 12 |
| 7 | | 4.548 | 11 |
| 8 | | 4.104 | 1 |
| 9 | | 4.105 | 2 |
| 10 | | 4.108 | 4 |
| 11 | | 4.293 | 8 |
| 12 | | 4.257 | 7 |



| w | | M | P |
|---|---|---|---|
| 1 | | 4.341 | 6 |
| 2 | | 4.584 | 10 |
| 3 | | 4.310 | 5 |
| 4 | | 4.292 | 2 |
| 5 | | 4.398 | 7 |
| 6 | | 4.639 | 11 |
| 7 | | 4.659 | 12 |
| 8 | | 4.304 | 4 |
| 9 | | 4.299 | 3 |
| 10 | | 4.237 | 1 |
| 11 | | 4.498 | 9 |




```
Export["/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/g1.eps",
 g1, ImageSize → 1000]
Export["/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/g2.eps",
 g2, ImageSize → 1000]
Export["/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/g3.eps",
 g3, ImageSize → 1000]
Export["/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/g4.eps",
 g4, ImageSize → 1000]
```

```
/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/g1.eps

/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/g2.eps

/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/g3.eps

/home/researcher/Work/Research/Topics_Collaborators/PhD/atomicclock/Figures/g4.eps
```

```
#ifndef RAN_H
#define RAN_H
#include <time.h>
#include <climits>

/**
 * Press, Teukolsky, Vetterling, Flannery
 * NUMERICAL RECIPES - The Art of Scientific Computing - Third Edition (2007)
 * p. 342 / 343
 */
typedef int Int; // 32 bit integer
typedef unsigned int Uint;

typedef long long int Llong; // 64 bit integer
typedef unsigned long long int Ullong;

typedef double Doub; // default floating type
typedef long double Ldoub;

struct Ran {
        Ullong u,v,w;
        Ran(Ullong j) : v(4101842887655102017LL), w(1) {
                u = j ^ v; int64();
                v = u; int64();
                w = v; int64();
        }
        inline Ullong int64() {
                u = u * 2862933555777941757LL + 7046029254386353087LL;
                v ^= v >> 17; v ^= v << 31; v ^= v >> 8;
                w = 4294957665U*(w & 0xffffffff) + (w >> 32);
                Ullong x = u ^ (u << 21); x ^= x >> 35; x ^= x << 4;
                return (x + v) ^ w;
        }
        inline Doub doub() { return 5.42101086242752217E-20 * int64(); }
        inline Uint int32() { return (Uint)int64(); }
};

//include <climits>
//define UCHAR_MAX 255

unsigned time_seed()
{
time_t now = time ( 0 );
unsigned char *p = (unsigned char *)&now;
unsigned seed = 0;
size_t i;

for ( i = 0; i < sizeof now; i++ )
seed = seed * ( UCHAR_MAX + 2U ) + p[i];

return seed;
}//2U means unsigned 2 (which is not really needed)


#endif
#ifndef CHAIN
#define CHAIN
#define LOGAUTOCORR 6

#include "ran.h"

class chain {

private:

        double q;
        double Q[2][2];
```

```
68          double x[2][2];
69          unsigned int N;
70          char *k, *oldk; //error vector
71          double (*pathEnd)[2];
72          double (*oldPathEnd)[2];
73          double prob, oldProb;
74
75
76          unsigned int numCorrelations, keepIndex;
77          static const unsigned int autocorr = 1<<LOGAUTOCORR;
78          double keepLast[autocorr];
79          double correlation[autocorr];
80          double mean;
81
82          unsigned int i;
83
84  public:
85          /*Constructor*/
86
87          chain(const double s, const double a, const double d, const unsigned int Num) : q((s
    +1)/2), N(Num)
88          {
89                  Q[0][0] = q;
90                  Q[0][1] = 1-q;
91                  Q[1][0] = 1-q;
92                  Q[1][1] = q;
93
94                  x[0][0] = a-d;
95                  x[0][1] = 1-(a-d);
96                  x[1][0] = a+d;
97                  x[1][1] = 1-(a+d);
98                  k = new char[N](); /*the empty initializer () sets all values to zero*/
99                  pathEnd = new double[N][2];
100                 oldPathEnd = new double[N][2];
101                 oldk = new char[N]();
102                 resetCorrelationData();
103                 resetChain();
104         };
105
106         /*Destructor*/
107
108         ~chain()
109         {
110
111 /*              free(k);
112                 free(pathEnd);
113                 free(oldPathEnd);*/
114                 delete [] k;
115                 delete [] pathEnd;
116                 delete [] oldPathEnd;
117
118         }
119
120         void resetChain()
121         {
122                 for (i=0;i<N;i++)
123                 {
124                         k[i]=0;
125
126                 }
127                 calcProb(0);
128                 //oldProb=prob; should never be necessary
129                 //resetCorrelationData(); leave it to the user to decide
130         }
131
132 /*This function calculates the probability of occurence of a given markov chain (as given by k
    representing error vector)
```

```
133 */
134         double calcProb(unsigned int start)
135         {
136                 if (start == 0)
137                 {
138                         pathEnd[0][0] = x[0][(int)k[0]]/2;
139                         pathEnd[0][1] = x[1][(int)k[0]]/2;
140                         start++;
141                 }
142
143                 for(i=start;i<N;i++)
144                 {
145                         pathEnd[i][0] = (pathEnd[i - 1][0]*Q[0][0] + pathEnd[i - 1][1]*Q[1][0]) *
    x[0][(int)k[i]];
146                         pathEnd[i][1] = (pathEnd[i - 1][0]*Q[0][1] + pathEnd[i - 1][1]*Q[1][1])* x
    [1][(int)k[i]];
147
148                 };
149
150                 prob = (pathEnd[N-1][0] + pathEnd[N-1][1]);
151
152                 return prob;
153         }
154
155         void randomizeChain(Ran &randomgen)
156         {
157                 for(i=0;i<N;i++)
158                 {
159                         k[i] = floor(2*randomgen.doub());
160                 }
161
162                 calcProb(0);
163         }
164
165         inline double getProb()
166         {
167                 return prob;
168         }
169
170         inline void setChainZero(const unsigned int location)
171         {
172                 k[location]=0; /*0 < location < N  perhaps introduce a private start=location
    variable */
173         }
174
175         inline void setChainOne(const unsigned int location)
176         {
177                 k[location]=1; /*0 < location < N*/
178         }
179
180         inline bool checkChainOne(const unsigned int location)
181         {
182                 return (k[location] == 1);
183         }
184
185         inline void flipChain(const unsigned int location)
186         {
187                 k[location] ^= 1;
188         }
189
190         unsigned int flipChain(const double probFlip, Ran &randomgen) //overloaded returns first
    location flipped or N for none
191         {
192                 unsigned int location=N;
193
194                 for(i=0;i<N;i++)
195                 {
```

```
196
197                         if (randomgen.doub()<probFlip)
198                         {
199                                 k[i] ^= 1;
200                                 if (location == N) location = i;
201                         }
202                 }
203
204                 return location;
205         }
206
207         void backupProb(const unsigned int from)
208         {
209                 unsigned int j;
210                 for(i=0;i<2;i++)
211                 {
212                         for (j=from;j<N;j++)
213                         {oldPathEnd[j][i] = pathEnd[j][i];}
214                 };
215                 oldProb = prob;
216         }
217
218         void restoreProb(const unsigned int from)
219         {
220                 unsigned int j;
221                 for(i=0;i<2;i++)
222                 {
223                         for (j=from;j<N;j++)
224                         {pathEnd[j][i] = oldPathEnd[j][i];}
225                 };
226                 prob = oldProb;
227         }
228
229         void backupK(const unsigned int from)
230         {
231                 for (i=from;i<N;i++)
232                 {oldk[i] = k[i];}
233         }
234
235         void restoreK(const unsigned int from)
236         {
237                 for (i=from;i<N;i++)
238                 {k[i] = oldk[i];}
239         }
240
241         bool candidateAccept(Ran &randomgen,unsigned int expectedflip)
242         {
243                 backupK(0);
244                 unsigned int firstLocation = flipChain(expectedflip/(double)N,randomgen); //All
    sites are potentially flipped
245
246                 bool accepted = true;
247
248                 if (firstLocation < N) //if firstLocation = N it means no change was made and
    self-same candidate was accepted
249                 {
250                         backupProb(firstLocation);
251                         calcProb(firstLocation);
252
253                         if(randomgen.doub() > prob/oldProb)
254                         {
255                                 restoreK(firstLocation);
256                                 restoreProb(firstLocation);
257                                 accepted = false;
258                         }
259                 }
260
```

```
261                     return accepted;
262             }
263
264             void resetCorrelationData()
265             {
266                     numCorrelations = 0;
267                     keepIndex =0;
268                     mean =0;
269                     for(i=0;i<autocorr;i++)
270                     {
271                             correlation[i] = 0;
272                             keepLast[i]=0;
273                     }
274             }
275
276             void gatherCorrelationData()
277             {
278                     numCorrelations++;
279                     mean += prob;
280                     correlation[0] += prob*prob;
281
282                     for (i=1;i<=keepIndex;i++)
283                     correlation[i] += prob*keepLast[keepIndex-i];
284
285                     for (i=keepIndex+1;i<autocorr;i++)
286                     correlation[i] += prob*keepLast[autocorr+keepIndex-i];//(autocorr-1)-(i-(keepIndex
    +1))
287
288                     keepLast[keepIndex++]=prob;
289                     if (keepIndex == autocorr) keepIndex = 0;
290             }
291
292     //A few simpliying assumptions have been made in correlationTime: 1: numCorrelations is large, 2:
    Population Variance ~ Sample Variance (~satisfied when 1 is true)
293             double correlationTime()
294             {
295                     double integrate=0;
296                     unsigned int numOccur;
297
298                     if (numCorrelations >= autocorr)
299                     {
300                             for (i=1;i<autocorr;i++)
301                             {
302                                     numOccur = numCorrelations - i;
303                                     if (numOccur>0) integrate += correlation[i]/(double)numOccur;
304                             }
305
306                             double avg = mean/(double)numCorrelations;
307                             double var = correlation[0]/(double)numCorrelations - avg*avg;
308                             integrate = (integrate - autocorr*avg*avg)/var;
309
310                             return integrate;
311
312                     } else return NAN;
313             }
314     };
315
316     #endif
317     #ifndef MONTE
318     #define MONTE
319     #include <stdio.h>
320     #include <cmath> //c++ library
321     #include <stdlib.h>
322     #include "ran.h"
323     #include "chain.h"
324
325     #define EPS 2.2250738585072014e-308
```

```
326
327 struct onerun {
328 double mean,error,acceptance,correlationTime;};
329
330 const unsigned int onerunSize = sizeof(onerun)/sizeof(double);
331
332 /*This function calculates the output entropy (unrelativised (not divided by N)) using a Monte
    Carlo algorithm to sample from the possible markov chains*/
333
334 onerun run_MCMC_Entropy(chain &channelChain, const unsigned int flips, Ullong burnin, Ullong
    skips, const unsigned int logSamples, Ran &randomgen)
335 {
336
337         //Entropy Var
338         double prob, logProb = 0;
339         double m2 = 0, mean = 0, delta = 0;
340
341         //Monte Carlo Var
342         Ullong i, j, numAccepts = 0, numSamples = 1 << logSamples;//pow(2,logSamples);
343
344         for(i=0;i<burnin;i++) channelChain.candidateAccept(randomgen,flips);
345
346         for(i=0;i<numSamples;i++)
347         {
348
349                 for(j=0;j<skips+1;j++) {if (channelChain.candidateAccept(randomgen,flips))
    numAccepts++;}
350
351                 prob = channelChain.getProb();
352                 channelChain.gatherCorrelationData();
353
354                 if (prob > EPS) logProb = -log2(prob); else printf("Extremely rare event
    happenned that shouldn't have happened");
355
356                 delta = logProb - mean; //using old mean
357                 mean += delta/(i+1); //making new mean
358                 m2 += delta*(logProb-mean); //using new mean towards variance
359
360         }
361
362         onerun finalAnswer;
363         finalAnswer.mean = mean;
364         finalAnswer.error = sqrt((m2/(numSamples-1))/numSamples);    //m2/(numSamples-1) is the
    sample variance.... /numSamples is the variance of the estimator .... sqrt() is the standard
    deviation (biased) of the estimator, often called standard error
365         finalAnswer.acceptance = numAccepts/(double)(numSamples*(skips+1));
366         finalAnswer.correlationTime = channelChain.correlationTime();
367
368         return finalAnswer;
369
370 }
371 #endif
372 /*
373
374 Program Name: MCMC_Entropy_Vary_One_Parameter.cpp
375
376 Author: Ismail Akhalwaya
377
378 Title: Monte Carlo Simulation of Quantum Channel with Noise Memory
379
380 Description: The capacity of quantum channels with noise memory is a valuable and fascinating
    subject within Quantum Information Theory. The purpose of this program is to numerically
    calculate the output entropy of a toy channel with noisy memory. The channel is constructed from
    two depolarizing channels (providing the noise) selected by a markov chain (providing the
    'memory').
381
382 Programming Detail: The output entropy is calculated by summing the log(probability) over
```

```
exponentially many markov walks. A Markov Chain Metropolis-Hastings Monte Carlo Importance
Sampling approach is used in order to make the calculation feasible while sacrificing accuracy.
The code is parallelized using MPI to enable execution on a supercomputer. This program explores
how a statistic of the sampling (the entropy itself) varies as one parameter is varied. Thereby
trying to understand what effect that one parameter has on the algorithm.

There are eight parameters in total: four relating to the channel and four to the Monte Carlo
algorithm:

Channel:

1) s is a real number that specifies the Markov chain probability: -1 <= s <= 1. s=-1 equates to
a deterministic markov walk which flips at every step, s=0 equates to no memory walk, s=1 means a
no flip walk.

2) a is a real number which specifies the average noise of the two depolarizing channels:
0<=a<=1, a=0 means no noise, a=1 means complete noise.

3) d is a real number which specifies the difference from the average of the two channels: 0 <= d
<= Min[a,1-a]

4) chain is the length of the message/channel chain

Monte Carlo:

5) flips is the starting expected number of flips (average number of changed sites between
candidate and current error vector index)

6) burnin is the number of steps to discard at the beginning of a Monte Carlo run.

7) skips is the number of steps to discard during a run between draws that are used towards the
entropy.

8) logSamples is the log_2 of the number of Monte Carlo samples to be drawn by each task. (Can't
be more than 57/2 ~ 30 (base 10) = 100 (base2) because of Ran period of 10^57 and can't be more
that 19 (base 10) = 64 (base 2) because of unsigned long long int counter)

Required Command line input:

This program varies one of the parameters while keeping the other seven constant. Therefore the
program needs those seven constants to be provided. For the parameter that is to be varied, the
start, end and step size values are required.

This is specified by providing all the parameters in the order listed above. The one to be varied
is to set to its start value. After the eight entries the next three entries are the index, end
and step values for the to-be-varied parameter. The index is the zero-started count of the
parameter to vary in the list of variables above.

The last command line argument is the number of times to repeat everything. (SADNFBSSIESR)

Typical Call:
e.g. to vary the chain length:

mpirun -np 10 MCMC_Entropy_Vary_One_Paramter 0.8 0.7 0.2 10 1 100 20 25 4 100 10 30

Copyright 2012,2013

GPL 2.0

*/

#include <stdio.h>
#include <cmath> //c++ math library
#include "mpi.h"
#include "monte.h"
#include "chain.h"

/*The main section of the program contains the MPI code.
```

```
The code splits into multiple tasks with one special task (rank 0) that gathers the data and writes the output to file.*/

int main(int argc, char **argv)
{
        if (argc != 13) {printf("Incorrect number of arguments: %i.\n",argc-1);exit(1);};

        //MPI Var
        int numtasks, rank; //can't be unsigned because of the requirements of the MPI functions
        MPI_Init(&argc,&argv);
        MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        Ullong seed = time_seed() + (1<<(rank+30));//pow(2,rank+30);
        printf("I am alive, my name is %i. I am one out of %i threads. My seed is %llu.\n",rank,numtasks,seed);

        const char *var_names[8] = {"S","A","D","Chain-Length","Flips","Burn-in","Skips","Log-samples"};

        //Chain Var

        float s = atof(argv[1]); //should use strtof(argv[1],NULL,0) and if (errno != 0) {}
        float a = atof(argv[2]);
        float d = atof(argv[3]);
        unsigned int chainLength = atoi(argv[4]);

        //Monte Var

        unsigned int flips = atoi(argv[5]);
        unsigned int burnin = atoi(argv[6]);
        unsigned int skips = atoi(argv[7]);
        unsigned int logSamples = atoi(argv[8]); //Log_2 of the number of samples

        const unsigned int vary_index = atoi(argv[9]);

        const float runEnd = atof(argv[10]);
        const float runStep = atof(argv[11]);
        unsigned int repeat = atoi(argv[12]);
        unsigned int repeat_count;

        float runStart;

        switch (vary_index) {
                case 0: runStart = s; break;
                case 1: runStart = a; break;
                case 2: runStart = d; break;
                case 3: runStart = chainLength; break;
                case 4: runStart = flips; break;
                case 5: runStart = burnin; break;
                case 6: runStart = skips; break;
                case 7: runStart = logSamples; break;
                default:  printf("Error: incorrect index.\n"); exit(1);
        }

        const unsigned int numRuns = floor((runEnd-runStart)/runStep + 1 + EPS);
        if (numRuns <= 0) {printf("Incorrect Run Values.\n");exit(2);};
        const unsigned int runs_each =  numRuns/numtasks;
        const unsigned int runs_extra = numRuns%numtasks;
        const unsigned int max_runs = runs_each + 1;
        printf("runs_each %i, runs_extra %i\n",runs_each,runs_extra);
        unsigned int myRuns;

        onerun results[repeat][max_runs], tempResult;

        float currentRun;

        int i;
```

```
496          Ran randomgen(seed);
497          chain *channelChain;
498
499
500
501  /*      //For debugger
502          if (rank==2)
503          {
504          i=0;
505          while (0 == i)
506          sleep(5);}
507
508          //For debugger
509  */
510
511          if (vary_index > 3)
512          channelChain = new chain(s,a,d,chainLength);
513
514          unsigned int my_would_be_step_extra;
515
516          for (repeat_count=0; repeat_count < repeat; repeat_count++)
517          {
518                  my_would_be_step_extra = (runs_extra - (rank + 1) + repeat_count*runs_extra
     +numtasks)%numtasks;
519                  myRuns = runs_each + (my_would_be_step_extra < runs_extra);
520
521  //this assigns the overuns in such a way that the more computationally intensive jobs (assuming
     intensivity increases) get assigned to the lower ranked threads (because they had either equal or
     lower load due to the cycles of reflection assignment below, but because of repeating the whole
     cycle we then shift the assignment away from those assigned last round. old: ((runs_each%2)? rank
     + repeat_count*runs_extra : numtasks - (rank+1) - repeat_count*runs_extra)%numtasks;
522  //+numtasks is to turn the remainder operator % into the mathematical mod "wrap around" function
523
524                  //printf("repeatcount: %i, mytask: %i,mywouldbestepextra: %i, myruns: %i
     \n",repeat_count,       rank,my_would_be_step_extra,myRuns);
525
526
527                  for(i=0; i < myRuns; i++)
528                  {
529                          if (i<runs_each) currentRun = runStart + runStep*(i%2? (i+1)*numtasks -
     (rank+1) : rank + i*numtasks);
530                          //share tasks by flipping order every round
531                          else currentRun = runStart + runStep*(my_would_be_step_extra +
     i*numtasks);
532                          //got an extra run
533
534                          switch (vary_index) {
535                                  case 0: s = currentRun; channelChain = new chain
     (s,a,d,chainLength); break;
536                                  case 1: a = currentRun; channelChain = new chain
     (s,a,d,chainLength); break;
537                                  case 2: d = currentRun; channelChain = new chain
     (s,a,d,chainLength); break;
538                                  case 3: chainLength = (unsigned int)floor(currentRun + EPS);
     channelChain = new chain(s,a,d,chainLength); break;
539                                  case 4: flips = (unsigned int)floor(currentRun + EPS); break;
540                                  case 5: burnin = (unsigned int)floor(currentRun + EPS); break;
541                                  case 6: skips = (unsigned int)floor(currentRun + EPS); break;
542                                  case 7: logSamples = (unsigned int)floor(currentRun + EPS); break;
543                                  default:  printf("Error: incorrect index.\n"); exit(1);
544                          }
545
546                          if (vary_index == 5) channelChain->resetChain(); else channelChain-
     >randomizeChain(randomgen);
547
548                          channelChain->resetCorrelationData();
549
```

```
550                         tempResult = run_MCMC_Entropy
    (*channelChain,flips,burnin,skips,logSamples,randomgen);
551
552                         tempResult.mean = tempResult.mean/chainLength; //Relativize Entropy
553                         tempResult.error = tempResult.error/chainLength;
554
555                         results[repeat_count][i] = tempResult;
556
557                         if (vary_index < 4) delete channelChain;
558                 };
559         };
560                         if (vary_index > 3) delete channelChain; //clean-up
561
562         if(rank==0)
563         {
564
565                 double collectResults[numtasks*repeat*max_runs*onerunSize];
566
567                 MPI_Gather
    (results,repeat*max_runs*onerunSize,MPI_DOUBLE,collectResults,repeat*max_runs*onerunSize,MPI_DOUBLE,0,
568
569                 char filename[400];
570                 char printfString[400]="\0";
571                 unsigned int sim_myRuns, sim_rank, sim_my_would_be_step_extra,
    sim_count_Runs_Done, k;
572
573                 for (k=0;k<8;k++)
574                 {
575                         strcat(printfString,var_names[k]);
576                         if (k == vary_index) strcat(printfString,"-Start");
577                         if (k < 3) strcat(printfString,"_%3.2f_"); else strcat(printfString,"_%
    i_");
578                 }
579
580                 switch (vary_index) {
581                         case 0: s = runStart; break;
582                         case 1: a = runStart; break;
583                         case 2: d = runStart; break;
584                         case 3: chainLength = (unsigned int)floor(runStart + EPS); break;
585                         case 4: flips = (unsigned int)floor(runStart + EPS); break;
586                         case 5: burnin = (unsigned int)floor(runStart + EPS); break;
587                         case 6: skips = (unsigned int)floor(runStart + EPS); break;
588                         case 7: logSamples = (unsigned int)floor(runStart + EPS); break;
589                         default:  printf("Error: incorrect index.\n"); exit(1);
590                 }
591
592                 if (vary_index < 3)
593                 {
594                         strcat(printfString,"End_%3.2f_Step_%3.2f_repeats_%i_numtasks_%i.txt");
595                         sprintf
    (filename,printfString,s,a,d,chainLength,flips,burnin,skips,logSamples,runEnd,runStep,repeat,numtasks)
596                 } else
597                 {
598                         strcat(printfString,"End_%i_Step_%i_repeats_%i_numtasks_%i.txt");
599                         sprintf
    (filename,printfString,s,a,d,chainLength,flips,burnin,skips,logSamples,(int)floor(runEnd + EPS),
    (int)floor(runStep + EPS),repeat,numtasks);
600                 }
601
602                 FILE *out = fopen(filename, "w" );
603                 if(out == NULL) {printf("Creating a new file error.\n"); exit(3);}
604
605 //              strcat(filename,"\n");
606 //              fprintf(out,"%s",filename);
607
608
609                 strcpy(printfString,"initializeLabels = {\0");
```

```
610                 for (k=0;k<8;k++)
611                 {
612                         strcat(printfString,"\"");
613                         strcat(printfString,var_names[k]);
614                         strcat(printfString,"\"");
615                         strcat(printfString,", ");
616                 }
617                 strcat(printfString,"\"vary_index\", \"runEnd\",\"runStep\",\"repeats\",\"numtasks
    \"}\n");
618
619                 fprintf(out,printfString);
620
621                 strcpy(printfString,"initializeValues = {\0");
622
623                 for (k=0;k<8;k++)
624                 {
625                         if (k < 3) strcat(printfString,"%3.2f, "); else strcat(printfString,"%i,
    ");
626                 }
627
628                 if (vary_index < 3)
629                 {
630                         strcat(printfString,"%i, %3.2f, %3.2f, %i, %i}\n");
631                         sprintf(filename, printfString, s, a, d, chainLength, flips, burnin,
    skips, logSamples, vary_index, runEnd, runStep, repeat, numtasks);
632                 } else
633                 {
634                         strcat(printfString,"%i, %i, %i, %i, %i}\n");
635                         sprintf(filename, printfString, s, a, d, chainLength, flips, burnin,
    skips, logSamples, vary_index, (int)floor(runEnd + EPS),(int)floor(runStep + EPS), repeat,
    numtasks);
636                 }
637
638                 fprintf(out,"%s",filename);
639
640
641                 fprintf(out,"dataLabels = {\"TaskNo\",\"%s\", \"Entropy\", \"Std Error\",
    \"Acceptance\",\"Correlation\"}\ndata = {",var_names[vary_index]);
642
643         for (repeat_count=0; repeat_count < repeat; repeat_count++)
644         {
645
646                 fprintf(out,"{\n");
647                 sim_count_Runs_Done=0;
648
649                 for (sim_rank=0; sim_rank < numtasks; sim_rank++)
650                 {
651                         sim_my_would_be_step_extra = (runs_extra - (sim_rank + 1) +
    repeat_count*runs_extra + numtasks)%numtasks;
652                         sim_myRuns = runs_each + (sim_my_would_be_step_extra < runs_extra); //MAY
    BE ZERO!!!! so it messes up naive bracket printing
653
654                         for (i=0; i < sim_myRuns; i++)
655                         {
656                                 if (i<runs_each) currentRun = runStart + runStep*(i%2? (i
    +1)*numtasks - (sim_rank+1) : sim_rank + i*numtasks);
657                                 //share tasks by flipping order every round
658                                 else currentRun = runStart + runStep*(sim_my_would_be_step_extra
    + i*numtasks);
659                                 //got an extra run
660
661                                 fprintf(out, "{%i,", sim_rank);
662                                 if (vary_index < 3) fprintf(out, " %f",currentRun); else fprintf
    (out, " %i",(int)floor(currentRun + EPS));
663                                 for (k=0;k<onerunSize;k++) fprintf(out,", %15.13f", collectResults
    [(sim_rank*repeat*max_runs+ repeat_count*max_runs + i)*onerunSize + k]);
664
```

```
665                                 if (sim_count_Runs_Done < (numRuns - 1)) fprintf(out, "},\n");
666                                 else fprintf(out, "}\n");
667
668                                 sim_count_Runs_Done++;
669
670 //                              if (sim_rank<(numtasks-1)) fprintf(out, "},\n");
671 //                              else {if (i<(sim_myRuns-1)) fprintf(out, "},\n"); else fprintf
    (out, "}\n");};
672                         };
673                 };
674
675                 if (repeat_count<(repeat-1)) fprintf(out, "},\n\n"); else fprintf(out, "}}\n");
676         };
677
678                 fclose(out);
679
680         }
681         else
682         {
683                 MPI_Gather
    (results,onerunSize*max_runs*repeat,MPI_DOUBLE,NULL,0,MPI_DOUBLE,0,MPI_COMM_WORLD);
684         }
685
686         MPI_Finalize();
687         return 0;
688 }
689
690 /*
691
692 Program Name: channel_entropy_full.cpp
693
694 Author: Ismail Akhalwaya
695
696 Title: Full Entropy Calculation of Quantum Channel with Noise Memory
697
698 Description: The capacity of quantum channels with noise memory is a valuable and fascinating
    subject within Quantum Information Theory. The purpose of this program is to numerically
    calculate the output entropy of a toy channel with noisy memory. The channel is constructed from
    two depolarizing channels (providing the noise) selected by a markov chain (providing the
    'memory').
699
700 Programming Detail: The output entropy is calculated by summing over exponentially many markov
    walks.
701
702 Required Command line input:
703 There are 4 required input parameters which specify the properties of the quantum channel.
704
705 1) The first parameter (internally called s) is a real number that specifies the Markov chain
    probability: -1 <= s <= 1. s=-1 equates to a deterministic markov walk which flips at every step,
    s=0 equates to no memory walk, s=1 means a no flip walk.
706
707 2) The second parameter (internally called a) is a real number which specifies the average noise
    of the two depolarizing channels: 0<=a<=1, a=0 means no noise, a=1 means complete noise.
708
709 3) The third parameter (internally called d) is a real number which specifies the difference from
    the average of the two channels: 0 <= d <= Min[a,1-a]
710
711 4) The fourth parameter is the starting length of the markov chain to simulate.
712
713 5) The fifth parameter is the ending length of the markov chain to simulate.
714
715 Output: The average output entropy is written to the screen.
716
717 Typical Call:
718 ./entropy_full 0.5 0.7 0.1 10 30 5
719
720 Copyright 2012
721
```

```
722 */
723
724
725 #include <stdio.h>
726 #include <math.h>
727 #include <stdlib.h>
728 #include "chain.h"
729 #include <omp.h>
730
731
732 #define NaN nan
733 //#define UCHAR_MAX 255
734 #define EPS 2.2250738585072014e-308
735
736 //typedef unsigned long long int Ullong;
737
738 /*This function calculates the output entropy in full, that is without using the Monte Carlo
    Algorithm. It is exponential in running time as a function of the chain length.
739 */
740
741 double SPhirhoFull(double s, double a, double d, unsigned int N)
742 {
743         double YNP;
744         double sumS=0;
745         Ullong i;
746         int j;
747         chain channelChain(s,a,d,N);
748
749         YNP = channelChain.calcProb(0);
750         sumS = YNP*log(YNP);
751
752         for(i=1;i < pow(2,N);i++) //"i=0" has been done above. i=2^N -1 is the last one.
753         {
754                 for(j=N-1;j>=0;j--) //we're adding one in binary
755                 {
756                         if(channelChain.checkChainOne(j)) {channelChain.setChainZero(j);}//carry
757                         else {channelChain.setChainOne(j);break;};//set channel to 1
758                 }
759 //              printf("after loop, j=%d\n",j); should never be -1 because break is always used
    to exit loop
760                 YNP = channelChain.calcProb(j);
761                 sumS += YNP*log(YNP);
762         };
763
764         sumS = -sumS/log(2);
765
766         return sumS;
767 }
768
769 int main(int argc, char* argv[])
770 {
771         if (argc != 7) {printf("Incorrect number of arguments: %i. Expecting six arguments.
    \n",argc);exit(10);};
772
773         const float getS = atof(argv[1]);
774         const float getA = atof(argv[2]);
775         const float getD = atof(argv[3]);
776         const unsigned int getN_start = atoi(argv[4]);
777         const unsigned int getN_end = atoi(argv[5])+1;
778         const unsigned int getN_step = atoi(argv[6]);
779         double entropy;
780         int threads,thisthread;
781         int procs = omp_get_num_procs();
782
783         printf("Num processors:%d\n",procs);
784
785
```

```
786 #pragma omp parallel for schedule(dynamic)
787         for (int count=getN_start; count<getN_end;count=count+getN_step)
788         {
789                 entropy = SPhirhoFull(getS, getA, getD, count);
790                 threads = omp_get_num_threads();
791                 thisthread= omp_get_thread_num();
792                 printf("Thread#: %d/%d, # messages: %d, Entropy: %.10e
    \n",thisthread,threads,count,entropy/count);
793         }
794         return 0;
795
796 }
```