universität wien

DISSERTATION

Titel der Dissertation

"Intelligent Detectors:
Data Processing of n-Dimensional Detector Arrays"

Verfasser

Mag.rer.nat. Roland K. J. Ottensamer

angestrebter akademischer Grad

Doktor der Naturwissenschaften (Dr.rer.nat.)

Wien, November 2009

Studienkennzahl lt. Studienblatt: A 091 413

Dissertationsgebiet lt. Studienblatt: Astronomie

Betreuer: A.Univ.-Prof. Dr. Franz Kerschbaum

Für Herbert

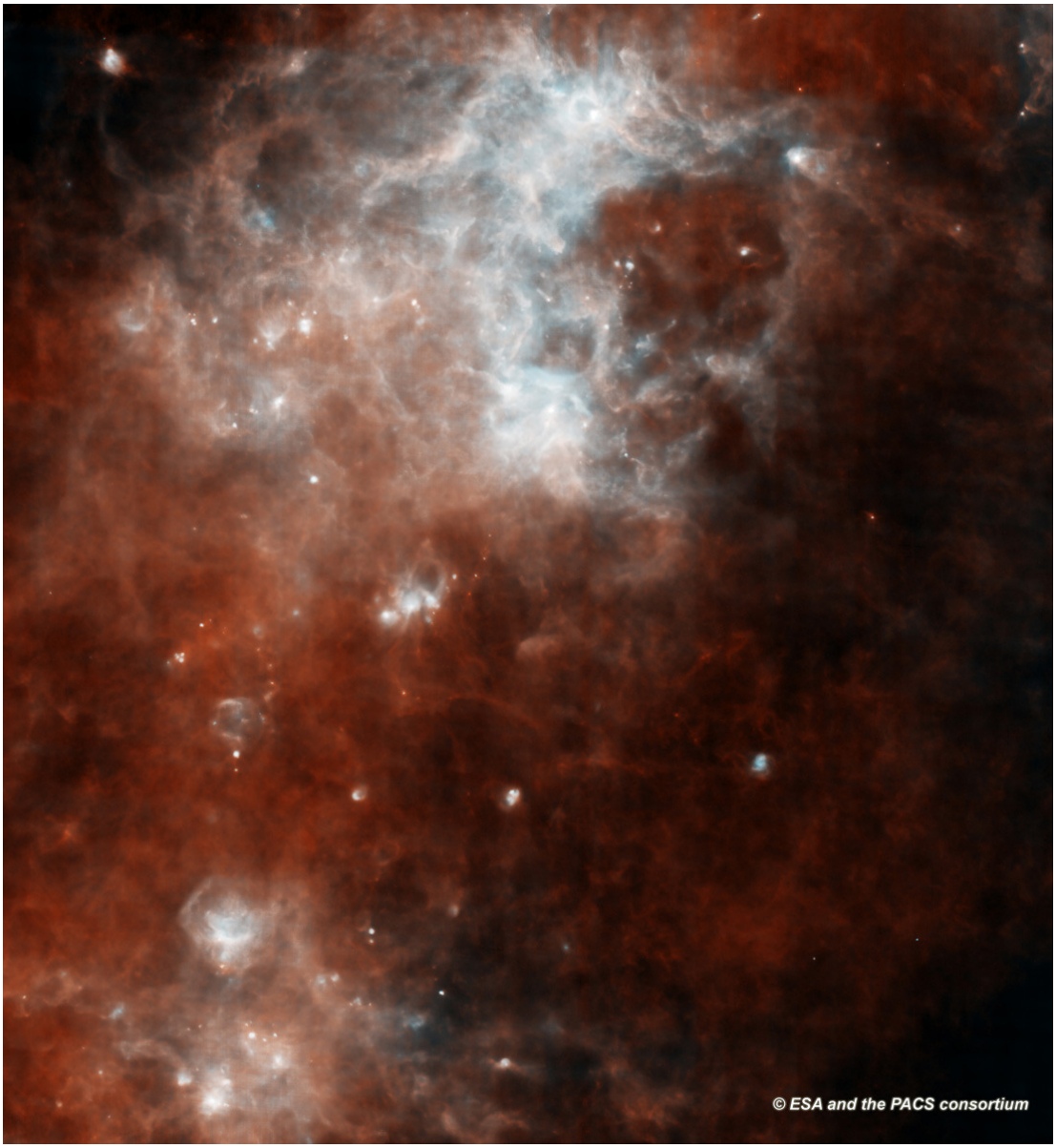


Figure 1: **First parallel mode scan with PACS.** A two by two degree scan of the Milky Way in the constellation Southern Cross shows the cold ISM in great detail and makes us enthusiastic about upcoming discoveries with this fantastic instrument. Without our on-board processing, such images would not be possible.

Abstract

Imagine the following problem: you have an instrument on a satellite that makes 50 measurements for good reason, but only 5 values can be transmitted to ground. *What's the best way to deal with this?* – Ever more ambitious astronomical missions with ever growing detector arrays are faced with exactly this problem. They produce enormous amounts of data at every instant, yet observational and power budget constraints severely limit communication and result in a discrepancy between the available data rate and the raw science. For this very reason a number of satellites have been equipped with on-board computers that perform lossy and lossless processing steps to master this issue. Obviously, there are good and bad choices of how these 5 values could be made up, but of course you want to keep as much scientific information as possible. Ideally, your instrument would *reduce*[1] the data for you, so you put a little astronomer inside and make it an *intelligent detector*.

This thesis contains the knowledge and a good deal of experience that are necessary for the development of such astronomical on-board software for satellites. The key elements in the development are the understanding of the *scientific purpose*, knowledge of the *physical properties of the detector*, the comprehension of the *mathematical operations* involved in data processing and the consideration of the *technical and observational circumstances*.

What is presented herein is mainly drawn from the turbulent final development stage of the on-board reduction and compression software for the PACS instrument of the *Herschel Space Observatory*, where almost every suitable concept from information theory has been confronted with its test data to make it an intelligent detector. Since almost a decade I am part of the team that developed this mission-critic software and during the last four years I took over the responsibility to develop the flight model version to bring the necessary changes that would guarantee a working instrument. After many years of hard work, Herschel has finally been launched on the 14^{th} of May, 2009. During the first months of operation the instrument performance of PACS has been verified and all there is to say is that the scene is set for the next three or more years to bring exciting discoveries in the *cold universe*.

[1] Beside the literal meaning of *decreasing the amount*, *"data reduction"* is the usual term for data analysis in astronomy.

Contents

Prologue

Astronomers used to observe the sky with the naked eye for thousands of years, with no better tools than just rods and the like, later astrolabes and quadrants. The invention of the telescope 400 years ago revolutionised the field and changed our view of the world. A few hundred years later, faint distant objects became observable with the advance of photography. In the 20^{th} century, the silicon revolution brought again large innovations to science. On the one hand computers reduced the workload of calculations and large scale simulations could be carried out, on the other hand the telescopes themselves were updated with much better electronic detectors. When we speak of the present, we also use the term *Space Age*, because since 50 years we are able to send forth our telescopes into space, enabling us to measure objects and wavelengths that are out of reach from the ground.

> Computer science is no more about computers than astronomy is about telescopes.
>
> —*Edsger Wybe Dijkstra*

Motivation

The rapid development of detector technology with more pixels and higher efficiency leads to scientifically more valuable data, but also to much higher data rates. Especially satellites at far distances, such as the Lagrangian points or solar system probes and missions with a limited lifetime cannot afford lengthy downlink periods and must therefore reduce the data to the available amount in real-time. The first missions that were equipped with a dedicated data compression unit were the Voyager probes. Since then many satellites had to solve this problem in their own specific way, but mostly they could go ahead with a lossless compression module. At present there are missions that don't need to carry out on-board data compression, but others need to intensively process their raw data.

The *Hubble Space Telescope* can be seen as an example for the rapid progression of detector technologies. During its 19 years of operation, Hubble has had five service missions that brought new instruments. As a consequence of the higher data rates, the HST data management system had to be upgraded as well. Figure 2 shows how the pixel numbers have grown throughout the years. Due to the low earth orbit Hubble can transmit its data mostly without compression. A mission where on-board compression is a key component is Herschel. Compared to previous infrared space telescopes, the

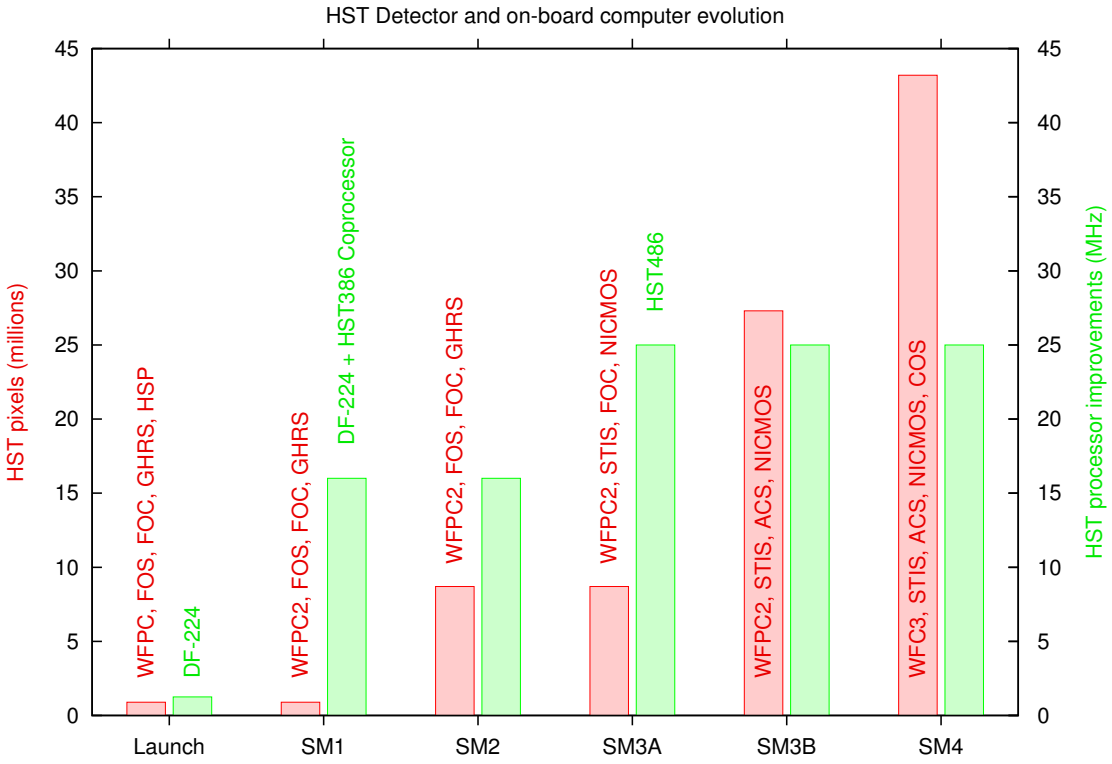


Figure 2: **HST detector evolution**. This figure shows how the HST has evolved during the five service missions with respect to pixel numbers and on-board processing power. The HST was launched with a payload of 5 instruments, but the High Speed Photometer had to be exchanged with the corrective optics COSTAR during the first service mission. The latest SM 4 was finally made in May 2009 replacing the WFPC2 with WFC3 and the COSTAR optics with the Cosmic Origins Spectrograph.

most obvious difference is its sheer size, but also the number of pixels increased from just a few to several thousand. As Herschel operates at L2,[1] even lossy reduction steps have to be carried out on board to be able to transmit the data within the available telemetry budget. It is an unpleasing but undeniable fact that current and future missions will no longer provide raw data to the community, but at least partly reduced data products. On-board compression software has become a complex and sophisticated mission-critical component.

About this Thesis

The next 150 or so pages deal with considerations on feasibility and implementation of on-board reduction and compression concepts. These are given with respect to the Herschel/PACS instrument.[2] The first three chapters deal with fundamentals – properties of science data, decorrelation and data compression. These are the three steps in science data compression and I describe how I applied them to PACS in the middle part. This recipe and experiences from Herschel/PACS can be well applied to subsequent projects with a need for on-board data processing, as this is done in the last chapter about SPICA/SAFARI.

[1] The second Lagrangian point in the Sun-Earth system, in a distance of 1.5 million km from Earth.
[2] PACS is short for Photodetector Array Camera and Spectrometer.

Intended Readership

This thesis contains much of what I have been dealing with during the development of the PACS on-board software. Besides the fact that it is needed to get the doctoral degree, I have written it as a reference for myself and other people who want to understand how science data compression works and what PACS is doing. I wanted to create an essential introduction and valuable reference for both scientists and engineers that are involved in space projects. I decided to put considerable effort into the chapters about the fundamentals by not using existing software packages, but by programming everything from scratch so that I could stress out the important details for implementation.

Source Codes

Moreover, I decided to include some short sources in plain C from my rich pool of little programs whenever they are of any help. You are welcome to experiment with them, but I don't guarantee their proper functioning, even if many of them are used *right now* at the time of writing, 1.5 million km above our heads.

Quotation Boxes

As in many modern books, I added quotes to fill up the page layout. These quotes were selected to be entertaining in the first place, but also to have a vague connection to the surrounding text to encourage careful reading. Many of them are taken from real life, but the majority stems from my nighttime study of the *Encyclopaedia Britannica* [Hoi07]. These can be identified by two acronyms – EBm stands for *Micropaedia* and EBM for the *Macropaedia*.

A Word on Notation

The use of computer-oriented units follows the SI standard now. kiB (kibibyte) stands for 1024 bytes, whereas kB (kilobyte) would stand for 1000 bytes. Other prefixes like MiB (Mebibyte) and GiB (Gibibyte) are used accordingly. However, most of the time kiB and MiB will be used, as they usually give more attractive numbers.

1

Science Data

Science is how we call our pursuit for knowledge of the physical world around us and by data we mean any type of information mostly in digital form. The data we use to derive scientific results are either measures taken with any kind of instrument or results from numeric simulations. Imprecise or incomplete models as well as numeric insufficiencies lead to errors in the latter, but progress in computer hardware and numerical methods allows for results that are in closer agreement with theoretical expectation. Although all computational aspects treated in this text can be applied to synthetic data, I will further concentrate on *real data* only, i.e. science data that are not synthetic, but were taken by measurement. In this case noise plays a central rôle, as it defines how precise the taken measure is, or, how much scientific information can be derived from it.

> Every dataset contains noise. Some datasets contain signals.
> —*Paul Marchant, Univ. Leeds*

This chapter contains concepts that are needed as a basis for all kinds of science data processing. I use the language the astronomer is familiar with to memorise statistical fundaments, make familiar with laws of information theory and explain mathematical relations in preparation for the chapters that follow. What is most important here is the understanding of what data are made of and what the consequences for transmission and storage are. Although my motivation lies in data compression, the goal is ultimately the same as in data analysis: in order to extract the maximum scientific information from a dataset, the noise must first be characterised and a way of distilling the signal must be found. Once the concept of entropy is established as a measure for noise it will become clear that the biggest fraction of a dataset is used up by the noise.

The last section of this chapter deals with basic lossy reduction steps of science data to attack the initial problem of reducing the amount of data to a few values. Among these classical reduction steps is averaging of course. I will show how easily the trivial task of averaging can develop into a complex problem under certain circumstances – such as on board an astronomical satellite. One of the algorithms that I have developed for PACS averages and quantises in a way that ensures a minimum of degradation. Its performance on test data is estimated and the source code is given.

The chapter is divided into the following sections:

1 Signal and Noise

A typical observational measure in astronomy is a compound of a source signal with a number of different sources of noise. From the measure we hope to get some information about the source, but it is equally important to know how precise the result is. Stars are perfect specimen for demonstrating the intrinsic uncertainty of a signal. First of all, a star is not a constant source, but emits radiation at a variable rate, depending on its astrophysical properties. So obviously, the measure we take is determined by *when* the observation is made. In addition to that the nature of physical phenomena involves *quantum noise*. Especially for weak sources that are sending us their photons as single events with irregular time intervals inbetween it is important *how long* the observation is made. This actually represents the fundamental limit of the achievable signal-to-noise ratio and determines the exposure time. In case of photon detection with optical systems this is referred to as *photon noise*. Photon noise follows a Poisson probability distribution and is therefore proportional to the square root of the signal. Various other sources in the fore- or in the background as well as environmental effects pollute our measurement, but the biggest noise contributor is the detection system itself, which

is above all limited by the *readout noise* of the readout electronics. This component follows a normal distribution.

In short, we have an intrinsically uncertain signal embedded in an amalgam of other noise sources. This is where the central limit theorem comes into play and adds all noise sources up to follow a normal distribution. In the end we may have taken a digital measure, but a single measure doesn't tell us anything, it has *no information* as long as the *signal to noise* ratio is not known. This is also well reflected in the *variance*. You *can* calculate the variance for a single measure, but it's either 0 or ∞ depending on whether you apply *Bessel's correction* or not. A second measure already improves this – if it is identical, then you know that either there is no noise and you don't need to measure any more or – more probably – that there's something wrong with your measurement. Ideally, the second value taken will be relatively close to the first one and now speculation can start what the *true* value might be. In other words, we can only use the information of the signal if we know how uncertain it is.

> Noise, n. A stench in the ear. Undomesticated music. The chief product and authenticating sign of civilization.
>
> —*Ambrose Bierce, The Devil's Dictionary, 1911*

Two characteristics determine the type of noise: its amplitude distribution (the *histogram*) and its spectral density (the *colour*). These are largely independent of each other. This chapter first treats the probability distributions relevant for astronomical data which describe how uncertain a value for a sample is, before the characteristics of the power spectrum are being discussed. But now let me recapitulate the basic principles from probability theory and statistics.

Statistical Basics – Compressed

Our datasets are affected by noise and thus at least one of the underlying processes is indeterministic, or better called a stochastic or *random process*. Even if the past is known, there are many possibilities such a process might go to, but some paths are more probable than others. Its states at any instant, its future evolution are described by an associated *probability distribution*,[1] which tells us how probable each possible outcome is. The output of a stochastic process is a *random variable* X and a sequence of observations x_i of a random variable is also known as a *time series*. In notation a distribution is usually given by a letter (or its name) describing the type with the parameters in parentheses like $\mathrm{N}(\mu, \sigma^2)$ for normal distribution.

Mean μ

The probability distribution of a random variable is often characterised by a small number of parameters, which also have a practical interpretation. The most important one is of course the *expected value* $E(X)$, or equivalently, the (arithmetic) *mean* μ of the random variable with distribution $P(x_i)$. It is defined by $E(X) = \sum_i x_i P(x_i)$ for discrete random variables and $E(X) = \int_{-\infty}^{\infty} xP(x)dx$ for continuous random variables. Keep in mind that there is a difference between the *true* parameters of the distribution,

[1] In the discrete case the probability distribution is given by the probability *mass* function, whereas a probability *density* function is defined for continuous random variables, which gives the probability for the sample interval.

which are normally not known, and parameters that are estimated from the available samples, like $E(X) = n^{-1} \sum_{i=1}^{n} x_i$. Fortunately, estimation gets better the more samples are taken into account, as will be explained beneath.

Variance σ^2

Once the mean is estimated, the next question is about statistical dispersion, i.e. how far away from this average the individual values of X typically are, a question that is best answered with the *standard deviation* $\sigma = \pm\sqrt{E[(X - E(X))^2]}$, the square root of the mean squared error (MSE), which in turn is known as the variance σ^2. A small modification, paying again attention to the difference between true and estimated distribution, known as *Bessel's correction* is usually applied to this. Instead of n the divisor $n - 1$ is used if the true average of the probability distribution $P(x_i)$ is not known but estimated – derived by averaging the available X.

alternatively, $\sigma_X^2 = E(X^2) - (E(X))^2$

Median

Other parameters – actually moments of $P(x)$ – are *skewness* and *kurtosis*, responsible for asymmetry and peakedness, respectively, but I rather mention the *median* value here, which plays an important rôle in astronomical data analysis due to the fact that it's almost as good as the mean[1] yet much less affected by outliers. The median is found by sorting the data values and picking the middle value if there is an odd number of data values, or the average of the two middle values if their number is even. In practice, the median has a disadvantage – it is relatively expensive to compute.

Square Root Law

One question when dealing with noise is what can be done to minimise its effect. This is where the *law of large numbers* comes into play. It states that the sample average converges to the expected value for $n \to \infty$. Yet we would also like to know how fast this converges, how many samples we have to take to reach a certain precision. For this purpose the *standard error* can be taken, which is the standard deviation of the error in the sample mean relative to the *true* mean. By simply applying error propagation to the mean $\mu = \sum_{i=1..n} x_i/n$ we get $\sigma_\mu = \pm\sqrt{(\sigma_1/n)^2 + ... + (\sigma_n/n)^2}$. Assuming that the uncertainty is the same for all samples $\sigma_{1..n} = \sigma$ we find that the error of the sample mean relative to the *true* expected value is $\sigma_\mu = \pm\sqrt{n(\sigma/n)^2)} = \sigma/\sqrt{n}$. To wrap this up in a memorable phrase, the precision of a value generally improves with $\sqrt{n}$.[2]

Random Numbers

It's time to become a little bit more familiar with random variables, because if we want to understand noise we need to know how to generate it as well. To do so we need random numbers, which are numbers drawn from a stochastic process, whose outcomes follow a probability distribution and no describable deterministic pattern. Essentially, a single random number is a sample of a random variable. Ideally, a stochastic process has no memory, i.e. each number is drawn regardless of what has already been drawn, but in practice it is not possible to exclude statistical correlations of a certain degree. This is also true for random number generation. In a computer, unless a specialised hardware device is used, several parameters such as mouse and keyboard states, interrupts and disk states are gathered in the so-called entropy pool to generate a seed for pseudo-random number generation.

[1] Recall that the median has 64% efficiency of the mean for a normal distribution.

[2] Note that this relation is general and not to be confused with the estimation of the SNR $= n/\sqrt{n} = \sqrt{n}$ which applies only for typical photon-noise dominated imaging.

A number of random number generators are available for all kinds of purposes. As an example, here is a very simple `rand()`-like implementation with the coefficients taken from the ADSP-21000 Family Development Tools 3.3:[1]

Listing 1.1: Random Number Generation

```
#define URAND_A_val 1664525
#define URAND_C_val 32767
#define URAND_CLIP 0x7fffffff

static unsigned int URAND_seed = 1;

unsigned int GetRandomNumber ()
{
  unsigned int RandomNumber = URAND_seed * URAND_A_val;

  RandomNumber += URAND_C_val; // multiplication and addition will
  URAND_seed = RandomNumber;   // overflow the 32 bits most of the
  RandomNumber &= URAND_CLIP;  // time, but we clip the result anyway.

  return RandomNumber;
}

Or, as a compact C macro:

#define GETURAND() (URAND_seed = URAND_seed * URAND_A_val +
    URAND_C_val, (URAND_seed & URAND_CLIP))
```

The implementation above returns a pseudo-random unsigned integer in the range $[0, 2^{32} - 1]$ with a period in the order of $2^{32} - 1$. For many reasons it is desirable to map this to the interval $[0, 1]$ through `(float)RandomNumber/URAND_CLIP+1.0`. The type of this algorithm is a linear congruential generator, i.e. of the form

$$X_n = (aX_{n-1} + c) \mod m \ .$$

The integer overflow from the multiplication is no problem, as the result is clipped by `0x7fffffff` anyway, which is the same as a modulo `0x80000000` operation.[2] So, in our case we have:

$$X_n = (X_{n-1} \times 1664525 + 32767) \mod 2147483648 \ .$$

Linear congruential generators are deterministic, as any single number determines the full sequence of numbers to come. Better pseudo-random number generators such as the *Mersenne Twister* [Mat98] abound, where a random variable depends on a larger sequence of previous ones, but the deterministic nature of numbers calculated in a computer cannot be eliminated without additional hardware that would sample for instance thermal resistor noise. Note that the `rand()` function from the C standard library implements such a linear congruential generator and with it most applications do so too. Although there were plans to include a true random number generator in the Pentium-III CPU, this feature is still not implemented in generic computer hardware. For a fine introduction on pseudo-random numbers including a table of other possible coefficients go for [Sch96]. In any case, we keep our fingers crossed that the random numbers from our generator will be uniformly distributed.

[1] The C version is pretty straightforward. Note that the macro version uses the binary (,) operator.

[2] $A \mod B$ can be calculated by the binary operation $A\&(B-1)$ if B is a power of 2.

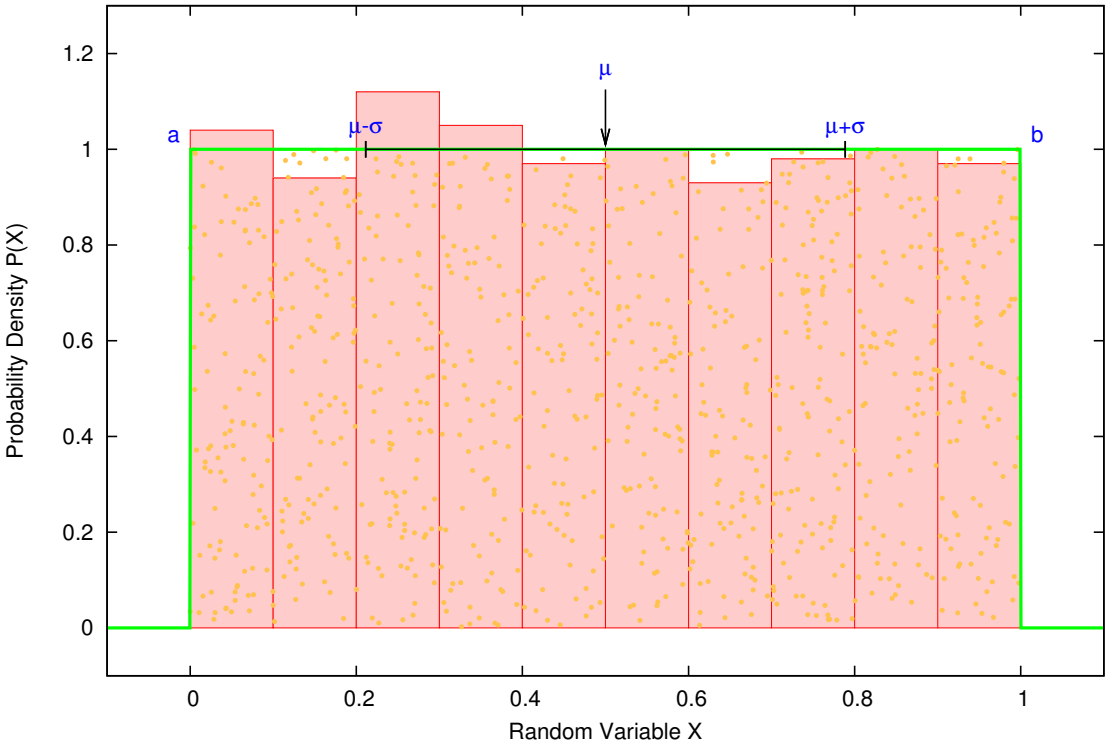


Figure 1.1: **Plot of the [0, 1]-uniform distribution U(1/2, 1/12)** (green line) with mean μ and standard deviation σ. The dots inside illustrate how a 2D distribution would look. The red bars are the histogram of the discrete 2D random samples.

Uniform Distribution

If a random variable is uniformly distributed, all numbers in a certain interval are equally probable. The uniform distribution $\mathrm{U}(\mu, \sigma^2)$ exists in a discrete and a continuous form and its importance is justified by two reasons. First of all, as shown above, it is easy to generate such random numbers. Secondly, a uniformly distributed random variable is the starting point for many conversion methods to other probability distributions, such as the *Box-Muller* transform [Box58] for the generation of random numbers following a standard normal distribution. Another method is to make use of the central limit theorem by summing 12 uniformly distributed random numbers and subtracting the value 6 to obtain a $\mathrm{N}(0, 1)$ distribution.[1] In general, inverse transform sampling can be used to get to all kinds of distributions [Dev86].

The (continuous) uniform distribution has the following characteristics:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b] \\ 0 & \text{for } x \notin [a, b] \end{cases}$$

$$\text{mean } \mu = \tfrac{a+b}{2}$$
$$\text{variance } \sigma^2 = \tfrac{(b-a)^2}{12}$$
$$\text{entropy } H = \mathrm{ld}(b-a)$$

[1] Of course, the numbers have to be independent and identically distributed with finite variance, which is not guaranteed by linear congruential generators, but it is a common mishabit to use them nevertheless.

Normal Distribution

The normal or Gaussian distribution is the most important one for scientific work. It was first introduced by Abraham de Moivre and used by Laplace and later also by Gauss to analyse the errors of measurements. Gauss was the one to give the equation for the bell curve. Figure 1.2 shows its typical shape. This distribution has the largest entropy for a given variance [Cov06], a very useful property in light of data compression as it tells us that Gaussian noise is the worst one to compress. For any given dataset with variance σ^2 one can immediately calculate the entropy of the normal distribution $H = \mathrm{ld}(\sigma\sqrt{2\pi\mathrm{e}})$ as an upper limit of the actual entropy of the dataset. The consequences of that will be outlined in Section 2.

> THE
> NORMAL
> LAW OF ERROR
> STANDS OUT IN THE
> EXPERIENCE OF MANKIND
> AS ONE OF THE BROADEST
> GENERALIZATIONS OF NATURAL
> PHILOSOPHY • IT SERVES AS THE
> GUIDING INSTRUMENT IN RESEARCHES
> IN THE PHYSICAL AND SOCIAL SCIENCES AND
> IN MEDICINE, AGRICULTURE, AND ENGINEERING •
> IT IS AN INDISPENSABLE TOOL FOR THE ANALYSIS AND THE
> INTERPRETATION OF THE BASIC DATA OBTAINED BY OBSERVATION AND EXPERIMENT
>
> —*Laudatio by William Youden*

The Central Limit Theorem

The reason why the normal distribution is so important is the central limit theorem, which states that the sum of independent random variables with finite mean and variance will be approximately normal distributed with $\mu = \mu_X + \mu_Y$. Yes, the random variables can even be drawn from different distributions! As natural processes usually involve a combination of several sources of noise, the chances are high that the outcome is well described by a normal distribution.

The normal distribution has the following characteristics:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

mean $\mu = \mu$
variance $\sigma^2 = \sigma^2$
entropy $H = \mathrm{ld}(\sigma\sqrt{2\pi\mathrm{e}}) = 1/2\,\mathrm{ld}(2\pi e\sigma^2)$

σ-Clipping

By looking at the equation above we see that the normal distribution's probability density function $f(x) > 0$ for all $x \in (-\infty, \infty)$. So theoretically a Gaussian random variable could assume just *any* value, although in practice such extreme outliers are not encountered. As most of the values concentrate around the mean, an objective way of getting rid of spoiled data is to discard samples with larger residuals than e.g. 3σ. This threshold is also indicated in Figure 1.2. Of course, this treatment can be applied to other distributions as well, though special attention must be paid to distributions with considerable asymmetry.

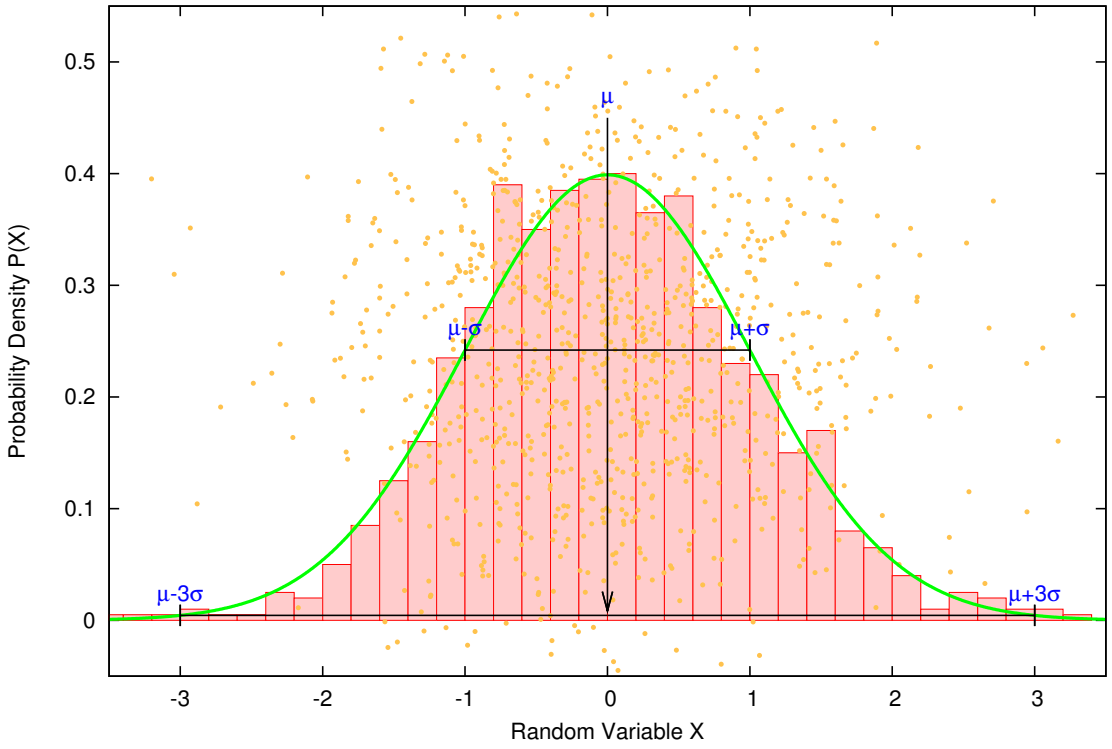


Figure 1.2: **Plot of the standard normal distribution N(0, 1)** (green line) with mean μ and standard deviation σ. The spots inside again illustrate how a 2D distribution of that kind would look.

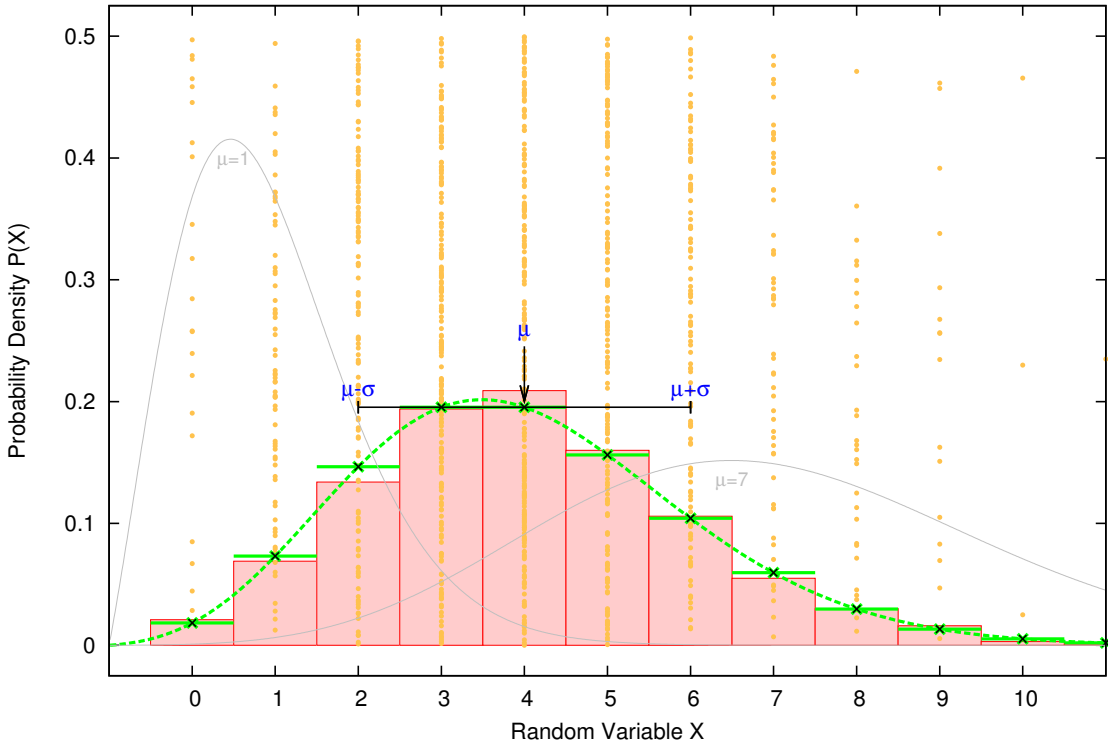


Figure 1.3: **Poisson distribution P(μ=4)**. The dashed green line indicates the continuous distribution and the green steps mark discrete intervals. The spots for the 2D plot are intentionally discrete in X this time. Other plots for μ=1 and μ=7 are also given.

Poisson Distribution

In search of an approximation to Bernoulli's binomial distribution, Siméon-Denis Poisson found what would later be known as the Poisson distribution. Also called the *law of rare events*, it describes the probability that a number of events occur in a fixed interval of time, space or any other measure with respect to a known average rate and independent of the last event. In astronomy, this distribution is of particular interest, as it expresses the photon noise,[1] the noise in images under low signal conditions. This distribution has only one parameter μ, because the variance $\sigma = \mu$. This makes signal-to-noise estimation particularly easy, because $\mathrm{SNR} = n/\sqrt{n} = \sqrt{n}$ as already mentioned before. However, in practice it's not that easy as the noise is usually made up of other components as well.

> Poisson sans boisson est poison.
>
> —*French proverb*

When simulating astronomical data for test purposes it is necessary to consider the Poisson noise component. Here is a little dirty AWK-function that can be used to generate Poisson noise:

Listing 1.2: Poisson Random Number Generation

```
function poisson(lambda) {
  L = exp(-lambda);       # set the limit for the event
  k = 0; p = 1;
  do {
       k = k + 1;         # count up k ...
       p = p * rand();    # randomly decrease the interval
     } while (p >= L)     # ... until the event takes place

  return k-1;
}
```

Poisson noise is prevalent in detection systems that are counting rare events, such as raindrops on a flower or photons on a pixel. For large μ this distribution can be approximated by a normal distribution, but the smaller μ is, the more asymmetric it becomes. That is, we cannot simply treat Poisson data as if they were Gaussian, which is what we would ideally want to do. Fortunately Francis John Anscombe found in 1948 a transform [Ans48] of Poisson noise to Gaussian noise, which reads $t(I) = 2\sqrt{I + 3/8}$ and creates a normal distribution with $\sigma = 1$. The way back is simply $I = 4t(I)^2 - 3/8$. A decade ago, this transform has been extended to a mix of Gaussian and Poisson noise [Sta98a], which is sufficient to be used on CCD data.

The Poisson distribution has the following characteristics:

$$f(x) = \frac{e^{-\lambda}\lambda^k}{k!}$$

mean $\mu = \lambda$
variance $\sigma^2 = \lambda$
entropy $H = \lambda[1 - \ln(\lambda)] + {} + e^{-\lambda}\sum_{k=0}^{\infty}\frac{\lambda^k \log(k!)}{k!}$

[1] More general designations are *shot noise* or even *quantum noise*.

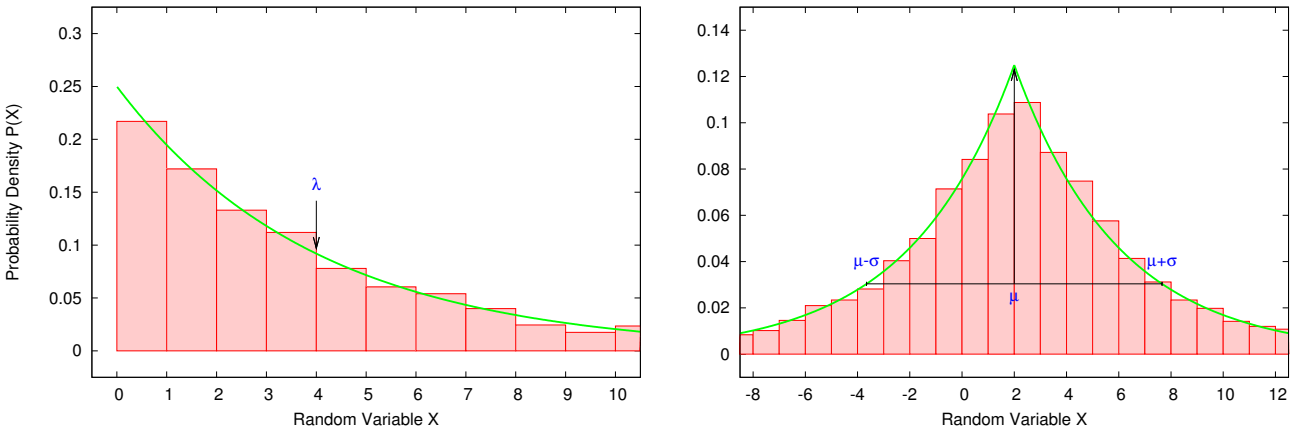


Figure 1.4: **Exponential distribution and Laplace distribution.** (Left) The exponential distribution "Exponential(4)" has only one parameter just like the Poisson distribution, whereas the Laplace distribution (Right) has the usual two parameters "Laplace(2, 16)".

Exponential and Laplace Distribution

If a process P_λ is Poisson distributed, then the times between the events in P_λ are exponentially distributed with parameter $1/\lambda$. To help you visualise this, look at the vertical spaces between the dots in Figure 1.3. The exponential distribution describes such intervals, like arrival times or nucleon decay times and is memoryless. Generation of an exponentially distributed random variable X from a uniform distribution U on $[0,\,1]$ is simply done with: $X = -\ln(1-\mathrm{U})/\lambda$.

The exponential distribution has the following characteristics:

$$f(x) = \frac{1}{\lambda} e^{-\frac{x}{\lambda}} \qquad x, \lambda > 0$$

mean $\mu = 1/\lambda$
variance $\sigma^2 = 1/\lambda^2$
entropy $H = 1 + \mathrm{ld}\lambda$

Laplace Dist.

This distribution, which is also known as the double exponential distribution, describes best the difference between two i.i.d. (independent and identically distributed) exponential random variables. For instance, a Brownian motion evaluated at an exponentially distributed random time is distributed that way. For us the Laplace distribution is important because many decorrelated/preprocessed datasets are well described by it [Yeh93]. In general, differences of consecutive correlated values tend to resemble this distribution [Sal07]. Remember this when viewing histograms in Chapter 2. Generation of Laplace-distributed data from a uniform distribution on $[0, 1]$ works by $X = \mu - \lambda\,\mathrm{sgn}(\mathrm{U} - 1/2)\ln(1 - 2\,|\mathrm{U} - 1/2|)$.

The Laplace distribution has the following characteristics:

$$f(x) = \frac{1}{2\lambda} e^{-\frac{|x-\Theta|}{\lambda}}$$

$$-\infty < x, \Theta < \infty, \lambda > 0$$

mean $\mu = \Theta$
variance $\sigma^2 = 2\lambda^2$
entropy $H = 1 + \mathrm{ld}2\lambda$

2 Entropy and Information

In contrast to its definition in thermodynamics, the entropy in information theory is a measure for *expected* uncertainty [Sha48] and not a measure for disorder. Basically, entropy is the logarithm of the number of states that can be assumed by a system, regardless of its actual order. Given the probability density function $f(x)$ of a distribution, it can be derived from

$$H(x) = E\left(\mathrm{ld}\frac{1}{f(x)}\right) = -\int_{-\infty}^{\infty} x \,\mathrm{ld}\, f(x)\, dx \quad .^{1}$$

Regardless of the distribution it can be estimated directly from the data of X using

$$H(X) = -\sum_{x \in X} p(x) \,\mathrm{ld}\, p(x) \quad ,$$

where the probability $p(x)$ is best taken from the relative frequency $c(x)/n$. In that sense entropy is the average length of the shortest description of X, the amount of information required on the average to describe the random variable.

Related Concepts

In principle that's all we need to understand science data compression, but here are some more related definitions that might help to grasp the overall picture. Just to remind you of the notations that are used in the following context involving the random variables X and Y, (X, Y) is read "X and Y", $(X;Y)$ is not really read and $(Y|X)$ is read "Y conditional on X" or "Y given X".

As two random variables can be expressed by a common distribution – a joint distribution $p(x, y)$, they also have a *Joint Entropy* $H(X, Y) = H(X) + H(Y|X)$. If there is a connection between these two random variables, if they are dependent, then conditioning reduces entropy and the *Conditional Entropy* reads $H(Y|X) \leq H(Y)$ with equality if X and Y are independent.

The best mathematical answer to the question, *"What is information?"* is given through the *Mutual Information*, which uses the discrepancy between the joint distribution and the product of the separate distributions.

$$I(X;Y) = H(X) - H(X|Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \,\mathrm{ld}\frac{p(x, y)}{p(x)p(y)} \quad .$$

This is the reduction of uncertainty in X due to the knowledge of Y. In other words it is a measure of the amount of information one random value contains about the other. If $I(X;Y) = 0$ then X and Y are independent. An excellent in-depth treatment of these concepts is [Cov06].

Entropy of Science Data

Aside from the noise in our dataset there is hopefully also some signal. Now that we know what entropy is we also understand that it is not the limit of ultimate compression, because it does not take into account the conditional entropy of the science content. Thus, as long as the dataset is not just pure noise, we can do better than what the entropy dictates if we want to compress the data.

[1] The *Logarithmus Dualis* is $\mathrm{ld}\, x = \frac{\ln x}{\ln 2}$.

A simple example would be to take 10 binary digits, 5 times 1 and 5 times 0. Then the sequences 1010011010 and 1111100000 both have the same entropy of 10 bits, but the latter is more easily compressible below that limit, because its bits are obviously not independent. To make use of the correlation in datasets for the sake of compression we need a better understanding of *independence*, which will be given now.

Covariance and Correlation

The *covariance* of two random variables is a measure of their interdependence, i.e. how synchronous they vary and *correlation coefficient* ρ is the covariance normalised to the standard deviations. Their equations are given by

$$\begin{aligned} \mathrm{Cov}(X,Y) &= E[(X-\mu_x)(Y-\mu_y)] = \\ &= E[XY] - \mu_x\mu_y \\ \rho(X,Y) &= \frac{\mathrm{Cov}(X,Y)}{\sigma_x\sigma_y} \ . \end{aligned}$$

A two-dimensional *covariance matrix* can be calculated for sequences of random variables. If X is an $n \times m$ matrix with n observations (rows) of m random variables (columns) and Y is an $n \times s$ matrix, then the $m \times s$ covariance matrix can be calculated. In the equation the average of the i^{th} random variable of X is noted as μ_i and the averages in Y are found in ν_i. I give it in a form which allows a direct translation to a programming language.

$$\mathrm{Cov}(X,Y) = \sum_{ij} = \sum_{j=1}^{s}\sum_{i=1}^{m}\sum_{k=1}^{n}(X_{ik}-\mu_i)(Y_{jk}-\nu_j)$$

For the special case $\mathrm{Cov}(X,X)$ the variances of the random variables are found in the diagonal. In Chapter 2 we will make use of that to evaluate the efficiency of different decorrelations. Note that if applied to 2D-data only the horizontal correlation is revealed. To depict the vertical correlation, the covariance matrix of the transpose needs to be calculated.

Cross Correlation

The *cross correlation* is a slightly different, but equally useful tool. For two functions $f(t)$ and $g(t)$ it is calculated by the convolution

$$\mathrm{CCF}(\tau) = (f * g)(t) = 1/T \int_0^T f(t)g(t-\tau)dt \ .$$

In correlating a function with itself we get the *autocorrelation*, that I give in discrete form $\mathrm{ACF}(t) = 1/n \sum_{i=0}^{n} f(i)\, f(i-t)$, because that way it is used for data analysis in later chapters. Note that $\mathrm{ACF}(0) = \sigma^2$. If we normalise the ACF by σ^2 we get again the correlation coefficient ρ.

Conditioning

A few paragraphs above the important statement *conditioning reduces entropy* was made. If two samples are not independent they are correlated and we can use one sample to predict the other one. The *decorrelated* samples will have a lower entropy if that operation was well done. A very simple predictor is to use the first sample as a predictor for the next one. Essentially, this is a differentiation. Now let's find out if this is a good decorrelation.

Take $x_1, ..., x_n$ from a discrete source X with a certain σ_x and let $\mu_x = 0$ without loss of generality. The operation we are interested in is $d_i = x_i - x_{i-1}$.

$$\begin{aligned}\sigma_d^2 &= n^{-1}\sum(d_i - \mu_d)^2 = {}^{1}\ n^{-1}\sum d_i^2 = E(d_i^2) = E(x_i - x_{i-1})^2 = \\ &= E(x_i^2 - 2x_i x_{i-1} + x_{i-1}^2) = E(x_i^2) + E(x_{i-1}^2) - 2E(x_i x_i - 1) = {}^{2} \\ &= 2\sigma_x^2 - 2E(x_i x_{i-1}) = {}^{3}\ 2\sigma_x^2 - 2\sigma_x^2\frac{\mathrm{ACF}[x]}{\sigma_x^2} = 2\sigma_x^2(1-\rho)\end{aligned}$$

On the one hand we can now estimate the correlation of our random variable concerning the differentiation:

$$\rho = 1 - \frac{\sigma_d^2}{2\sigma_x^2}\ .$$

For $\sigma_d^2 = \sigma_x^2$ we see that $\rho = 1/2$, i.e. for a correlation coefficient of 0.5 the noise after differentiation is still the same. If there is no correlation ($\rho = 0$), there will be an increase of $\sigma_d^2 = 2\sigma_x^2$, or $\sigma_d = \sqrt{2}\sigma_x$. If $0 < \rho < 1/2$, we have a smaller increase of σ_d^2 and if $\rho > 1/2$ we finally decrease σ_d^2 with our operation. For the interesting case where the data are completely dependent ($\rho = 1$) we get $\sigma_d^2 = 0$, simply because each successive element is fully determined by its predecessor. This little example also shows us the relevance of the autocorrelation for the interpretation of dependencies between symbols, or, redundancy. Considering the entropy of an i.i.d. ($\rho = 0$) Gaussian source $H = \mathrm{ld}(\sigma\sqrt{2\pi e})$ we see that differentiation increases the entropy by $\mathrm{ld}(\sqrt{2}) = 0.5$ bit per sample, which is certainly something we would like to avoid. The prediction of x_i from x_{i-1} can be improved by a weight coefficient which depends on ρ. Such an exercise and the inclusion of even more preceding samples is the scope of linear prediction, which is briefly treated in Section 2.1.

Multiscale Entropy

The biggest problem of Shannon's entropy definition [Sha48] is that it completely ignores patterns and correlation within the data. In [Sta98b] an approach was made towards finding a better measure of information with the aid of multiscale analysis, hence the term multiscale entropy. It is defined as the sum of the information of each scale of a dataset's wavelet transform. Furthermore the noise is modeled to estimate its effect on the transform coefficients and allow for filtering to extract the signal. Taking the *à trous* wavelet transform, the multiscale entropy for Gaussian noise according to [Sta98b] is

$$H(X) = \sum_s \sum_k \frac{w_s(k)^2}{2\sigma_s^2}\ .$$

In this definition $w_s(k)$ is the wavelet coefficient at scale s for pixel k and σ_s is the noise at scale s. That way a measure for information is established that takes into account correlation within the data as well as background noise.

[1] assuming $\mu_d = 0$ to guarantee $\mu_x = 0$

[2] As x_i and x_{i-1} stem from the same random variable we have $E(x_i) = E(x_{i-1})$ and because $\mu_x = \mu_d = 0$ we have $E(x_i^2) = E(x_{i-1}^2) = \sigma_{x_i}^2 = \sigma_{x_{i-1}}^2$.

[3] $E(x_i x_{i-1})$ is the auto-correlation function at lag 1: ACF(1) and ACF(x)/ACF(0) = ACF(x)/σ_x^2 = $\rho(x)$.

Kolmogorov Complexity

An entirely different approach towards the information content of a given dataset is made by trying to find a shorter description for it. The *algorithmic information content* of a dataset S is the length $K(S)$ of the shortest program (in a universal programming language) that would generate the dataset. Kolmogorov complexity [Kol65] is a theoretical tool that gives us a better understanding of compressibility, yet it cannot be used in practice because one of its theorems says that there is no algorithm that correctly computes $K(S)$ for all S. Simply go back to the example above with the zeros and ones where it is obvious that shorter representations can be found in the first case where $S = 1111100000$. A program that, given the number n would output n ones followed by the same number of zeros would do the trick. Obviously, for large n the program itself would be considerably shorter than the whole dataset S. In the other extreme case where S is a random sequence it is likely that the shortest program simply outputs S itself by a `print`-like statement.

One important result from algorithmic information theory is that $K(S)$ does not depend on the programming language apart from a fixed overhead. Although we never know if we have found the shortest description or not, the expected value of the Kolmogorov complexity is close to the Shannon entropy for the typical datasets we are interested in. For further diving into the matter I refer to [Say03].

> Poincaré had an unusually retentive memory for anything he read; moreover, he could visualize what he heard, a useful faculty because he could not clearly see at a distance the mathematical symbols that were on the blackboard.
>
> —*EBm "Poincaré, Henri"*

3 The Colour of Noise

The probability distribution was introduced as the first important characteristic of noise. *Colour* is the other one, telling us how a probability distribution evolves. The parameters of a distribution – the moments like μ, σ, the ACF, etc. – stay the same in case of a stationary process and thus the power spectral density remains almost constant throughout the frequency spectrum. In this case the colour is called *white* – paying tribute to the electromagnetic spectrum of white light.

As soon as the power spectrum is no longer at a constant level, we call it *coloured*. In such a nonstationary process the probability function is a function of time and the ACF is no longer constant. So, parameters derived from one data interval are no longer valid for other sections of the dataset and need to be newly estimated. To anticipate what will be discussed in later chapters, the consequence for data compression is that any kind of predictive technique needs to constantly adapt to the new noise characteristics. The same also holds true for entropy coders and, well, compression in general.

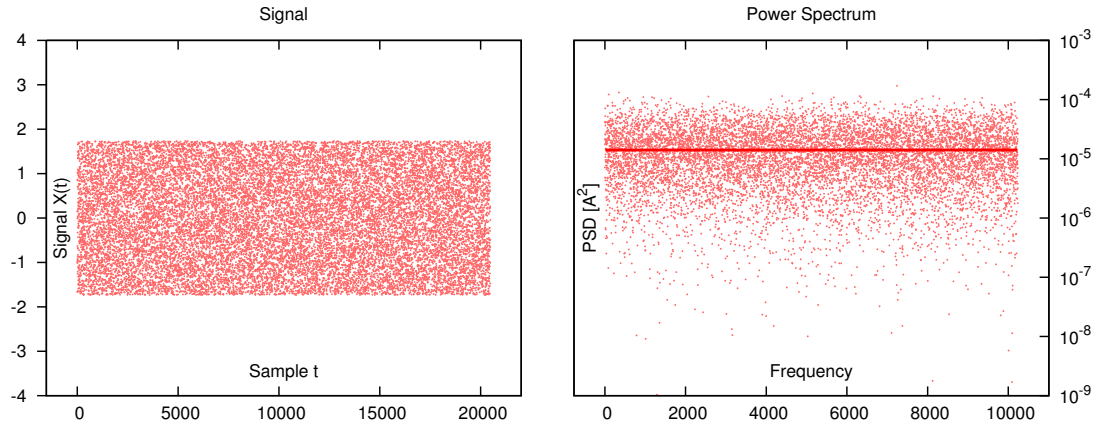


Figure 1.5: **White uniform noise.** (Left) The random variable X is drawn from a uniform distribution $X \sim \mathrm{U}(0,1)$. In the logarithmic power spectrum the noise amplitude of σ is indicated by the solid line.

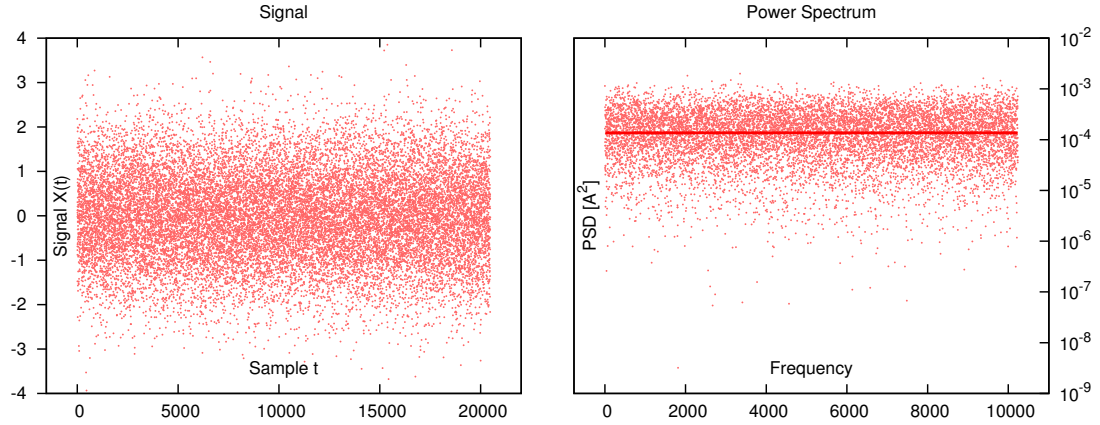


Figure 1.6: **White Gaussain noise.** (Left) $X \sim \mathrm{N}(0,1)$. (Right) The power spectrum is also flat and practically indistinguishable to white uniform noise.

White Noise

White noise is a random signal (or process) with a nearly constant spectral density, so that the signal contains equal power within a fixed bandwidth at any centre frequency. White noise is a valid approximation for many astronomical images, especially CCD ones [Ber06]. Uniform noise produced by a random number generator is normally white, unless it is specifically designed for another purpose. Figure 1.5 shows uniform noise and its power spectrum PSD$= 4|\mathcal{F}(f(t))|^2$ [1] depicts its frequency characteristics. So, a histogram of the left plot gives us an idea about the distribution and the PSD shows if any dependencies are contained in the dataset. White noise is therefore i.i.d.

White Gaussian

White Gaussian noise has the additional property of having a Gaussian amplitude distribution. This type of noise is prevalent in all kinds of electrical circuits and thus also in astronomical detectors. Such *thermal noise*, also called *Johnson noise* or even *Nyquist noise* is generated by the thermal agitation of the charge carriers inside an electrical conductor at equilibrium, which happens regardless of any applied voltage.

[1] The factor $1/n$ was included in $\mathcal{F}$, which then still gives half the amplitude $A/2$ as the unit of the transform coefficient. To compensate for this, the PSD is multiplied by 4.

Thermal noise is approximately white and the amplitude of the signal follows very nearly a Gaussian probability density function.

White noise is the starting point for science data compression. From this point of view any preprocessing or decorrelation operation has the purpose of *whitening* the signal. Note that the plots for the coloured noise examples (Figures 1.7 and 1.8) were generated with the white Gaussian dataset (Figure 1.6) as input.

Coloured Noise

So far I have treated a source as an i.i.d. random variable, where drawn samples adhere to the same *static* probability density function. In such a case the autocorrelation at lags different to 0 is – aside from the usual fluctuations – at a constant level and the power spectrum will be white.[1] If the noise spectrum is coloured we can hope to take advantage of inter-symbol redundancies. Taking the simple example of differentiating an i.i.d. source as an example, this operation creates purple noise, because the symbols are no longer independent. Each symbol of the difference set Y depends on two symbols from the original X and two succeeding symbols in Y even share one of the original components. Clearly, the differentiation eliminates large drifts at low frequencies and amplifies high frequency noise. This can be well observed by comparing Figure 1.8 with the white Gaussian dataset (Figure 1.6), which was used as its input.

Brown(ian) Noise

Brown noise or even *red noise* is the kind produced by Brownian motion, or, as I prefer to call it, by a random walk. An example in astronomy is non-destructive sampling during integration, which is the preferred way of reading certain types of detectors. While white noise can be said to have a $1/\nu^0$ character in the PSD, this type of noise follows a $1/\nu^2$ trend, which can be verified in Figure 1.7. Brownian noise can be easily produced by integrating white noise and a cheap but effective treatment is differentiation.

Purple Noise

Purple noise is somewhat the opposite of Brown noise. Its PSD stands out by its ν^2 trend and it can be produced by differentiating white noise. Note the increase of σ in Figure 1.8 compared with the i.i.d. source.

Pink Noise, etc.

Pink noise or $1/\nu$ noise is a signal or process with a PSD that is proportional to the reciprocal of the frequency. For pink noise, each octave carries an equal amount of noise power. The name arises from being intermediate between white noise $1/\nu^0$ and red noise $1/\nu^2$.

Other colours for noise exist in various branches of science and engineering (blue, grey, green, etc.), but their relevance is minor for the context of this thesis. Astronomical detectors nearly always have a more or less white Gaussian noise component and apart from that instrumental effects such as the *flatfield* dominate the scene.

[1] This is a good place to remember the Wiener-Khinchin theorem $\mathrm{ACF}(t) = \mathcal{F}(|f(t)|^2)$ which can be rephrased to state that the Fourier transform of the ACF is the power spectrum.

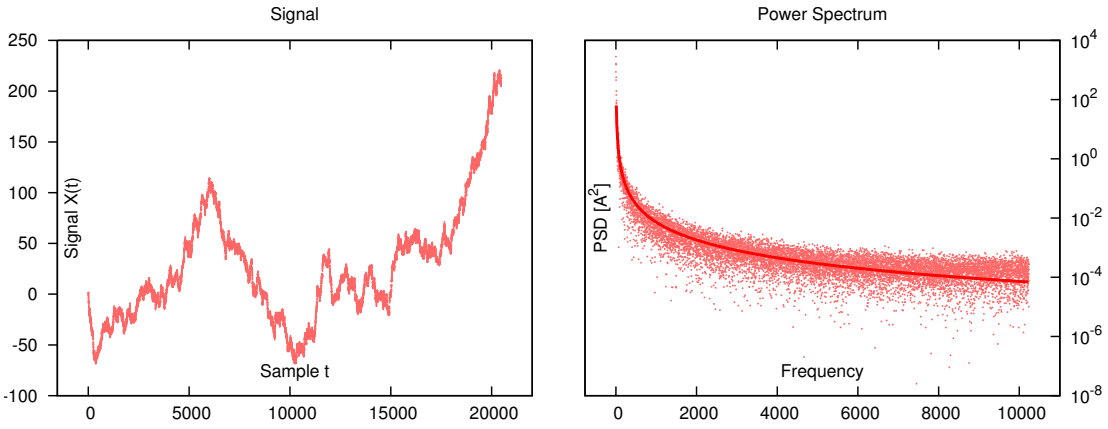


Figure 1.7: **Brown Gaussian noise.** (Left) The white Gaussian signal is integrated to generate Brownian noise. In contrast to other types of noise the signal itself already indicates its nature. The drunkard performing the random walk is this time twisted towards positive numbers. (Right) In the power spectrum the $1/\nu^2$ characteristic is revealed.

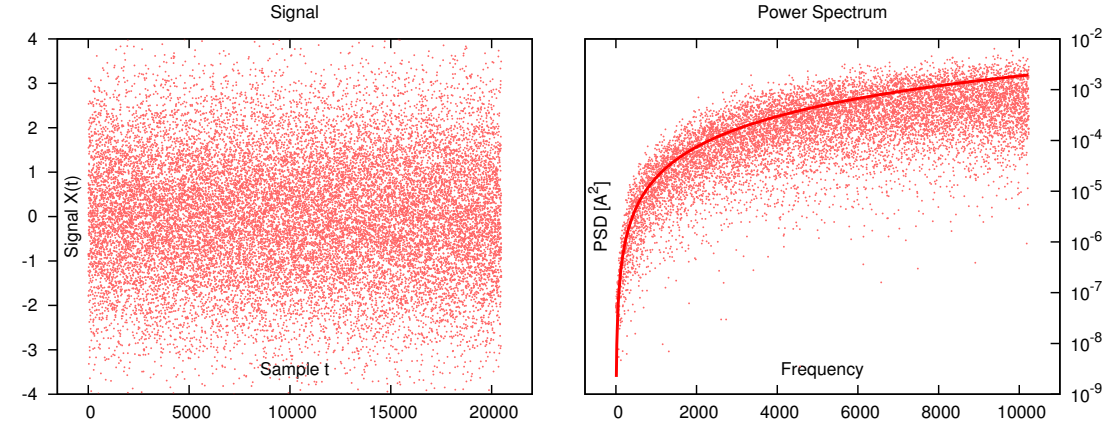


Figure 1.8: **Purple Gaussian noise.** (Left) While the signal itself is unremarkable, the PSD reveals its true nature (Right).

4 Sampling and Quantisation

The problems with science data already begin at the point where they are measured. It is usually desirable to correctly sample the signal in time and space, but *what exactly does that mean?*

Think of the signal as a continuous function $f(t)$. This function might be the amount of light that falls on a CCD pixel.[1] Problem number one is that we cannot measure all times t for various reasons, number two is that the analogue signal is mapped through A/D conversion to a coarser set of discrete values. The latter introduces additional quantisation noise and thereby increases the original noise by a certain factor. The answer to the former problem was given by Harry Nyquist in the early 20$^{\text{th}}$ century [Nyq24] and is nowadays known as the *sampling theorem*.

[1] From Ludwig Boltzmann and Max Planck we learned that the world is quantised. Especially when it comes to counting photons in the presence of quantum noise it is obvious, that the signal per se is not continuous. However, for our typical resolving capabilities we can confidently assume that the signal is originally continuous.

The Sampling Theorem

The sampling theorem is normally named after Claude Elwood Shannon, who proved it in [Sha49]. In his paper he writes:

> If a function $f(t)$ contains no frequencies higher than W cps, it is completely determined by giving its ordinates at a series of points spaced $1/2\ W$ seconds apart.

In a more modern way we would express it like: if a signal is bandlimited to W Hz, a sampling rate of $2W$ Hz is sufficient to reconstruct the original signal from the samples. So, if we want to measure certain details in time or space, we need to sample them with at least twice the precision. In this thesis uniformly sampled datasets are treated. It is however worth mentioning that the sampling theorem is also applicable to non-uniformly spaced samples. In this case the frequency resolution will be given by twice the average inverse Nyquist interval [Nyq24, Sha48]:

$$W = 2(n-1)^{-1} \sum_{i=1}^{n-1} (t_{i+1} - t_i)^{-1} = 2(n-1)^{-1}(t_n - t_1)^{-1} \quad .$$

An example is the way sound and music are digitised. The human ear is said to have a frequency response range from 20 Hz to about 20 kHz. Compact discs have a sample rate at 44.1 kHz to ensure frequency response up to the limit of human hearing.

In general it's a good rule of thumb to oversample by a factor larger than 2. However, recent mathematical research has shown that the Shannon sampling theorem is not complete and there are cases where even undersampled data can be perfectly reconstructed. These results and their impact on scientific measurement and data processing are known now by the term *Compressed Sensing* [Bob08].

Digitisation and Quantisation

The data type provided by typical astronomical CCD cameras or other kinds of detectors is in these days mostly a 16-bit (un)signed short integer. This is due to the employed analog-to-digital converters, considerations about gain, dynamic range and readout noise as well as the comfort of having two whole bytes per measure. The range of values is therefore between 0 and 65535 (or -32768 and 32767 if the data are signed). This digital sampling of analogue data is what we call digitisation.

Quantisation

If we speak in a more general context about mapping (continuous) values to a coarser discrete set of values, we use the term *quantisation*. Even integers can still be quantised: imagine a grid of even numbers where some input integer values need to be mapped to without modifying the scale. It seems natural to us that such an operation affects the data in a negative way and indeed, quantisation leads to an increase of the noise if the quantisation interval Δ (the inverse of the new resolution) is finer than $\sim \sigma^2$, which will normally be the case. An even rougher quantisation will quickly degrade the signal to a point where most of the information has been removed.

Now let's take a close look at a single quantisation interval $Q = [A, B]$ spanning over a distance $\Delta = B - A$. Any sample $s \in S$ that falls into $A \leq s \leq B$ will be represented by either A or B, depending on the quantisation strategy used. So we lose the information of the original shape of the probability distribution and make it uniform on Q. For a

better understanding imagine Q is one of the steps in a step-wise approximation to the original histogram of S. If the quantisation process works like normal[1] rounding, the same number of measures should fall between $[A, (A+B)/2]$ and $[(A+B)/2, B]$ if s is uniformly distributed in Q. If the latter condition is not fulfilled – for very peaked distributions together with a very large Q – we will get an additional increase in σ due to the asymmetry of s in Q. The maximum error between s and its quantised counterpart is $\pm(A+B)/2$. To express this error in a useful form we recall the variance of the uniform distribution and apply it to our interval $\sigma_Q^2 = (B-A)^2/12 = \Delta^2/12$. The variance of this error adds up with the original signal variance σ_s to an increased total variance of $\sigma^2 = \sigma_s^2 + \sigma_Q^2$.

Resampling

Quantisation must also be considered when the sample rate is converted. Resampling is the application of multivariate interpolation. Casual methods are point sampling, where pixel values are derived from the nearest neighbour, bilinear and bicubic (spline) interpolation [Pre07], kriging [Kri51] and transform-based resampling. In case the sample rate should be reduced (downsampled) by means of frequency reduction to the new bandwidth to avoid *aliasing*, the term *decimation* is also used. One should have these things in mind when designing or choosing a quantisation stage, especially when the linearity of the signal within the dynamic range is questionable. For science data one should also hesitate with other techniques than scalar quantisation, because these may lead to differences in the weighting of individual values.

Scalar Quantisation

Think of scalar quantisation like simple rounding, but in a more general form. The word *scalar* is due to the uniform quantisation grid, which resembles a process of zooming. Usual definitions found in literature like [Str05] are $q = \lfloor |s|\Delta^{-1} + 1/2 \rfloor \mathrm{sgn}(s)$ for a so-called *midtread* quantisation with reconstruction $s_q = q\Delta$,[2] or $q = \lfloor |s|\Delta^{-1} + 1 \rfloor \mathrm{sgn}(s)$, $s_q = (q - 1/2)\Delta$ for a *midrise* quantiser, where 0 is not a quantisation step.

Adaptive quantisation is different, because it maps the input to a non-uniform grid by a scaling function for example. The motivation behind is to give more precision to more probable values in non-uniform distributions at the cost of values that appear less frequently. In combination with *prefix codes* (see Chapter 3) this can make up for an effective lossy compression technique (e.g. in speech compression), which I do not recommend for astronomical datasets, where the interesting small bright portions of the data (the stars) are outweighted by the large background area at low values.

I also need to mention *vector quantisation* [Sal07], which is something entirely different. It merges input values into multidimensional vectors, defines pointers towards the most frequent ones in a code-book and quantises the pointers of less frequent vectors to the ones already available in the code book. This is also frequently used in speech compression systems, but equally problematic for science data.

What about $\Delta/2$?

There is one striking point in where rounding strategies differ and that's what happens if s lies exactly in the middle of Δ. In daily life we will round up a number like 7.5 to 8 without thinking about it. Blame your teacher, but this is simply not correct! By doing so a systematic error is made which leads to an increase in μ and an additional increase in σ. For floating-point data which differ a lot in their decimals this is no problem of course, but if the only value between integers is .5 this additional bit of information is simply wiped out.

[1] For now we don't pay special attention to the case $s = (A+B)/2$.

[2] $\lfloor\,\rfloor$ is the floor operator, the largest integer smaller than x.

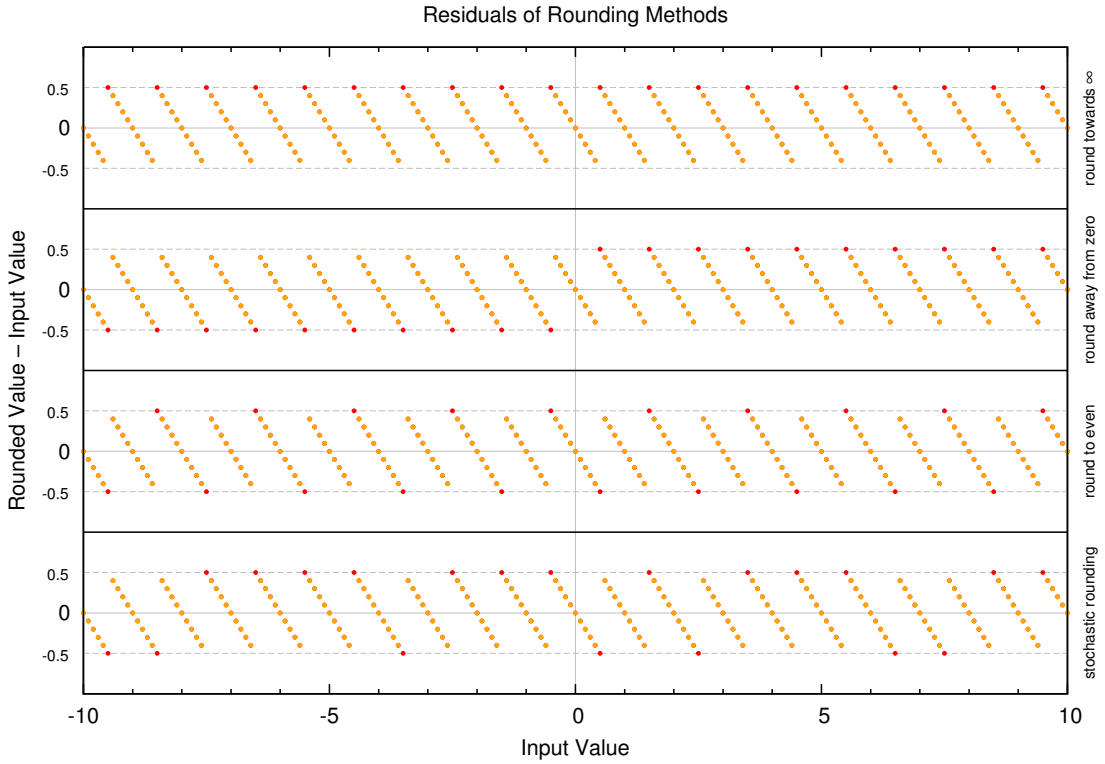


Figure 1.9: **Different rounding strategies in comparison.** A sequence of numbers increasing by 0.1 from -10 to 10 was generated and rounded to integer with different strategies. The residuals are shown as orange circles with the cases where .5 has to be rounded up or down in red colour. (Top) Rounding towards ∞ increases the mean of the dataset by 0.05. (Second line) Rounding away from zero leaves μ in the centre if as many positive as negative numbers are there, but it leads to a different scale around zero (only 9 points will be rounded to 0). (Third line) Rounding to even is a well-balanced method, but it introduces periodicities in the dataset. (Bottom line) Stochastic rounding is balanced and has no systematics if the randomisation is not too bad.

The most popular strategy to deal with this is to round towards the next even number. However, on small scales there is still a systematic effect left, which even leads to periodicities in the dataset. The one correct way of rounding is to *randomly* round up or down. This is finally called *stochastic* rounding. Some rounding strategies introduce a discontinuity around zero, like rounding towards or away from it. This should be avoided in any case for science data because it disturbs the metric around zero. Figure 1.9 tries to support what is written here with a graphical comparison of the mentioned strategies. We will continue with these matters shortly in Section 6, but first we need to have a measure for the quality of data.

> Specific organisms, such as *Brevibacterium linens*, in Limburger cheese result in a reddish brown surface growth and the breakdown of protein to amino nitrogen. The resulting odour is offensive to some, but the flavour and texture of the cheese are pleasing to many.
>
> —*EBM "Food Processing"*

5 Quality of Science Data

Given some sets of science data we need to to distinguish between good and bad measurements. In other words, we need a measure to assess the quality of a dataset. This depends on the type of data and their purpose, so several metrics exist.

Signal to Noise Ratio

With the statistical tools that were discussed so far we can intuitively take the ratio between μ and σ to express how much larger a bias-free signal S is in units of the noise N. Well, this is already the SNR used in astronomy. There the SNR can be very small and it is not uncommon to speak of "5σ, 1h", which means that a SNR of 5 is achieved in a one-hour measurement (remember the square-root law). On the other hand when dealing with bright sources the SNR can grow by up to several orders of magnitude. In astronomy it is conventional to give the SNR in normal scale units, whereas in technical disciplines decibels are preferred and both signal and noise are expressed by the respective power (the squared amplitude). Recall that *bel* is the common logarithm of a ratio, then one *decibel* is 1/10 bel of course, so if A is the amplitude, we have $\mathrm{SNR}_{db} = 10\log(A_S^2/A_N^2)$.

in astronomy, $\mathrm{SNR} = S/N$ in digital units

Error Metrics

The error metric we have been using so far was the *mean squared error* MSE (the variance σ^2) and its more useful friend the *root mean squared error* RMSE (the standard deviation σ). These metrics are useful especially in Euclidean space, so if we want to test rounding algorithms for instance we take the sum of the squared residuals (and normalise to the number of pairs if we like). In image processing it is not uncommon to use the *mean absolute difference* MAD, which is faster to compute and some authors even argue that this measure comes closer to human perception.

In mathematics, a metric is a "measure of distance", that is prerequisite to a norm – the "measure of size". Different notions of norms exist for vector spaces, defined by

$$\ell_p : ||x||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} ,$$

of which we recognise the ℓ_2-norm as the Euclidean and find ℓ_1 in the MAE.

Rate Distortion Function

The effect of quantisation on science data has been explained, but for lossy operations in general we also need to extend our understanding to the more general theory of rate and distortion. *Distortion* describes the error that is made by representing a source X by the distorted X_D, just like the quantisation error. We will use $d(x, x_D) = (x - x_D)^2$ as the measure of distortion and keep $D \leq E(X_D - X)^2$. The *rate* is of course the entropy of the dataset. So, given the entropy, we want to know what the effect of a given distortion is on the rate and equivalently, how large the distortion is if we reduce the

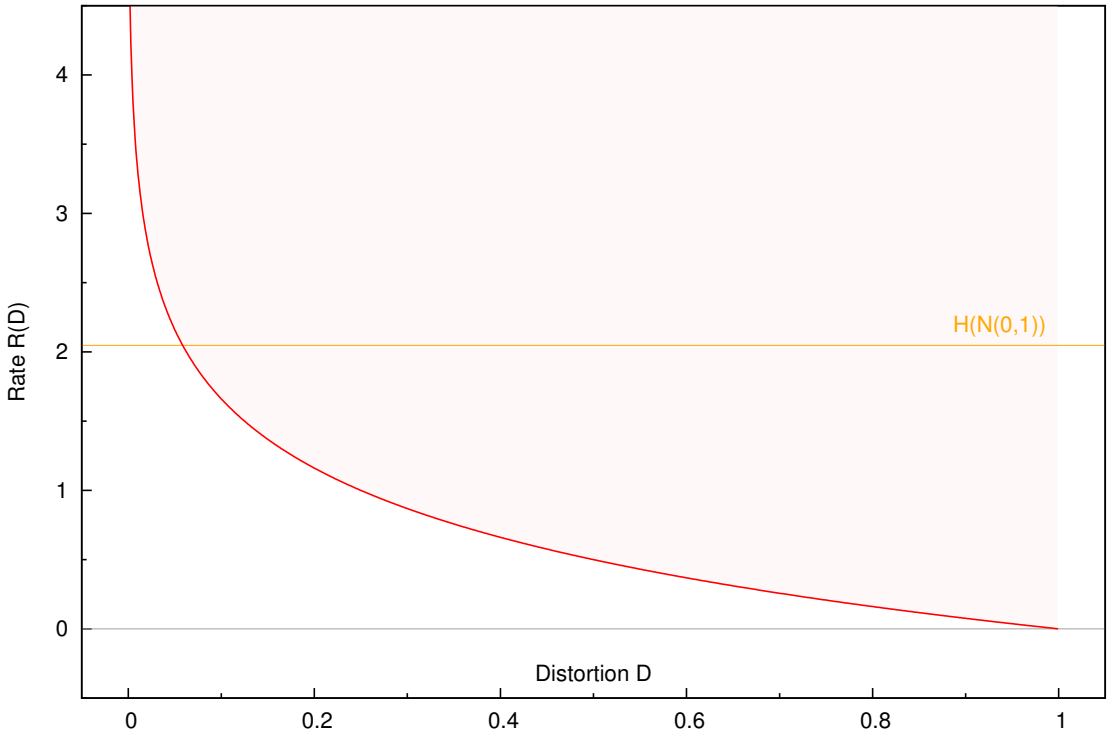


Figure 1.10: **Rate distortion function for N(0, 1).** The R(D) function illustrates the tradeoff between lossy compression and data quality. It is given in bits and represents the lowest limit of all achievable (R, D)-pairs. The unit of D is σ^2. (Orange line) The entropy for N(0, 1) is also shown in the plot. Close to zero R(D)$\rightarrow \infty$ owing to the fact that continuous samples have infinite precision and therefore infinite rate.

entropy to a certain rate. So we use the rate distortion function $R(D)$, which I define similar to [Cov06] by

$$R(D) = \min I(X; X_D) \quad \text{subject to} \quad p(x_D|x) : \sum_{(x,x_D)} p(x)p(x_D|x)d(x, x_D) \leq D \ ,$$

where $p(x)$ is the distribution of X. We had the definition of mutual information in Section 2, $I(X; X_D) = H(X) - H(X|X_D)$. For a $N(0, \sigma^2)$ source it can be shown that

$$R(D) = \begin{cases} \frac{1}{2}\mathrm{ld}\frac{\sigma^2}{D}, & 0 \leq D \leq \sigma^2 \\ 0, & D > \sigma^2 \ . \end{cases}$$

The rate distortion function gives the lower bound of the achievable rate distortion pairs (R, D) as depicted in Figure 1.10.

Another important finding from rate distortion theory is that combined symbols are less susceptible to distortion than individual ones, even if they are completely independent. This justifies the use of transforms together with quantisation as a very good strategy to reduce the rate at minimum distortion.

6 Combined Averaging and Rounding

You should already have a feeling that there are cases where averaging is not as simple as summing up values and dividing by their number. Quantisation effects must be considered as soon as integer arithmetic is involved. For the purpose of on-board data reduction I have also developed several algorithms that have the task of averaging a number of integers with optional rounding. Bit-rounding of integers means to map to a coarser grid, e.g. after 1-bit rounding all odd numbers will be gone, after 2-bit rounding only numbers divisible by 4 will be left etc. In here I present what I have put together as the combined averaging and rounding algorithm which is used in the PACS on-board software.

PACS FM Averaging

The good thing about the PACS signal processing units, which are the computers responsible for on-board reduction and compression, is that DSPs are used. They have quite complex instructions also in floating point precision and every instruction is executed in one single cycle. The combined averaging and rounding algorithm reads

$$m_r = \frac{1}{2^r n}\sum_{i=1}^{n} x_i + \frac{1}{2}\,\mathrm{sgn}\sum_{i=1}^{n} x_i - \frac{\mathrm{rand}(1)}{2^r n} \quad .$$

In here n is the number of samples to average, r is the number of bits of additional rounding and `rand(1)` is a random number being 0 or 1. The inputs are integers, the operations are carried out partly in float and after calculation the result is cast back to integer. Note that the mean m_r is right-shifted by r bits if bit-rounding is used.

Here are the sources for the combined stochastic FM averaging and rounding algorithm, consisting of `StatAveraging` and a helper function `StatRestRoland`, which takes care of the second and third term of the formula. You will see that this is a little bit more tricky than just $\sum x/n$.

Listing 1.3: subroutine for the randomisation

```
static float StatRestRoland (int average, float invden)
{
  int temp;
  float sign = 1.0;
  float stat = 0.5;

  if (average < 0)
  {
    sign = -1.0;
    stat = -0.5;
  }
  URAND_seed = URAND_seed * URAND_A_val + URAND_C_val;
  temp = (URAND_seed & URAND_CLIP);

  if (temp > URAND_RMXH)
    stat = stat - sign*invden;

  return stat;
}
```

In the following listing, `StatRest` is a function pointer towards `StatRestRoland`, but another rounding strategy (`StatRestKoryo`) could be selected. Also note that `roldiv` is a replacement for integer division.

Listing 1.4: stochastic averaging in C

```
int StatAveraging (int *source, int nbPixelsPerFrame, int nbFrames,
                   int *dest, int numavg, int RoundingBits,
                   int strategy)
{
  int pctr=0, bctr=0, sctr=0; // counters for pixels, blocks, samples
  int blocks = 0; // number of averages to generate
  int remain = 0; // remaining frames
  int avg = 0;
  int destctr = 0;
  int offset;
  int den;
  int rem_den;
  float favg, deninv, rem_deninv;

  // choose between two different kinds of rounding.
  float (*StatRest) (int average, float invden) = StatRestKoryo;
  if (strategy == K_ROUNDING_STYLE_ROLAND)
    StatRest = StatRestRoland;

  blocks = roldiv (nbFrames, numavg);
  remain = nbFrames - blocks * numavg;

  den = numavg * (1 << RoundingBits);
  deninv = 1.0f / (float)den;

  if (remain)
  {
    rem_den = remain * (1 << RoundingBits);
    rem_deninv = 1.0f / (float)rem_den;
  }
  else
    rem_deninv = 1.0f;

  for (pctr=0; pctr < nbPixelsPerFrame; pctr++)
  {
    avg = 0;

    // average the complete blocks first
    for (bctr=0; bctr < blocks; bctr++)
    {
      offset = pctr*nbFrames + bctr*numavg;
      avg = 0;
      sctr = 0;
      while (sctr < numavg)
      {
        avg += source[offset + sctr];
        sctr++;
      }

      favg = (float)avg*deninv + StatRest(avg, deninv);

      dest[destctr] = (int)favg;
      destctr++;
    }
```

```
55      // last incomplete block (it has less samples than numavg)
56      if (remain)
57      {
58        offset = pctr*nbFrames + bctr*numavg;
59        avg = 0;
60        sctr = 0;
61        while (sctr < remain)
62        {
63          avg += source[offset + sctr];
64          sctr++;
65        }

67        favg = (float)avg*rem_deninv + StatRest(avg, rem_deninv);

69        dest[destctr] = (int)favg;
70        destctr++;
71      }
72    }

74    return destctr;
75  }
```

The `// last` block is there for cases where the number of input values is not divisible without rest by the number of samples to average.

Performance

The performance of the averaging algorithm for PACS is best measured with random numbers. To resemble the detector noise on short time scales white Gaussian noise was fed with different μ and σ into the averaging algorithm. The increase of the averaged noise's σ with respect to the input is shown in Figure 1.11. For this test, the number of samples to average has been set to 4 and so the expected result for no quantisation is 0.5, as this operation reduces the noise by a factor $\sqrt{4}$. This is indeed achieved by the unquantised averaging in float. Two other lines in the plot compare the rounding algorithm that was used for the qualification model ("4/0 13.8", it rounds towards $+\infty$) with the new averaging ("4/0 R"). The results that are especially interesting for us are "4/1 R" for 1-bit rounding and "4/2 R" for 2-bit rounding. Of course, if the quantisation becomes comparable with σ, the noise is gradually cut away. Apart from that the "R" algorithm follows the theoretical prediction very well. We see that 2-bit rounding increases this to ~ 0.56, that is 12 percent, but it is not yet degrading the amplitude.

Behaviour around 0

For Figure 1.12 the amplitude of the noise was kept constant, but its mean was varied across 0 to see if a sign change causes an effect. Here the "4/2 K" strategy differs in that it has a peak around zero. The reason for this is that this one changes rounding direction at zero, which leads to a different scale at that position. All other candidates have no problems with this.

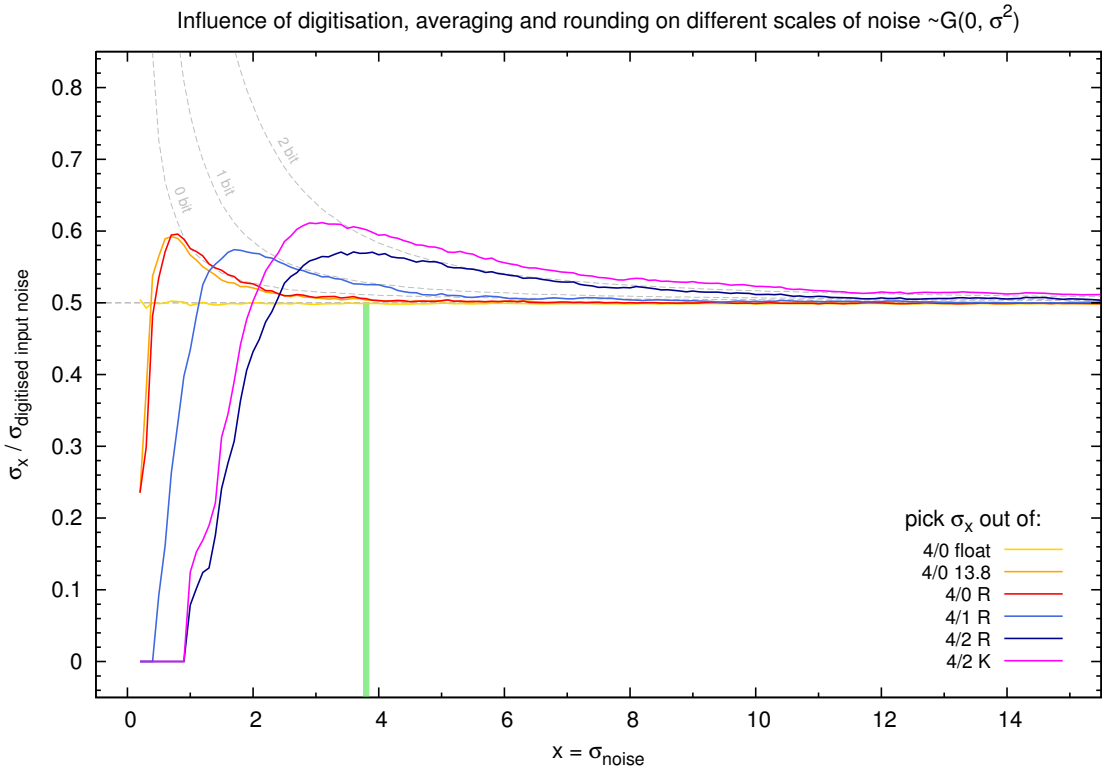


Figure 1.11: **Rounding for different noise amplitude.** In this plot the σ of the input noise is varied and compared with the rounded noise. The dashed grey lines show how the noise increase should theoretically look like. The green line indicates where noise of $\sigma = 7.59$ would be.

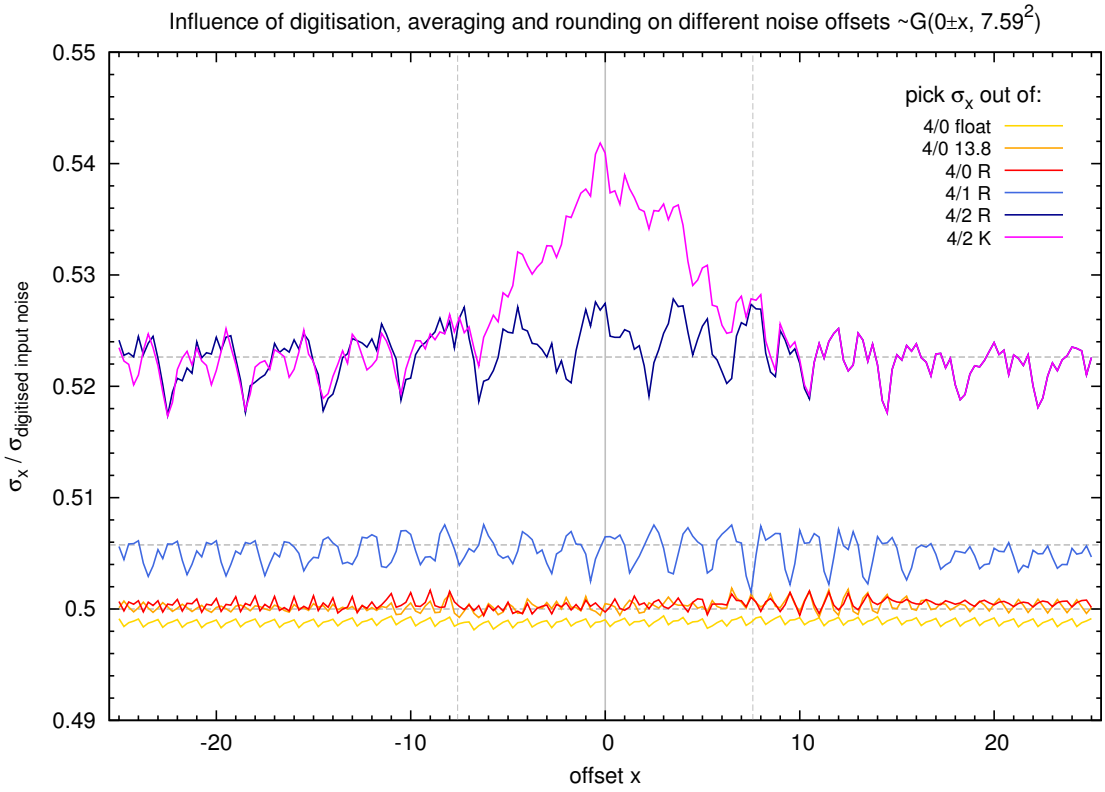


Figure 1.12: **Rounding behaviour around zero.** Different μ centred around zero were fed into the algorithms. The horizontal grey lines indicate theoretical predictions and the vertical ones mark the $\pm\sigma$ boundaries.

2

Decorrelation

This chapter is essentially about representing signal and noise. The starting point may be a dataset with all kinds of features on all scales, but what is desired is a reversible representation in as small values as possible, ideally just noise around zero. This can be achieved by prediction to take the conditional entropy out of the data. A simple predictor would be the last seen value and the encoded offset would then just be the difference. Such preprocessed data can then be passed on to a compression back-end, an entropy encoding algorithm.

Another way to achieve a flat representation is decorrelation[1] through *linear transforms*. The interesting scientific information – the signal energy – is concentrated by means of a change of bases in a few large coefficients, whereas the majority of coefficients are found to be small around zero. This now wide-spread principle has some more advantages, because it allows almost any degree of lossy operation close to the rate distortion limit. In the last two decades *wavelets* and all kinds of other *-lets* have joined this family of algorithms. Decorrelation is now at the core of almost every lossy compression, the most prominent ones being JPEG, MPEG and MP3 – three standards that use the discrete cosine transform. Many modern lossy compression schemes follow a three step approach by performing decorrelation, quantisation and encoding in three separate steps.

> Roland, I didn't know I was so ignorant about compression.
> —*Renato Orfei during coffee break at IFSI*

This chapter begins with linear prediction and immediately afterwards we dive into the reign of linear transforms. There I concentrate on orthonormal transforms, because they have favourable features for science data. With only a few exceptions, wavelets are still poorly understood in the astronomical community. One reason for this is that there are many ways leading to them with discouraging words like *multiscale analysis*, *subband decoding* and *filter banks*. These are more or less just names for the same thing and I will try to avoid them. Instead my approach is purely through matrix multiplication, even if this is not the usual way for some of the presented transforms, but astronomers rather have a sympathy for pure algebra than for electronic filters. With the exception

[1] Concerning terminology I need to clarify that an operation which acts as a decorrelation is at the same time a correlation as well, it merely depends on the point of view. The reason for this is that after a (de)correlation, variables which were correlated are then decorrelated and vice versa.

The chapter is divided into the following sections:

of the continuous Fourier transform I will also concentrate on discrete transforms only, simply because continuous transforms have no relevance on discrete uniform datasets. Most of the transforms have already been used in space applications and I will show their strengths and weaknesses with an illustrative example data file as well as probe their performance on real science data.

1 Predictor and Corrector

Linear prediction is the estimation of signal values from linear combinations of previous samples [Rab78]. This technique is popular in speech compression and analysis, in the analysis of radar applications and seismic waves. Among the examples for use in on-board software is also PACS, but higher order predictors have not been used so far due to their increased computational cost for little additional decorrelation. Basically it works like predicting the current sample by its neighbours (the already encoded values) and encoding the error to its actual value (the corrector). That way low frequency components are more or less absorbed and what is left is the high frequency noise. Of course a prediction based on the last-seen values can be optimised by weighting them. Finding these weights is the task of linear prediction. This concept is also found in the more general context of DPCM (Differential Pulse Code Modulation) [Say06, Jay84]. It is central to many speech compression algorithms like CELP [Sch85] (variants are used in mobile phones) or lossless audio compression like FLAC [Sal07] and related with various higher-order prediction techniques in text and image compression.

Differentiation

The easiest predictor p_n for a sample s_n to use is the previous sample s_{n-1}. Then the corrector, i.e. the error of the prediction is $e_n = s_n - s_{n-1}$. The consequences of such a prediction which is essentially a differentiation have been discussed in Section 1.2. To give the short version of what was written there, the variance after differentiation becomes any value between 0 and 2 times the original variance, depending on the

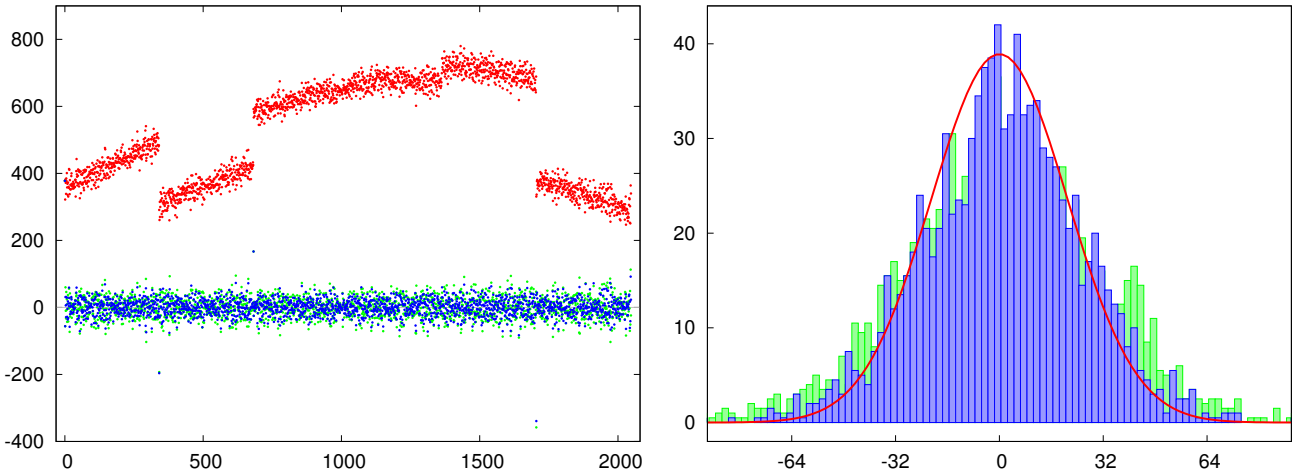


Figure 2.1: **Decorrelation by running average prediction.** (Left) The synthetic input values, as plotted in red colour contain a drift, noise and larger steps. The green dots show how a differentiation would bring these down to zero. The blue dots show how a running average with threshold performs. (Right) Histogram of the differences (green) and the correctors (blue) at a bin size of 2. The differences have a σ of 33, the correctors have σ=27.6. The red line is a Gaussian envelope of the original noise σ=20.9.

correlation in the dataset. Figure 2.1 shows how simple differentiation already brings down the data to just noise around zero.

Running Average

A better predictor is a running average, where $p_n = m_n$, $m_n = (m_{n-1} + s_{n-1})/2$ and $m_0 = s_0$. Internally, the predictor is adjusted and the error is encoded. For such a prediction we can already add a weighting parameter to s_{n-1} which can be used to make the running average $m(n)$ more resistant to high noise or outliers. For step-like functions it might be wise to adjust the running average with more aggression. It can be a good idea to include a threshold in the correction step. If the error e_n is found to be above that threshold, the running average shall then take on the new value instead of adjusting the mean. Such a decorrelation is also shown in Figure 2.1 in comparison with simple differentiation. With respect to the input data noise, differentiation increases the standard deviation σ by a little more than a factor $\sqrt{2}$ for this dataset, because the signal component (drift and jumps) is not taken completely out. The running average leads to an increase by a factor 1.32, which is definitely better, but also suffering from the drift. This can be further reduced by using higher order prediction with self-adjusting weights, ideally down to almost no increase of σ.

Linear Prediction

The trick in LPC (Linear Predictive Coding) of higher order is to determine the weights for the particular dataset. If our predictor is to the order of k, our prediction is $p_n = \sum_{i=1}^{k} w_i s_{n-i}$ from the last $s_{n-1}, ..., s_{n-k}$ values. The error of that prediction with respect to the actual sample s_n is then $e_n = s_n - \sum_{i=1}^{k} w_i s_{n-i}$. The coefficients w_i can then be calculated by minimising the expected value of the squared error: $\min Ee_n^2$. This minimisation has to be made either over the entire dataset or just smaller chunks of data, depending on whether the probability distribution is stationary as well as on the general circumstances of the application. These relations are treated in more detail and with an algorithmic outline in [Say06]. To give you an idea about the complexity

here is how I digested the matter:

$$\begin{aligned}\frac{\partial}{\partial w_j} E\left(s_n - \sum_{i=1}^{k} w_i s_{n-i}\right)^2 &= -2E\left(s_n - \sum_{i=1}^{k} w_i s_{n-i}\right) s_{n-j} := 0 \\ \Rightarrow \sum_{i=1}^{k} w_i E s_{n-i} s_{n-j} &= E s_n s_{n-j}\end{aligned}$$

This is the so-called Wiener-Hopf equation [Kam98]. Assuming that the s_n are stationary we can set $E\, s_{n-i} s_{n-j} = \mathrm{ACF}(i-j)$ and with the k equations we retrieve a system:

$$\begin{bmatrix} \mathrm{ACF}(0) & \mathrm{ACF}(1) & \mathrm{ACF}(2) & \dots & \mathrm{ACF}(k-1) \\ \mathrm{ACF}(1) & \mathrm{ACF}(0) & \mathrm{ACF}(1) & \dots & \mathrm{ACF}(k-2) \\ \vdots & & & & \\ \mathrm{ACF}(k-1) & \mathrm{ACF}(k-2) & \mathrm{ACF}(k-3) & \dots & \mathrm{ACF}(0) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} = \begin{bmatrix} \mathrm{ACF}(1) \\ \mathrm{ACF}(2) \\ \vdots \\ \mathrm{ACF}(k) \end{bmatrix}$$

Solving the equation system takes a considerable amount of CPU power, which is a problem for on-board applications. Especially the many autocorrelations take most of the time, the matrix inversion is not so costly, because it has a special form (Toeplitz) for which fast algorithms such as Levinson-Durbin [Say06] can be used. In the non-stationary case the elements $E\, s_{n-i} s_{n-j}$ cannot be replaced by the ACF and the derived matrix needs to be inverted with slower techniques. This makes it particularly problematic for on-board data reduction.

If the weights need not be adaptive they can as well be calculated once on an offline dataset. In this case I can recommend LPC, because the operations left amount to just a few additions and multiplications for each sample to be made and the prediction is definitely better than simple differentiation. Note that LPC as presented herein works on one-dimensional data, though it is possible to extend the concept to higher dimensions at increased CPU cost.

2 A Zoo of Transforms

I tend to call any operation that uses more than one sample to compute a new one a (de)correlation, simply because existing dependencies are on the one hand resolved and new ones are made. However, the family of algorithms that dominate in the field of decorrelation is the one of *linear transforms*. Among these especially the orthonormal transforms are useful for us, because they ensure that the signal energy is conserved (then also called *unitary transform*). I guess that everyone has already heard the word *wavelets* and at the same time only a vague idea about what they are and what they can do for us. One reason for this may be that the Fourier transform is so well established in astronomy and in many fields still the method of choice that the demand for wavelets is small.

Countless books have been written about transforms and wavelets. The definite guide to the Fourier transform is [Bra99a]. If the mathematical theory of wavelets is in focus, then there is hardly a way around the first three chapters of [Dau92]. The best reference on transforms and applications for the astronomer is however provided

by [Sta98a] and [Sta06]. Good practical use concerning data compression is provided by [Sal07]. But before consulting these works I propose to read through this chapter first. Before we go on, here are a few definitions.

Basic Definitions

A *transform* is a mathematical relation that projects data from one basis onto another. A *basis* spans the vector space with finite linear combinations of n linearly independent vectors generating an n-dimensional subspace of $\mathbb{R}^n$. This sounds awful, so let me circumscribe it this sloppy way: the basis vectors work just like the axes of the new coordinate system. The standard basis on the Cartesian plane is $e_1 = (1, 0), e_2 = (0, 1)$. A basis is *orthogonal* if the basis vectors are mutually perpendicular and *orthonormal* if they additionally have a length of one. Therefore an orthonormal transform preserves all lengths and angles and is energy conserving. Once a dataset has been transformed, the values are called *transform coefficients*. In many frequency-isolating transforms the first coefficient (corresponding to a frequency of zero) is called the DC coefficient, which is not seldomly the average and all others are the AC coefficients, conceivable as differences. A matrix – an image – with many zeros (or negligibly small values) and few large values is a *sparse matrix*. The opposite would be a *dense matrix*. It seems natural that a sparse dataset is easier to compress than a dense one.

> The term matrix was introduced by the 19th-century English mathematician Arthur Cayley, who developed the algebraic aspect of matrices.
>
> —*EBm "matrix"*

1D and 2D

The most important thing about the transforms we want to use is that they are reversible. The transform is carried out by matrix multiplication where the rows of the transform matrix W contain the basis vectors. Matrix multiplication can of course be inverted: $A = W^{-1}(WA)$. Consult your algebra book to find out that if an $m \times n$ matrix A (m rows and n columns) is multiplied with an $n \times p$ matrix B the operations to carry out $C = AB$ are $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ ($i = 1..m$, $j = 1..p$) and C is an $m \times p$ matrix.[1] Something that we will definitely need is to deal with the *transpose* $(AB)^T = B^T A^T$. The good thing about an orthogonal basis W is that $WW^T = W^T W = I$ (the identity matrix) and $W^T = W^{-1}$, so that the one-dimensional transform is $T_{1D} = WA$ and the way back is $A = W^{-1} T_{1D}$. Note that normalising factors need to be considered if W is not orthonormal. For better memorability I write

$$A = W^{-1}(WA) = WWA.$$

It is important to stress that a transform through matrix multiplication is a one-dimensional operation on the columns of A. However, this can be easily extended to any number of dimensions because of separability. To do so, we consecutively apply the basis change along each dimension separately. To make a 2D-transform of an image A by our orthogonal basis W we transform the transpose of the 1D-transform: $T_{2D} = WT_{1D}^T = (T_{1D} W^T)^T = WAW$. Analogous to the 1D case I put it into the memorable equation (this is a nice exercise to build from scratch)

$$A = W(W(W(WA)^T)^T) = WWAWW.$$

[1] Remember that matrix multiplication is commutative, but not associative, i.e. $AB \neq BA$, but $(AB)C = A(BC)$.

Normalisation

In that sense the 2D Fourier transform of the image A is $\mathcal{F}_{2D} = \mathcal{F}\mathcal{F}(A)^T$ and the way back is $\mathcal{F}^{-1}\mathcal{F}^{-1}(\mathcal{F}_{2D}A)^T$. Not quite! We have not taken into account that the discrete Fourier basis is not normalised, so the cleanest thing would be to include a scalar factor $1/\sqrt{N}$ in $\mathcal{F}$ as well as in $\mathcal{F}^{-1}$, but the way the Fourier transform is normally defined the factor $1/N$ is included in the inverse transform. Note that $1/\sqrt{N}$ is energy conserving, whereas $1/N$ conserves the dynamic.

> His lectures at Cambridge attracted very few students; [...]
> —*EBm "Arthur Cayley"*

Before we begin our journey through that jungle of transforms we need proper test data. Astronomical images tend to be already sparse, as they have few large values and lots of dark background. Such an image will not be very helpful to judge the energy compaction capability of a transform. Therefore I have made up another test image which will allow for better comparison. Section 3 at the end of this chapter contains transforms of some typical astronomical datasets and to assess the performance on one-dimensional data we will continue to use the dataset from Figure 2.1.

La Mona Roli

The test image that I am using to display the properties of each transform in this chapter is a small image that I made more than a decade ago, cropped and put to greyscale. It serves better for explanation than real astronomical data, because it has recognisable features even in the transformed domain. With 256×256 pixels in 8 bit greyscale it has a raw size of 65536 bytes. In Figure 2.2 the image is shown as well as its covariance matrices and the histogram. A covariance matrix is interpreted the following way:

- It is a symmetric matrix, so only focus on the columns, the lines are not interpreted.
- On the diagonal lies the σ^2 of the respective column. Large positive values should be found here.
- The vertical offset of a nonzero value from the diagonal indicates a correlation for this column at that shift. If the value is normalised by the variance you get the correlation coefficient.

The histogram is self-explanatory with one thing to add: the cuts are centred on μ and set to show 99.75% of the area, so that outliers have no negative effect on the scale. This is also the way the cuts for scaling the brightness in all images to follow are made unless otherwise indicated.

Evaluation

A little bit further down we will start to transform Mona Roli with different bases and try to assess the effects of that operation. Whenever necessary, the mean μ and standard deviation σ, the entropy H (H_S for the entropy per symbol) and the multiscale entropy M will be given. Note that the M I use is inspired by the multiscale definition

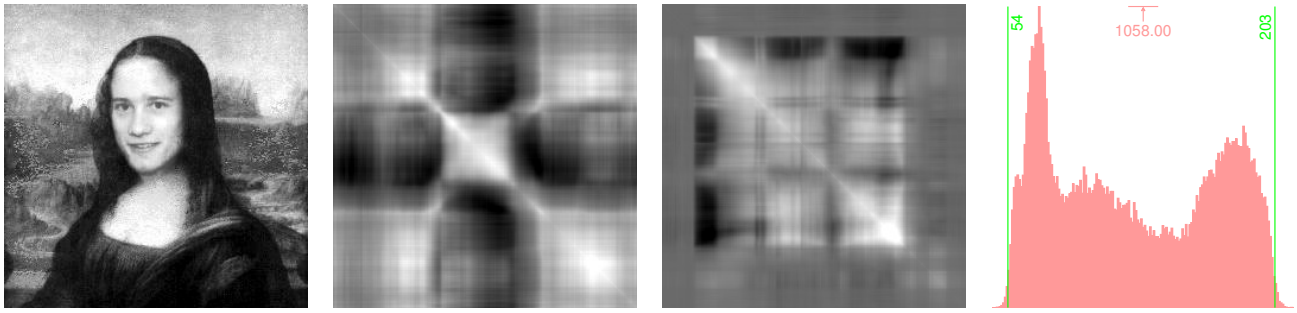


Figure 2.2: **Mona Roli test image, covariance matrices and histogram.** (Left) The original input image is an 8-bit 256×256 greyscale image. (Middle left) The covariance matrix of the input image quantifies vertical correlation. (Middle right) Covariance matrix of the transpose to get an impression of horizontal redundancies. (Right) Histogram of the input image.

in Section 1.2, but pretty much different. As I wanted a lossless measure comparable with H, I chose to use the entropy of the decimated CDF 9/7 integer wavelet transform in standard decomposition. Telling you what all of these words mean is the purpose of this chapter. For now it's sufficient to know that M will be much closer to the unknown compression limit (remember the Kolmogorov Complexity) than H. For residual images the MSE, its square root σ and the mean absolute error MAE are useful measures of distortion. Sometimes the signal energy $E = n^{-1}\sum x^2$ (not to be mixed with the expected value!) or its square root are used for argumentation.

$\mu = 123.9282 \quad \sigma = 45.9376 \quad H = 468235 \quad H_S = 7.145 \quad M = 273288 \quad \sqrt{E} = 132.168$

Mona Roli image statistics

In addition to the entropy it may be of interest which values standard lossless compression algorithms achieve. For that purpose I have chosen the *deflate*[1] algorithm, the PNG image format (which also uses *deflate*, but combined with a very simple prediction – differencing – of neighbouring pixels) and JPEG-LS, previously known as LOCO-I [Wei98]. Here are the results:

uncompr.	H	Deflate	PNG	JPEG-LS	M
524288 bits	468235	420432	338680	322336	273288

So to summarise, we can do better than the entropy limit because of the correlation between pixels, but even with the best lossless technique not even a factor of 2 is achieved.

Distortion

The histogram counts the occurrence of each value regardless of any inter-pixel dependencies. Similarly, the entropy, which is made up of the probabilities of the individual values, says nothing about the correlation of the dataset. Now let me pick two pixels from the image at random and simply exchange them and let me do this a number of times. Figure 2.3 illustrates the effect of this cruelty. This operation has no impact on the symbol frequencies and thus the entropy stays the same as the histogram does. So, once more we see that Shannon entropy is a bad measure for the information contained in the image. In the covariance matrix we can observe how the correlation

[1] Read about this lossless compression algorithm in Section 3 of Chapter 3. For now it is sufficient to know that it is the heart of the widespread *zip*.

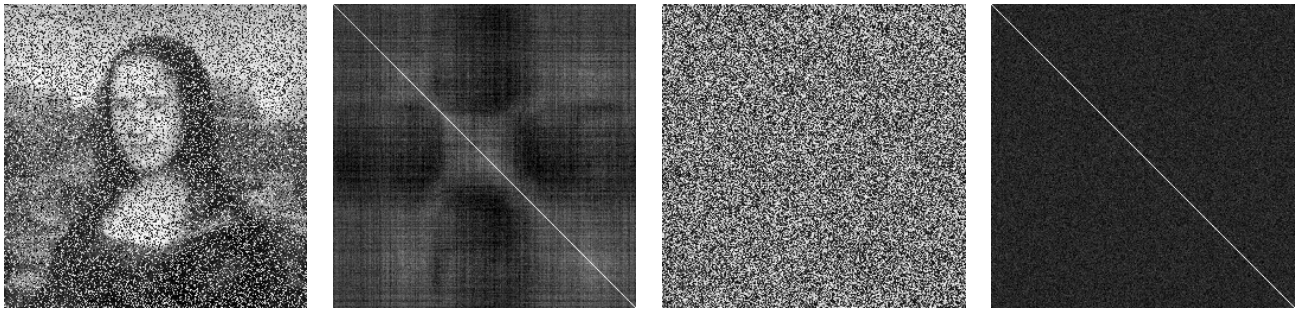


Figure 2.3: **Mona Roli distorted.** The correlation can be easily distorted without altering the histogram by randomly swapping pixels in the image. (Left) 50 percent of the pixels are displaced. Mona Roli is still recognisable. (Middle left) The covariance matrix has indications of randomness as well as for correlation. (Middle right) More than 97 percent of the pixels have been displaced. (Right) Covariance matrix shows no more correlation.

Figure 2.4: **Inverse.** (Left) Mona Roli inverse. This operation correlates each pixel with all the others, but without performing any energy compaction. (Middle left) Covariance matrix of the inverse: the data are almost perfectly correlated. (Middle right) A single value at row 8, column 17 has been changed from almost zero to zero and the inversion was undone. (Right) Residual image for the defect inverse matrix.

fades away. If the heavily distorted image is now given as input to *deflate*, a size of 58880 bytes (471040 bits), or 7.19 bit/pixel is achieved.

(De)Correlation

Ideally we would like to have an operation that perfectly decorrelates the input data. Such operations exist, as for example *matrix inversion*. In Figure 2.4 the inverse is shown, along with its covariance matrix in which the diagonal has disappeared, indicating close to perfect correlation. However, one reason why matrix inversion is not the perfect operation for data compression is that it has no energy compaction at all. The other one is that it is very susceptible to modification and therefore prohibiting any kind of quantisation.

Reduction

The goal of the next experiment is to show how very simple reduction operations can be used to reduce the amount of data by a factor 64, i.e. down to 1024 bytes and to estimate the distortion and show the consequences of such a dramatic reduction. Probably the easiest way to reduce the sheer amount is to subsample the data, that is, throw away samples and only keep a fraction. This will certainly lead to a loss of information, unless the dataset is highly oversampled. Averaging neighbouring pixels seems to be a much better way to reduce an image. In Figure 2.5 I give three useful ways to reduce the spatial resolution, these are picking a central pixel for each 8×8 block, picking

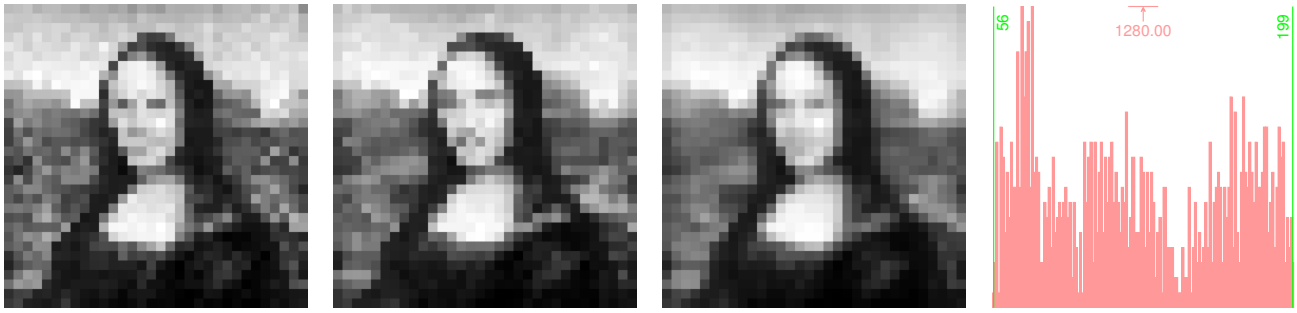


Figure 2.5: **Mona Roli reduced.** The input image brought to lower resolution in different ways by a factor 8 yields a 32×32 image. (Left) Coarse version with aliasing, where each 8×8 block is sampled only once at the centre. (Middle left) Here the median of each block is taken. (Middle right) The average of each block with its smooth appearance. (Right) Histogram of the average.

the median and averaging. Clearly, *any* single pixel of a block is a bad representative for that area, especially for astronomical images, where this method would become a lottery whether a star or some background is picked and the result can have a very different appearance (aliasing). The remedy against this is to restrict the bandwidth of the image S first by $\mathcal{F}_{2D}^{-1}\mathcal{F}_{2D}(S)\cdot M$, where M is a mask acting as a low-pass filter. This operation can also be achieved by simply smoothing the image by means of averaging. But averaging the blocks in our image is a drastic operation as well, because it creates values in the histogram that did not exist before and reduces the contrast. A way to circumvent this is to use the median as it retains the most contrast yet uses information from the whole block, even if it is not as precise as the average. On the other hand it is almost immune to outliers, which is a nice feature to have for measurements in space. The residuals of the three images are:

	subsampled	median	average
MSE/MAE =	263.2/9.88	204.0/8.98	140.7/7.65

So, for the Mona Roli the best choice in reducing the resolution is still good old averaging. Note that the histogram has now a very spiky appearance. This leads to the problem that although we reduced the number of pixels by a factor 64, the entropy needs not to follow that decrease. In our case it has, but be aware that smoothing can actually increase the sample entropy!

> Her designs were noted for combining eccentricity with simplicity and a trim neatness with flamboyant colour. In 1947 Schiaparelli's new colour, "shocking pink," was the sensation of the fashion world.
>
> —*EBm "Schiaparelli, Elsa"*

The Walsh-Hadamard Transform

In the WHT everything starts with the real-valued Hadamard matrix H, which was first constructed by Sylvester [Syl67] and later extended by Hadamard [Had93]. The

basis vectors in the square Hadamard matrix are the Walsh functions [Tho86], so the entries consist of values $+1$ or -1 only. Each row is a basis vector and because they are linearly independent we have an orthogonal system. Depending on how the Walsh functions are arranged in H we get differently ordered matrices with different characteristics. The most frequently used orders are *natural* order for the original construction and *sequency* order for the rearranged version that has frequency-distinctive properties.

Construction of the Matrix

The matrix can be constructed via a recursive formula. Starting with the 1×1 dimensional $H^1 = (1)$ the N-th order matrix H^N is constructed by copying H^{N-1} also to the right and down and filling the lower right corner with $-H^{N-1}$. Graphically:

$$H^1 = 1$$
$$H^N = \begin{pmatrix} H^{N-1} & H^{N-1} \\ H^{N-1} & -H^{N-1} \end{pmatrix}$$

In Figure 2.6 H^{256} is shown in natural and sequency order. In order to perform the transform, H is then simply multiplied with the data vector or matrix and if it is necessary to conserve the energy a scalar factor $1/\sqrt{N}$ needs to be considered in the 1D transform. The result of the 2D-transform of Mona Roli is shown in Figure 2.7 along with the covariance matrix of the transformed image. This has some similarities with the miscarried experiment of the inverse, but altering a single pixel has virtually no effect (not shown) any more. This property, which can be ascribed to the fact that combined symbols are less susceptible to distortion, is why this transform is useful in digital holography [Yar03].

Sequency Order

By sorting the basis vectors according to their spatial frequency, that is, how often the sign changes on the coordinates, we derive the sequency ordered Walsh-Hadamard matrix and the transform will usually lead to a concentration of coefficients with higher energies in the top-left corner. The result can now be interpreted much like the modulus of a Fourier transform. From the first coefficient on, the transform corresponds to increasing spatial frequency and therefore smaller image details. For the first coefficient (top-left in a 2D transform) the first column of H and the first row, both containing only ones, were involved in calculation, as well as the entire input image. Thus, if a dynamic range conserving normalisation has been made, the first coefficient is the mean of the original image! As each coefficient is the weighted sum of a row and a column vector of H with the input this brings us to another way of looking at the transform.

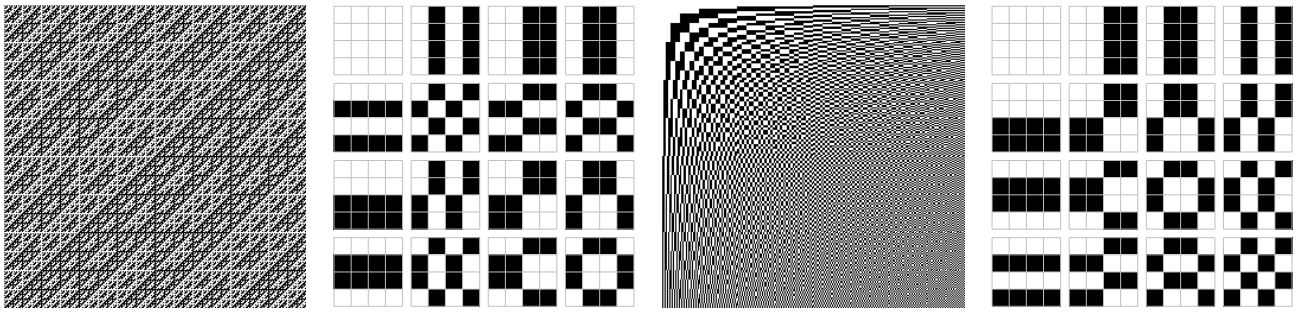


Figure 2.6: **Hadamard matrices and basis images.** (Left) The original Hadamard matrix in *natural order*. Black is -1 and white is 1. (Middle left) Basis images for a 4×4 kernel derived in natural order. (Middle right) The matrix brought to *sequency order*. (Right) 4×4 basis images derived in sequency order.

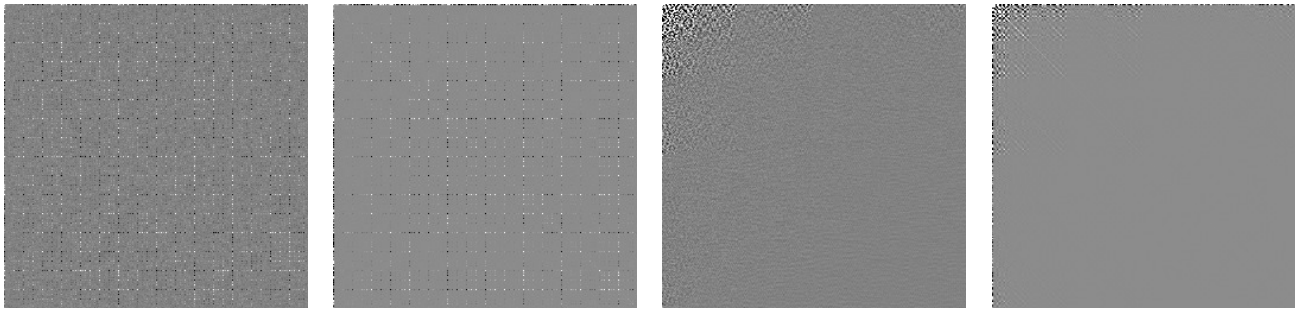


Figure 2.7: **Mona Roli in the WHT.** (Left) Transform of the image on the naturally ordered Hadamard basis. (Middle left) Covariance matrix. (Middle right) Transform through the sequency ordered basis. (Right) Covariance matrix.

Another Take

The operations in a matrix multiplication can be taken apart and set together in a different way to arrive at the same result. If we multiply a column vector with a basis vector we get a kernel image. We do that for all permutations and get 16 kernel images for a 4×4 matrix, that is, for every pixel in our input image we have a kernel image. In the Hadamard matrix the first row and the first column are all ones and the resulting kernel image is thus – all ones. If we compute the cross correlation of the image with that kernel, we get the average. By correlation of our image with each of the 16 kernel images we get the transform coefficients. Figure 2.6 shows these kernel images. Note that they are given for a 4×4 kernel, and not for a 256×256 kernel, as we would need it for Mona Roli. But two important things become visible: each coefficient is calculated taking the whole image into account, and that the kernel images which are derived from the sequency ordered bases are responsible for the frequency ordering.

Just as explained before, 1D transform works like $T_{1D} = HA$ and $A = HT_{1\{mathrmD}$. This will transform the columns independently. To achieve a 2D transform the rows need to be 1D transformed as well, so the transform is applied another time with the transpose of H and we get $T_{2D} = HAH^T$ and $A = HTH^T$. For the sake of energy conservation a factor $1/||H|| = 1/\sqrt{N}$ should be included in the matrix multiplication. The Hadamard matrix in sequency order has two advantages: it is symmetric and therefore $H = H^T$ and the basis vectors are sorted with increasing frequency, therefore separating spatial frequencies. We remember that the inverse of an orthonormal matrix is its transpose. So, the equations become more simple for the sequency ordered Hadamard matrix: $T_{2D} = HAH$ and $A = HT_{2D}H$.

WHT and Compression

The WHT if performed in integer by omitting the normalisation almost doubles the entropy of Mona Roli to 789378 bits. This is because the dynamic range is increased to $R_{2D} = N^2R$, respectively increased by $2\,\mathrm{ld}N$ bits. Of course, if sequency order is used, the same value is achieved, because the values of the coefficients are the same, only their location in the new basis is different. As with many transforms, the strength lies within quantisation. In natural order the coefficients must be treated equally, but in sequency order unnecessary details can be filtered out. To illustrate its reaction to such action, Figure 2.9 compares four different approaches, but each one bringing the entropy down to 1/64th of its original value, that is 8192 bits. Quantisation in the transform domain can be done in different ways (also see Figure 2.8):

- Windowing: a section (probably around the top left corner) is left untouched and the rest is set to zero. This is a low-pass filter.

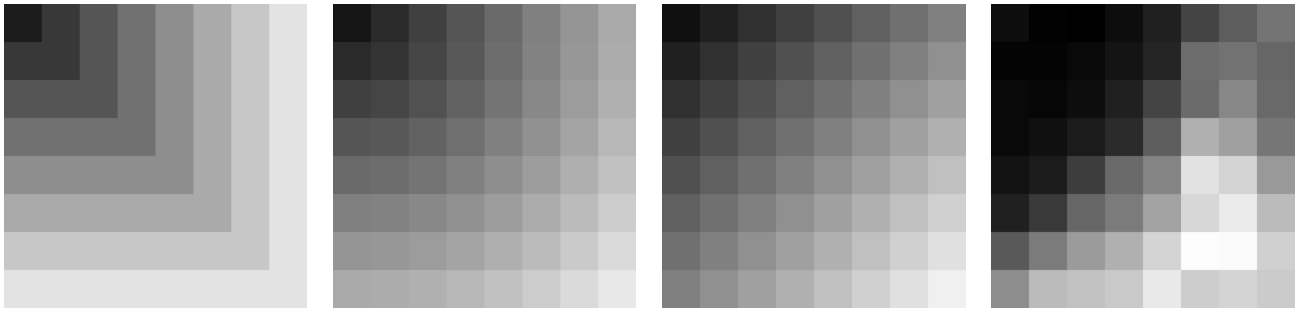


Figure 2.8: **More intricate quantisation tables.** (Left) Quantisation zones are cubes. (Middle left) Radial gradient. (Middle right) Linear gradient. (Right) JPEG luminance table.

- Hard thresholding: every value $|x| < k$ is set to zero. k is chosen such that the data rate is achieved.
- Iterative thresholding: the largest hard threshold k is chosen iteratively to give an error $< \epsilon$. This quickly increases the complexity and has thus its largest potential for blocked transforms.
- Rounding: the values are rounded to an integer grid which is coarse enough to satisfy the data rate. A cheap way to do this is by scalar multiplication with integer truncation.
- Radial/Diagonal/Quadratic gradient: the rounding grid is adapted to the distance from the top-left corner.
- Weight Map: the coefficient matrix is component-wise multiplied with a weighting matrix.

Whatever operation is used, note that it is not wise to modify the DC coefficient, because this alters the mean of the whole image. It is also comprehensible that scientists generally develop allergic reactions to quantisation in the frequency domain, because this is essentially a nonlinear operation. Uniform quantisation in the time/space domain is linear, because a large value is rounded as a small one would. In the frequency domain this operation emphasises the large coefficients that correspond to the features with more signal energy and thus affects the photometry.

	hard thr.	uniform quant.	windowing	gradient
MSE/MAE	146.5/8.05	122.1/8.36	134.38/8.39	142.65/8.83

The numerical results for Mona Roli are a little bit against perception. Winner is the WHT with everything equally rounded, but also the hard threshold performed better than the gradient methods, which require the sequency ordered matrix. These did slightly worse than a simple binning would have done. This is due to all the zeros that were still counted in the entropy calculation.

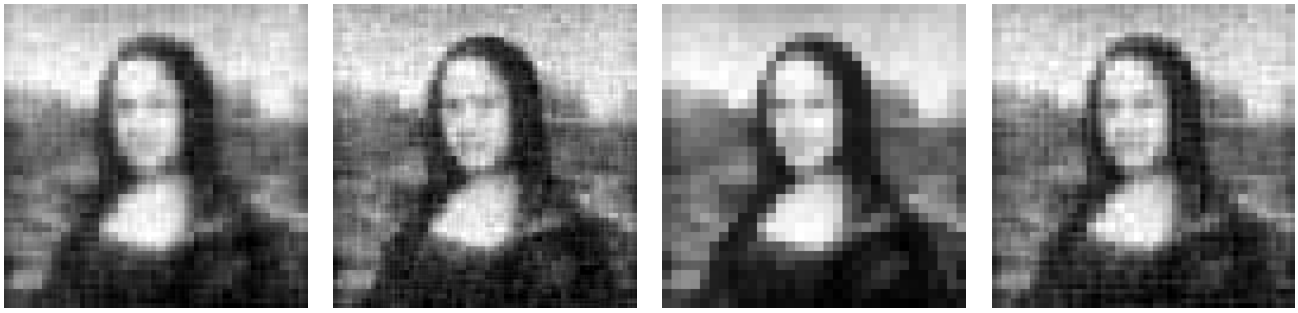


Figure 2.9: **Quantisations with the WHT.** (Left) Quantisation achieved with a hard threshold. (Middle left) Rounding all coefficients equally. (Middle right) A combination of windowing the 32x32 corner and rounding. (Right) A linear rounding gradient emerging from the corner.

Compaction and Entropy

In case of the WHT in sequency order or any other energy-compacting transform we can take advantage of the fact that the few large coefficients that are related to the image content are concentrated, whereas the coefficients related to the noise fill the rest of the coefficient matrix. In other words, once the quantisation has removed the small noise coefficients, what is left are mostly the large coefficients in the top-left corner. Thus, the lots of zeros in the lower right part don't need to be considered in compression and in entropy calculation. If we remove a symbol "0" from a dataset, what impact has this on the entropy H'? To answer this question we work a little bit with the entropy definition. k is the number of different symbols x, each one appearing $c(x)$ times in a set of n values. Note that for H the symbol entropy is taken.

$$
\begin{aligned}
H &= -\sum_{x=0}^{k} p(x)\mathrm{ld}\, p(x) = -\sum_{x=1}^{k} p(x)\mathrm{ld}\, p(x) - p(0)\mathrm{ld}\, p(0) \\
p(x) &= \frac{c(x)}{n}, \quad p'(x) = \frac{c(x)}{n-c(0)} = p(x)\frac{n}{n-c(0)} \\
H' &= -\sum_{x=1}^{k} p'(x)\mathrm{ld}\, p'(x) = -\sum_{x=1}^{k} p(x)\frac{n}{n-c(0)}\mathrm{ld}\left(p(x)\frac{n}{n-c(0)}\right) = \\
&= -\frac{n}{n-c(0)}\sum_{x=1}^{k}\left(p(x)\mathrm{ld}\, p(x) + p(x)\mathrm{ld}\frac{n}{n-c(0)}\right) = \\
&= -\frac{n}{n-c(0)}\sum_{x=1}^{k} p(x)\mathrm{ld}\, p(x) - \frac{n}{n-c(0)}\mathrm{ld}\frac{n}{n-c(0)}\cdot\sum_{x=1}^{k} p(x) = \\
&= \frac{n}{n-c(0)}(H + p(0)\mathrm{ld}\, p(0)) - \frac{n}{n-c(0)}\mathrm{ld}\frac{n}{n-c(0)}\cdot(1-p(0))
\end{aligned}
$$

An example: the total entropy of the sequence "1230000000" is 13.57 bits (1.357 for a symbol), whereas the entropy of "123" is 4.75 bits (1.583). From the equations we see that the biggest saving is not due to the fact that all 0 are thrown away and no longer contribute to the total entropy, but from the overall reduction of n.

> Version of the bicycle reinvented in the 1860s by the Michaux family of Paris. Its iron and wood construction and lack of springs earned it the nickname boneshaker.
> *—EBm "velocipede"*

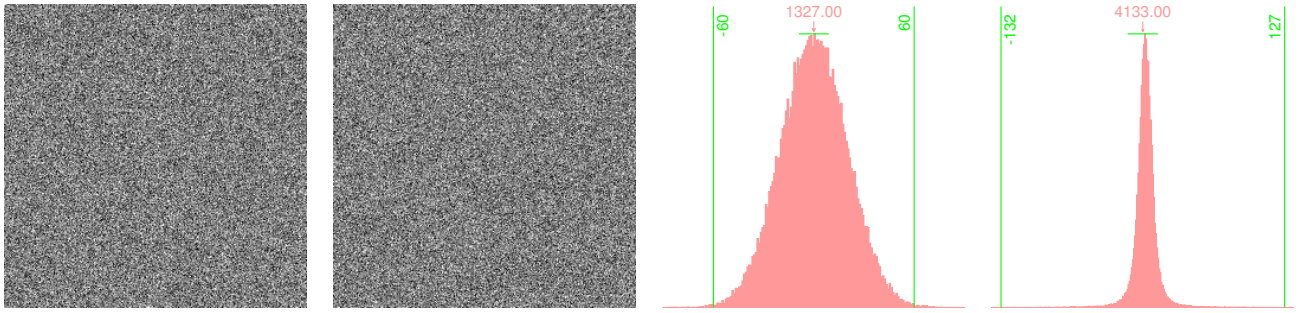


Figure 2.10: **Noise in the WHT.** (Left) Gaussian input noise with μ=0, σ=20 is constructed. (Middle left) The Gaussian noise transformed in the sequency-ordered WHT. (Middle right) Histogram of the transformed Gaussian noise, still with $\sigma = 20$. (Right) Histogram of the WHT transformed Mona Roli image (μ=0.57, σ=132.17). The cuts are as in the 2D transformed image in Figure 2.7.

Noise in the WHT

Figure 2.10 shows the results from a little experiment. Gaussian noise was transformed to see what its transform coefficients will look like. Recall that the Hadamard matrix contains only ones and minus ones. In combination with matrix multiplication the operations in the transform become simple additions and subtractions of the original input values. From the central limit theorem we know the effect of this: the input distribution will mutate into a Gaussian. However, if there are correlations in the dataset, they will emphasise the respective coefficients and the resulting shape resembles more a Laplacian.

Its Use for the Astronomer

The WHT is neither the best analysis tool, nor very good for compression. However, it may be the only solution if the computational resources are limited, because it can be implemented using additions only. Apart from that it is used in radio astronomy to implement phase shifting for many antennae [Tho86] and in Hadamard spectroscopy [Nel70]. In natural order it has been used as an error-correcting code, as in the Mariner and Voyager missions [Eva03]. In sequency order it can serve as a poor-man's replacement of the DCT. In the H.264/AVC video compression standard the WHT is used to decorrelate the DC coefficients of a previously blocked DCT [Sal07]. Among the space missions that were using the WHT for decorrelation is Cassini [Rob04].

The Haar Transform

In 1910 Alfréd Haar constructed the Haar basis [Haa10], which is the most basic wavelet transform. It is often said to be the most simple one, which may be true for the multiscale approach (see below), but construction from the basis matrix is not as easy. In contrast to the Hadamard transform, the Haar transform is still widely used today. By way of the Haar transform I will establish a connection between linear transforms and wavelet-based decompositions in general. This is possible, because the Haar basis can also be constructed in another way, using a scaling function and a wavelet function.

Construction of the Matrix

We construct the basis of our transform matrix from top to bottom, starting with the first line, which is all ones, because it shall give the DC coefficient, which is the mean of the input if properly normalised. We will see that the other lines will play a

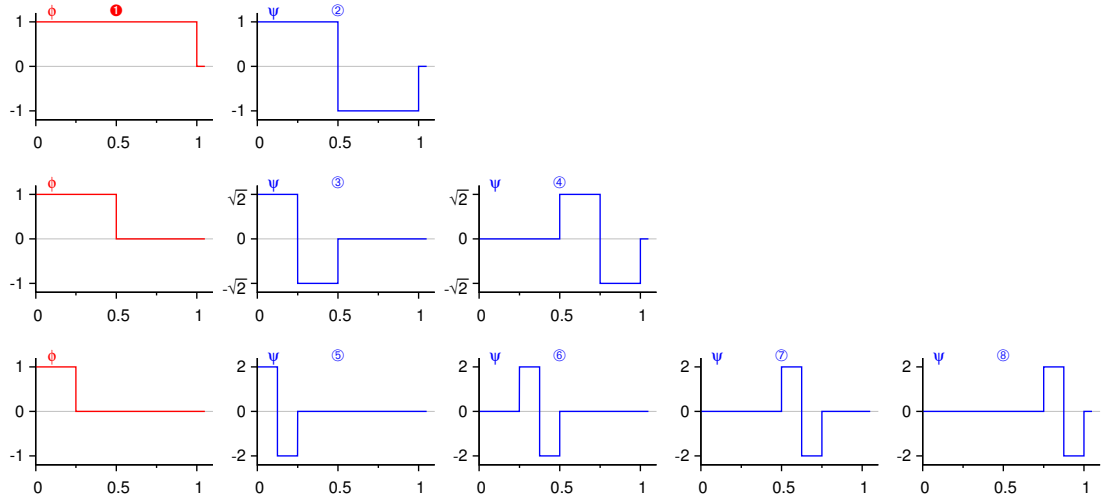


Figure 2.11: **Haar wavelet construction.** ❶ Scaling function, ② Mother wavelet and how the following scales are made up. The encircled numbers mark the basis function (line number) of the Haar matrix. Note that position 1 given on the abscissa corresponds to the length N of the discrete input dataset.

rôle as differences at different scales. Figure 2.11 is meant to be helpful for studying this paragraph. Our second line is the mother wavelet function at full size. In case of the Haar basis this is the step function

$$\psi = \begin{cases} 1, & 0 \le t < 0.5 \\ -1, & 0.5 \le t < 1 \end{cases} .$$

Each line will now contain a wavelet which is derived from the mother wavelet by scaling and shifting. So line three has the wavelet function at half size. At this scale we have two possibilities where to insert the wavelet, on the left side or the right side. We start on the left and leave the rest with zeros, and in the next line we start with zeros and fill the second half. The next scale is again cut in half[1] and we continue to paste the scaled and shifted wavelet into the basis. Here comes the formal definition, where the wavelet function is usually denoted with ψ. In the continuous case we use real-valued parameters for scaling $p > 0$ and shift q. So we write

$$\psi_{p,q}(t) = \frac{1}{\sqrt{p}} \psi\left(\frac{t-q}{p}\right) .$$

For a discrete transform it is necessary to choose discrete values for p and q in the scaling p^{-s} and shift kqp^{-s} at scale s with shift index k. Inserting this in the equation above leads to

$$\psi_{s,k}(t) = \frac{1}{\sqrt{p^{-s}}} \psi\left(\frac{t - kqp^{-s}}{p^{-s}}\right) = \sqrt{p^s}\psi(tp^s - kq) .$$

In here the fixed scale q is set according to the dimension of the basis vector, for example $q = N = 256$ for the 256×256 Mona Roli image. We can now construct the transform matrix:

[1] We use a dyadic scaling, but in fact the scaling factor should be adapted to the wavelet function.

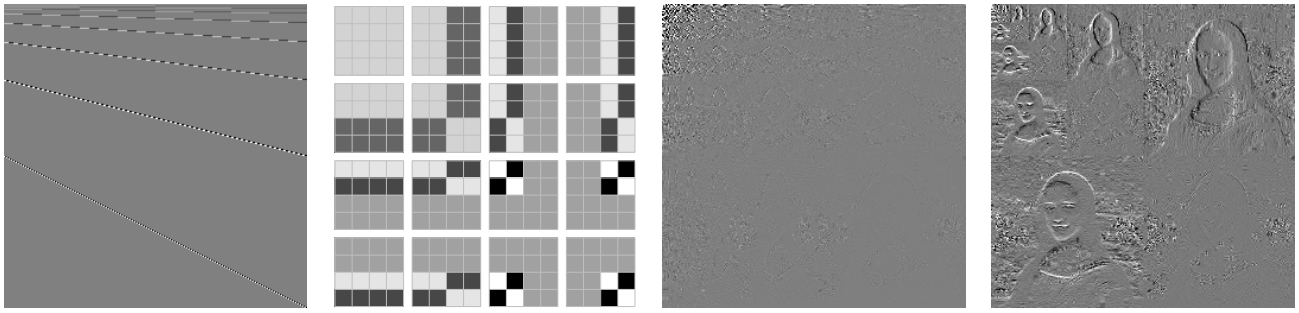


Figure 2.12: **Matrices of the Haar transform.** (Left) The Haar matrix for a 256×256 dataset. Black is −1 and white is 1. (Middle left) Basis images for a 4×4 kernel. (Middle right) Mona Roli Haar transformed to the standard basis. (Right) Pyramidal subband decomposition.

$$W = \frac{1}{\sqrt{N}} \begin{pmatrix} \phi(t) & = & 1 & 1 & 1 & 1 & \dots \\ \psi_{0,0}(t) & = & 1 & 1 & 1 & 1 & \dots \\ \psi_{1,0}(t) & = & 1 & 1 & 1 & 1 & \dots \\ \psi_{1,1}(t) & = & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & \dots \\ \psi_{2,0}(t) & = & 0 & 0 & 0 & 0 & \dots \\ \vdots & & & & & & \\ \psi_{S,K}(t) & = & 0 & 0 & 0 & 0 & \dots \end{pmatrix}$$

The Haar transform is said to be simple because its mother wavelet is a step function. A little bit further down the CDF 9/7 wavelet can be found. Its construction is more or less the same but with a different mother wavelet and scaling function.

A Different Approach

Now that we have gone through the maths here is the relief: it's not necessary to use it. Look at the basis matrix or at the kernel image in Figure 2.12. What this transform essentially does is to compute averages of different sizes of the image and their differences. A sequence of N numbers (N should be even) can be represented in $N/2$ averages of pairs and $N/2$ differences. This can be iterated on the averages until 1 average and $N-1$ differences are left. An example:

Input	1	3	3	-1	1	2	3	2
1. avg/diff	2	1	1.5	2.5	-1	2	-0.5	0.5
2. avg/diff	1.5	2	0.5	-0.5	-1	2	-0.5	0.5
3. avg/diff	1.75	-0.25	0.5	-0.5	-1	2	-0.5	0.5

The result is the Haar transformed original sequence! In the 2D case this averaging scheme can be applied to all rows and then to all columns and the result is a 2D transformed image. In this case it is called *standard decomposition*. It is identical with the result obtained via matrix multiplication. However, we could decide to only scale down once in each dimension. Then we have a dataset where in the top left is a 2× averaged image of the original, to the right and below are horizontal and vertical differences (*detail*) and in the bottom right quarter are the diagonal differences. The next iteration would then no longer work on the whole dataset, but continue with the top left quarter and so on. This is called *pyramidal decomposition* and it is also shown in Figure 2.12. Several advantages arise from this kind of decomposition. Above all, it saves CPU cycles, quantisation is easier and better encoding strategies can be used. On the other hand this is no longer a linear transform in the strict sense and there is

Figure 2.13: **Mona Roli Haar compressed.** (Left) Standard decomposition with a hard threshold. (Middle left) Standard decomposition equally rounded. (Middle right) Pyramidal decomposition with scale-dependent rounding (Right) Pyramidal decomposition with a linear quantisation gradient.

less decorrelation achieved. Several other decomposition strategies exist as well, but I conclude this subsection with the message that this multiscale approach is the link towards filter banks.

What about Mona Roli?

In analogy with the Hadamard transform I tried again to find an optimal quantisation for Mona Roli. The resulting pictures are presented in Figure 2.13. In terms of numbers we have:

	hard thr.	uniform quant.	scale dep.	gradient
MSE/MAE	114.53/7.64	102.14/7.51	127.70/8.35	123.51/8.06

> The characteristic *Lüfterl* ("Vienna air"), a light breeze blowing from the northwest and west, provides relief on hot summer evenings.
>
> —*EBM "Vienna"*

The Fourier Transform

Baron Joseph Fourier (1768–1830), a French egyptologist and gifted mathematician, introduced series with sines and cosines for the solution of differential equations and extended this concept into the so-called Fourier integral. Today, Fourier analysis is a major branch in mathematics. To the astronomer it is one of the most important analysis tools. Compared to the transforms that we have treated so far, the Fourier transform has some major differences. First of all, it is complex-valued. Even if only real-valued data are provided, the result will be complex. Secondly, the transcendent functions used in the transform kernel lead to new effects, the most striking one being that the data in the transform domain are no longer localised, i.e. you will not be able to identify a single feature from the spatial domain as a distinct symptom in the frequency domain and vice versa. These features are comprised in the basic theorems found in any authoritative introduction about the Fourier transform, such as [Bra99a]. Of these I will point out the convolution theorem, which has been mentioned earlier and the shift theorem, which states that shifting the input does not affect the amplitude but the phase.

The Fourier transform $\mathcal{F}$ of a continuous function $f(t)$ and its inverse $\mathcal{F}^{-1}$ are

$$\mathcal{F} = \hat{f}(\nu) = \int_{-\infty}^{\infty} f(t)e^{-i2\pi\nu t}dt \qquad \mathcal{F}^{-1} = f(t) = \int_{-\infty}^{\infty} \hat{f}(\nu)e^{-i2\pi\nu t}d\nu$$

The definitions above are seen in different notations and forms, essentially differing whether the factor 2π is found inside the integral or already put as $1/2\pi$ or $1/\sqrt{2\pi}$ in front.

Modulus and Power Spectrum

A Fourier transform of time or space to the frequency domain is a very valuable analysis tool. Interpretation of the frequency spectrum is best done with the modulus $|\mathcal{F}(\nu)|$, because amplitude (the real part) and phase (the imaginary part) are not very compelling to the human eye. However, it is customary to use $|\mathcal{F}(\nu)|^2$ instead and refer to it as the *power spectrum*.

The Discrete Fourier Transform

The discrete Fourier transform (DFT) is the basis for dealing with sampled data, i.e. all kinds of signals. In the simplest case we assume that the data are uniformly sampled. When a discrete Fourier transform is made of real-valued input, it is noticeable that the original N points will lead to real and imaginary parts that also have N points each, but of which (almost) half of them are complex conjugates, i.e. simply mirrored with opposite sign and therefore normally redundant. More on how to interpret the coefficients is found a little further down when the question will be raised, *"What are these coordinates?"*

The DFT of a discrete function $f(k)$ and its inverse are defined by

$$\mathcal{F} = \hat{f}(u) = \frac{1}{N}\sum_{k=0}^{N-1} f(k)e^{-i2\pi\frac{uk}{N}} \qquad \mathcal{F}^{-1} = f(k) = \sum_{u=0}^{N-1} \hat{f}(u)e^{-i2\pi\frac{uk}{N}}$$

The complex function $\hat{f}(u)$ can be split into a real and imaginary part (because $e^{i\Theta} = \cos\Theta + i\sin\Theta$), which will then lead to a form that is advantageous for programming:

$$\begin{aligned} Re(\hat{f}(u)) &= \frac{1}{N}\sum_{k=0}^{N-1} f(k)\cos(2\pi uk) \\ Im(\hat{f}(u)) &= -\frac{1}{N}\sum_{k=0}^{N-1} f(k)\sin(2\pi uk) \end{aligned}$$

Another way is to write it using modulus and argument:

$$\hat{f}(u) = |\hat{f}(u)|e^{i\arg\hat{f}(u)}$$

with $|\hat{f}(u)|^2$ being the power spectrum and $\Theta(u) = \arg\hat{f}(u)$ being the phase.

As the Fourier transform is separable, the 2D DFT can be written as two serial 1D transforms. Here are minimalist DFT functions for real input in AWK for playing around (In here $\mathcal{F}$ is normalised by $1/N$):

Listing 2.1: DFT and IDFT

```
function dft (data, ftrans, ftransi, lines) {
  for (u=0; u < lines; u++)
  {
    sumr = 0; sumi = 0;
    for (x=0; x < lines; x++)
    {
      t = 2*3.1415926536 * u*x / lines;
      sumr += data[x] * cos(t);
      sumi += data[x] * -sin(t);
    }
    ftrans[u] = sumr / lines;
    ftransi[u] = sumi / lines;
  }
}

function idft (rdata, idata, synth, lines) {
  for (u=0; u < lines; u++)
  {
    sumr = 0; sumi = 0;
    for (x=0; x < lines; x++)
    {
      t = 2*3.1415926536 * u*x / lines;
      sumr += rdata[x] * cos(t);
      sumi += idata[x] * -sin(t);
    }
    synth[u] = sumr + sumi;
  }
}
```

Fourier Basis

Just as we have done with all transforms so far, we will now construct the transform matrix W from scratch. Basically, it amounts to

$$w = e\frac{-2\pi i}{N} = \cos\frac{-2\pi}{N} + i\sin\frac{-2\pi}{N}$$

The column and row indices running from 0 to $N-1$ are used as powers p, q in w^{pq} to construct the DFT matrix W, which is then used in the same way as in the transforms before – through matrix multiplication.

$$W = \begin{pmatrix} w^{0\cdot 0} & w^{1\cdot 0} & \dots & w^{(N-1)\cdot 0} \\ w^{0\cdot 1} & w^{1\cdot 1} & \dots & w^{(N-1)\cdot 1} \\ \vdots & & & \\ w^{0\cdot (N-1)} & w^{1\cdot (N-1)} & \dots & w^{(N-1)\cdot (N-1)} \end{pmatrix}$$

Note that the normalisation ($1/\sqrt{N}$ for a symmetric one) needs to be taken into account as well. For the backward transform, w^{-pq} is used. The matrices and kernel images are shown in Figure 2.14.

Swapping

If we transform Mona Roli with the DFT, we find that the large coefficients are not concentrated in the top-left corner, but they show up in every corner (cf. Figure 2.15).

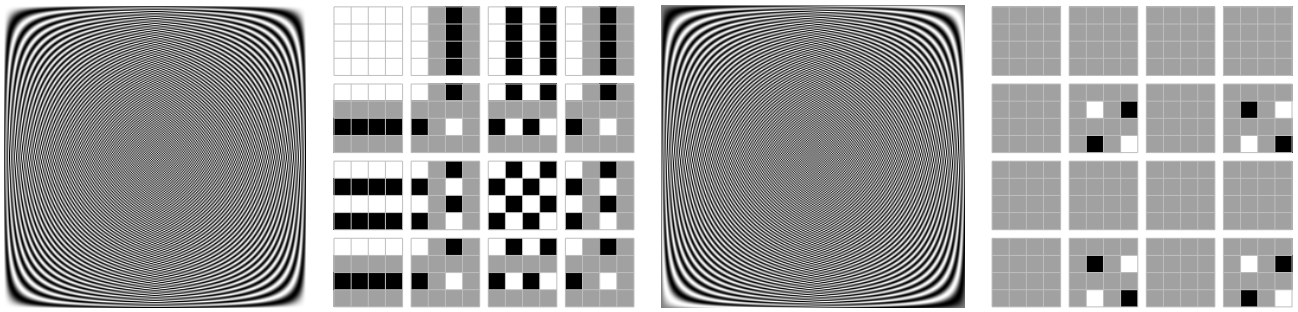


Figure 2.14: **Matrices of the DFT.** (Left) Real part of the Fourier basis. (Middle left) 4×4 Kernel for the real part. (Middle right) Imaginary Fourier basis. (Right) Kernel images for the imaginary part.

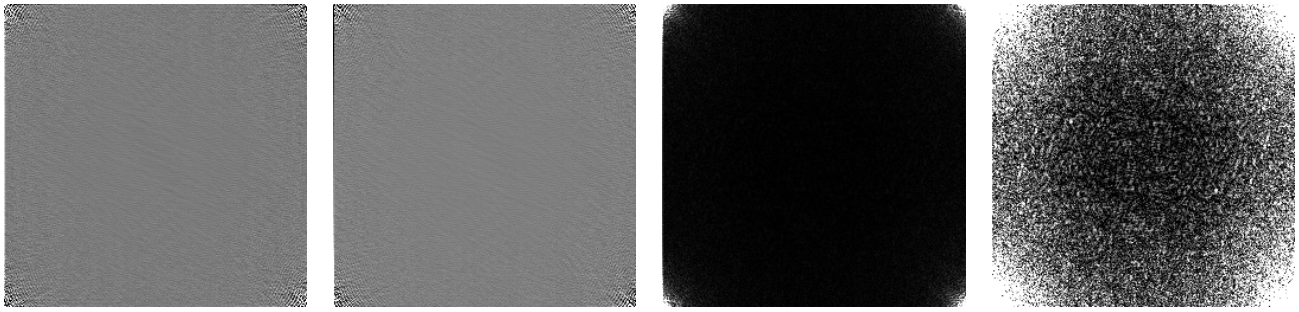


Figure 2.15: **Mona Roli in the DFT.** (Left) Real part of the transformed image. (Middle left) Imaginary part. (Middle right) Modulus and (Right) Modulus with enhanced contrast.

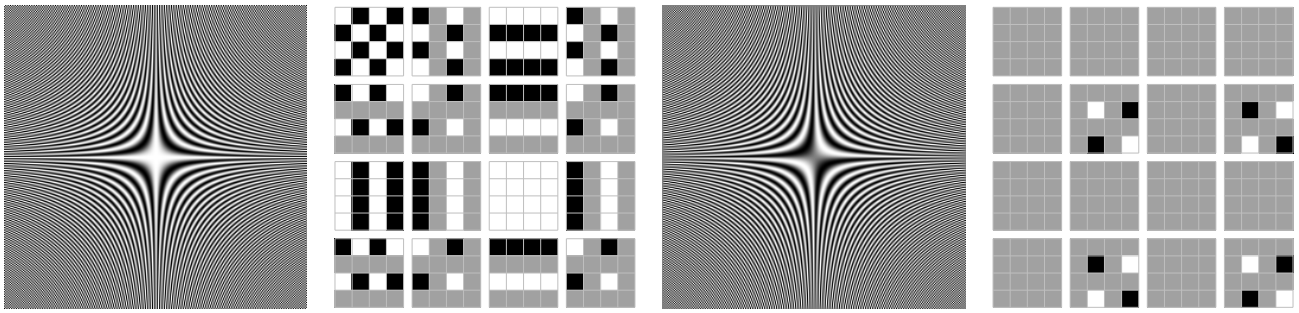


Figure 2.16: **Matrices of the swapped DFT.** (Left) Real part of the swapped Fourier basis. (Middle left) 4×4 Kernel for the swapped real part. (Middle right) Swapped imaginary Fourier basis. (Right) Kernel images for the swapped imaginary part.

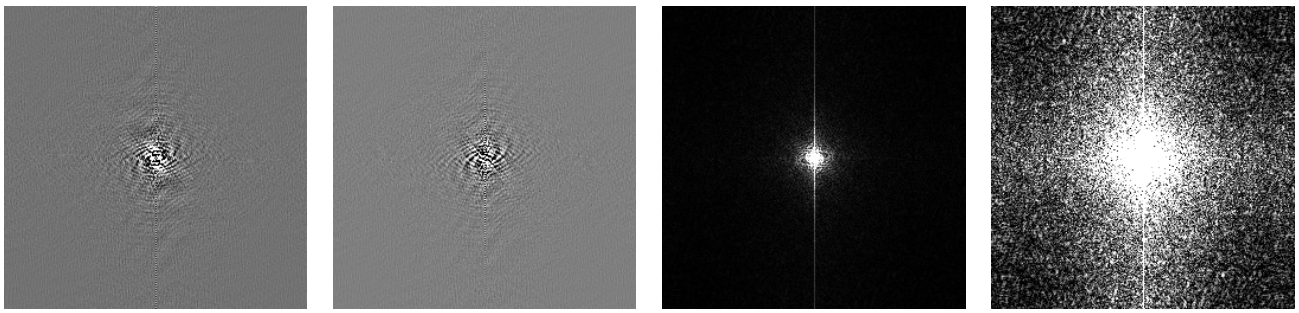


Figure 2.17: **Mona Roli in the swapped DFT.** (Left) Real part of the swapped transform. (Middle left) Imaginary part of the swapped transform. (Middle right) Modulus of the swapped transform. (Right) Modulus with enhanced contrast.

Figure 2.18: **DFT as a compression.** (Left) Mona Roli compressed with the DFT. (Middle left) A circular filter window with a radius of 32 pixels. (Middle right) Low-pass component (modulus). (Right) High-pass component (modulus).

Since this is not very appealing to the human eye, we swap the quarters of the image so that the interesting low frequencies are now concentrated in the middle. This can be achieved in two ways, either by swapping the result or by swapping the basis. The latter is better for complexity reasons and the swapped matrix is shown in Figure 2.16. How Mona Roli looks in the swapped DFT is shown in Figure 2.17.

Compression in the DFT

Now the question is whether the features of the DFT can be used for data compression. Similar to the transforms we have discussed so far, we want to see the effects of quantisation and filtering. The latter is very nice to achieve in the DFT and in fact high or low-pass filtering is one of the major applications of this transform. Figure 2.18 shows the result of uniform quantisation to a factor 64 in entropy and a filtering in the Fourier domain. Note that the used circular filter has a radius of 32 pixels and thus the low-pass image is only reduced by a factor of around 10 in entropy.

The result in numbers for the uniformly quantised coefficients is MSE/MAE = 83.20/6.93, which is actually an excellent performance compared with what we had so far. However, there are many inconveniences with the DFT for use in compression which are overcome by the DCT (see below).

What are these coordinates?

Fourier analysis is a very powerful tool, however, there is this mind-boggling thing about the Fourier coordinates. Assume you have made a transform of N uniformly spaced real values, indexed from 0 to $N-1$. The first real coefficient r_0 corresponds to the mean. The next one is r_1 with a frequency of $1/1$,[1] r_2 is with $1/2$ and so on until the one leading the second half $r_{N/2}$, which corresponds to the highest frequency $1/(N/2)$. Coefficients r_0 and $r_{N/2}$ are special in that they have no imaginary part. In that sense for real-valued input it is only necessary to keep N DFT coefficients, these are the first $N/2+2$ of the real part and $N/2-2$ from the imaginary part (omitting i_0 and $i_{N/2}$). For complex input we have $2N$ input values (counting real and imaginary part separately) and also all $2N$ transform coefficients are needed in this case.

What happens to the units?

A sine wave which runs through 2 full periods on an interval of 256 points, i.e. it has a spatial frequency of 2 and a period of 128, will create a spike of the size one half the amplitude times N in the modulus of the third transform coefficient $m_2 = \sqrt{r_2^2 + i_2^2}$. Note that if the wave for that frequency does not run over the whole dataset, but only over a fraction of it, the amplitude will also be reduced to that fraction.

[1] A full period over the dataset – so the biggest "feature" to be captured is $N/2$ in size.

What happens to the noise?

Gaussian noise stays the same in orthonormal transforms, all other distributions will be modified. The inverse transform can be used to generate white Gaussian noise from a uniform distribution. The resulting noise is Gaussian with $\sigma = \sqrt{n/12}$ if both, the real and the imaginary part are uniformly distributed on $[0, 1]$. If only the real part is available and the imaginary part is set to 0, then σ must be corrected by a factor $1/\sqrt{2}$. Obviously, this is a consequence from the central limit theorem in Chapter 1.

DCT and DST

The Discrete Cosine Transform is the most widespread transform, used in the JPEG [Pen93] and MPEG standards [Wat01] and also found in modified form in MP3, AAC [Bra99b] and virtually every other lossy audio codec. It has very good energy compaction for correlated datasets. In general, the cosine transform agrees with the Fourier transform for an even[1] function (*cos* itself is an even function). The similarity between the basis matrices suggests that the DCT is the top-left quarter from the DFT and the choice for this quarter is that it has the most resemblance with the ordered WHT. Its counterpart, the sine transform, is bad for correlated data, but has good energy compaction capability for small correlations. A commonly used definition for the DCT is DCT-II:

$$\mathrm{DCT}(k) = \sqrt{\frac{1}{2N}} \sum_{n=0}^{N} f(n) cos\left(\frac{(n+1/2)k\pi}{N}\right) \quad .$$

The basis matrices of the orthogonal DCT and DST with the respective coefficients $c_{p,q}$ and $s_{p,q}$ are:

$$\begin{aligned} c_{p,q} &= \sqrt{\frac{1}{N}} cos\frac{(q+1/2)p\pi}{N} & p = 0, q = 0..N-1 \\ c_{p,q} &= \sqrt{\frac{2}{N}} cos\frac{(q+1/2)p\pi}{N} & p = 1..N, q = 0..N-1 \\ s_{p,q} &= \sqrt{\frac{2}{N+1}} sin\frac{(q+1)(p+1)\pi}{2N} & p, q = 0..N-1 \quad . \end{aligned}$$

In Figure 2.19 the bases and kernel images are shown. Do not miss to compare them with Figure 2.14. There is no doubt[2] that Mona Roli is a correlated dataset and

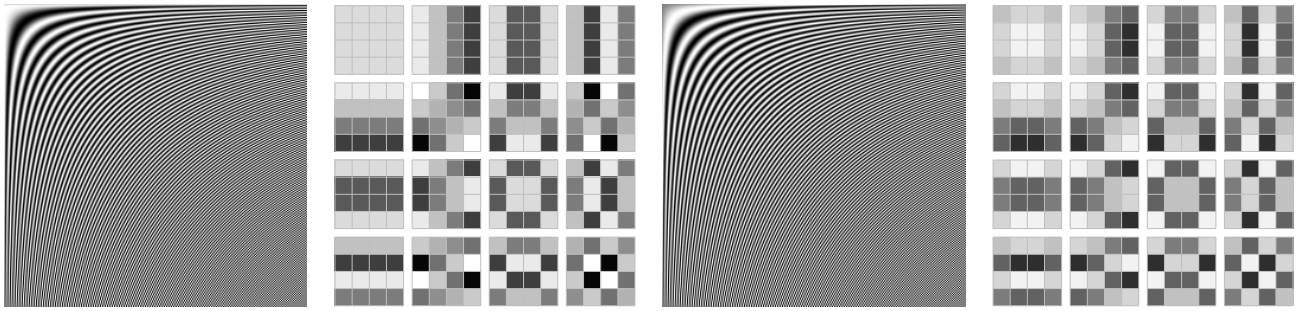


Figure 2.19: **Matrices of the DCT and DST.** (Left) The orthonormal DCT matrix for a 256×256 dataset. (Middle left) Basis images for a 4×4 kernel. (Middle right) DST matrix. (Right) DST basis images.

[1] $f(x) = f(-x)$

[2] There may be deviating views.

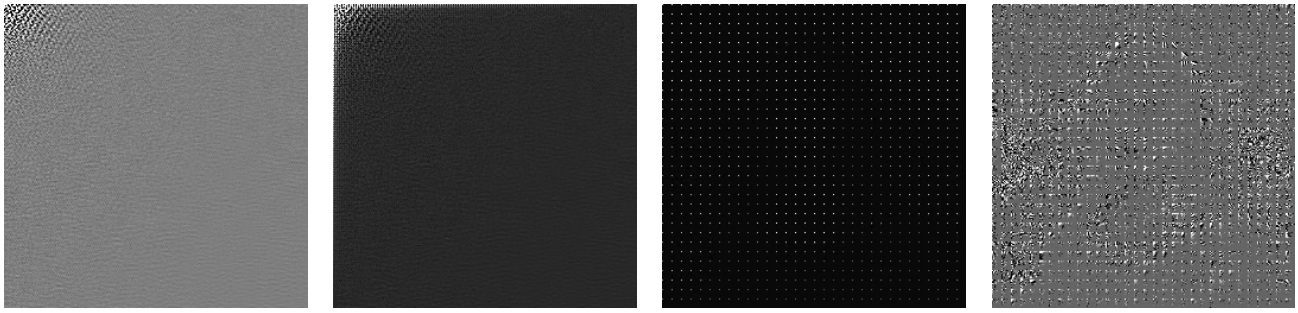


Figure 2.20: **DCT and DST in action.** (Left) Mona Roli DCT transformed. (Middle left) Mona Roli DST transformed. (Middle right) Mona Roli in the 8×8 blocked DCT. (Right) Same with higher contrast.

Figure 2.21: **DCT, DST and quantisation.** (Left) DST evenly quantised. (Middle left) DCT with hard threshold. (Middle right) rounded DCT. (Right) DCT with JPEG luminance quantisation table.

in Figure 2.20 we observe energy compaction as expected. The transform coefficients of DCT and DST have the same μ and σ, but the MAE of the DCT is smaller, which indicates better compaction. This is also confirmed with the impression we get from the quantised Mona Roli in Figure 2.21 and the numbers support this finding as well.

	DST uniform	DCT hard thr.	DCT uniform	DCT JPEG
MSE/MAE	131.64/8.67	79.58/6.52	73.83/6.50	81.85/6.94

Blocking

From these results we understand why the DCT is so popular in signal processing. However, there is still this problem with algorithmic complexity. One way to reduce the number of operations that are involved in matrix multiplication is to decompose the transform matrix W by factorisation and reduce the complexity from N^2 to $N \operatorname{ld} N$ the way that the Cooley-Tukey [Coo65] scheme does. The Cooley-Tukey algorithm divides the dataset in two, takes the separate transforms and combines them. In case of a 1024×1024 image, the FFT reduces the $2\,(\text{dimensions}) \times 1024\,(\text{lines}) \times 1024^2 = 2\,\text{Gi}$ operations to $2 \times 1024 \times 1024 \operatorname{ld} 1024 = 20\,\text{Mi}$ operations, but for applications in digital imaging – including astronomical observation – this can still be too much. Obviously the largest savings in complexity can be made by reducing N. By segmenting the image into smaller pieces – blocks – a reduction of the complexity is achieved at the cost of low frequency correlation. The tradeoff made in the JPEG standard and which has subsequently been used in blocking transforms is to go for a blocksize of 8×8 pixels. That way enough correlated data points are available and the complexity is reduced to $128 \times 128(\text{blocks}) \times 2 \times 8 \times 8 \operatorname{ld} 8 = 6M$ operations. Among the problems related

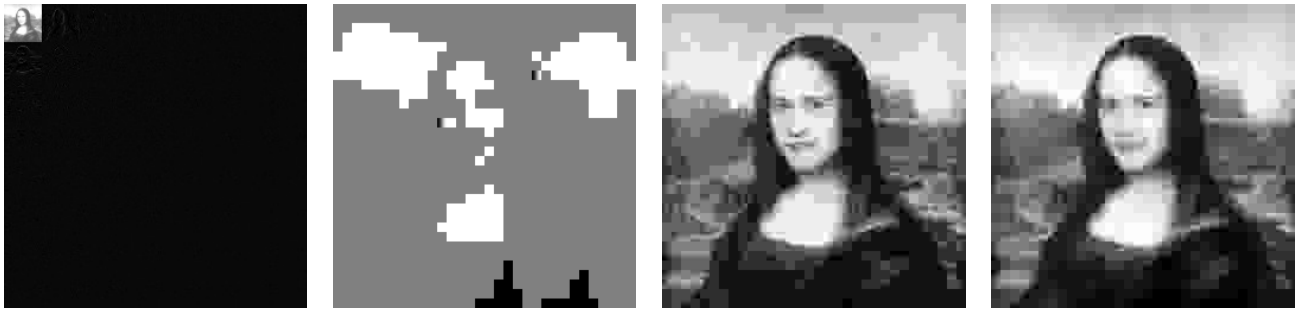


Figure 2.22: **The blocked DCT in action.** (Left) Blocked DCT coefficients reordered. (Middle left) Blocked DCT using JPEG luminance quantisation table. (Middle right) Blocked DCT (JPEG) with twice the data rate. (Right) Same with linear gradient quantisation table.

to blocking are of course the blocking artefacts, i.e. residual effects from quantisation that become visible between adjoining blocks. Another problem is that with blocked transforms no *very high* quantisation can be achieved, because each block has one DC coefficient. In case of Mona Roli, we have 32×32 blocks, as shown in Figure 2.21. These 32×32 coefficients alone already have an entropy equal to or higher than a 32×32 scaled image. This kind of interpretation of the blocked coefficients allows us, however, to rearrange them in a certain way resembling the multiscale approach (also shown in Figure 2.22). This is achieved by collecting all coefficients that correspond to the same spatial frequency and regrouping them in a sub-image of the size N/blocksize with the block coordinates as coordinates within the sub-image. All DC coefficients form an image in the top left corner, the next AC coefficients form the image next to that one and so on. This reordering allows for easier adaptive quantisation and more efficient back-end compression. The correlations that are left in the dataset due to the blocking can even be removed now by a further transform.

Blocking Mona Roli

It has been mentioned before that entropy is not a fair measure for an energy-compacting transform, because the coefficients of interest are concentrated around the DC coefficient and by considering the quantisation one could get rid of all the zeros that still contribute to the entropy. Not taking this into account, the results for Mona Roli are very bad for the blocked DCT (cf. Figure 2.22). As soon as we no longer consider the lower right half of the transformed image and only keep the top left half,[1] the results become again encouraging, however, this optimisation is applicable to any compacting transform. Here are the results of the blocked DCT:

	DCT JPEG	DCT JPEG x2	linear gradient
MSE/MAE	1718.7/37.38	85.5/7.07	73.18/6.11

DCT in Space

The DCT leaves no doubt that it is an efficient transform for correlated data. Through the JPEG and MPEG standards it is used in everyday life, though we would wish that no blocking had been used. These 8×8 blocks are part of the standard and by lifting it a new standard would have to be made. For applications in space, implementations in hardware are available from different companies and thus the DCT is in use by the majority of commercial satellites for data compression [Yu09]. One of the more recent uses with astronomical relevance was in the ESA Huygens Lander

[1] At this level of quantisation the nonzero coefficients are found in an even smaller region.

[Rüf92]. DCT with quantisation is a very good choice if the photometric quality is not crucial, but for precise measurements this kind of nonlinear operation is not desirable. To use the DCT without quantisation as a pure decorrelation tool is not possible due to the coefficients represented in float. However, the DCT exists also in forms where the operations are carried out in float, but the coefficients are in integer, thus leading to a fully reversible decorrelation. Examples of such are discussed in [Wei01] and [Iwa04] and a general way of deriving integer-to-integer mappings is presented in [Plo04].

The CDF 9/7 Wavelet Transform

The drawbacks of the use of a blocked DCT in JPEG are the blocking artefacts and the problem that very high compression is not possible due to the many DC coefficients. For this and other reasons the JPEG committee drafted a new standard based on wavelets that satisfies modern requirements and especially removes the mandatory blocking from the transform. This standard, known as JPEG2000 [ISO04], had its rusty start and is still poorly supported in computer applications, but the transform that was selected is worth having a look at, especially since it has become the new CCSDS recommendation for space applications [CCS05].

Filter Banks

Instead of constructing the transform matrix (Figure 2.24) I describe the approach via *filter banks* here, which is also used in the CCSDS standard [CCS05]. Even more material is found in [CCS07]. In [Say06] the link between the multiscale approach and the way how filter banks work is explained. The Cohen-Daubechies-Feauveau 9/7 wavelet uses two sets of analysis filter coefficients – so-called *taps* – to derive the average and detail components. Nine low-pass and seven high-pass coefficients are used. The one dimensional wavelet transform of a dataset s with $2N$ samples is achieved by calculation of the coefficients $C(j)$ (low-pass component) and $D(j)$ (high-pass component) in the following way:

$$C(j) = \sum_{n=-4}^{4} h(n)s(2j+n) \quad D(j) = \sum_{n=-3}^{3} g(n)s(2j+n+1) \quad j = 0..N-1 \ .$$

On the boundaries the input data s need to be mirrored according to $s(-m) = s(m)$ and $s(2N-1+m) = s(2N-1-m)$. The inverse operation is carried out by the following synthesis filters:

$$s(2j) = \sum_{n=-1}^{1} q(2n)C(j+n) + \sum_{n=-2}^{1} p(2n+1)D(j+n)$$
$$s(2j+1) = \sum_{n=-1}^{2} q(2n-1)C(j+n) + \sum_{n=-2}^{2} p(2n)D(j+n) \ .$$

In both, the even and odd part, $j = 0..N-1$. The filter coefficients for analysis and synthesis are given in the table beneath. Note that the low-pass coefficients h and q each add up to $\sqrt{2}$, whereas the high-pass coefficients g and p add up to approximately zero. The filter operation itself is conceivable as a convolution/correlation of the filter with the dataset, the low-pass derived from even and the high-pass from odd positions.

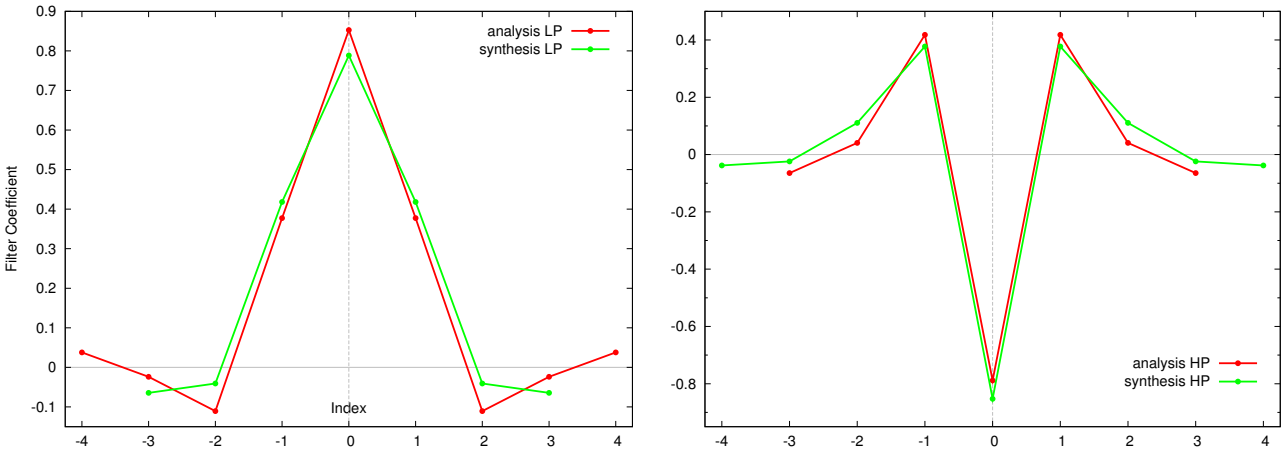


Figure 2.23: **CDF 9/7 filter coefficients.** (Left) The low-pass filter with the nine coefficients for analysis in red colour and the seven coefficients for synthesis in green colour. (Right) Likewise, the seven analysis coefficients of the high-pass filter in red and the nine synthesis coefficients in green colour.

index j	LP filter $h(j)$	HP filter $g(j)$	LP synth. $q(i)$	HP synth. $p(i)$
0	0.852698679009	-0.788485616406	0.788485616406	-0.852698679009
1/-1	0.377402855613	0.418092273222	0.418092273222	0.377402855613
2/-2	-0.110624404418	0.040689417609	-0.040689417609	0.110624404418
3/-3	-0.023849465020	-0.064538882629	-0.064538882629	-0.023849465020
4/-4	0.037828455507			-0.037828455507

Note that this operation decomposes the input only by one level, i.e. into one half of coarse detail and one half of fine detail. In order to achieve a multiresolution decomposition, the process needs to be iterated on the coarse detail coefficients to arrive at a standard decomposition.

Mona Roli in CDF 9/7

Our test image has gone through quite a number of transforms, with the unblocked DCT being the best one so far. In Figure 2.25 the image which is reduced in entropy by a factor 64 has some cross-like artefacts when uniform scalar quantisation is used. The thresholded image on the other hand has a very smooth appearance. However, all cases achieve substantially better results than in the DCT.

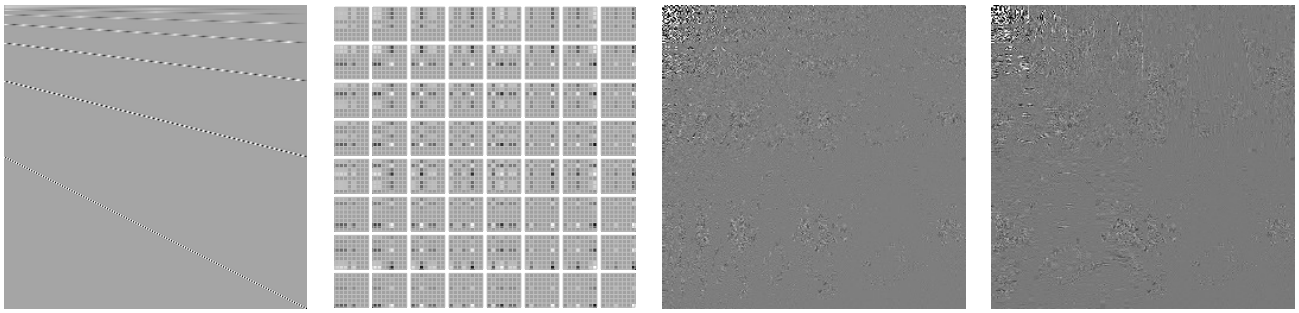


Figure 2.24: **Matrices of the wavelet transform.** (Left) CDF 9/7 basis matrix. (Middle left) 8×8 kernel. (Middle right) Mona Roli standard decomposition. (Right) Mona Roli in pyramidal decomposition.

Figure 2.25: **Wavelet quantisations.** (Left) Pyramidal hard thresholding. (Middle left) Pyramidal decomposition with rounding. This result is what you will achieve in JPEG2000 as well. (Middle right) Map of standard deviations of the Pyramidal decomposition. (Right) Standard decomposition with rounding.

	pyr. hard thr.	pyr. uniform	standard uniform
MSE/MAE	67.72/6.02	61.50/5.78	68.25/6.16

In contrast to the DCT, where uniform quantisation leads to a nonlinear operation, this is no longer the case here. For the synthesis of a single value in one dimension the same number of transform coefficients – one from each scale – need to be considered and the scales are all of the same unit. Thus quantisation with the wavelet transform keeps the photometry intact (of course, apart from the rounding, which is linear) and is thus also applicable to astronomical data. In the CCSDS recommendation an integer approximation to the CDF 9/7 transform is given, which can be used for lossless compression. This differs to JPEG2000, where the CDF 5/3 integer wavelet is used for lossless data. In the domain of space missions the discrete wavelet transform is now successively replacing the DCT, the most prominent astronomical example so far being the two Mars Exploration Rovers. More on their imaging system can be found in [Mak05] and a nice detailed description of the software is [Lit05].

KLT

The Karhunen-Loève Transform, also known as the *Hotelling transform* or *Principal Component Analysis* derives its transform matrix from the data and achieves therefore optimal decorrelation [Mac94]. This sounds great, but the truth is that this transform is not practical for compression, as it also requires to transmit the basis matrix (or alternatively, the ACF), which essentially doubles the amount of data. In light of this fact the other severe drawbacks are almost negligible, but for completeness, the KLT is not separable, it has very high computational complexity ($> N^2$) and there is no trick to speed it up. As the performance of the DCT comes asymptotically close, it is seldomly ever used in image compression, and usually authors stop at this point. I, however, will give you an idea of how it works with Mona Roli.

Here is a short step-by-step howto for the KLT. If A is an $m \times n$ matrix, eventually centred by subtracting the mean of each column from the vector coordinates, we will first calculate the $m \times m$ covariance matrix $C = \mathrm{COV}(A)$. Then the eigenvectors and eigenvalues are derived from C. Next we sort the eigenvectors according to the eigenvalues. The result is the $m \times m$ basis for the 1D KLT. Each basis vector is a principal component and the way to reduce the amount of data is to only keep the first

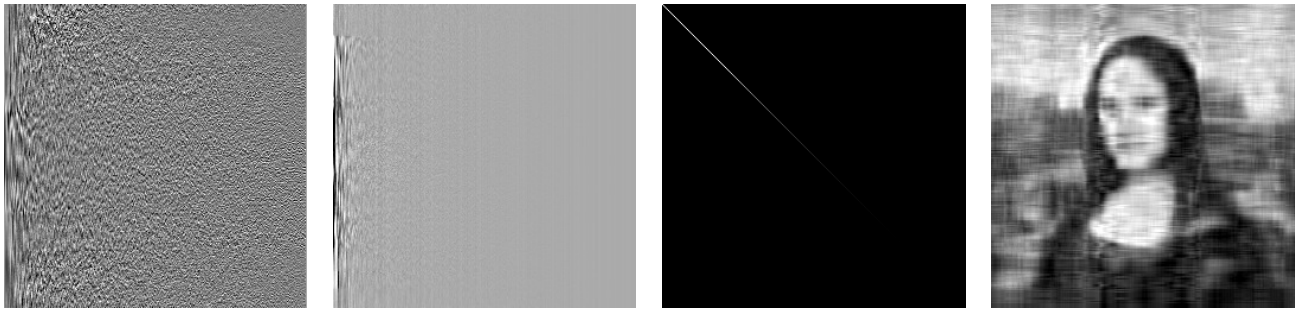


Figure 2.26: **Mona Roli in the KLT.** (Left) The basis matrix derived for the Mona Roli image. (Middle left) Mona Roli transformed (Middle right) Covariance Matrix of the transformed image. (Right) Reconstruction from the first 12 principal components, quantised.

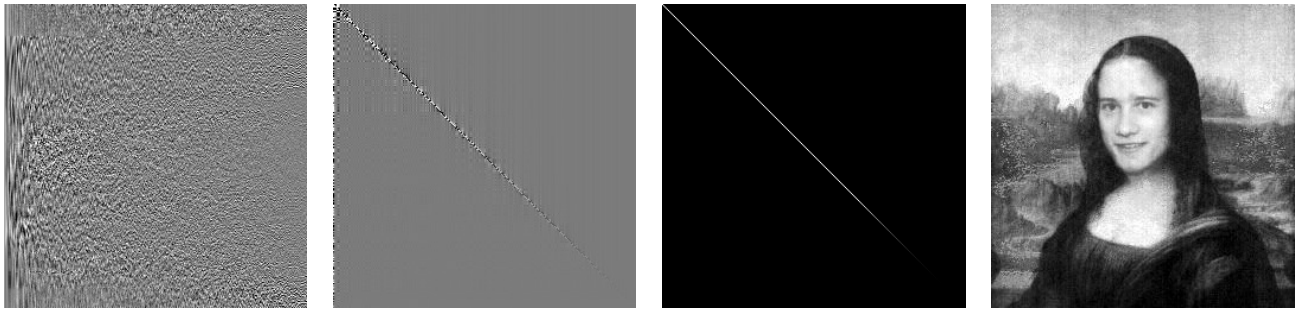


Figure 2.27: **Mona Roli in the 2D KLT.** (Left) The basis for the second dimension. (Middle left) Mona Roli transformed (Middle right) Covariance Matrix of the transformed image. Even the covariance matrix of the transpose looks like that now. (Right) Almost perfect reconstruction even if reduction factors up to 100 are used. No, there is no mistake with this image.

$k < m$ vectors whose eigenvalues contain a certain amount of energy (adding eigenvalues up to a certain threshold). The KLT is performed thus with a reduced $k \times m$ basis. Of course, that way only a transform in one dimension is made. To achieve a 2D KLT, we need to transpose the result and repeat every step from above. Note that each dimension has its own basis, which is why the KLT is not called separable.

The KLT makes a perfect decorrelation, but for reconstruction the basis (or bases in case of more dimensions) is needed again. Of course, every input image has its own KLT basis, but if we allow to sacrifice a bit of decorrelation we can assume that the input data are stationary and keep the basis for a number of data frames.

Mona Roli in the KLT

One consequence of perfect decorrelation is that you can almost throw away your transform coefficients and yet your reconstructed image will be almost identical to the original. In Figure 2.27 the bases were kept intact and the coefficients were reduced in entropy by a factor 100 through quantisation. Yet the reconstructed image differs almost not from the original, with MSE/MAE= 7.41/2.14. This is of course not fair, because the bases are known to the de-compresser. In Figure 2.26 I tried to put the record straight and combined the selection of the first few principal components in an 1D KLT with quantisation so that the basis and the transform coefficients together come up with an entropy of 1/64 of the original image. The result is now discouraging, with MSE/MAE=148.91/9.60, but my optimisations were also rather crude. Real-world applications of the KLT in data compression are rare, but there is lively work in that direction, e.g. [Pen06].

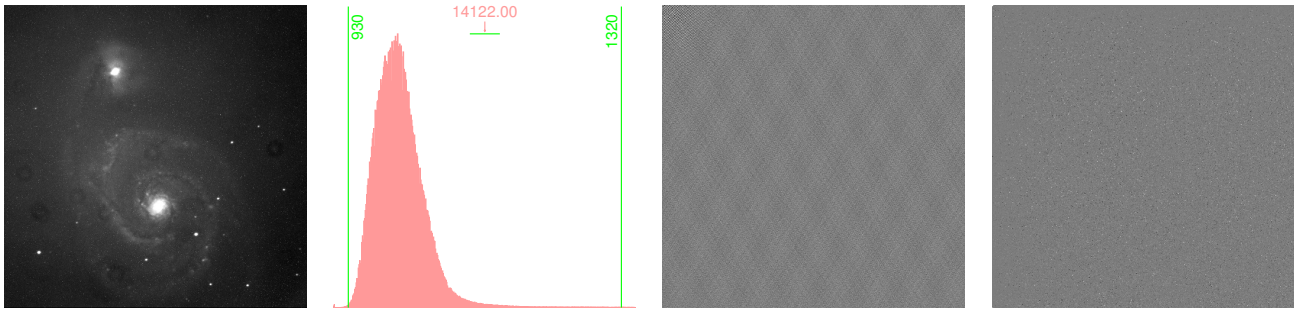


Figure 2.28: **M51** (Left) This picture is an uncorrected 5 minute exposure (V filter) of M51 that I made in early 2004 on the *Vienna Little Telescope*. (Middle left) Histogram of the input image. (Middle right) Discrete cosine transform. (Right) CDF 9/7 transform (standard decomposition). Now we wish Mona Roli back.

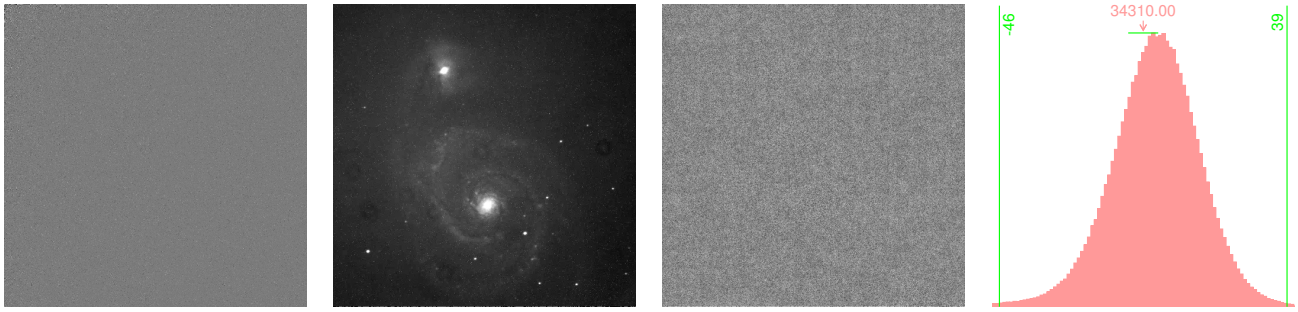


Figure 2.29: **M51** (Left) Haar transformed (standard decomposition). (Middle left) Image after HCOMPRESS with a factor 10 in entropy. (Middle right) Residual. (Right) Histogram of the residual.

3 Transforms of Typical Datasets

All of the transforms presented in this chapter can be used for data of any dimension, mostly because they are separable, i.e. a n-dimensional transform is achieved by successive one-dimensional transforms. However, as noted before, astronomical CCD images tend to be sparse in almost any representation, so that a transformed image is vastly as grey as in Figure 2.28 (right).

From this we see once more that an astronomical image consists mainly of noise in terms of entropy. In the example shown below, the dataset has $H_S = 7.083$ and $M_S = 6.567$, so that compared with the uncompressed image, where a sample is stored in 16 bits, a factor 2 of lossless compression seems to be easily achievable. However, we could have the requirement for higher compression and from what we have learned so far, this will only work with lossy steps. In Figure 2.29 I used HCOMPRESS [Whi92] to compress down to $H_S = 0.7$. This algorithm uses a 2D Haar transform with thresholding and it is part of the popular CFITSIO library [Pen99]. This is a dramatic reduction and a close inspection at higher magnification reveals first visual differences. The variance of the residual is comparable with the readout noise.

This little example impressively shows that large lossy compression factors are achievable without much loss of scientific information. To find the best tradeoff between compression and quality one would determine the R/D function where R would be the compressed data amount and D a specific quality measure (like distortion of photometry or astrometry of the data). To get a better feeling about the power of

linear transforms for compression, filtering and analysis I will now show plots of some typical 1D datasets including PACS test data.

1D Data

Let's begin with the data file we are already familiar with, the one used in the beginning of this chapter (Figure 2.1) to illustrate simple prediction. Our aim was to decorrelate the data to bring the entropy down to the noise $\sigma = 20.9$. In the Haar transform we get averages and differences, but as the differences are made from the averages they should not lead to the observed noise increase we had in differentiation. To find out whether this assumption is true, take a look at Figure 2.30. As we know the Haar wavelet produces fractional coefficients we multiply each scale with the respective factor to obtain integers. The integer approximation to the CDF 9/7 wavelet used here is lossless and works much better on this dataset, as you can see in Figure 2.31. The entropies H_S of this dataset in their different representations are given in the table below. So, the performance of the wavelet is okay, but it is not superior to a simple running average on this dataset.

original	noise	differences	runavg	Haar (amp)	CDF 9/7i
8.41	6.37	6.91	6.61	7.55	6.66

Ramp Data

Detectors that are read in a non-destructive way generate so-called *ramps*, where the signal is proportional to the slope of the ramp. An example for this are Ge:Ga photoconductors for the far-infrared, such as the ones used for PACS and FIFI LS [Ros02]. The data shown in Figure 2.32 stem from tests I made with the flight spare model of PACS at MPE on the 1st of April in 2009. Data from two pixels of the red detector array are shown in succession, each one providing 512 samples that are arranged in 8 ramps. These ramps are almost linear but differ in intercept due to the so-called reset noise and in slope due to various effects, one of them being the chopping that was exercised. There will be a lot more about this in Chapter 5.

It is not trivial to determine the entropy-relevant noise in ramp data due to reset noise, nonlinearities and the inherent random walk. Essentially, the systematic components – I summarise them by the term *the signal* – have to be removed to obtain the noise (this can be seen as the main goal of decorrelation). One way to do so is to calculate a *keyramp* and look at the residuals. For a dataset as short as the one shown in the plot we simply state for now that the entropy of the noise is not smaller than half a bit per symbol compared to the entropy of the differences. Figures 2.32 and 2.33 compare differentiation with the Haar and CDF 9/7 wavelet. Here are the numbers for comparison:

original	differences	Haar (amp)	CDF 9/7i
9.89	6.75	8.67	5.87

A simple differentiation does a good job, but actually cannot bring the numbers down to 0 but to a value around the mean difference. A similar effect is seen in the Haar wavelet, but obviously its main problem is the mapping to integer since it is not really meant to be used without quantisation. The CDF wavelet suffers from its filter

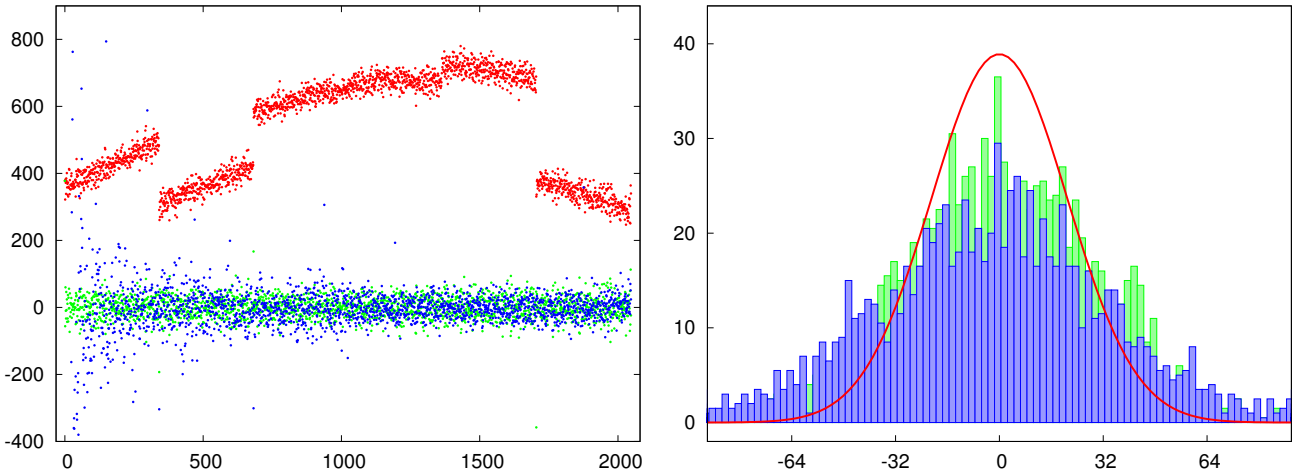


Figure 2.30: **Decorrelation by the Haar transform.** (Left) The input data in red were Haar transformed and mapped back to integer. For this reason the first coefficients are quite far away from zero. Plotted in green colour are the differences for comparison. (Right) The histogram shows that the whole operation is not good for lossless decorrelation.

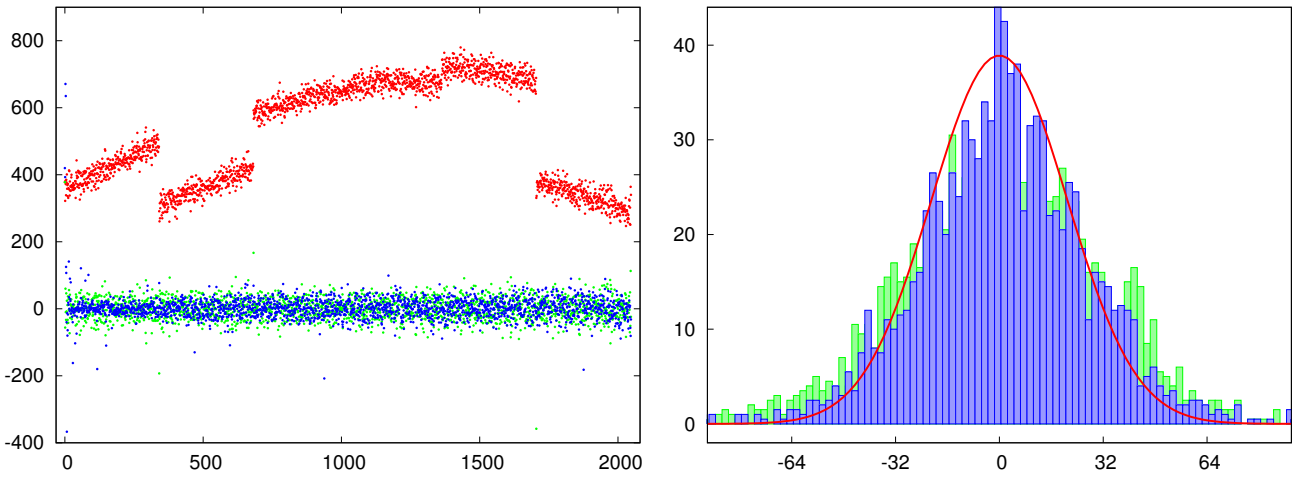


Figure 2.31: **Decorrelation by the CDF 9/7 wavelet.** (Left) The advantage of this integer approximation to the CDF 9/7 wavelet is that the decorrelated values are in integer and don't need to be scaled to be lossless.

lengths in a way that it produces large values at all scales for every discontinuity in the dataset, yet it still performs best. For this kind of data, an approach based on prediction works best, because we can take advantage of a priori information about the signal. Such a very specialised prediction model will only work well with the data it has been designed for. Transforms are not tuned to specific datasets, but they work well on all kinds of data. Chapters 4 and 5 contain an in-depth treatment about prediction-based decorrelation stages that I developed for the PACS detector arrays.

To put it in a nutshell, if lossless compression is to be done, use prediction or any other preprocessing that allows to model the signal. For lossy compression choose a transform-based approach if possible, but keep a sharp eye on the distortion.

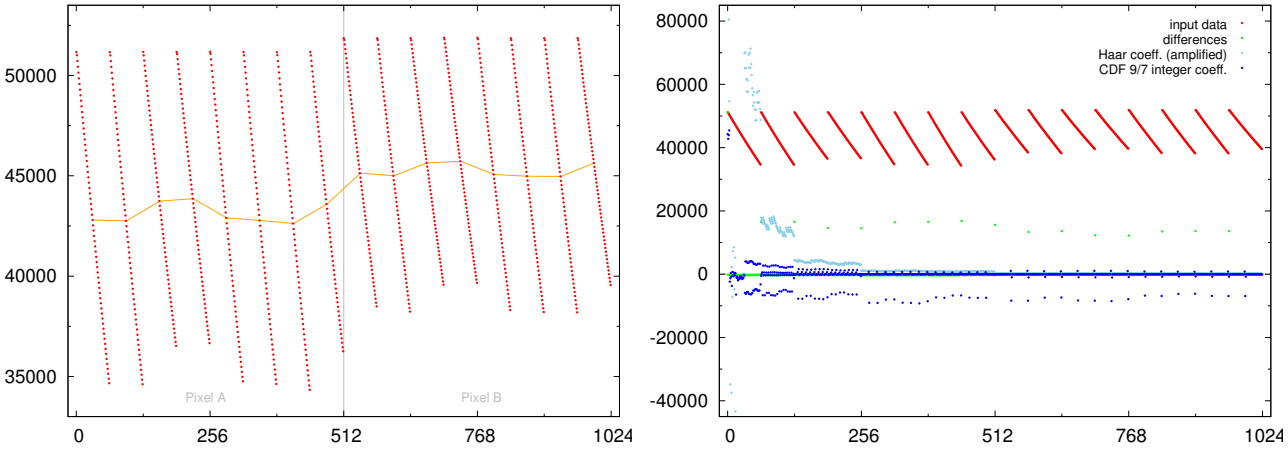


Figure 2.32: **Ramp input data.** (Left) The raw readouts from two pixels, each one read at 256 Hz with a reset interval of 1/8 second. The orange line connects every 32nd readout to give an impression of the variation which is also due to chopping. (Right) The data shown again in red colour, along with their differences (green), the Haar transform coefficients (light blue), and the CDF 9/7 integer coefficients in blue. Note that the Haar coefficients have been mapped to integer.

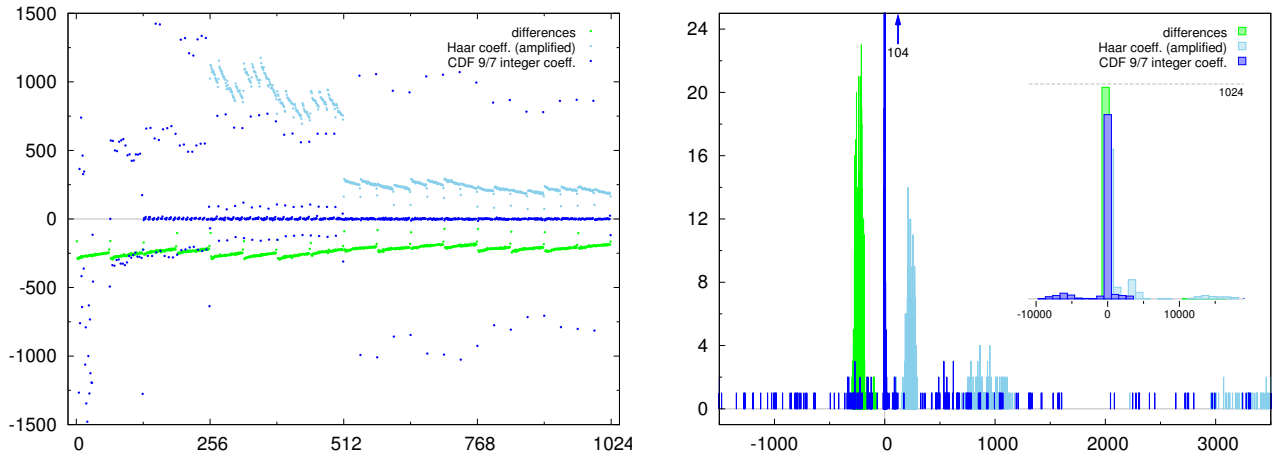


Figure 2.33: **Properties of transforms.** (Left) A closer look at the coefficients to allow for better comparison of the performance. (Right) Histogram of the decorrelated values with a bin size of 1. Note that the complete range of values is not shown. The inset histogram shows a larger range with a bin size of 1024. The three datasets have been slightly shifted for better visibility. These plots indicate that the CDF wavelet brings down the bulk of values to zero yet suffers from many coefficients with large values.

> He invented and introduced the symbol ∞ for infinity. [...] Wallis' life was embittered by quarrels with his contemporaries, including the political philosopher Thomas Hobbes, who characterised his *Arithmetica Infinitorum* as a "scab of symbols", and the Dutch mathematician Christiaan Huygens, whom he once tricked with an anagram concerning a possible satellite of Saturn.
>
> —*EBm "Wallis, John"*

3

Encoding and Compression

It's hard to tell what the first concept was that can really be called a *compression*. I tend not to consider *Braille* or *Shorthand*, but rather Morse codes that were developed from the late 1830s on [Hoi07], which are a – well, actually bad – example for variable-length ternary encoding. These codes basically cover the requirements from entropy encoding, at least after refinement by Morse's assistant Alfred Vail, that encoded frequent symbols with shorter codes at the cost of rare symbols that get assigned longer codes. Such *variable-length* codes are central to many encoding algorithms.

·— — ···· ·— — ···· ·— — ···· — —·· · —·· ·— — · ·· · ··— — —·· ···· —

What hath God wrought!
—*First message transmitted over the first telegraph line in 1844.*
(in American Morse Code)

Data compression can be achieved in different ways. There are *basic techniques* like replacing *run-lengths* of symbols with a symbol/number pair, though they stay quite far away from the entropy limit unless very special datasets are used. *Statistical methods* regard the symbols as independent and aim at encoding the individual symbols to their entropy limit. Whether this is achieved by variable-length codewords or other tricky methods such as arithmetic compression does not matter. The model i.e. the statistical assumption that is made about each symbol determines how a symbol is encoded. *Dictionary methods* are very different to that, they assume that the symbols are dependent and seek for replacing recurring symbol patterns – phrases – with shorter reference tokens. This is achieved by a dictionary that has of course some relation with statistical modelling, but the main feature of a dictionary is to contain symbol phrases of all lengths. Of course, there are still various algorithms left that are hard to classify, either having features of both groups or standing even outside, roughly categorised as *context-based*, but classification is anyhow not the relevant aim of this chapter.

Dictionary methods and context-based techniques are preferred by all kinds of multi-purpose data compression, such as *zip*.[1] Yet in this chapter I mainly concentrate on statistical methods because the scientific data we want to compress are affected by a noise component that is not well handled by that kind of technique. Some methods

[1] What we call *zip* today is a data container format originally introduced by Phil Katz' ZIP program. Nowadays it uses the *Deflate* algorithm for compression, which is closely related to LZ77.

The chapter is divided into the following sections:

presented herein, especially among the statistical ones, assume that the data to be encoded are already decorrelated. As in the previous chapter, each algorithm is discussed in terms of fitness for space applications, in particular for data with a large noise component. Apart from [Sal07], a fast introduction into basic lossless data compression focusing on space applications is [Kor90], a similar one is [Her96].

Terminology

Astronomers are normally not familiar with the vocabulary used in data compression. Although the used terms are simple and mostly self-explaining, I don't want you to guess what is written in here.

A *symbol* is the smallest input entity of a compression algorithm, it can be a 16-bit measure, a sample, a character, or any number of bits grouped together. A single-bit symbol is called a *flag*. By *stream* a sequence of bits is meant, similarly a *string* is a sequence of characters or symbols. A *token* is just like a flag, a symbol, or a combination of both. A *codeword* is an output symbol or token. When I speak of a *key*, a value is meant that is necessary to interpret another. As you can certainly imagine, the meaning of these terms depends strongly on the context used.

1 Basic Methods

A variety of compression techniques is easy to understand, but hard to classify. For example, masking nonzero values with bits or *move-to-front* coding [Ben86]. Such algorithms have been created for all kinds of purposes, but their compression performance is usually bad unless very special datasets are used. In here I will only pay a little more attention on VBWL compression and on run length encoding, of which the latter one is still a very widespread technique.

Variable Block Word Length

VBWL encoding [Kor90] is a very rudimentary technique of encoding values to smaller codewords. Small groups of input symbols are offset, either by the first value, by the mean or the median, the minimum or the maximum. Then the maximum number of bits required to encode the offset symbols is determined, and encoded with the symbols in the output bit stream. This technique obviously has some issues, it assumes a smooth input sequence, it may require mapping the offset values to positive or to add additional key information such as the position of the minimum within the group and so on. Even the computational complexity is pretty high compared with better encoding techniques. Its overhead can lead to an increase of the data length after compression for noisy or highly oscillating input. However, if the range of values of a dataset is pretty large and the compression requirement is not tight, it is at least better than nothing.

Run Length Encoding

One of the simplest yet widely used algorithms is RLE, as it is very easy to implement, with very little resource requirements and is pretty effective on structured data with little entropy. A run length is a sequence of identical symbols. One variant of RLE would be to send the number of repetitions with every symbol. Another variant is to use an escape character – usually an infrequent symbol – to mark a run length and leave single standing symbols untouched. This already greatly reduces the overhead. Depending on the memory, the escape character could be chosen from the input data. If only run lengths of a single symbol – such as zero – should be considered, then even the escape character can be spared. RLE can be made memoryless and has a complexity $\sim N$, but noisy data are also increased instead of reduced depending on the implementation. Run length encoding is good for huge run lengths, such as the dots on a black and white page. In the early days of image compression, where only few colours were available, this type of encoding was pretty effective. Even in modern compression schemes it can be found as a *back-end* (the final encoding step) to the Burrows Wheeler transform (see Section 4 below) for example.

RLE and Statistics

For us, RLE is a good starting point for some theoretical considerations. Given a set of symbols we may be interested in the number of run lengths we can expect. This depends of course on the set size, on the number of different symbols and their probabilities and on the length of the run we are searching for. The number of ways some symbols can be arranged is given by their *permutations*. N different symbols would have $N!$ permutations,[1] but no run lengths. If any symbol S can occur any number of times, we have S^N permutations.

Now let N be the size of the dataset, $c_1, c_2, ..., c_s$ count the symbols 1...s, then the number of permutations is $N!/(c_1!c_2!...c_s!)$ (note that $\sum c_i = N$). In case the symbols have equal counts this reduces to $N!/(sc!) = (N-1)!/(c-1)!$. For a binary set of k ones and $z = N-k$ zeros we get $N!/(k!(N-k)!) = \binom{n}{k}$ permutations. The probability for a specific binary string of k ones and z zeros given the probability of $p_z = 1 - p_k$ is

[1] Instead of computing the factorial $N! = \prod_{i=1}^{N} i$, $0! = 1$ the Stirling approximation $N! \approx \sqrt{2\pi N}(N/e)^N$ can be helpful for large N and the Gamma function $N! = \Gamma(N+1)$ is useful to get continuous values.

$P = p_k^k p_z^z = p_k^k (1-p_k)^{N-k}$ and thus the probability, that an arbitrary combination of k ones and z zeros is found is $\binom{n}{k} P$. Now look at P. This is the *Geometric distribution* (the discrete version of the Exponential distribution). Remember the Exponential distribution showed how intervals between events are distributed. In our case the intervals are the runs of zeros and the events are the ones. Now take a look back at Figure 1.4. Depending on p_k the probability for longer runs will decrease and we are confronted with the problem of finding optimal codewords to replace the run lengths. This is what Golomb codes are essentially about and so this is a perfect transition to dive into statistical compression methods.

2 Statistical Methods

Here are some more terms that are used in connection with statistical methods. A *prefix code* or *instantaneous code* is one where no codeword is the prefix of another codeword. The opposite, an *ambiguous code* would not be uniquely decodable. If the code is *universal*, then it assumes nothing particular about the probability distribution of the input data. *Variable-length codes* compress data by assigning shorter codewords to symbols with higher probability. The aim of such an algorithm is to represent a symbol in a codeword of the size of the symbol's entropy. However, a symbol usually has an entropy which is not an integer, like $H_S = 2.34$ and the variable-length code has to use the unused noninteger fraction of the third bit as well. In that sense a variable-size code has always a penalty due to that fractured bit except if the symbol entropies are integers, that is, if the symbol probabilities are negative powers of two. Careful study of this section will make you understand the last phrase.

The data to be compressed by statistical methods need to be decorrelated, otherwise there is not much compression to be achieved.

Huffman Codes

In 1952 David Huffman [Huf52] worked out the first universal method for constructing an optimal prefix-free variable length code for a finite alphabet, which became the subject of decades of research and is still used as back-end in many widespread compression methods such as in the popular ZIP, JPEG or MP3 formats.

The Huffman code is built as a binary tree from the sorted table of symbol probabilities. It starts at the bottom, selects the two least probable symbols and combines them to a new symbol with a probability equal to the sum of the two. The symbols are distinguished by a single bit 0 or 1. Now we have $n-1$ symbols and we repeat this procedure until there is only one symbol left. We end up with a binary tree. To find the codeword for a particular symbol we follow the tree from top to bottom and collect all 0 or 1 that we find to form the codeword. As the less frequent symbols are now at the bottom, we have to follow the longest path and collect most bits. Figure 3.1 gives a visual description that should be easy to follow.

The problem with Huffman coding is that it needs all of the input data to construct the Huffman tree. This tree or information to construct it, such as the frequency table need to be transmitted as well. The computational complexity to build the tree is $\sim N$, as is the complexity to actually encode the symbols. For a given dataset, there exist a

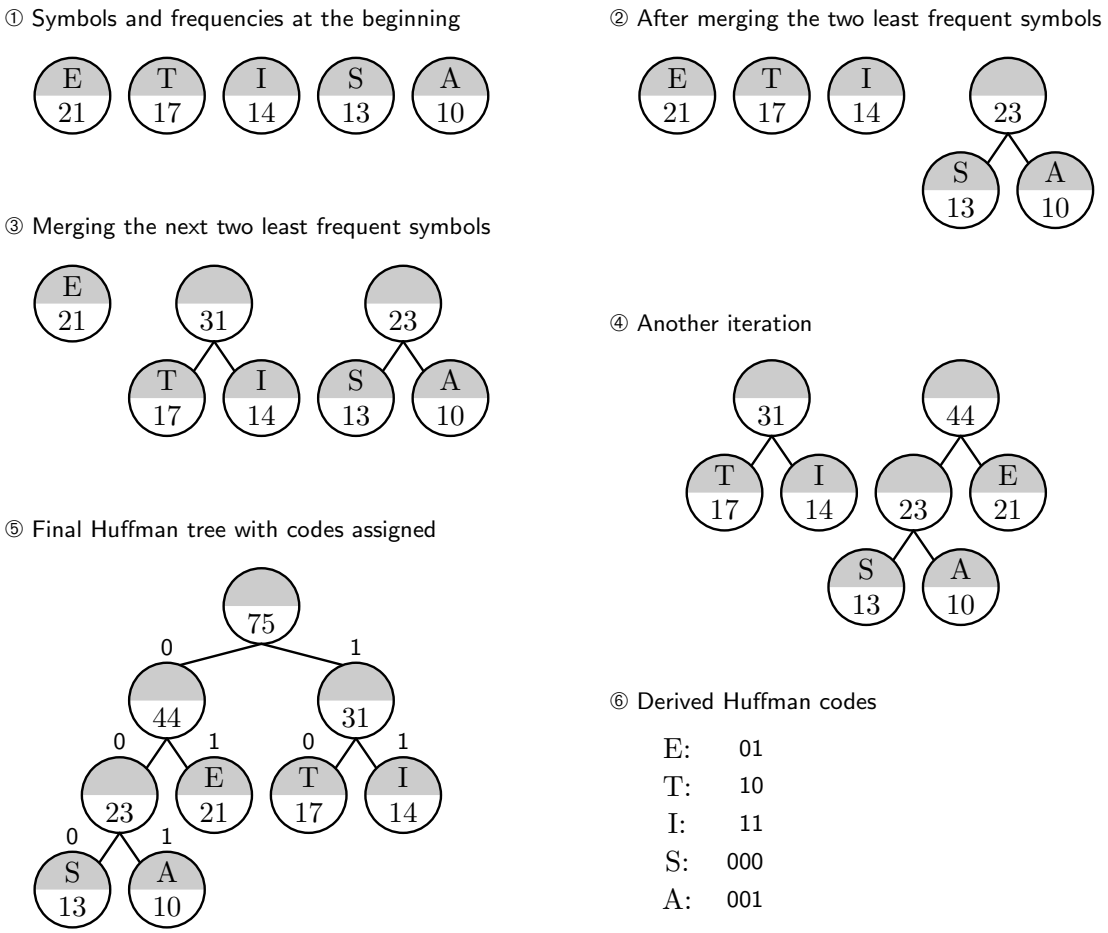


Figure 3.1: **Huffman tree buildup.** An example input alphabet of five symbols is given together with their quantity. The frequencies are counted and the sorted frequency table is the starting point for forming the Huffman tree. In the end a table is established where prefix-free binary codewords are assigned to the symbols.

number of optimal codebooks (= no codebook with smaller average code length exists), because codewords of the same size can be assigned differently.

Static vs Adaptive

A Huffman table that is derived from the input and never changed throughout the buffer is called static. Static tables have a very good statistical model of the buffer in the beginning, but towards the end of the buffer they become inefficient, because the more symbols we encode, the less are left as possible candidates towards the end of the buffer. For instance, the last symbol would not have to be encoded at all, because if we count which symbols we already encoded and compare that with the frequency table we know which symbol is left. So it would be best to modify the frequency table after each encoded symbol and rebuild the Huffman code, but obviously this would dramatically increase the computational complexity. A compromise is a semi-static model, which is updated from time to time, like rebuilding the Huffman tree for each data chunk. A fully adaptive code has the advantage of not needing to transmit a frequency table, because it is regenerated during de-compression. One could start the encoding with a code table containing only a "new symbol" escape code which would initiate the update of the frequency table with the given raw symbol. As more and more symbols are seen the frequency table approaches the probabilities of the sequence's statistic. However,

in the beginning a lot of "new symbols" would be seen, so that it is normally better to start with a coarse frequency table instead. Adaptive Huffman codes can be inefficient for small datasets or large alphabets.

> The problem of developing computer programmes, or software, has long been a painful one.
>
> —*John R. Rice in the EB 1975 Yearbook of Science and the Future*

Golomb and Rice Codes

Huffman codes have some disadvantages for the on-board compression of science data. First of all, it's a relatively resource-intense task to establish a binary tree and the data have to be accessible to do so. Even if a previously fixed code is used, it has to be available as a table in memory, which can be a problem if the range of values is as already large as 2^{16}. In his amusing paper [Gol66], Golomb introduced in 1966 codes which seek to compress run lengths by assigning shorter codewords to smaller values. These prefix codes may be used in a way similar to Huffman codes, but they can be derived on a per symbol basis without the need for a code table.

We remember that run lengths are geometrically distributed, a run of $n-1$ zeros with a trailing 1 in a binary source has a probability of $p_k(1-p_k)^{n-1}$. The codewords of the Golomb code are optimal for geometrically distributed data. However, a parameter m is needed to adjust to the distribution. This parameter is chosen best with $p^m = 1/2$ or, equivalently, the median of the (run length) symbols, which makes sure that half of the symbols are encoded with the shortest codeword length.

Code Construction

Here is a quick outline of how to construct a Golomb code of parameter m. Note that we need decorrelated positive integers as input. If we build a Golomb code of parameter m, we begin with asking ourselves about the number of members in the first group and the number of bits needed there. A group is a set of codewords of the same length. All groups have m members except the first group, which has less if m is not a power of 2. A rule of thumb is to use $m = \mathrm{ld}(H(x) - 2)$ as a starting point. We answer the former question by looking at the second group g(1), which has m members. g(0) has $2^{\lceil \mathrm{ld}(m) \rceil} - m$ members (in case of g(0)=0 we use m). The number of bits in the first group is $\lceil \mathrm{ld}(m) \rceil$. The first code in the first group is 0 and changing the group consists of adding 1 to the previous element followed by a left-shift, so that the new group has now a codeword of a length longer by one bit. In principle, a codeword is constructed from the quotient $q = \mathrm{int}(n/m)$, coded in unary,[1] and the remainder $r = n - qm$. In cases where m is a power of 2, it's as simple as attaching the unary coded q to the remainder. Golomb codes, where m is a power of 2, are also called Rice codes. They are especially favourable to compute, because the groups are all of the same length and the first element of a new group has a remainder of 0. This saves a lot of operations, especially one division, which is otherwise necessary in the construction of a codeword.

Here are sources for generating Golomb codes and Table 3.1 lists the first codewords for several values of m.

[1] The unary code of a number n is $(1 << n) - 1$. Its bitwise inverse is the fundamental sequence.

Listing 3.1: Rice code

```
int mk_Rice (int m2, int value, int *enc)
{
  int q,r,g; // group = nr of leading bits
  int rl;    // remainder length
  int m;     // get m from the input m2 = 2^m

  // calculate outside function for better performance
  m = 1 << m2;
  rl = m2 + 1; // length of rest

  g = value >> m2;  // group nr, nr of leading bits
  q = (1 << g) - 1; // prepare the left side
  r = value - g*m;  // prepare the right side

  *enc = (q << rl) | r; // form the codeword

  return rl+g; // returns the length of the codeword
}
```

For a general Golomb code the binary length $c = \lceil \mathrm{ld}(m) \rceil$ and the number of elements in the first group become important, and we have to consider that the first codeword of a group starts with a different base that depends on the number of codewords in group 0.

Listing 3.2: Golomb code

```
int mk_Golomb (int m, int value, int *enc)
{
  int g0, l0, b, g, q, lg;
  int len;

  l0 = log(m)/log(2) + 1; // codeword length in group 0
  g0 = (1 << (int)ceil(log(m)/log(2))) - m; // members in group 0
  g0 = g0 == 0 ? m : g0; // for powers of two we fix g0 = m

  if (value < g0) // group 0
  {
      *enc = value;
      len = l0;
  }
  else // other groups
  {
      b = (g0 << 1);      // form the base codeword
      g = (value-g0)/m;   // this group is which one
      lg = l0 + g;        // it has lg remainder bits
      q = (1 << g) - 1;   // prepare the left side in unary
      *enc = (q << l0+1) + b + (value-g0)-g*m; // composed codeword
      len = l0 + g + 1;   // length of the cw
  }

  return len;
}
```

There are of course also some traps in the Golomb code. First of all, the different lengths of codewords require bit stream encoding, which may take more CPU load than the Rice code generation itself. On the other hand, the size of the codeword quickly runs away for small parameters m. Assuming a maximum codeword length of 32 bits,

n	m = 1	m = 2	m = 3	m = 8	m = 14	m = 256
0	0	00	00	0000	0000	000000000
1	10	01	010	0001	0001	000000001
2	110	100	011	0010	00100	000000010
3	1110	101	100	0011	00101	000000011
4	11110	1100	1010	0100	00110	000000100
5	111110	1101	1011	0101	00111	000000101
6	1111110	11100	1100	0110	01000	000000110
7	11111110	11101	11010	0111	01001	000000111
8	111111110	111100	11011	10000	01010	000001000
9	1111111110	111101	11100	10001	01011	000001001
10	...	1111100	111010	10010	01100	000001010
11		1111101	111011	10011	01101	000001011
12		11111100	111100	10100	01110	000001100
13		11111101	1111010	10101	01111	000001101
14		111111100	1111011	10110	10000	000001110
15		111111101	1111100	10111	10001	000001111
16		1111111100	11111010	110000	100100	000010000
17		1111111101	11111011	110001	100101	000010001
18		...	11111100	110010	100110	000010010
19			111111010	110011	100111	000010011
20			111111011	110100	101000	000010100
21			111111100	110101	101001	000010101
22			1111111010	110110	101010	000010110
23			1111111011	110111	101011	000010111
24			1111111100	1110000	101100	000011000

Table 3.1: **Golomb codes for several parameters m.** For example, the Golomb code of parameter $m = 3$ for the number 8 is 11011.

only 31 different symbols can be encoded with the unary code, 62 for $m = 2$, 91 for $m = 3$, 232 for $m = 8$, 394 for $m = 14$, 6144 for $m = 256$. To cover the 16-bit range of values with a Rice code, a parameter $m = 4096$ would be required, leading to 12-bit words in the first group. One way to deal with that is to choose an escape code from the codebook, signalling that the next value shall be interpreted as a 16-bit raw value. In that way we can deal with statistical outliers without the need for a compromise in the m parameter. At last, remember that Golomb and Rice codes require a decorrelated positive integer input sequence, a requirement that no other algorithm in this chapter has, although in general it can be profitable for all.

CCSDS

In 1997 the CCSDS adopted a modified Rice code with an extension to low-entropy data as lossless compression standard for space applications [CCS97, CCS06]. The recommended *e_Rice* algorithm encodes blocks of 16 samples with different encoding options and selects the shortest one afterwards. This makes it sort of an adaptive entropy coder, but in a rather crude way. It is worth mentioning that larger symbols with low entropy are treated with symbol splitting.

Golomb and Rice codes have passed their zenith and the niche that is left now is when a very CPU efficient implementation is needed. Instead of going after the *e_Rice* standard I recommend to carefully adapt your own Golomb code to the instrument data. In Chapter 6 such an approach is made for a potential compression system for the SPICA/SAFARI instrument.

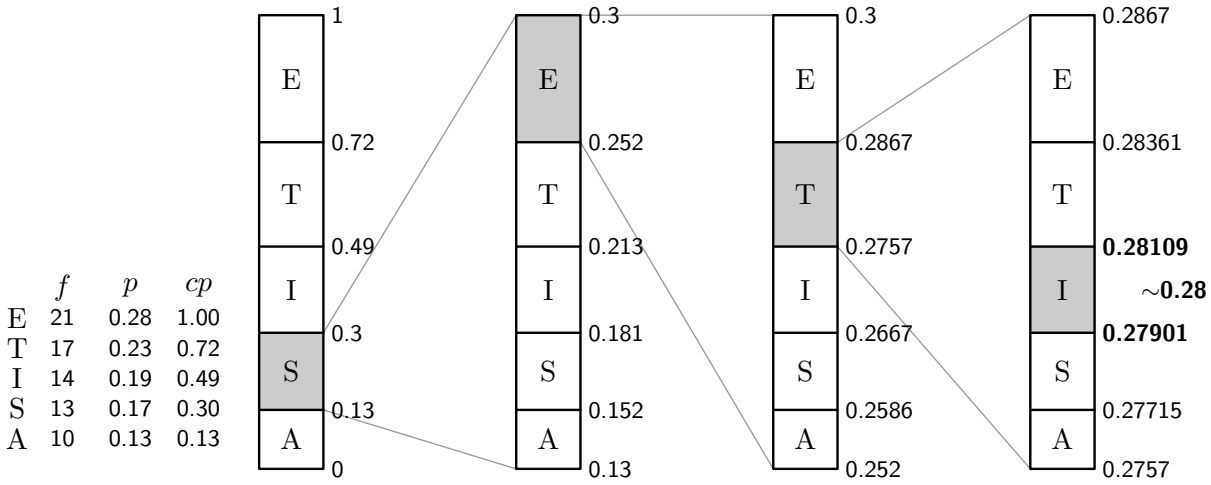


Figure 3.2: **How arithmetic coding works.** The important thing is to understand the cumulative probability table and how it is iteratively applied to the subintervals after every symbol. If the single probabilities p are successively added together, the table of cumulative probabilities cp is achieved. Note that the p need not be sorted. The interval borders of the symbol to encode are selected to be again subdivided by intervals projecting the cumulative frequencies. This process continues until all symbols are encoded. In our example, the string "SETI" is represented by any number between 0.27901 and 0.28109.

Arithmetic Coding

Generally, variable length codes share the problem that a symbol with a fractional entropy is encoded in an integer number of bits. By that the compression has an average inherent inefficiency of half a bit per symbol. One countermeasure is to combine symbols, but then the code table is becoming increasingly complex. The final solution to this was provided by a different approach in 1987 [Wit87]. In arithmetic coding, the exact sequence of the symbols is expressed by means of a fractional number. Once more, the starting point is a *frequency table*, but at least it does not have to be sorted as in Huffman coding. The symbol probabilities are added up in a cumulative probability table such that the total sum of probabilities give 1. We could now express a symbol by giving a fractional number falling into one of the probability intervals. Then the interval corresponding to the current input symbol is selected as the new range and the pile of probabilities is scaled into this interval to go on with the next symbol etc. In the end, the fractional number will be increasingly long, but only as long as to identify the last interval. Figure 3.2 shows how a string is encoded by a fractional number. An implementation of this algorithm must of course solve the problem that the encoded fractional number quickly grows out of the data type's precision. Thus, a renormalisation step has to be included from time to time.

The decoding procedure works very much the same way as the encoding process. The encoded fractional number is compared with the nested intervals until the original string is reconstructed. The statistical model of the data is very well separated from the encoding process itself. This allows to develop models that are specifically trimmed towards the dataset. As with Huffman codes, the frequency table can also be updated after every symbol to obtain a fully adaptive model. One consequence is that the cumulative frequency table has to be recomputed, but efficient algorithms based on balanced binary trees as described in [Sal07] have been developed to lower the computational complexity.

Range Coding

The CPU intense part of arithmetic coding is to calculate the new interval borders. In [Mof98] a number of improved variants were presented that work with lower hardware requirements and in [Sch98] range coding, employing further optimisations on the output encoder, is described. A full implementation of arithmetic compression can be found e.g. in [Str05].

Arithmetic compression is optimal, it encodes a decorrelated dataset down to the entropy limit. If the data model (the cumulative frequency table) is not an exact match to the actual statistics, the compression performance is not much reduced. Unsurprisingly, arithmetic compression works as the backbone in the most efficient compression schemes to encode transform coefficients as in JPEG2000 or compression tokens as the ones produced by dictionary based techniques.

3 Dictionary Based Techniques

In correlated datasets combinations of symbols may occur frequently like words in a piece of text do. It seems plausible that if recurring patterns are present, compression can be achieved by replacing the repeated patterns with shorter references. Such a compression strategy consists of two steps, to construct a *dictionary* where the patterns are kept during compression from the input and to make use of it to encode the patterns as tokens. Applications exist, where such a dictionary can be static, but the true potential of this family of algorithms lies within the adaptive dictionary. Most dictionary based algorithms can be divided into two families that stem from two papers of Lempel and Ziv in 1977 [Ziv77] (the LZ77-family) and 1978 [Ziv78] (the LZ78-family).

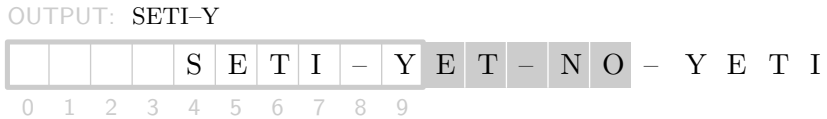
LZ77

In LZ77, a buffer window consisting of a *look-ahead buffer* and a *sliding window* is scrolled over the symbols and matches of strings between the two are encoded as *phrase tokens*. The sliding window functions as the dictionary and the longest match that is found is encoded by the position and length of the substring in the sliding window. This is a bit hard to explain, but with the help of Figure 3.3 you will understand the mechanism. First, symbols are loaded into the look-ahead buffer until it is full. So far the dictionary is empty and thus no match can be found. A single unmatched symbol is encoded by a *symbol token*, which is the symbol itself. To distinguish between symbol and phrase token, an extra bit – a prefix – needs to be attached to the token. The optimal size of a phrase token depends on the size of the sliding window as well as on the look-ahead buffer, which are typically around 4096 symbols (so we need 12 bits to encode the offset) for the sliding window and 32 symbols (5 bits) for the look-ahead buffer. In this partitioning a symbol token is 9 bits (1 for the prefix and 8 for the symbol) and a phrase token 26 bits (1+12+5+8 for the unmatched next symbol). By including the next unmatched symbol in the token one prefix bit is saved. In another variant of LZ77 unmatched symbols are also encoded in phrase tokens with a length parameter of 0. This way the prefix can be spared. After encoding the whole buffer is scrolled by the number of symbols that were encoded and the search for the longest match is again started at the first symbol in the look-ahead buffer.

① The input string is partly loaded into the look-ahead buffer and the sliding window is empty.

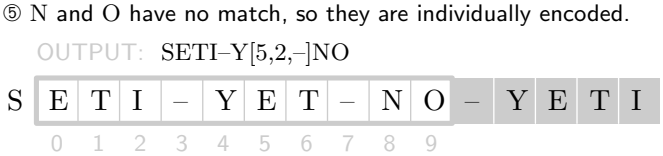
② No match for S is found, so it is encoded to output and slided into the window.

③ Same story for the next five symbols.



④ ET has a match, encode the index of the match and its length and the next symbol, then shift.

⑤ N and O have no match, so they are individually encoded.



⑥ –YET has a match, so encode it and terminate with next symbol I.

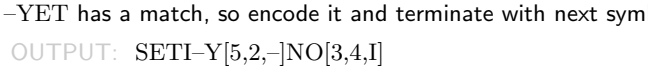


Figure 3.3: **How LZ77 works.** Sliding window and look-ahead buffer are scrolled over the input symbols. Matches are encoded by phrase tokens which are compound of offset, length and the next unmatched symbol.

Deflate

A modified version of LZ77 is *Deflate*, which is actually the most widespread compression algorithm today. It is part of the *zlib* compression library, the core of the file formats ZIP, GZIP, PNG, Postscript and PDF. Its main modification is that it looks up Huffman codes for the offset and length parameter pairs in the tokens. A good description with further references can be found in [Sal07].

Performance on Science Data

In everyday life, dictionary-based methods are preferred over statistical methods on datasets like a typical computer user's files. However, their compression performance depends on recurring patterns, but how likely are recurring patterns in white noise? Obviously, this depends on the noise amplitude, and we can also say that larger buffers have a better chance of finding a match. The probability of a match can be calculated in a similar way as we did this for run lengths above, thus it follows a geometric distribution. In Figure 3.4 I show the result of a practical experiment with *Deflate*, where the compression performance of 4 bit noise is shown as a function of data amount. We see that it follows the cumulative distribution function of the geometric distribution, but it never reaches the entropy limit because of the encoding overheads of the tokens.

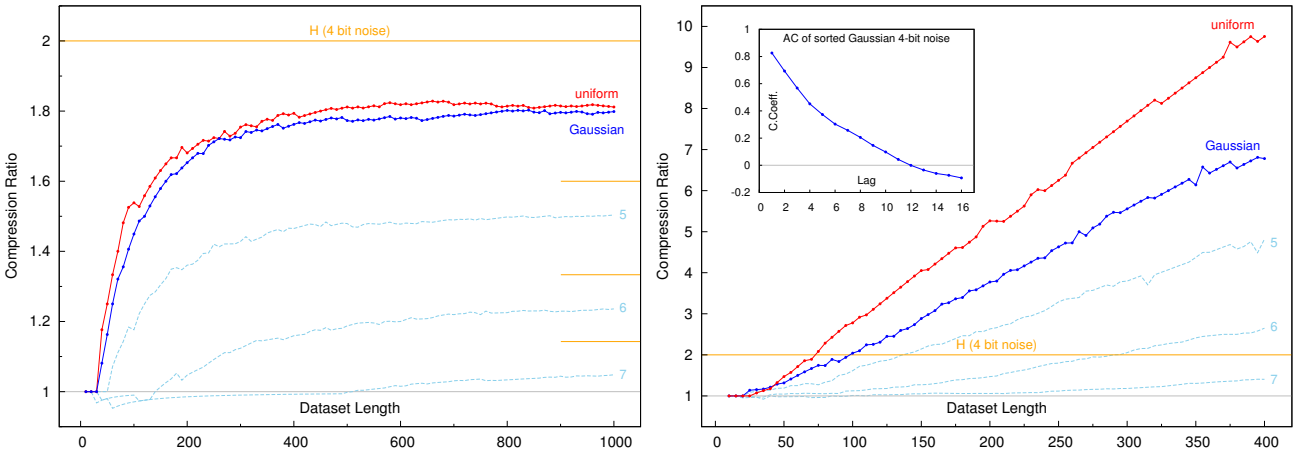


Figure 3.4: **Performance of Deflate on noise.** (Left) Datasets with different lengths containing 4 bits of uniform noise (red line) and Gaussian noise (blue), encoded in 8 bit, have been compressed with *Deflate*. The compression ratio achieved depends on the length of the dataset. The noise is uncorrelated, thus the best factor to achieve is 2. *Deflate* needs a certain amount of data to achieve any compression at all and even if enough data are available, it stays quite away from the entropy limit. (Right) The same datasets were correlated by sorting the values. The algorithm now well exceeds the entropy limit for large datasets. However, again enough data must be available to have efficient performance. The inset shows the interesting correlation coefficients for a sorted Gaussian noise dataset with a length of 64. ρ=0.5 lies between 3 and 4, which is the average size of a run length in the dataset.

Contrariwise a correlated structured dataset can be compressed far better than the entropy limit would suggest, but in the presence of noise this advantage is quickly nullified. For this reason it's much better to go for a statistical method in combination with a preprocessing or decorrelation step for science data.

4 Context Based Compression

In context-based compression, the statistical properties are not known beforehand, but estimated from the data themselves and the already encoded data are used to provide more efficient compression. This is done by looking at the *context* – the neighbourhood – in which a symbol occurs. In the course of compression several such contexts are established and every symbol is looked up in them to encode it in the proper context. A number of algorithms follow this strategy, such as PPM (Prediction with Partial Match) [Cle84],[1] which is used e.g. by RAR [Sal07] and other schemes. I will not explain this one here, but an even more astonishing context-based algorithm, which is the relevant part in *bzip2* [Sal07].[2]

Burrows Wheeler Transform

The BWT [Bur94] is not conceivable as a mathematical transform, it does not manipulate the symbol values, it merely rearranges them. It is also referred to as *block*

[1] The astronomer might mix this acronym with the Piecewise Parabolic Method which is a popular extrapolation for hydrodynamic calculations.

[2] *bzip2* uses BWT and MTF followed by Huffman coding.

sorting, which is a somewhat more suitable designation. Originally published in 1994 [Bur94], block sorting is one of the newer concepts in data compression and it is used in the popular *bzip2* software as well as in *szip* [Sch97]. Sorting itself does not compress anything, it removes the structure of the data, yet the entropy stays the same. But a sorted dataset has run lengths and is thus compressible with basic techniques. However, to reconstruct the original data from a sorted dataset we also need the original index of each symbol. The BWT makes use of dependencies between successive symbols to obtain an almost sorted order from which reconstruction is possible with only a single index value. This sounds like magic and I really wonder how Burrows and Wheeler figured it out, but here is how it works. Note that the entire dataset needs to be available to the algorithm.

Let S be a string of n symbols. We form an $n \times n$ matrix M whose rows are the cyclic shifts to the left. In the next step the rows of the matrix are sorted in the usual way, where two rows with the same start are ranked according to the remaining symbols. The BWT is then the last column L of the sorted matrix, plus the index of the row that now contains the original string S. To get a better understanding of it look at Figure 3.5 and concentrate on L, where the preceding symbols to the sorted ones are found. If the symbols of S are not completely independent, then L contains some run lengths. Just imagine some text that contains frequent combinations such as *ch*, where the letter *h* is preferably preceded by *c*. In the BWT this combination would sort the *h* in the first line of M and thereby concentrate the *c* in L.

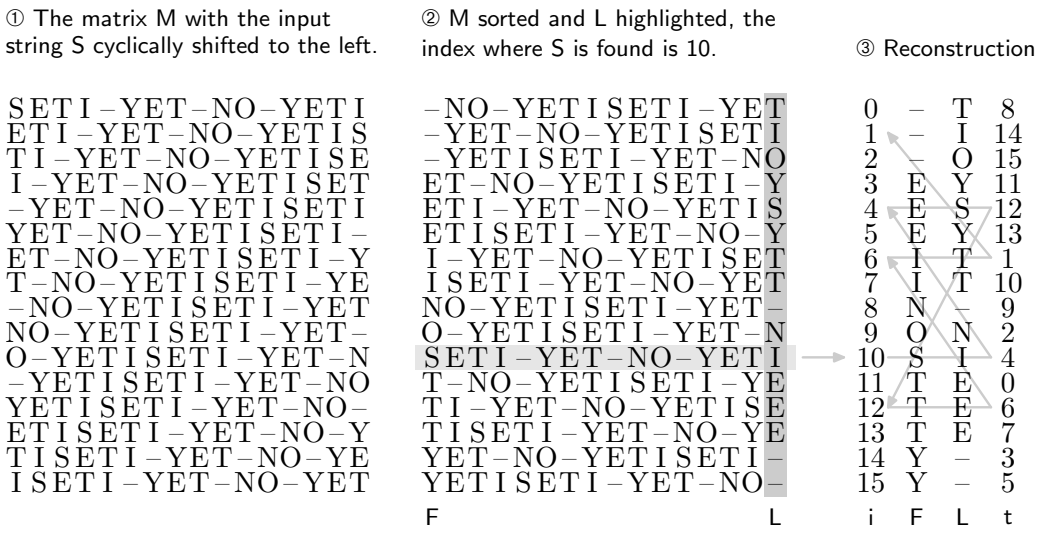


Figure 3.5: **BWT in action.** The input string S="SETI–YET–NO–YETI" is transformed to the string L= "TIOYSYTT–NIEEE—" and the index 10. L contains runs of identical symbols and is thus easier to compress.

As already mentioned before, the BWT is only a preprocessing, it actually does not compress. In their original paper from 1994 [Bur94], the authors propose a move-to-front algorithm with Huffman or arithmetic coding. *szip* for example, uses a range coder [Sch98]. Depending on the data even RLE works well, but also a simple decorrelation or predictor with an entropy coder can lead to good results. The good thing is that even in the worst case of noisy data it does not worsen the situation. The BWT has computational complexity similar to dictionary-based compression methods, but achieves better compression. Its efficiency increases with n, but it needs to be taken

into account that the memory requirements increase with n^2 due to the sorting. This is strongly speaking against its use for on-board processing.

Reconstruction

Reconstruction is a little bit more tricky, but Figure 3.5 should give the necessary support. We only have available L and the start index. The first step is to reconstruct F by sorting L. After that the mapping t is constructed. It contains the indices where each symbol of F is located in L. Two guidelines must be followed for reconstruction: always start at the top and do not consider already looked-up indices. Remember that the symbols in L cyclically precede the symbols in F, so starting at the correct index i we use the vector t to get the predecessor of each symbol. Output the symbol in F at i, update i from t and repeat until S is complete.

Now that we know how BWT works we can also tell that the efficiency of it depends on AC(1). In case of white noise it is of course useless and as it only depends on AC(1) and does not make use of other lags it is not a good decorrelation as well. So the verdict for science data is: if it doesn't help it doesn't do much wrong, except that it uses a lot of resources, especially memory.

5 Compression of Floats

I have added this section for two reasons: astronomical data are sometimes of a floating point data type and there seems to exist no satisfying lossless compression algorithm for floating point data. It is well conceivable because of the way that fractional numbers are represented in the float data type, these data are very hard to compress, as two similar numbers have very different binary representations. For example, the numbers 1.233 and 1.234 differ in 4 bits. A floating-point number according to IEEE 754 [IEE87] is made up of 1 sign bit S, $E = 8$ bits for the exponent and $M = 23$ bits for the mantissa, with a radix of 2. The exponent is biased with $B = 2^{E-1} - 1 = 127$. Thus, a number is represented by $(-1)^S(1 + M) \cdot 2^{E-B}$. This is suboptimal, because the same number can be represented in different ways, e.g. 16 can be $1 \cdot 2^4$, $2 \cdot 2^3$ or $8 \cdot 2^1$ etc. For this reason the numbers are usually normalised, i.e. the exponent is adjusted to get a fraction starting with 1, which can then be omitted (thus the $1 + M$ term). As a consequence, two numbers with the same magnitude will at least have a similar exponent. This is the basis for simple techniques, such as bitwise subtraction or the binary XOR-operation, followed by a binary arithmetic encoder. That way it is possible to achieve a decorrelation that leads to about 30 percent compression.

More efficient methods are based on prediction or on decorrelating transforms as discussed in [Gam04], where an extension to the JPEG2000 standard is presented. Strategies for lossless compression of floats have recently been developed, especially in lossless audio compression [Ghi04], but also in scientific applications [Lin06].

The good thing is that floats usually permit a certain degree of quantisation, as their precision is not equal to the last digits anyway. If the values are not too far apart in magnitude, they might be mapped to integer and then be compressed. Otherwise it is necessary to transform the data before the mapping. Note that all transforms presented in Chapter 2 can be used without modification for floats as well, but special attention must be paid to special floating point values like infinities and NaNs.[1]

[1] In float, infinity is represented by $E = 255$ with $M = 0$ and *not a number* (NaN) is $E = 255$, $M \neq 0$.

4

PACS FM Photometer Software

During the years 1997–2008 the reduction and compression software for PACS was developed under the lead of our institute, with partners from the Technical University and *Joanneum Research*. Several publications were written during this time [Bis00, Ker00, Ott02a, Ott02b, Rei04, Ott04b, Bel05, Ott05, Ott08]. Part of the challenge was to develop a reduction system for a detector whose actual performance was to be known several years later in the future. It was somehow not a big surprise that the *flight model* (FM) generation of detector hardware for PACS gave rise to adaptation of the already qualified flight software. In this and the following chapter the FM reduction and compression system that I developed for the PACS flight model during 2006–2008 is described in detail.

> Time is the fire in which we burn.
> —*Delmore Schwartz, "Calmly We Walk Through This April's Day", 1937*

I start with a brief explanation of the PACS instrument and draw an outline of the problem with the available data rate. After that, an analysis of FM test data of the bolometer is given and the derived on-board data processing steps are discussed. Chapter 5 describes how the processing chain has been established for the spectrometer and contains sections with implementation details for both detector types, especially concerning the back-end encoder.

With the knowledge obtained in the previous chapters it should be possible to follow the described processing steps hereafter, understand why they were implemented and also see where further improvements could be made. Keep in mind that the key in developing on-board software algorithms is to trade the limited hardware resources for scientific information content.

1 Herschel/PACS in a Nutshell

ESA's *Herschel Space Observatory* was successfully launched on May 14$^{\text{th}}$, 2009 with an Ariane V ECA and – being the biggest space telescope ever built – is now

The chapter is divided into the following sections:

about to explore the far-infrared universe with unprecedented sensitivity [Pil08]. Three instruments are carried on board the satellite, HIFI for frequencies ranging from 480–1910 GHz [Gra09], SPIRE for wavelengths between 200 and 670 micron [Gri08] and PACS for the 55–210 micron range [Pog06]. PACS is built by a European consortium led by the *Max Planck Institut für Extraterrestrische Physik* in Garching. It is actually a two-in-one instrument with four independent detector arrays, offering two basic and mutually exclusive observing modes:

- Imaging photometry over a field of view of 1.75×3.5 arcmin^2, with full sampling of the telescope point spread function. Two independent bolometer arrays are used and the bands are 60–85 μm or 85–130 μm for what we call the *blue* channel as well as 130–210 μm for the *red* channel.
- Integral-field spectroscopy between 55 and 210 μm with a spectral resolution of R∼940–5500 (55–320 km/s) and an instantaneous 16 pixel coverage of ∼1500 km/s, over a 47×47 arcsec^2 field of view using two Ge:Ga photoconductor arrays.

From QM to FM

As soon as data from first ground tests of the FM detectors were available, it became clear that the on-board compression software of the qualification model was unfit for the kind of data and that new schemes had to be developed as quickly as possible for the upcoming test campaigns. The biggest problem was that both the bolometer data as well as the spectrometer data were so different to earlier models resulting in by far too high data rates at full CPU load. The main reason for this was that the implemented back-end compression using BWT and a range coder (*pacs_codec*, based on *szip*) [Bel04] had been a bad decision for the noisy data and the decorrelation stage lost too much efficiency concentrating on spatial redundancies. Although the new reduction and compression schemes are still not optimal, up to ∼20 percent above the entropy limit, they are a big step into the right direction making the best of the available hardware.

The first key element in the evolution from QM to FM was to concentrate on temporal instead of spatial redundancies. Analysis of the FM datasets showed that the best predictor for a pixel readout is its previous readout and not the neighbouring pixels. Thus decorrelation of the pixel readouts instead of the frames is the best strategy to deal with the main signal component of a frame, which is for both detector types the *bias*. Ingredient number two was to include an additional quantisation step – rounding, as presented in Section 1.6 – to deal with the fact that the noise is actually oversampled, depending on the *gain* used. The third important change was to implement an entropy coder which also considers large values, such as values produced by chopper transitions and glitches. The modular concept still allows for upgrades during flight to respond to unforeseen matters and to increase performance.

PACS Raw Data Dilemma

PACS with its novel bolometer arrays for photometry and the two Ge:Ga spectrometer arrays has by far the largest number of pixels of the Herschel payload and also yields the highest data rates. In photometry, the array for the shorter wavelengths has 2048 pixels, whereas the array for the red end has 512 pixels. Both are read at 40 Hz with 16 bit precision. This leads to a raw science data rate of 1600 kibit/s.[1] In addition to that, a header consisting of sixteen 32-bit parameters is attached to each science data frame for each array, adding another 40 kibit/s.

In spectroscopy, the two matrices are made up of 25 stacks of 18 pixels for each of the two arrays and they are read non-destructively at 256 Hz with 16 bits precision, producing ramps of normally 1/4 second length. An additional electrical column of 18 pixels is added, so the total output yields 3744 kibit/s. We also have 16 header parameters in spectroscopy, yielding 256 kibit/s. The problem with these data rates is that operational and power budget constraints severely limit communication with Herschel and result in an allowed sustainable data rate of 120 kibit/s.[2] This is typically 16 times smaller than the raw science output of the PACS photometer and up to 40 for the spectrometer. Note that even the raw header data from one single spectrometer would already be too much for transmission! For this reason, a signal processing subunit had to be included in the warm electronics of PACS which performs on-board reduction and compression of the science data. Section 2 below describes the lossy and lossless steps involved in photometry and Chapter 5 has all the details for spectroscopy as well as the back-end encoder that is used for both kinds of data.

On-Board Resources

PACS is a highly computerised instrument, essentially containing four independent computers, partially backed up by redundant units. Two of them are found inside the Signal Processing Unit (SPU), which is where the reduction and compression software is implemented. What all computers have in common is that they are equipped with an 18-MHz ADI 21020 Digital Signal Processor (DSP) and an SMCS 332 chip for communication. The detector arrays are read by the Detector and Mechanism Controller

[1] Binary prefix: 1 kibit = 1024 bits, SI prefix: 1 kbit = 1000 bits.

[2] Daily ground contact with the New Norcia Station is limited to 2 hours per 24 hours with a data rate of 1.5 Mbit/s. Taking the factor $(24-2)/2$ into account as well as protocol and housekeeping (diagnostic values) overheads leads to the 120 kibit/s of sustainable science data rate.

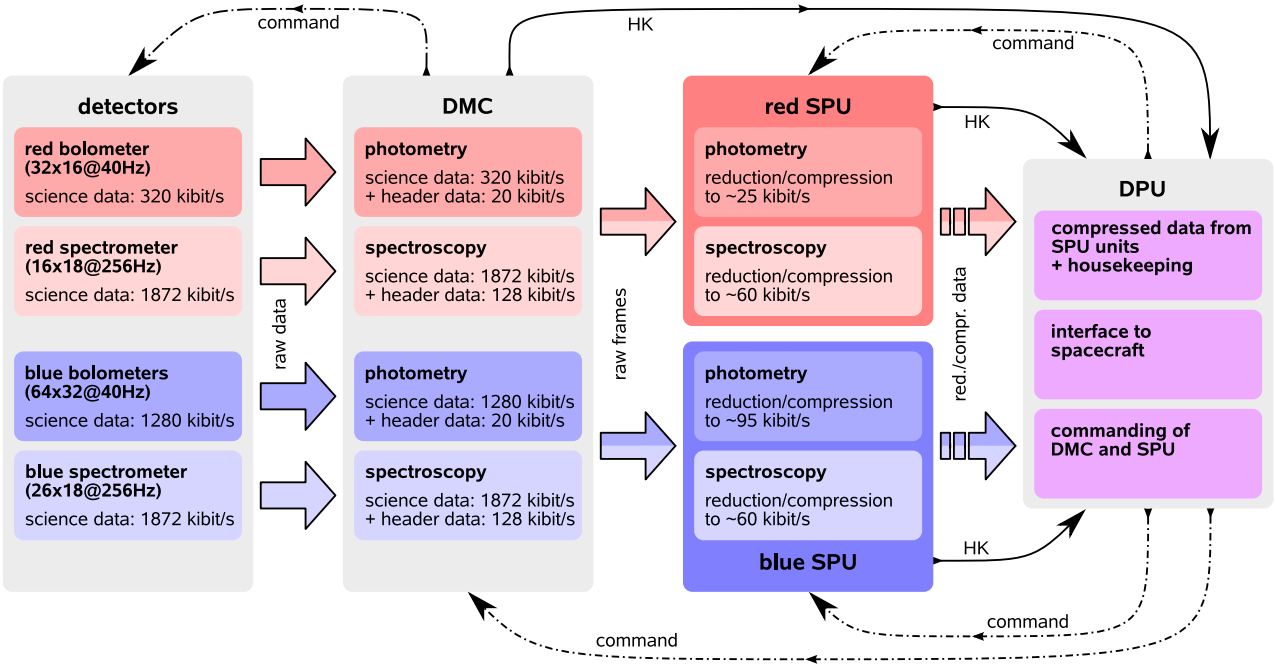


Figure 4.1: **Data flow inside the PACS warm electronics.** From left to right: either the bolometer arrays or the Ge:Ga arrays are used at a time, depending on the mode of observation. Their data are read by the DMC, which structures them into data frames. Then reduction and compression is done by the SPU units, where a compressed data stream is produced and sliced into packets that get wrapped by the DPU to become standard telemetry packets. Aside from the nominal science data flow, each subunit is obliged to send housekeeping at intervals of ∼2 seconds.

(DMC) built by *Centre Spatial de Liège*. The data are structured into frames, attached with a header consisting of 16 32-bit words and sent to the two SPUs, one for the *red* and one for the *blue* channel. This header has parameters to identify the observation and information about the instrument setup. Among them is also the compression mode that should be used by the SPU. The SPU reduces and compresses the data according to the data type, compression mode and the contents of on-board parameter tables, which are loaded during instrument set-up. More details about this can be found in the aforementioned publications, especially in [Ott04b, Ott08]. The SPU software is what was contributed by the University of Vienna[1] and the SPU hardware was built by CRISA who were contracted by the *Instituto de Astrofísica de Canarias*. Each SPU has 4 MiB of data RAM and 3 MiB of program memory. This means that only a few seconds of raw data can be buffered. After the data have been processed in the SPU units they are shipped in packets to the Data Processing Unit (DPU), which has an interface to the spacecraft and also acts as the instrument command centre. The DPU hardware is built by *Carlo Gavazzi* under *Istituto di Fisica dello Spazio Interplanetario* (IFSI) contract and the DPU Software is also contributed by IFSI.

Figure 4.1 shows the four on-board computers that are involved in nominal data flow inside PACS. Regardless whether the instrument is operating in photometry or in spectroscopy, the data flow of the red and the blue arrays goes separate ways through dedicated DMC and SPU units. Only the DPU, where everything comes together before storage in the solid state mass memory of the spacecraft, handles both channels. This warm electronics chain is backed up by a redundant counterpart, in case that the hardware of a nominal unit becomes damaged in any way.

[1] Our partners were the TU Vienna and Joanneum Research in Graz. For credits see the epilogue.

2 Photometry with the FM Bolometers

When PACS is operated as a photometer, the two bolometer arrays are in use [Bil06]. The many-stage readout electronics of the bolometers allow for different bias and gain adjustments, but most importantly, the gain applied at the warm electronics control box determines the magnitude of both, signal and noise. Just think of your stereo – if you increase the volume you also amplify the noise. In PACS we either set a low gain for observing bright sources or high gain to maximise the sensitivity. As this has an impact on the dynamic range the bias needs to be adjusted to the gain as well. Different ways of reading the detector elements are also available, leading to varying output data types that are essentially distinguished whether they are signed or unsigned.

Bolometer Data

The photometers can be seen as movie cameras and the generated data are thus easily understandable. However, in practice the detector technology has some properties that prohibit the use of popular movie compression schemes, such as the MPEG standards. First of all, we are talking about science data and thus we have noise that must not be discarded. In the FM we will see that the readout noise can vary between 3 and 8 bits at signal levels between typically 20000 and 60000 ADU. Figure 4.2 gives you a preview of a characteristic data frame from the blue array. Above all, its look is dominated by the bias pattern. Even after bias subtraction, the variation between neighbouring pixels is still bigger than the readout noise owing to varying sensitivity. Therefore we can quickly guess that it is better to concentrate on the temporal redundancies rather than on cosmetic aspects within the frame, an assumption that will be substantiated on the next pages. The plots in Figure 4.3 show how a pixel signal typically looks like. The drift is in part due to $1/f$ noise and in part to instrumental effects. Histograms of the noise reveal its Gaussian nature and facilitate estimates for entropy encoding. You are now invited to speculate about what kind of processing or combination of operations will reduce the size of the data by a factor 16 yet retain as much information as possible. There are many strategies that will work for now, but be warned – most of them fail as soon as the chopper comes into play. Now let's go on for a more detailed analysis.

> Hello Roland,
>
> According to our French colleagues, the following data are representative for flight and should therefore be compressed within the allocated bandwidth:
>
> Day 20070331: File = FILT_Phot_spu_datarate_20070331_01.tm
>
> Best regards,
> Helmut
>
> *—email received on the 9^{th} of April in 2007*

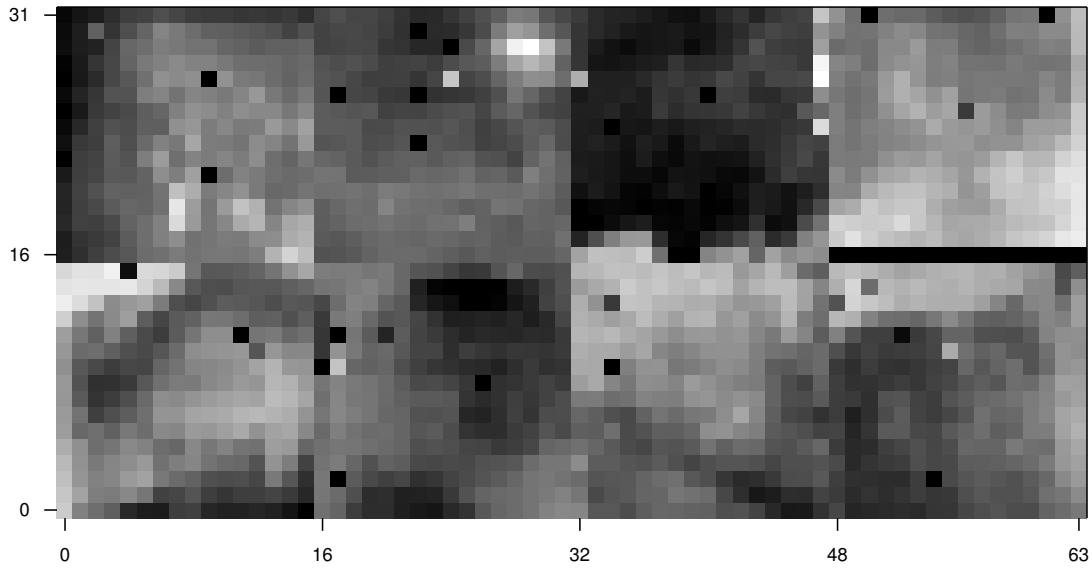


Figure 4.2: **Appearance of a single frame from the blue bolometer array.** A single frame is dominated by the bias pattern. Even a strong point source is not easy to spot without background subtraction. Different sub-matrices that constitute the full array are easily distinguishable. The level of brightness maps the 16-bit range to black=0 and white=65535. Clearly visible is a line of dead pixels on the right side. In total there are ∼25 dead pixels, which is slightly more than one percent.

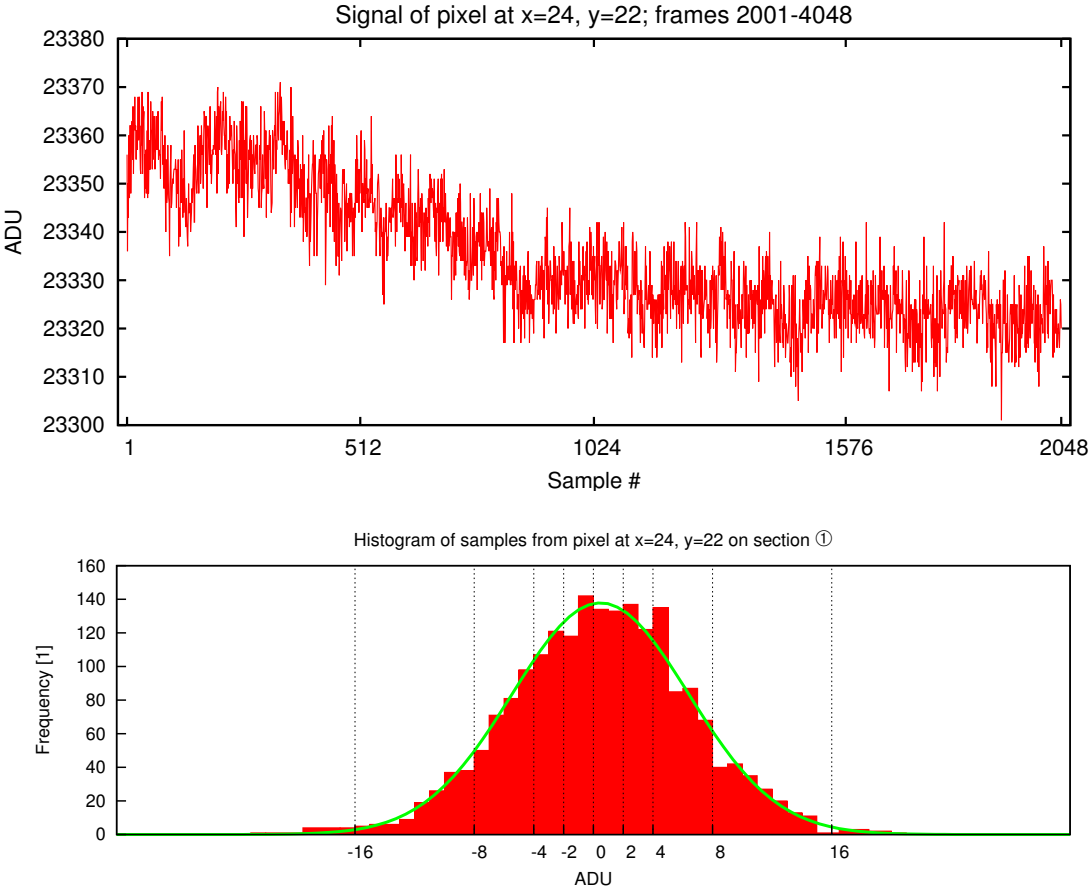


Figure 4.3: **Typical signal for a single bolometer pixel.** The data shown correspond to 51 seconds of data. Below is the noise histogram (not the histogram of the data), fitted with a Gaussian distribution of the same variance. The entropy for this pixel is 4.56 bits/sample.

3 Analysis of the Data

Representative test data are vital in the design of a tailored data processing system. These were not available before 2007 to us and so any reduction and compression that we had implemented before was finally put to the test when the data arrived. Unsurprisingly, we quickly figured out that major modifications had to be made.

The *Flightdata* file

The file *FILT_Phot_spu_datarate_20070331_01.tm* is the result of a test that was made through a generic test observation procedure. Its data are said to be what we expected at that time to have in space, but the data rate is between 140 and 190 kibit/s, which is too high above the aimed bandwidth. The file contains 736 compressed entities[1] for the blue channel and 185 for the red one. Decompression provides 22058 blue reduced frames (averages of 4 raw frames) and 22086 reduced frames from the red array. For each array 3 rotating pixels of additional raw data are also available.

The SPU data reduction was performed in the *default compression mode*[2] with the so-called *velvet* FM compression scheme, which is already adapted to the FM data as described in Section 4, yet the data rate is still too high. In this chapter I will show why this is the case, come up with ideas about what can be done better and what the finally implemented solution is. The analysis concentrates on the blue channel only, since it contributes 4/5 of the data rate. Figure 4.4 displays the accurate numbers of the compressed science data and reveals the 6 sections of different instrument setup.

Data Rate Limits

Ideally, the data rate should be around 120 kibit/s, which is the number you get if you divide the DTCP downlink by the rest of the day, subtracting what is allocated to housekeeping. This number however, cannot take into account the overheads in the actual execution of the observation and so the limit is softer in practice, more likely around 135 kibit/s. The instrument has also no problem to swallow a data rate of 185 kibit/s, but above that a different bus profile in the DPU-spacecraft communication interface – *burst mode* – has to be used, otherwise there will be loss of data. In burst mode data rates up to ∼300 kibit/s go through, but of course this cannot be a basis for sustained data generation.[3] Coming back to Figure 4.4, it's okay if the data rate exceeds 120 kibit/s as long as everything stays well below the blue horizontal line, which is not the case here. Although only the blue channel was examined, the estimation of the total size of both was made by multiplication with 1.25.

Typical Pixels

Just to see what the data look like, a pixel can be picked (in Figure 4.5 it's three of them) to plot the signal for all the frames in the file. Comparing data rate and signal (Figure 4.4 with Figure 4.5) reveals very good agreement of the 6 sections. One can easily guess that sections ① to ③ and ④ to ⑥ do the same thing (start staring on chopper position A, then do the chopping, end on position B), but with different detector settings. Basically, noise forms the biggest part of the data rate, but the signal also contributes, as can be seen by comparing the sections ② and ⑤ (chopping) with the other sections, where just staring is performed.

[1] A *compressed entity* is composed of a few seconds of compressed headers and data. It reflects the way the data are buffered and processed by the SPUs.

[2] Four samples are averaged, all pixels are selected and the data type is a 16 bit unsigned short.

[3] The numbers provided are rough estimates that I found during tests with the FS.

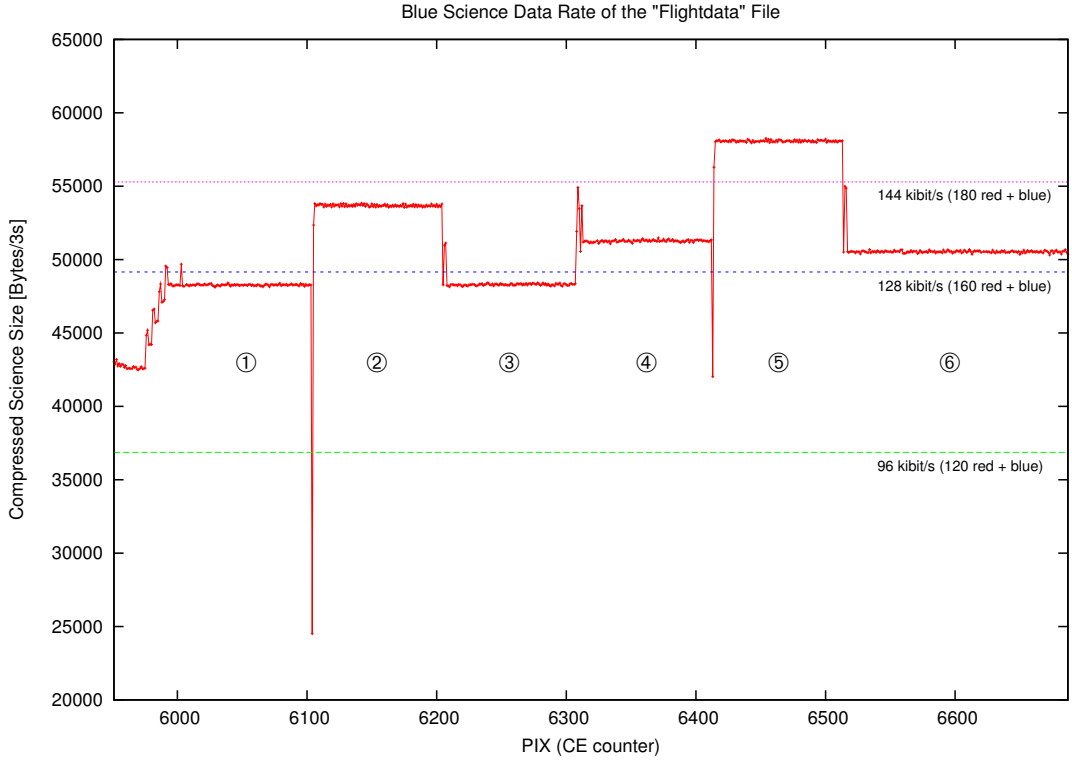


Figure 4.4: **Data rate of compressed science data.** The plot shows the size of the compressed science data without additional raw channels, DMC headers and the PUS TM packet overhead. All these make up for an additional 2–3 kibit/s here.

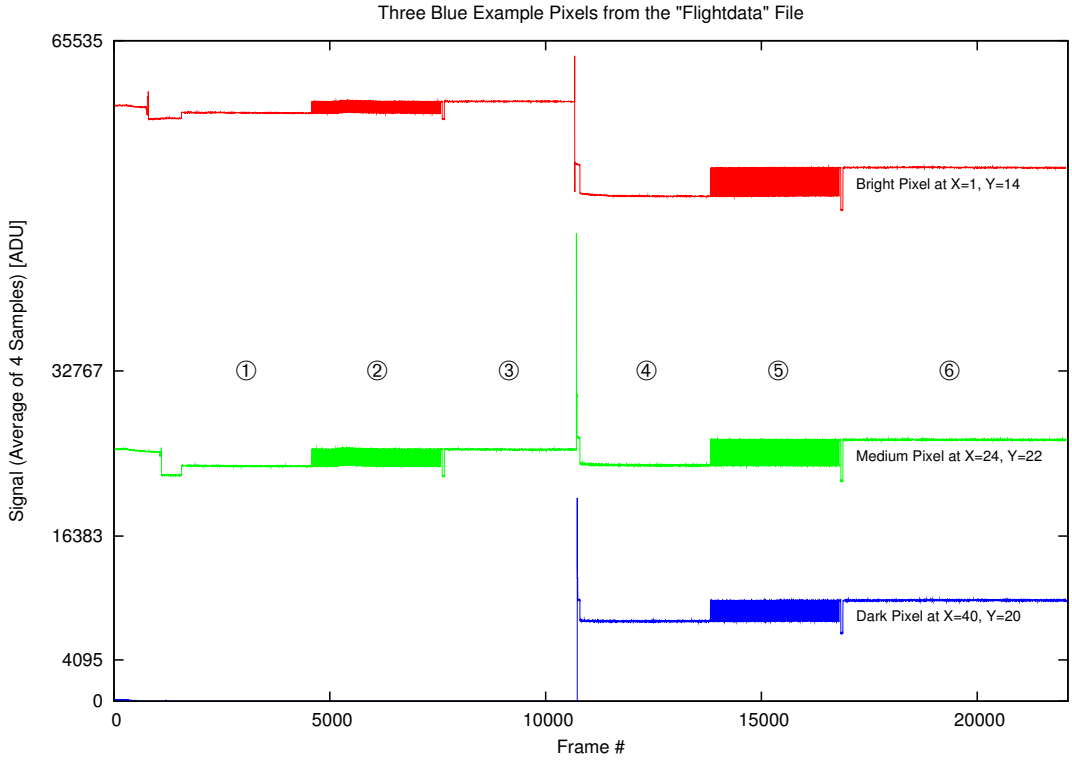


Figure 4.5: **Signal of three example pixels.** Three pixels have been arbitrarily selected to cover high, medium and small numerical values. Note that the measurement lasts for ∼40 minutes.

The first three sections also differ from the rest in another way. Obviously, some dark pixels (see the blue line in Figure 4.5) are also brought into play with the different bolometer settings (presumably the bias) and now have their share on the telemetry. Even from the three pixels picked for Figure 4.5 (by the way, their location is marked in Figure 4.6), some educated guesses can be made. First of all, the difference of the three pixels mostly comes from the bias offset. Another thing is that a comparison of the bright (red) and the medium (green) pixel on section ② shows that the readout value is not a measure for sensitivity, because the green pixel responds stronger to the signal than the red one (compare the dynamic on section ②). A few pages down the noise of the staring sections will be studied, but for now let's concentrate on the 2D image.

The Image

If neighbouring pixels tend to show pretty much the same signal, then this correlation can be exploited during compression. Figure 4.6 shows frame 1 (this one is as good as any one of the 22058 frames) of the dataset. One point to investigate now is whether the spatial (2D image frame) or temporal (1D pixel signal) similarities are bigger. Although by simply looking at the image one may conclude that a pixel does not have too much in common with its neighbours, it could well be that at least their sensitivity or even their noise is similar. The next frame in sequence (Figure 4.7 left) is pretty much the same, which is what we expected because we are already aware of the bias pattern. To see how big the similarity is, their difference is calculated and another plot is made (Figure 4.7 right), still showing hardly any distinction in the 16-bit range of values. Setting the cuts to the lowest 6 bits and redoing the plot for Figure 4.8 finally reveals the noise. This range setting corresponds to an amplification of 1024.

Space vs Time

With these images we want to answer whether a single value has more in common with its neighbourhood or with its own previous values. To do so, a single difference frame is not enough, but we need to create a *noise map* by averaging all difference frames that belong to the same section and plotting the standard deviation. This is of course a bit fishy, because as we know from Chapter 1, the σ will be overestimated by a factor $\sqrt{2}$ in case the noise is uncorrelated. The problem of detector drift is also not sufficiently treated that way.[1] The results for two sections are shown in Figure 4.9 and 4.10. So, *on average* we would have to decorrelate and encode such an image if we decided to concentrate on the spatial redundancies. I guess your already trained eye has some justified concerns about this idea.

Spatial Redundancies in the 2D Signal

It's time to find out if there is a systematic signal left in a difference frame. For that purpose a plot of the difference image data seen as a 1D signal is made. For the support of interpretation, we will use autocorrelation and the modulus of the Fourier transform. In a dataset of uncorrelated noise we will only find a Dirac impulse at 0 lag and some *grass* at other lags in the AC. Note that in the modulus plot the *half* amplitude is given.

[1] One of my favourite ways to correct a drift for determining the noise is to subtract a few low-pass scales of the *à trous* wavelet transform [Hol89]. In this case a varying background is taken out leaving the higher frequency noise untouched.

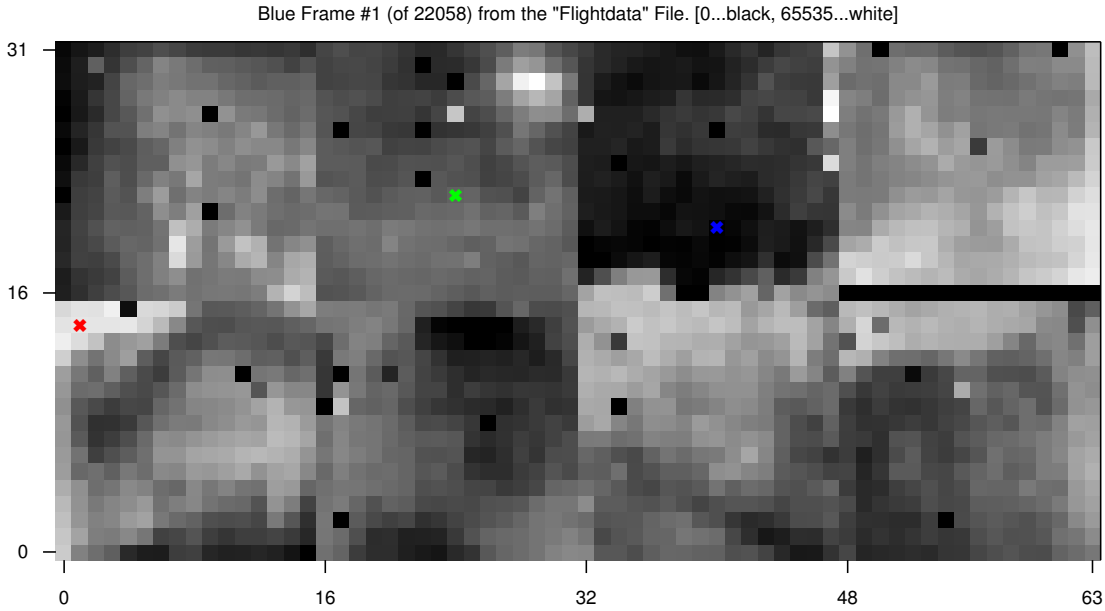


Figure 4.6: **A single frame.** This plot shows frame 1 of the reduced data with indicators of the three pixels from Figure 4.5. Black and white linearly span the 16 bit range of values.

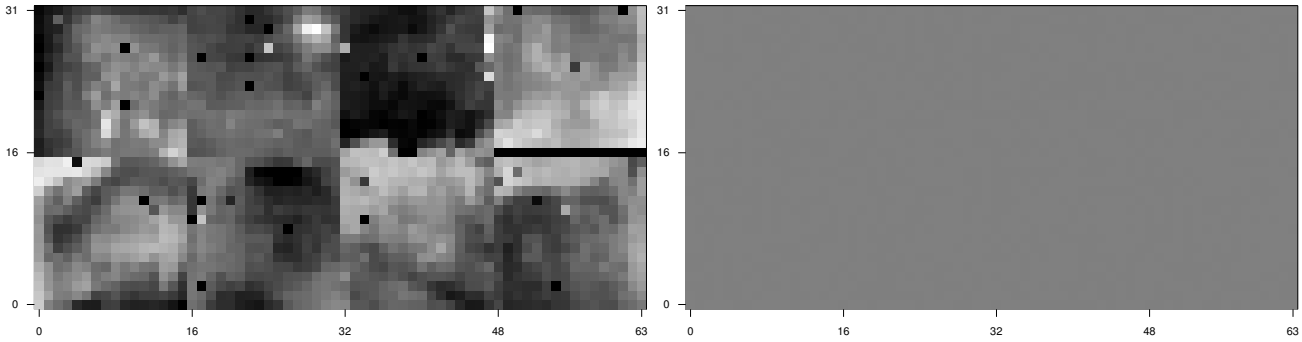


Figure 4.7: **The next frame.** (Left) Here is frame 2 seen as an image. The difference to frame 1 is hardly recognisable. (Right) The difference between frame 1 and 2. Still, a challenge to see...

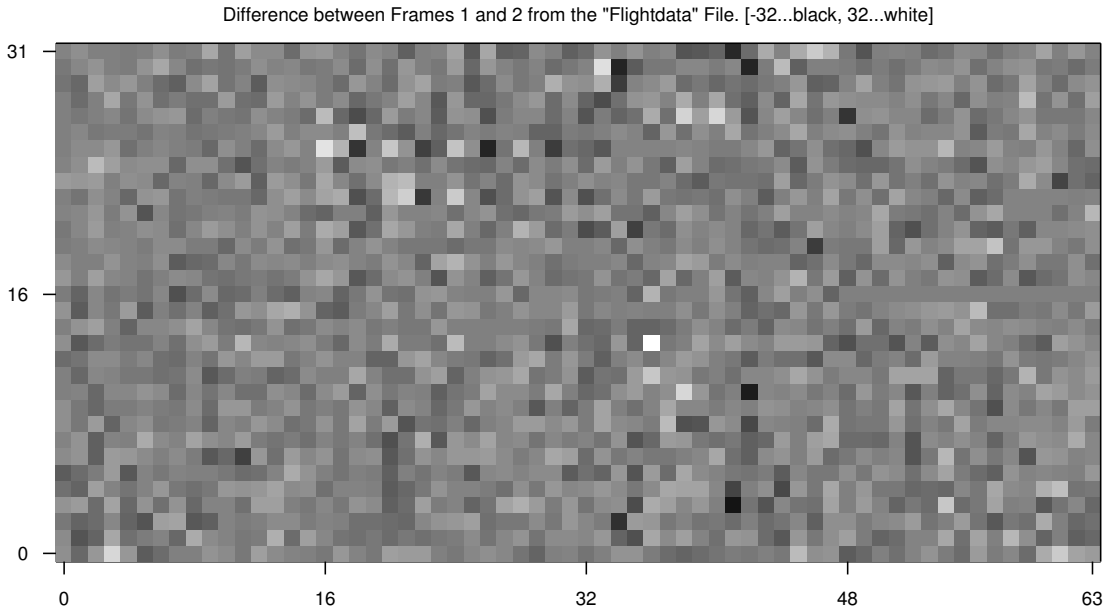


Figure 4.8: **Difference frame with different cuts.** Adjusting the brightness reveals a noise pattern.

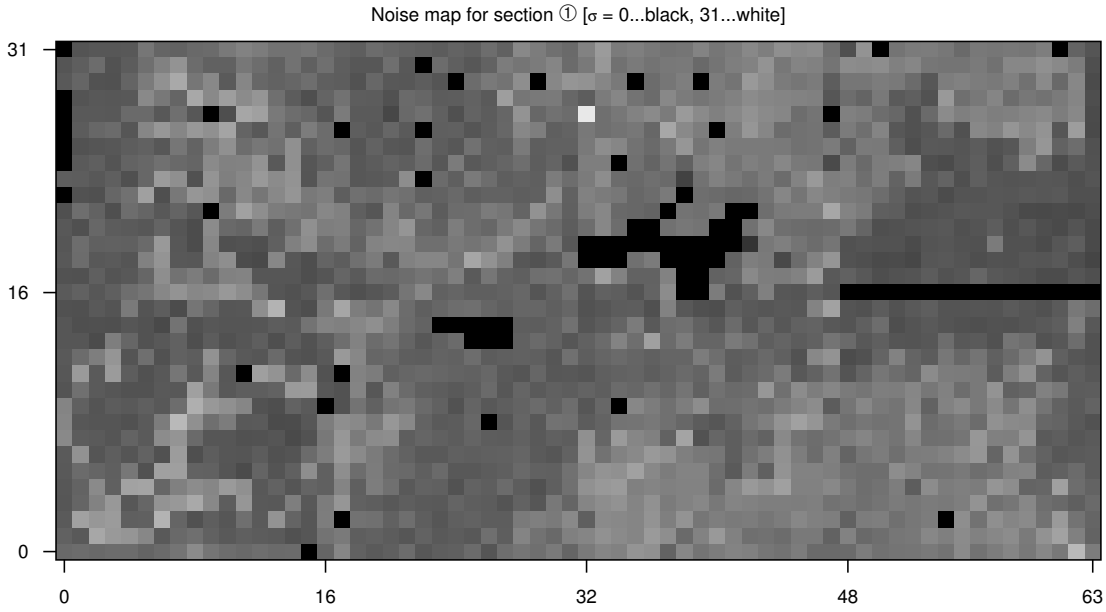


Figure 4.9: **Noise map for section ①.** The mean $\sigma = 8.483$ in the plot becomes 5.998 if corrected by $\sqrt{2}$. Note that the "dark" pixels neither have a signal nor any noise.

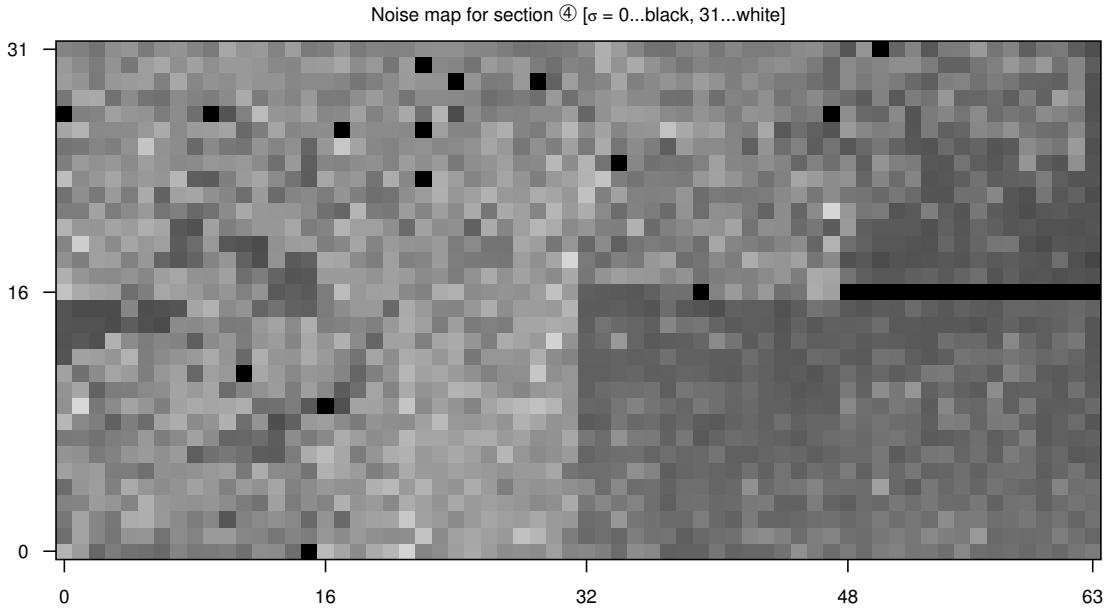


Figure 4.10: **Noise map for section ④.** It doesn't look much like the map for the first section, because of the different bias settings. $\sigma = 11.003$ (the corrected value is 7.780). The previously *dark* pixels that are brought to life by the different detector settings can now be identified, but the cost for this is that the noise has also increased quite a lot.

Figure 4.11 shows the difference frame if it was read from left to right and from top to bottom. A close look makes us spot the 16 dead pixels with zero signal right in the middle of the plot. The autocorrelation shows what was expected: there is no redundancy in these data. The modulus has a peak at a period of 8 pixels, which is by far too weak to attract any attention.

Order and Colour

However, there is a problem with this plot. Reading the image from left to right and from top to bottom is not what the bolometer output looks like. The detectors are simply not read that way and the SPU does not get the data ordered like that, but in a more complicated way. Essentially the eight sub-matrices are read in a fancy way to be compatible with *the three ways of Saclay's fuzzy logic* [Sau09]. Figure 4.12 is the same plot made for the same data, but in the order the SPU gets them. Even here, there is more or less the same story to tell: no signal, no redundancies, just noise of approximately white colour. The missing piece to be inserted now is the histogram.

Entropy of a Noise Frame

Fortunately, the histogram does not depend on the reading order. A quick look at it (Figure 4.13) is enough to suspect a Gaussian-like distribution. A comparison with the calculated μ and σ shows acceptable agreement. For an estimate of the compression of these data we determine the entropy as explained in Section 1.2.

We remember the data will be encoded down to the entropy limit if they are uncorrelated (otherwise the limit can be beaten!) and there are no overheads in the symbol/probability encoding. Ideally, we will use arithmetic coding to achieve this, but if we have to use fixed Golomb or Huffman codes, then the compression will perform very likely around $H_S + 0.5$ for each symbol to be encoded. It is trivial now to calculate the entropy for the difference frame, which is $H = 9203.29$ bits ($H_S = 4.494$ bits per sample).[1] Remember that of all the methods presented in Chapter 2 differentiation was simple yet quite effective. In this spirit we can use the obtained H to estimate what a compression consisting of frame differentiation and entropy encoding would achieve.

A First Encoding Attempt

Encoding a 3-second buffer of such data by leaving the first frame untouched and generating difference frames of the remaining 29 averaged frames would result in 2048×16 bits (for the first frame) + 29×9203.29 bits (for the difference frames) = 299663.41 bits. In 1 second, that would be slightly less than 100 kbits for the blue channel only. Adding the red channel adds up to ∼125 kbit/s. Considering the increased symbol entropy due to a fixed codeword size yields ∼140 kbit/s. These numbers can be compared with the data rate that has actually been achieved. In Figure 4.4 it's ∼140 kibit/s (red+blue) for the first unnumbered short section, which is the one we drew our data for this calculation from. This leads to the assumption that the actual encoding has an overhead of half a bit per symbol, although it uses arithmetic compression. Further down it will be shown that most of this comes from an additional step in the decorrelation stage which is needed for chopped observations and the encoding is performing slightly above the entropy limit due to some trade-offs that had to be made.

[1] Note that these data are already averaged frames, where four raw frames were reduced to one, so the noise in the original data was even a factor $\sqrt{4}$ higher, but on the other hand we increased the noise by $\sqrt{2}$ by the differencing, so originally, the noise is half a bit higher in the raw frames than in the delta frames.

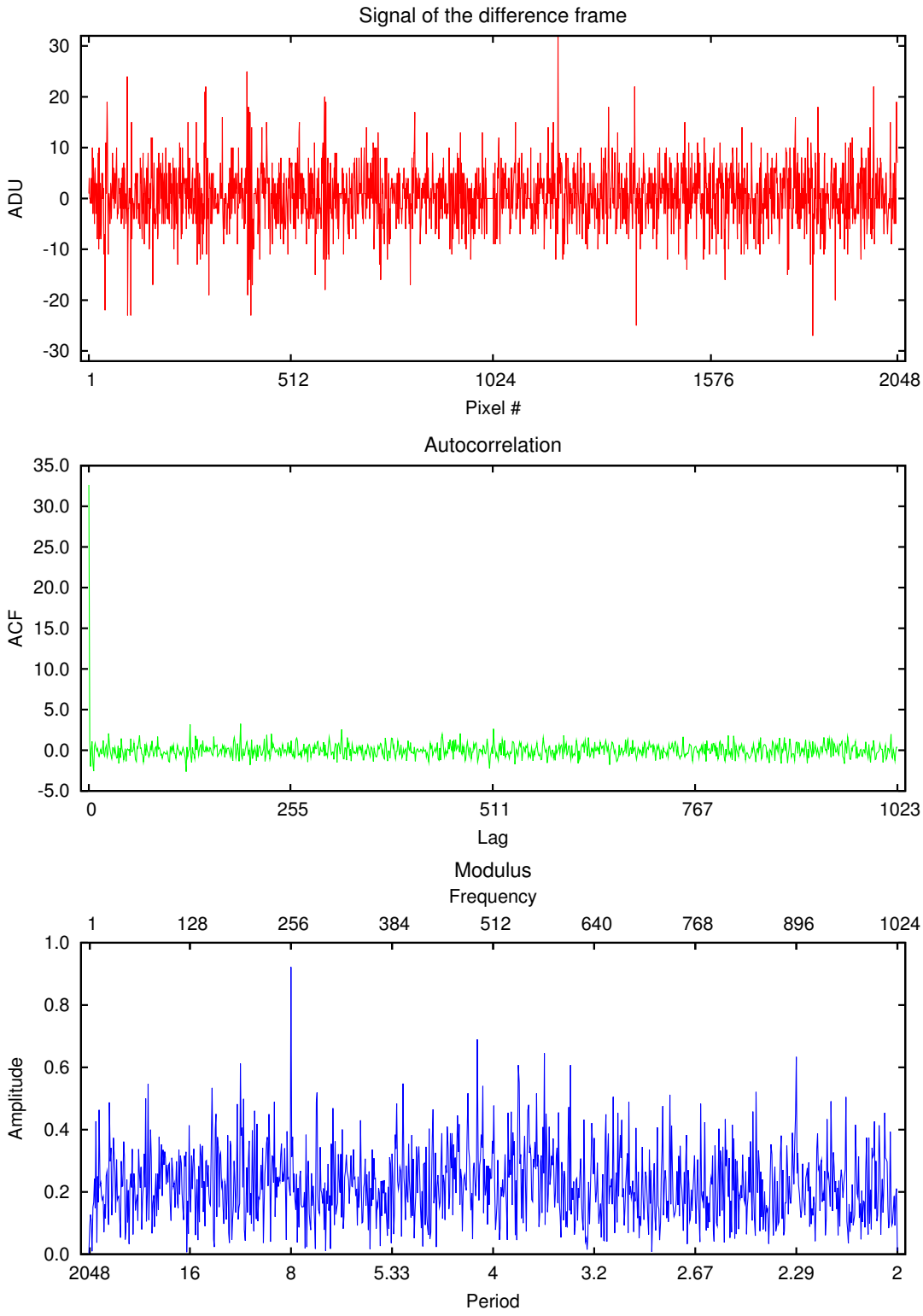


Figure 4.11: The difference between frame 1 and frame 2 plotted as a 1D dataset in *text reading order*.

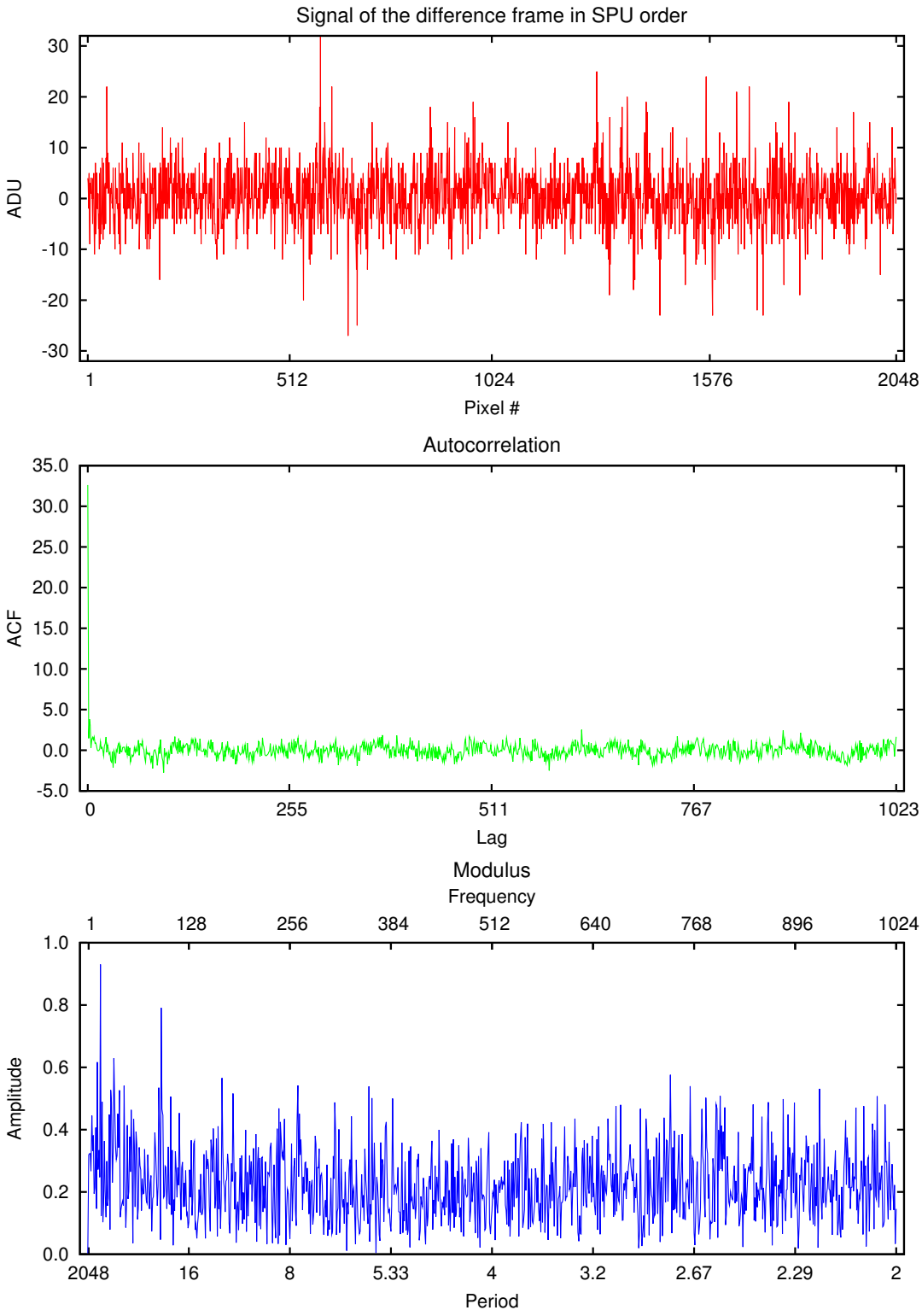


Figure 4.12: The difference between frame 1 and frame 2 plotted as a signal in *spu order*.

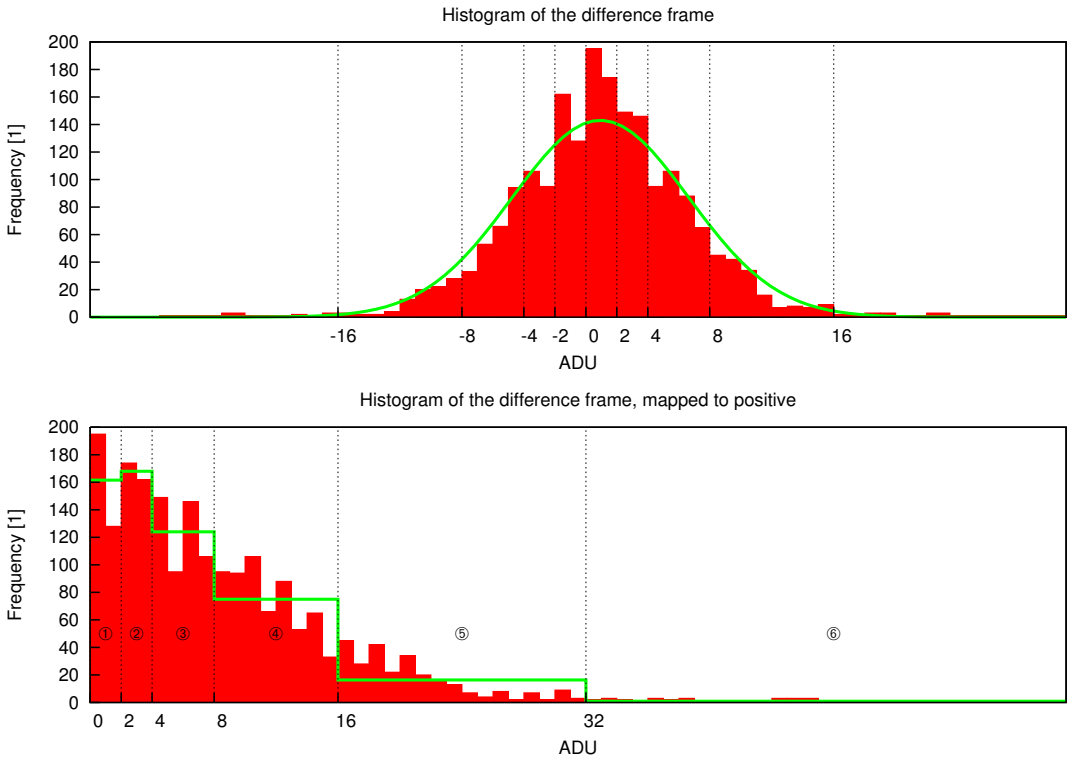


Figure 4.13: **Histogram of the difference frame.** A Gaussian error distribution curve with the derived μ=0.442 and σ=5.712 is in good agreement (green line). The ticks on the x-axis have been chosen to resemble a binary encoding. The second histogram has been mapped to positive: a positive number a becomes b=2a and a negative number is mapped like b=−2a−1. This is done because it is obligate to encode positive values in many algorithms. No, it does not increase the entropy. The green line here is the mean number of elements on a binary digit. The encircled numbers give the number of bits needed to encode a number from that bin. Find out more in the text.

The Signal on the Sections

Now it's time to concentrate on the temporal characteristics of the data. For each of the signal plots, 2048 frames have been taken (for direct comparison with the spatial analysis) from one of the sections. The first plots to be made with that are the bright and the medium pixel on section ① (see Figures 4.15 and 4.16). With this scale of the ordinate quite a drift comes out of the staring data. The autocorrelation impressively shows how much this drift dominates over anything else. Finally, the modulus shows no serious periodicities.

Making a histogram out of these data to estimate the noise would be meaningless because of the contained drift, unless the data are properly preprocessed. In this case I decided to subtract the mean from the data on slices of 30 frames, which is three seconds of averaged data,[1] and by making a histogram out of that, which can be considered an acceptable method because $\mu \approx 0$.

[1] This is equivalent to the size of a compressed entity for the blue bolometer. Since the SPU has only a few MB of memory and must process the data in pieces, 120 raw frames (3 seconds) are buffered and processed to 30 reduced frames (averages). As a consequence, the SPU cannot see redundancies over a longer time than 3 seconds. The red channel has less pixels and the respective SPU can thus buffer 12 seconds of data.

The derived histograms (Figures 4.17 and 4.18) show again a Gaussian noise distribution. The *bright* pixel has a smaller σ and will need less bits to encode. The entropy for the *bright* pixel is $H = 4.246$ bits per sample. The *medium* pixel has $H = 4.596$. Here is another estimate of the data rate: 2048 (pixel) $\times$ (16 (the key frame) + 29 (frames) $\times$ 4.5) = 300032 bits (again $\sim$125 kbit/s for red+blue), which is a similar number as in the spatial counterpart. However, this estimate does not take the signal (the drift was eliminated by the slicing process) into account. Such a drift is easily taken out of the data by calculating the differences between the frames and we know that the uncorrelated high frequency noise is increased by a factor $\sqrt{2}$. This adds half a bit per sample and the estimated data rate becomes more like $\sim$145 kbit/s. Taking a fixed symbol entropy as a base for our calculation finally leads to $\sim$160 kbit/s. Again, this can be compared with the data rate overview (Figure 4.4), where we see that the data rate of section ① is around 155 kbit/s for red and blue.

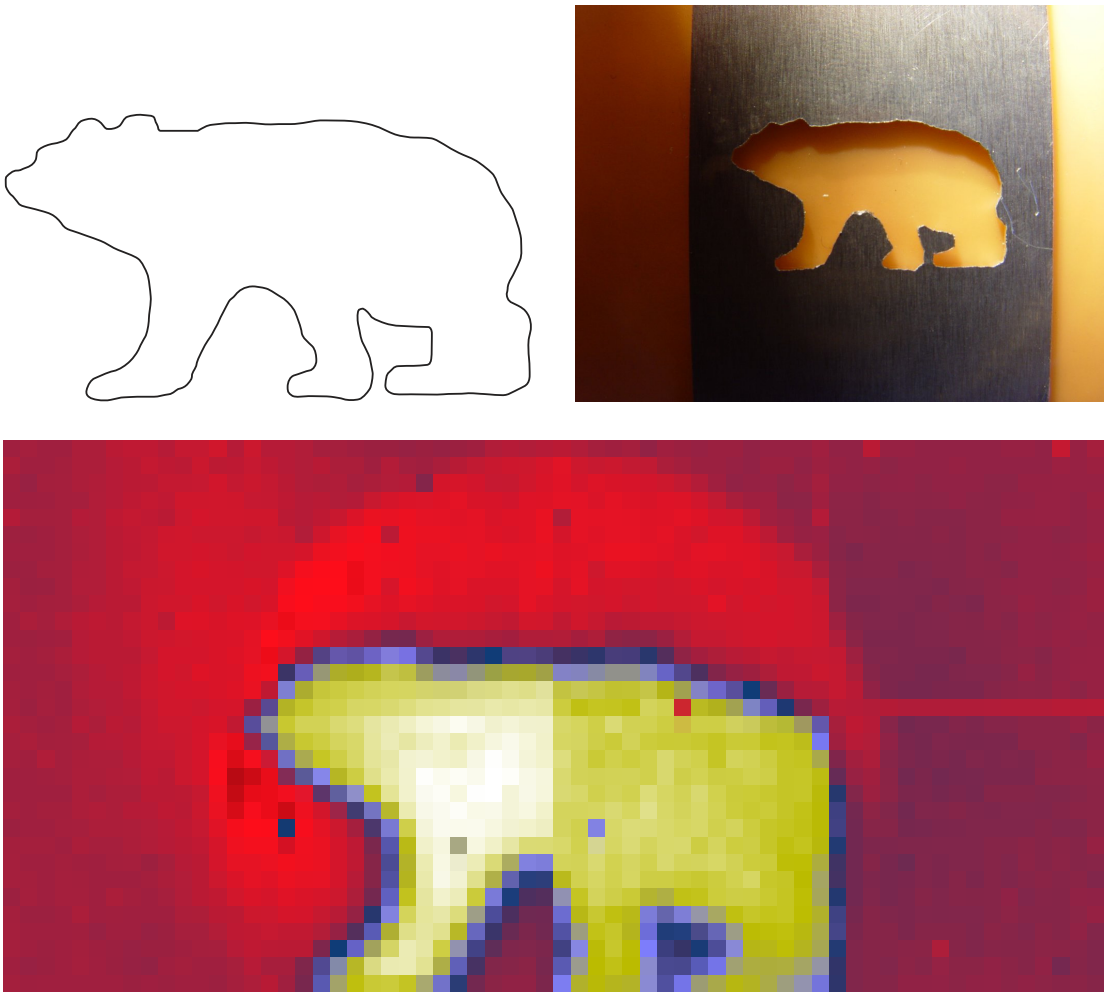


Figure 4.14: **Polar bear measurements.** On the 22nd of December in 2006, the last day of the *FM-ILT cold – part I* tests of PACS, the consortium mascot *Pacsi Bear* was used to get a more complex image onto the arrays. As a by-product of these measurements, the so-called *bear crosstalk* was discovered. (Top left) The original design. (Top right) The fabricated aperture that was moved with an external XY-stage in front of the cryo window. (Bottom) The background-subtracted bear measurements of the blue array. Images provided by Albrecht Poglitsch and Eckhard Sturm.

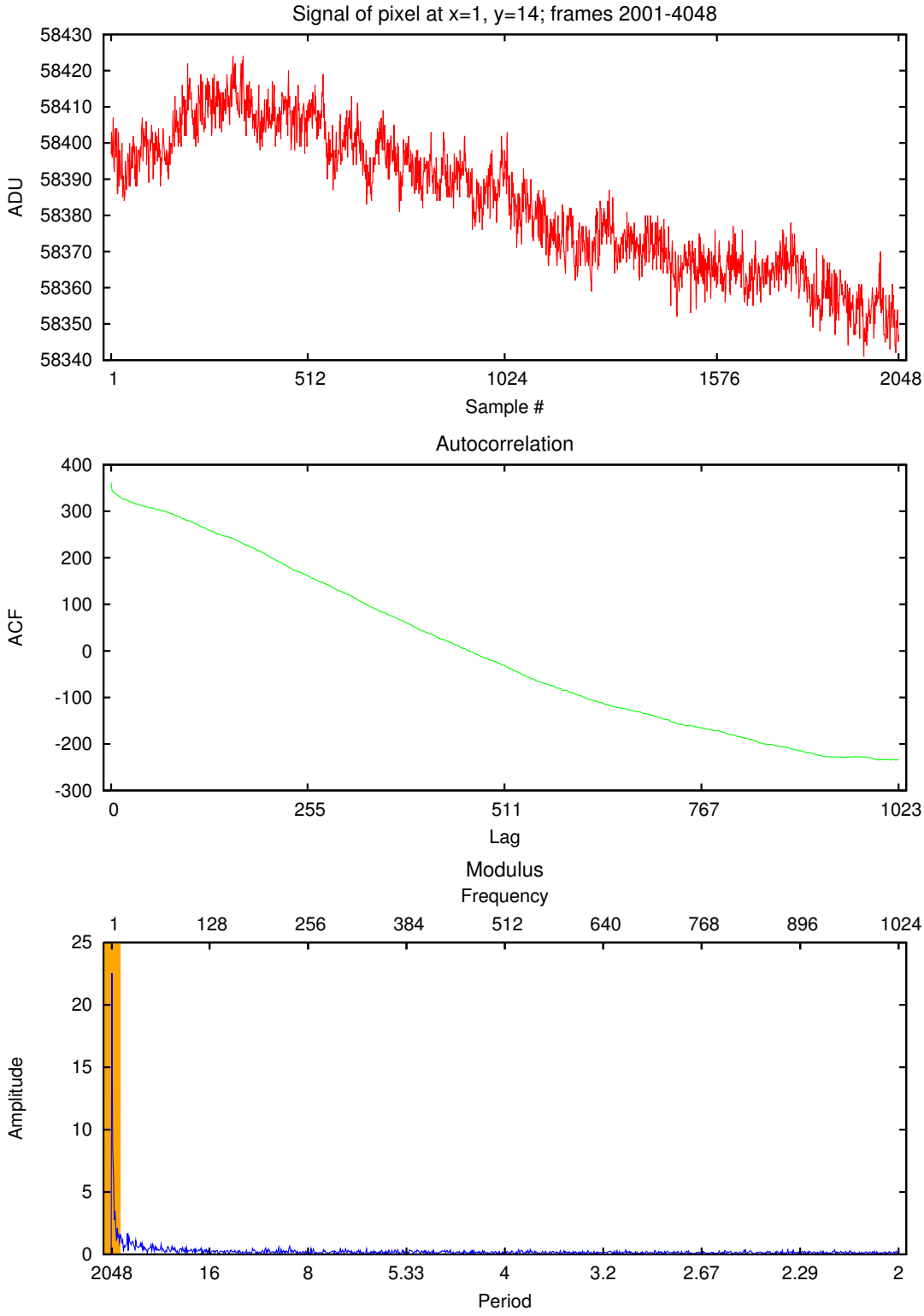


Figure 4.15: **The signal of the bright pixel on section ①.** Periods in the orange segment are not accessible by the SPU, because the data buffer is too small.

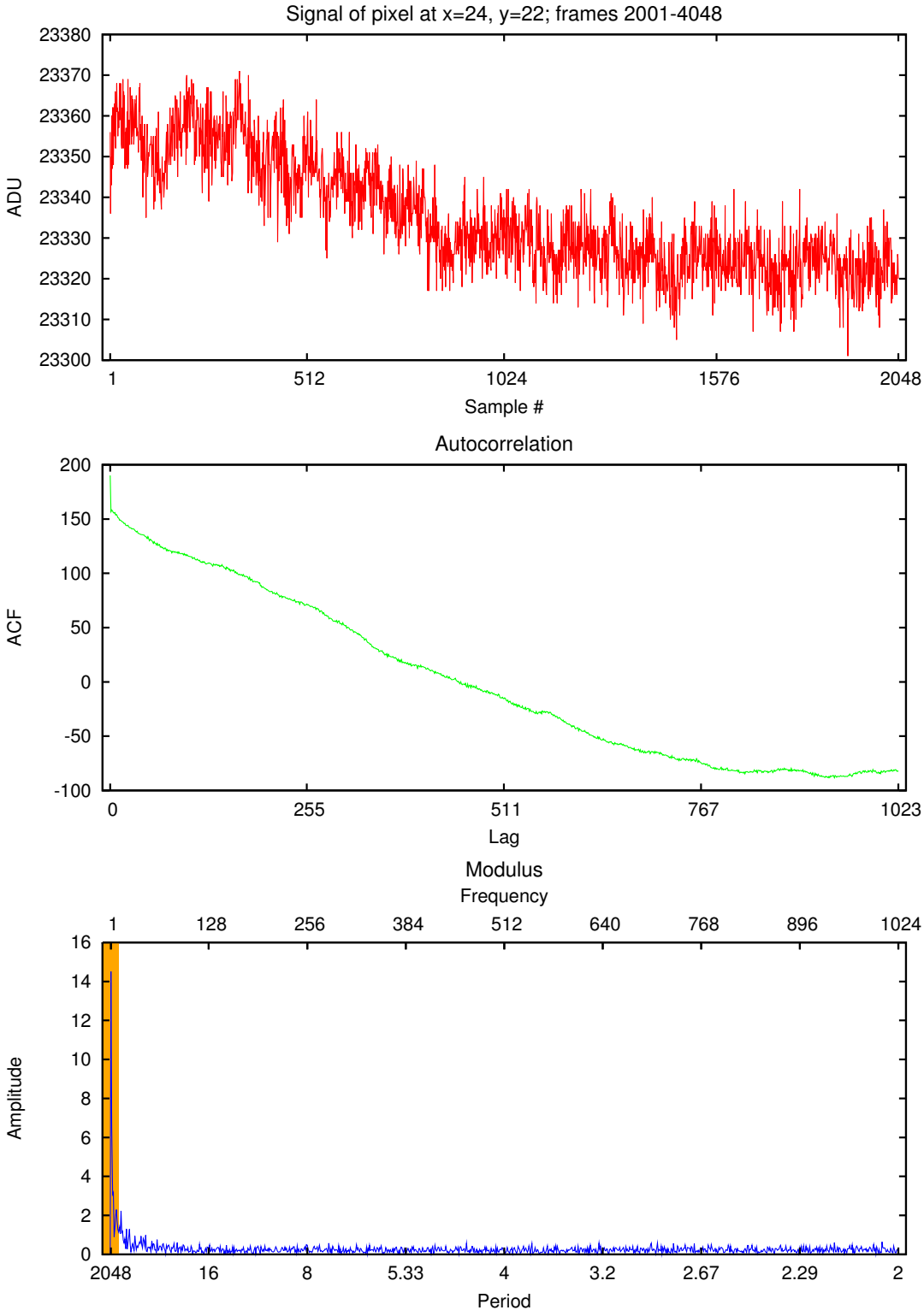


Figure 4.16: **The signal of the medium pixel on section ①.** The low frequency drift follows the same trend as the *bright* pixel, which indicates that this is a systematic component of the signal.

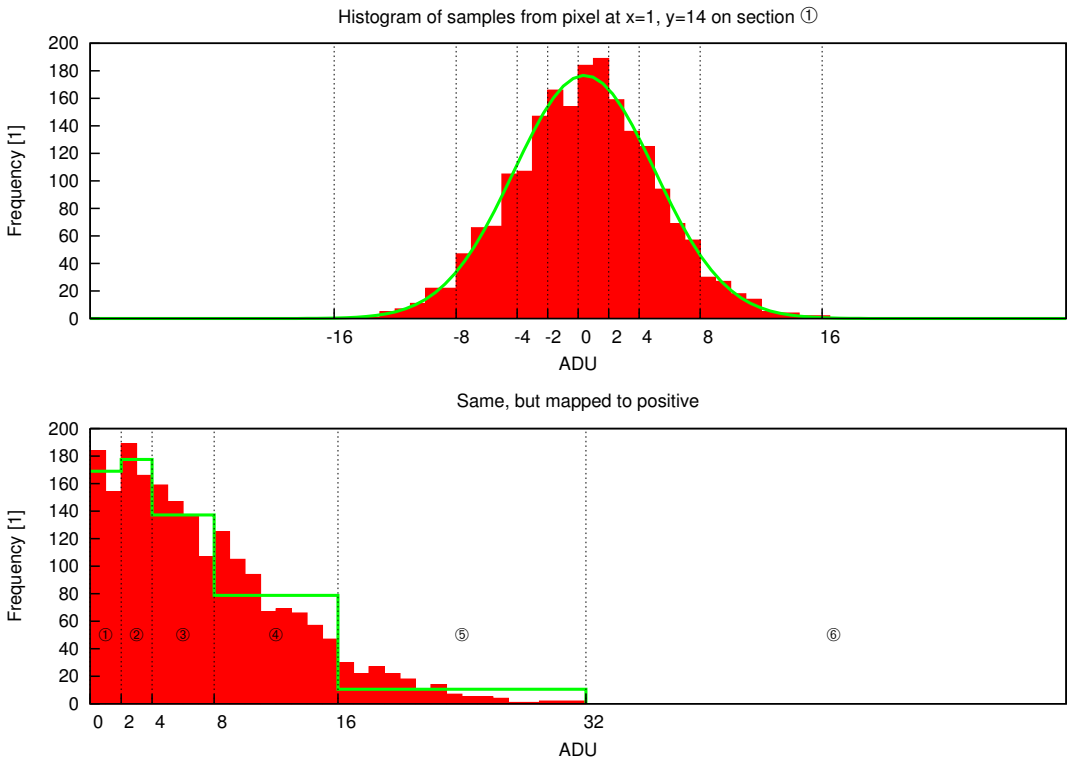


Figure 4.17: **Histogram of the signal of the bright pixel on section ①.** $\sigma = 4.624$

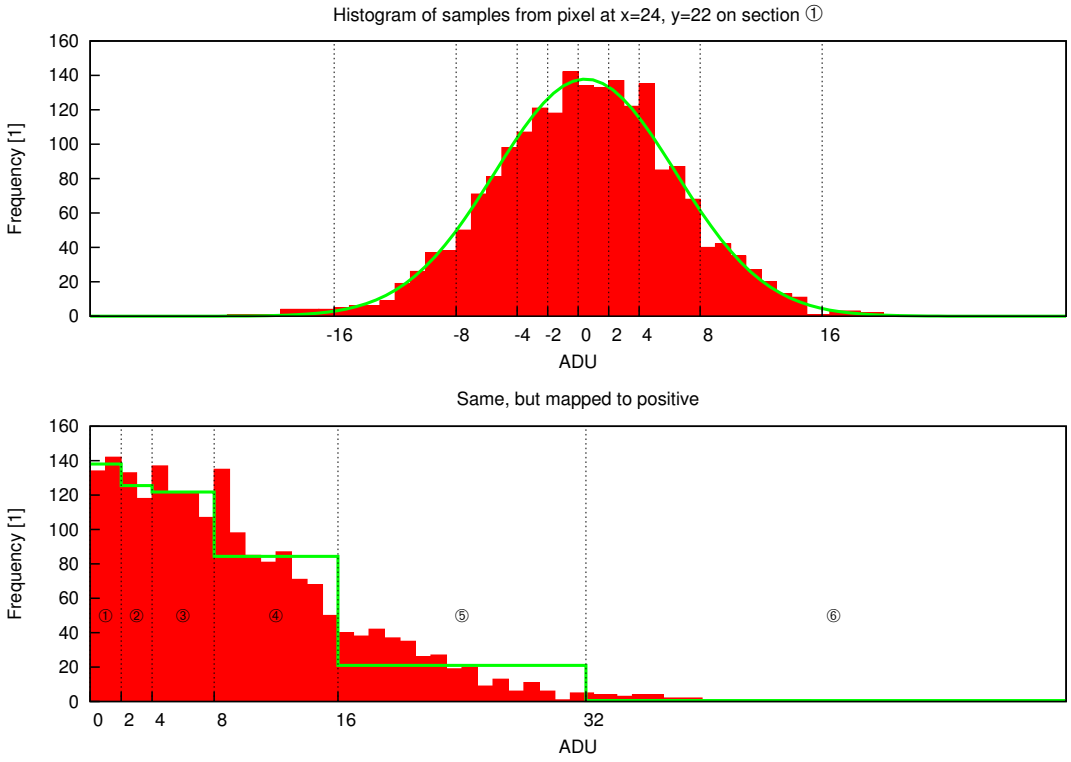


Figure 4.18: **Histogram of the medium pixel on section ①.** $\sigma = 5.926$

The Effect of Chopping

The data we have seen so far were made by *staring* at a certain source, such as one of the two blackbodies inside PACS. But to account for the high background signal in the observed wavelengths and to compensate instrumental effects, the standard way of observing with the bolometer is to have a chop/nod pattern at a defined frequency, in our case one half to several Hz, leading to sections of only a few raw samples on the same chopper plateau. Although this creates a square wave like pattern in the data, it is difficult to take advantage of during compression. The main reason for this is that the chop rate is not in the frame header and therefore not directly known to the compression software. What is in the header is the chopper position, a parameter that is also affected by noise and thus not ideal for interpretation on board. Above all, the on-board averaging is synchronised by a distinct header flag regardless of the chopper position. Chopping is one of the main problems to consider in the design of a proper decorrelation step.

Our two example pixels on section ② show the expected square pattern in Figures 4.19 and 4.20. We still see the drift and the readout noise in the data, but above all, the chop signal now dominates. All the frequencies that compose the signal (on this data window) are accessible to the SPU. The problem here is that the chopper signal spans a range of 1600 ADU. Making differences does not help to get rid of that. It's even worse: the deltas vary between +1600 and −1600 now and this has to be encoded. Within 30 averaged frames, 2–3 such transitions happen in this dataset, each one taking more than just the 4.5 bits to encode. Mapping that to positive and allowing for values between 0 and 4096 (2^{12}) adds $2.5 \times 12 \times 2048 = 61440$ bits to the blue channel in 3 seconds, i.e. ~ 25 kbit/s for red+blue. This can be compared with the actual data rate (Figure 4.4): Section ② is 20 kibit/s higher (for red and blue) than section ①.

Higher Bias/Gain Settings

The last question to be raised here is what makes the difference between section ① and ④. The second plot (Figure 4.5) showed that some weak pixels now also contribute to the data rate. But what about the noise? To answer that, the plots we are familiar with (Figures 4.21 and 4.22) and histograms (Figures 4.23 and 4.24) for the bright and the medium pixel are made. The entropy for the bright pixel is now $H_S = 4.275$ bits per sample and $H_S = 5.135$ for the medium pixel. To put it in a nutshell, the higher σ also leads to a higher entropy, which requires up to 0.5 bits per sample more to encode. That is up to $0.5 \times 29 \times 2048 = 29696$ bits (blue, 3s), which amounts to ~ 12 kbit/s for the red and the blue array. The actual data rate (Figure 4.4) shows an increase of 10 kibit/s as estimated for red and blue.

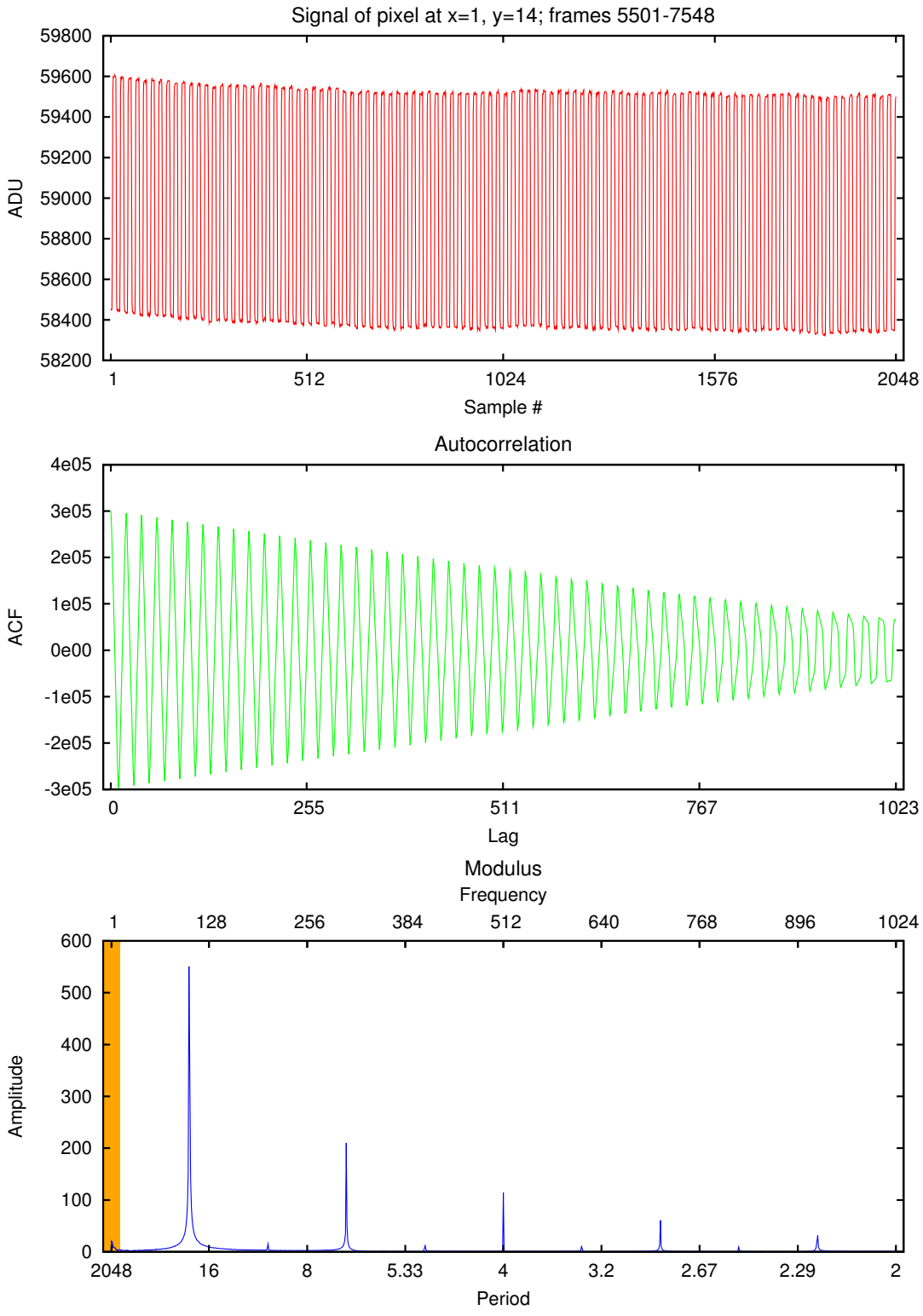


Figure 4.19: **The chopped signal of the bright pixel on section ②.**

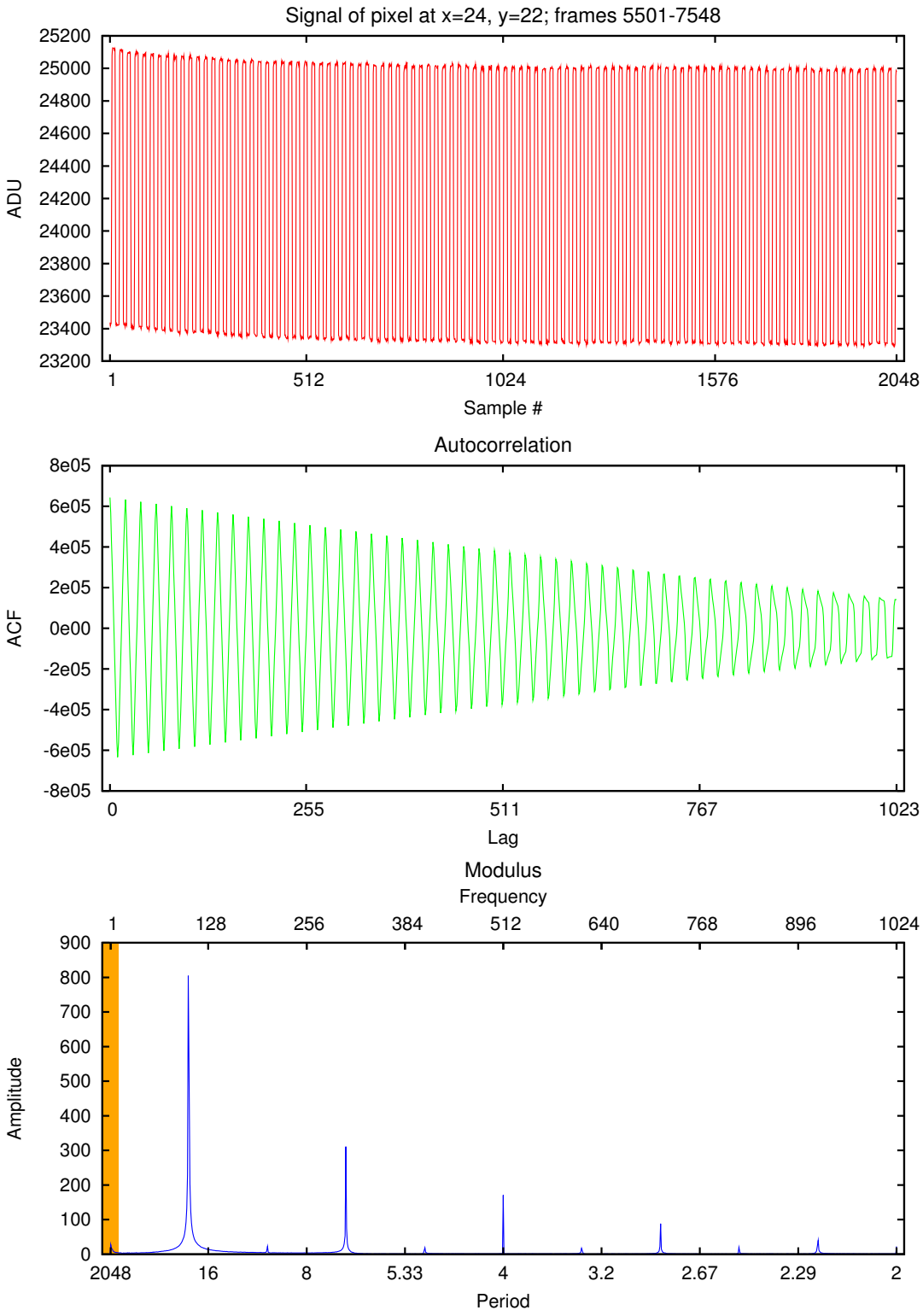


Figure 4.20: **The signal of the medium pixel on section ②.**

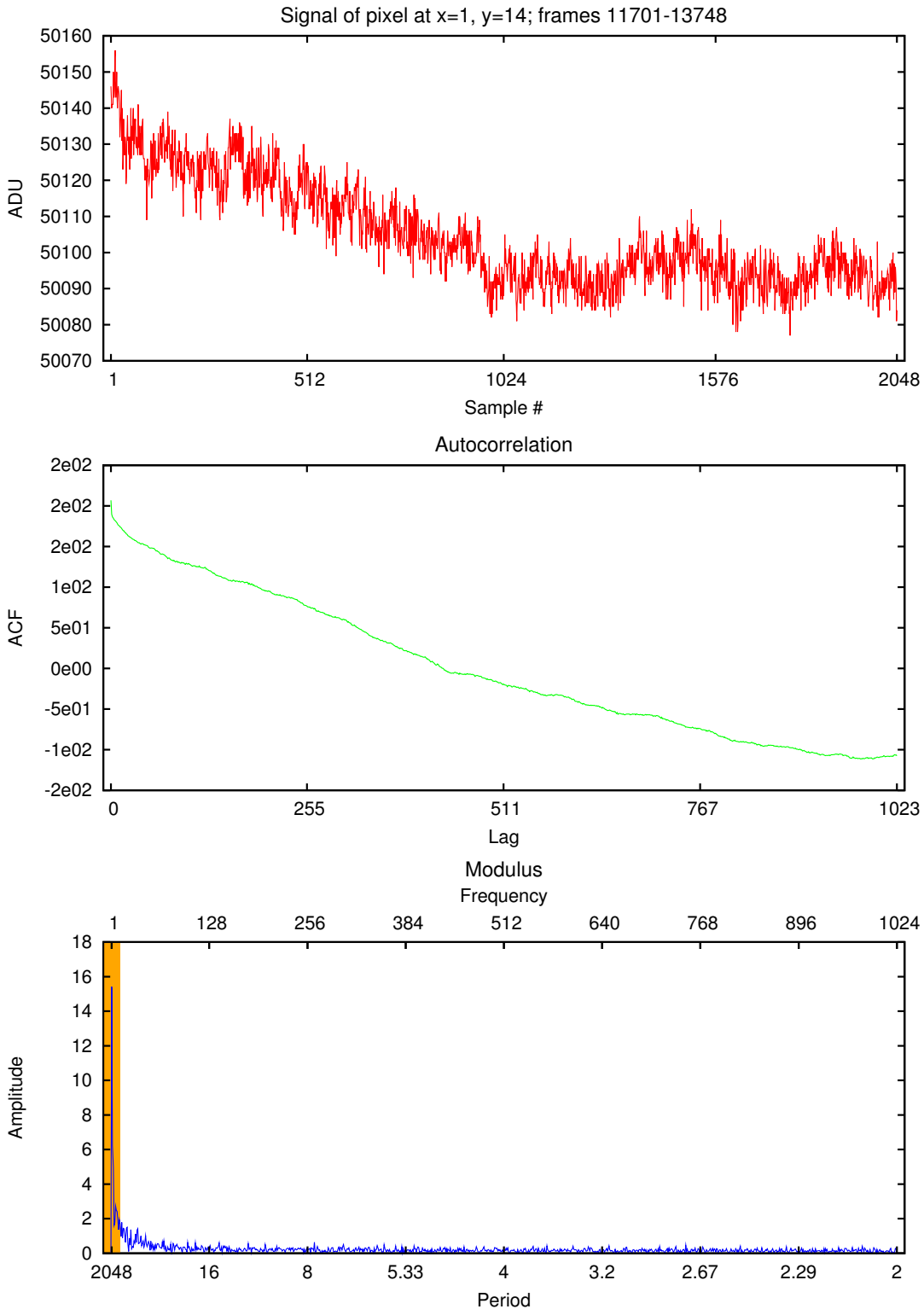


Figure 4.21: **The signal of the bright pixel on section ④.**

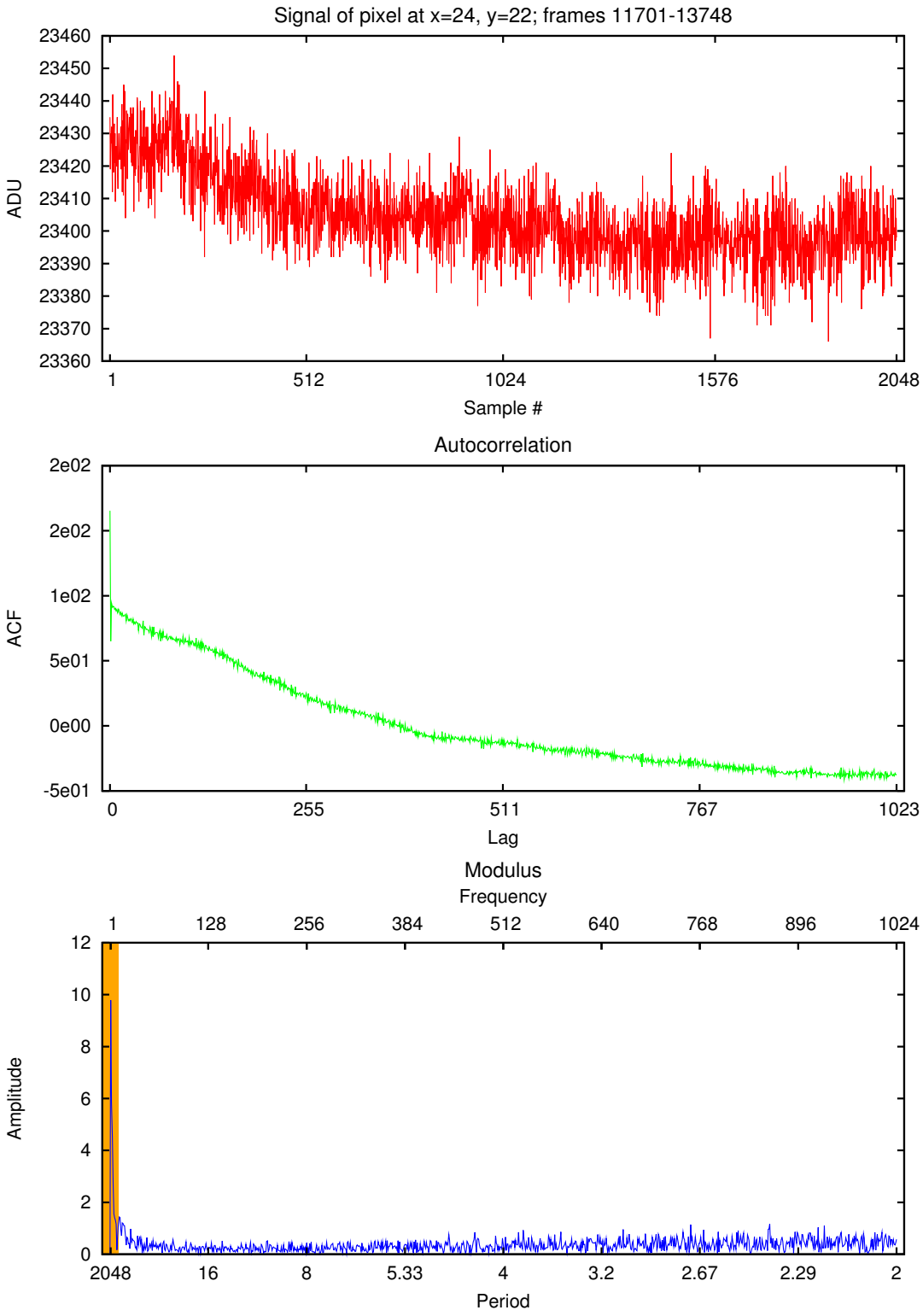


Figure 4.22: **The signal of the medium pixel on section ④.**

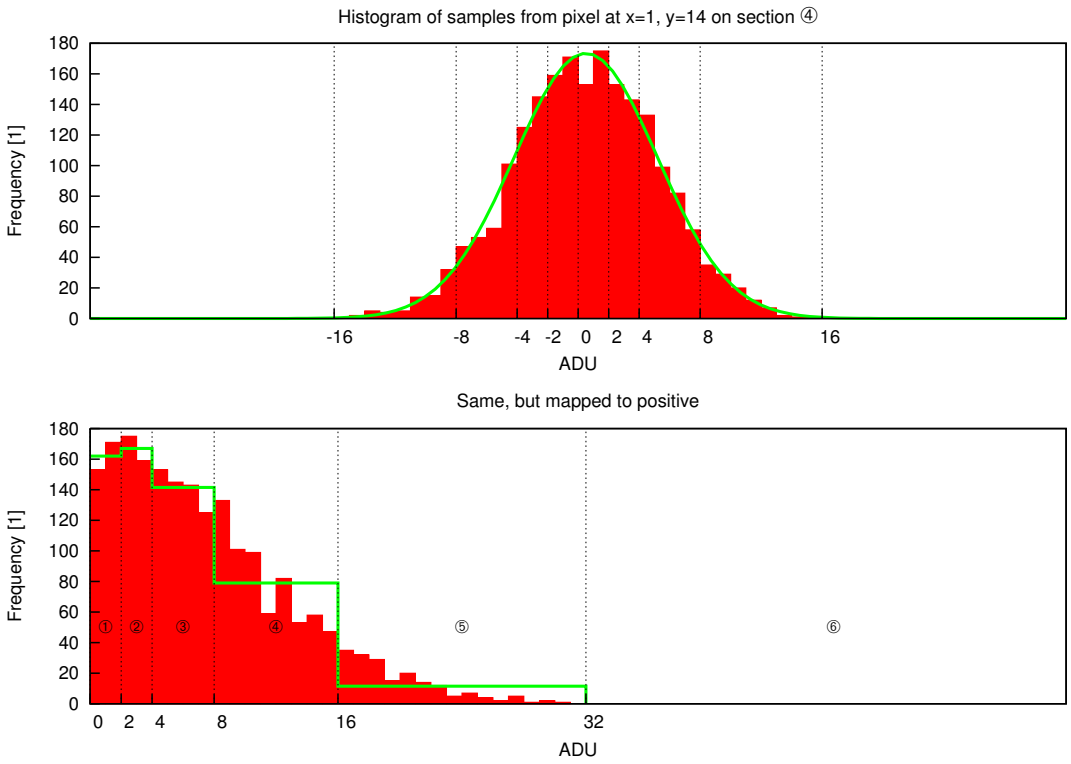


Figure 4.23: **Histogram of the signal of the bright pixel on section ④.** $\sigma = 4.711$

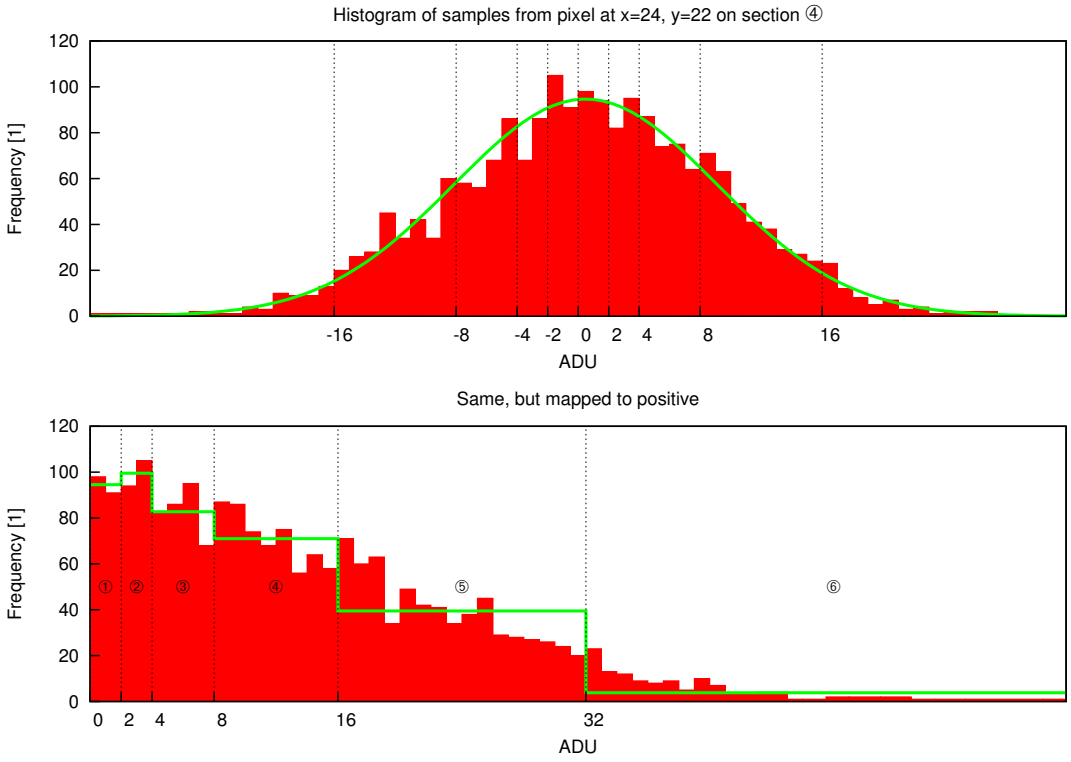


Figure 4.24: **The signal of the medium pixel on section ④.** $\sigma = 8.631$

4 The Devised Reduction Scheme

During summer 2006 I developed a new lossless compression scheme for photometry and gave it the code name *velvet*. After that I rewrote the reduction part to add stochastic rounding (as described in Chapter 1) and optional decimation (frame dropping). During observations in *default* compression mode 4 frames are averaged and further processed by the *velvet* scheme, which will decorrelate and entropy code the data. This section will now discuss the operations in detail.

At the core of data reduction stands averaging of 4 frames, which reduces the readout noise by a factor of 2 (i.e. 1 bit), but the averages still span over the whole 16 bit range and the noise varies from 3–7 bits, as was shown in the previous section. The next step is to separate signal and noise, with *signal* comprising the bias offset including any form of drift, the chopping pattern and the actual source signal. This is approached in the decorrelation stage, which ought to produce a dataset as sparse as possible, with more or less the readout noise remaining around zero. The particularities of the data had to be considered in the design of the decorrelation, but with the limited processing resources in mind. The easiest operation with yet great effect is a simple differentiation of the data, with the side-effect of increasing uncorrelated noise by a factor of $\sqrt{2}$. In terms of data rate, 1 bit of noise gives an overhead of 10 (averaged frames) $\times$ 2560 (red and blue pixels) = 25 kibit/s. This can be avoided by key-framing or offsetting from an average (or median) frame, but this is a hopeless attempt for data that are drifting and even chopped. Several predictor/corrector algorithms ranging from a running average to a full linear prediction of second order have been considered, but although there is a benefit for staring data they offer no real solution for chopped data. Block-sorting transforms exceeded the available CPU resources and orthonormal transforms were regarded problematic due to the increase of the dynamic in the coefficients.

So a simple differentiation for each pixel was chosen, followed by reformatting the data to an ordering allowing frames differentiation. These 2 differentiations increase the noise by 1 bit, but their combination eliminates very well the signal redundancy, any drift and even the chopping, with operations that all work in place and barely need any processing power. The result is that only 5–10% of the decorrelated values exceed an 8-bit range even for chopped observations, which is a good starting point for the back-end encoder (described in Chapter 5).

Step by Step

The input data received from DMC are in what I call *frame order*, where pixels are increased before frames in the dataset. Figure 4.25 has plots of the first 8192 samples. Each successive processing step will now be described with plots of the intermediate result. Two plots always go together, one for a *staring* observation and the other one for an observation where *chopping* was performed. Note that every plot comes with a magnified detail to the right. A given zoom factor applies to the ordinate (the y-axis).

1: Reordering

The first processing step is to discard data from pixels that were deselected, but normally the whole frame is kept and so we will omit this step here. After that, the data are rearranged to what I call *pixel order*, where the samples of the same pixel are grouped together. This is shown in Figure 4.26.

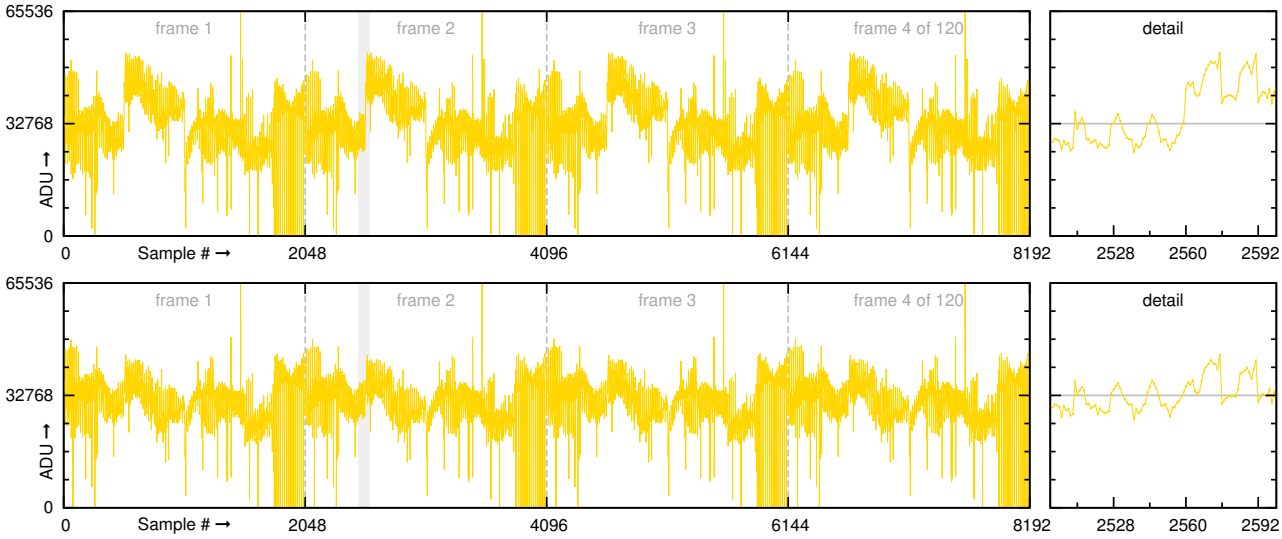


Figure 4.25: **Raw input data of a staring and a chopped observation.** The frames of an observation look much the same, whereas within a frame hardly any features span across several pixels. A grey marked region is shown in the magnified detail plot. Not too much difference is visible between the staring (top) and the chopped observation (bottom).

Listing 4.1: Pixelize

```
/*
  Transforms a buffer where data are stored pixel by pixel
  into a buffer where they are stored frame by frame.

  SRC contains: p0f0, p0f1, p0f2, ... p0fF, p1f0, p1f1, p1f2, ...
  DEST contains : p0f0, p1f0, p2f0, ... pPf0, p0f1, p1f1, ...
*/

void Pixelize (int *src, int srcSizeInWords, int pixelsPerFrame,
               int *dest)
{
  int i, j;

  int nbFrames = roldiv(srcSizeInWords, pixelsPerFrame);

  for (i=0; i < nbFrames; i++)
  {
    for (j=0; j < pixelsPerFrame; j++)
    {
      dest[j*nbFrames + i] = src[i*pixelsPerFrame + j];
    }
  }

  return;
}
```

2: Sign and Average

After reordering the averaging is done, but the averaging algorithms are implemented for signed integer input, which is provided by the detectors if they are read in the so-called *double differential mode*. The more frequently used *direct mode* provides unsigned integer input, which needs to be mapped to the signed data type. This is done by subtracting 0x8000 (in decimal: 32768) from each sample. This mapping and the averaging are illustrated in Figure 4.27.

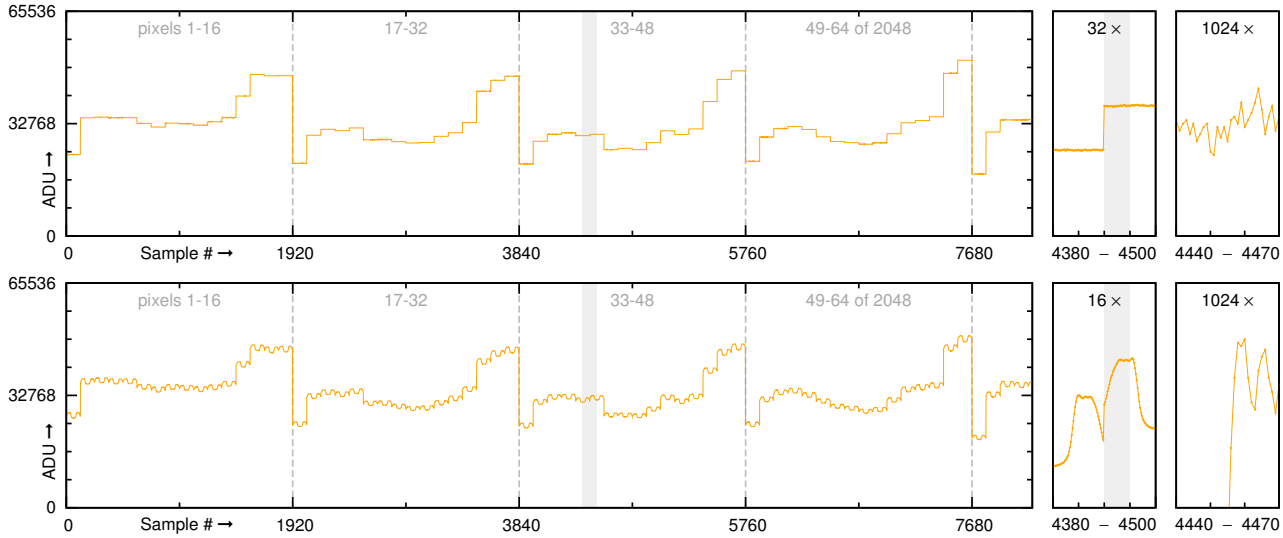


Figure 4.26: **Raw input data rearranged.** The pattern that is visible is more or less the bias. In the magnification to the right the readout noise is revealed. By comparing the staring (top) with the chopped (bottom) dataset we find that the signal variation due to the chop is clearly visible, especially in the magnification.

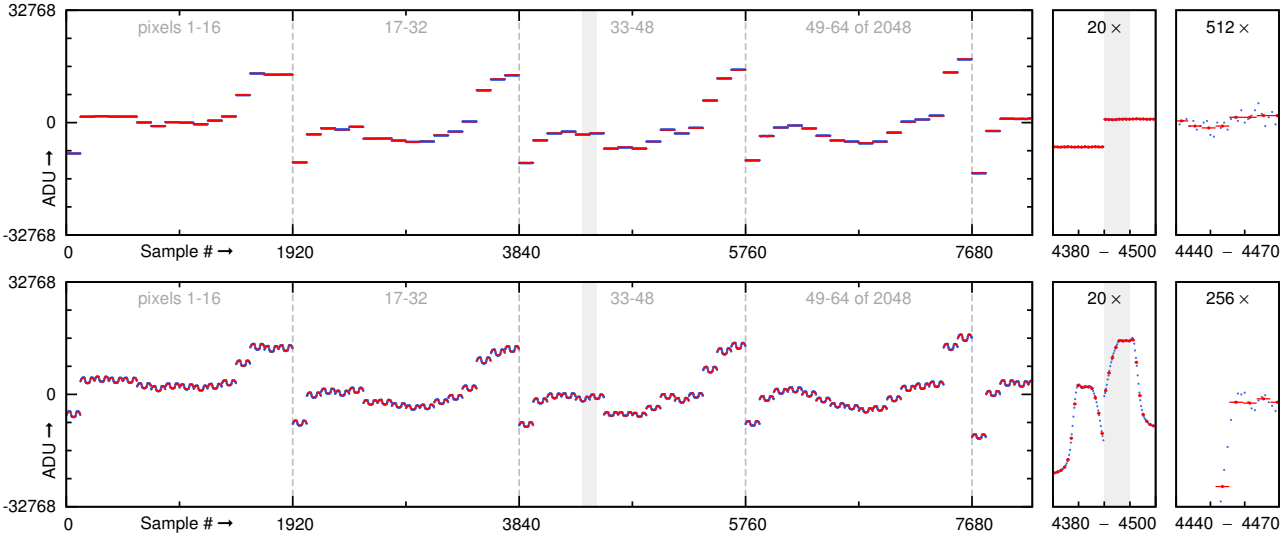


Figure 4.27: **The dataset signed and averaged.** When inspecting the averages (indicated by red dots and dashes) we observe that the noise is well reduced by this operation. The averaging process needs to be synchronised with the data to avoid averaging over chopper transitions. In this dataset the chopper moves so slowly that this is not an issue. The averaged data have to be compressed in the next processing steps and then sent to ground.

3: Differentiation

The averaged data still need to be compressed, but what is the best way to do this? As we are dealing with noisy science data the recommended strategy is to go for a combination of decorrelation and entropy coding. Note that the decorrelation step will determine most of the compression efficiency. For various reasons I decided to use differencing here. Differences are made between successive frames, but not across frame transitions. Figure 4.28 shows the result of this operation and the code listing is also given.

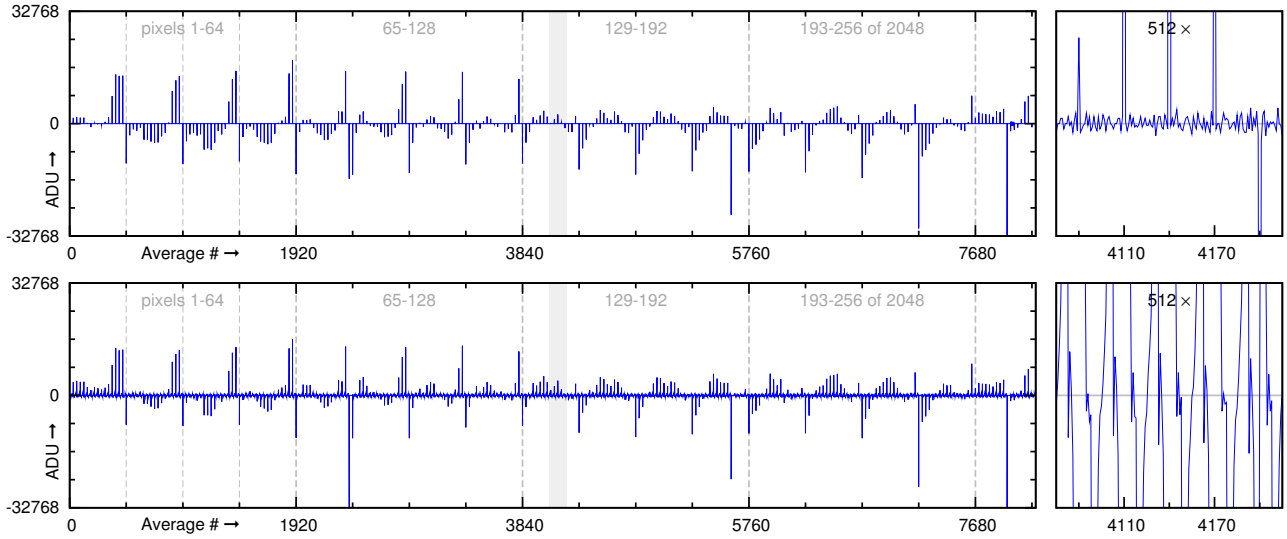


Figure 4.28: **The averages after differencing.** Every pixel starts with a spike which is a remainder of the bias, but the samples are already concentrated around zero. The magnification uncovers that there are systematic differences between staring (top) and chopped data (bottom).

Listing 4.2: Delta step-wise

```
/*
  Differentiate a ''pixelized'' buffer from back to front,
  but not over pixel boundaries (every nbFrames).
*/

void DeltaStepwise (int *buf, int bufSizeInWords, int nbFrames)
{
  int i,j;

  for (i=bufSizeInWords-1, j=1; i>0; i--)
  {
    if (j < nbFrames)
    {
      buf[i] = (buf[i] - buf[i-1]) &0xffff;
      buf[i] = buf[i] > 0x7fff ? buf[i] | 0xffff0000 : buf[i];
      j++;
    }
    else j = 1;
  }

  return;
}
```

In the next step we bring back the data into *frame order*. As a result of the differentiation that was done before, the first frame is still the same, whereas the other ones contain the residuals (see Figure 4.29). So far no correlation between pixels was taken into account and a collective signal change – such as chopping – will affect whole frames by large offsets. The signal variation by a chop is not the same for each pixel because of the flatfield and the inhomogeneous signal itself, but a chopped frame will no longer contain noise around zero, but noise around a value which is the mean signal change across the frame. In the next step, we have to get rid of this.

4: Reordering

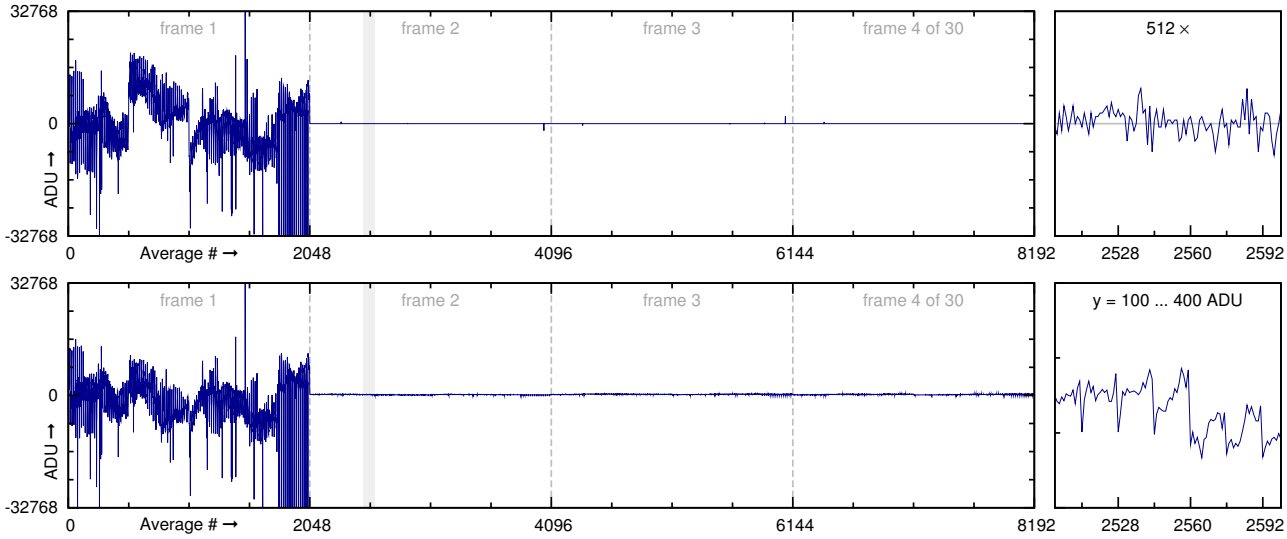


Figure 4.29: **Data brought back to frame order.** Apart from the first frame which contains the bias the remaining ones are decorrelated. The chopped dataset (bottom) shows small systematic offsets, e.g. ∼200 in the magnification.

Listing 4.3: Frameify

```
/*
  Transforms a buffer where data is stored frame by frame
  into a buffer where data is stored pixel by pixel.

  SRC contains : p0f0, p1f0, p2f0, ... pPf0, p0f1, p1f1, ...
  DEST contains: p0f0, p0f1, p0f2, ... p0fF, p1f0, p1f1, p1f2, ...
*/

void Frameify (int *src, int srcSizeInWords, int pixelsPerFrame,
               int *dest)
{
  int i, j;

  int nbFrames = roldiv(srcSizeInWords, pixelsPerFrame);

  for (i=0; i < pixelsPerFrame; i++)
  {
    for (j=0; j < nbFrames; j++)
    {
      dest[j*pixelsPerFrame + i] = src[i*nbFrames + j];
    }
  }

  return;
}
```

5: Differentiation

In order to deal with the chopping a second differentiation across *frame order* is done. This step brings chopped frames back down to noise around zero, but also increases uncorrelated noise by $\sqrt{2}$. A running average for decorrelation can do a slightly better job, hardly being worth the extra effort. The result of the second differentiation is plotted in Figure 4.30. The listing for the second differentiation goes over the whole buffer and makes no pauses as the first one did.

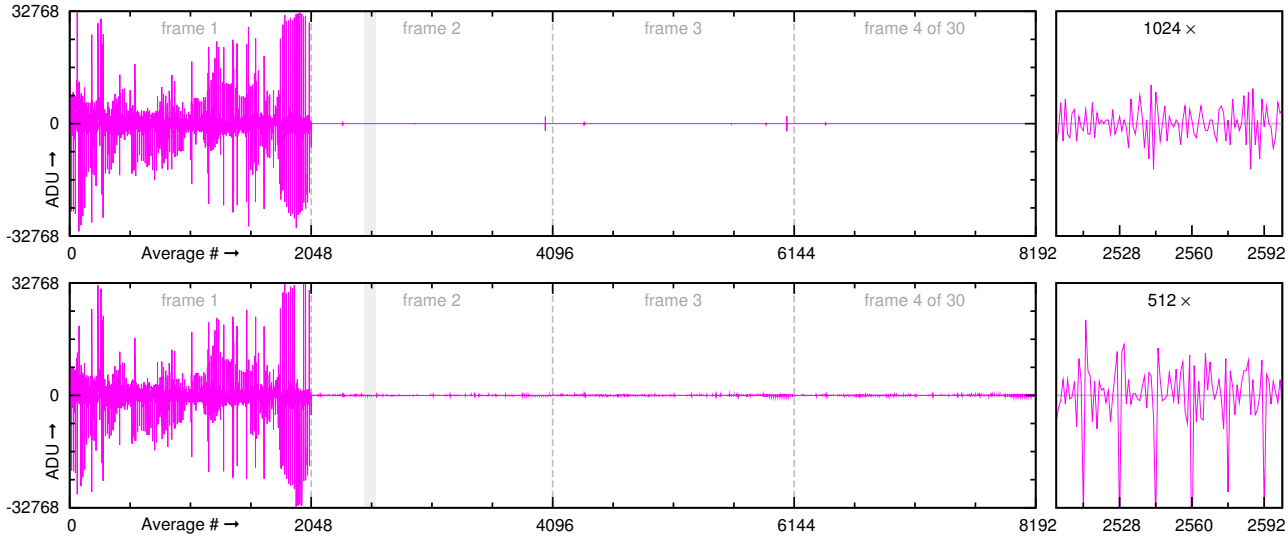


Figure 4.30: **The dataset after the second differentiation.** This operation is a countermeasure against chopping, but at a relatively high cost. Single spikes become a ± spike pair and uncorrelated noise is increased by half a bit.

Listing 4.4: Delta

```
/*
  Differences are made in place, from bottom to top,
  with 16 bit sign extension. Works in place.
*/

void _Delta (int *buf, int words)
{
  int i;

  for (i=words-1; i>0; i--)
  {
    buf[i] = (buf[i] - buf[i-1]) &0xffff;
    buf[i] = buf[i] > 0x7fff ? buf[i] | 0xffff0000 : buf[i];
  }

  return;
}
```

Before the data can be passed to the entropy coder, they need to be mapped to positive (shown in Figure 4.31). We see that our dataset consists of a turbulent first frame, i.e. 2048 values that fall into a 16-bit range mainly due to the bias. There will be hardly any compression done with these. But the remaining 29 reduced frames are well concentrated in a 5-bit range and will be compressed by roughly a factor of 3. This is best seen in Figures 4.32 and 4.33.

6: Mapping

Listing 4.5: Map2pos

```
/*
  This function transforms signed data to unsigned data by folding
  the negative part into the positive one. The peak of the original
  distribution should be around zero. Positive numbers are multiplied
  by two and negative ones are inserted in between. It works in place.
*/
```

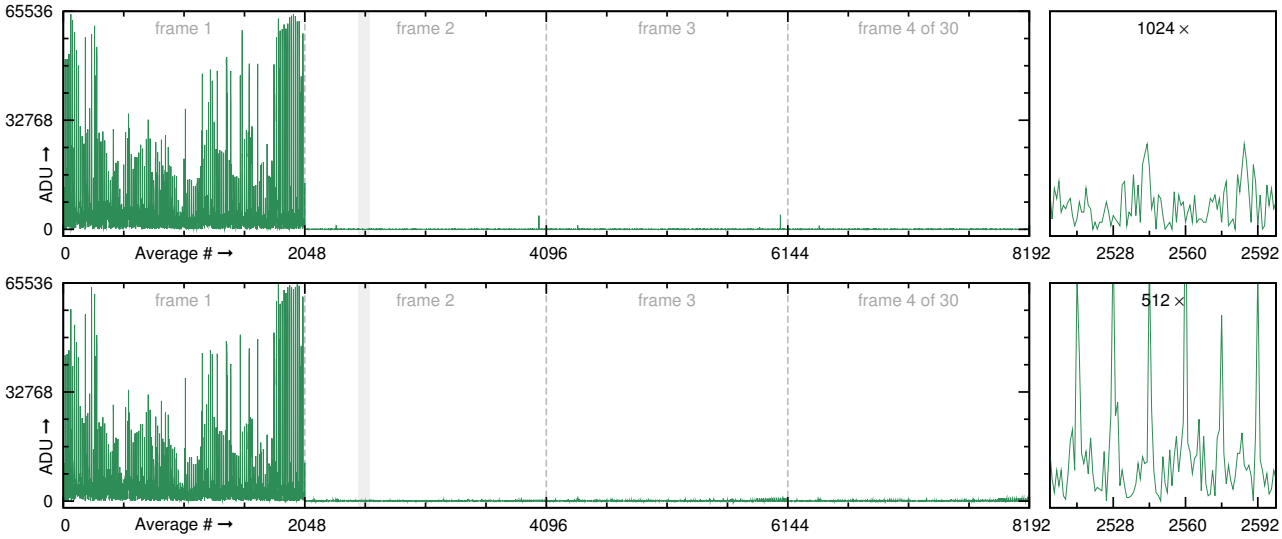


Figure 4.31: **The decorrelated and mapped dataset.** This buffer is now passed to the entropy coder, where the actual lossless compression is performed. The chopped dataset still shows spikes in the magnification (a 7-bit range is shown), which is a sign of imperfect decorrelation.

```
8  void Map2Pos32 (int *buf, int sizeInWords)
9  {
10   int i;
11
12   for (i=0; i < sizeInWords; i++)
13   {
14     buf[i] = buf[i] < 0 ? (buf[i] * -2 -1) & 0xffff : buf[i] * 2;
15   }
16
17   return;
18 }
```

Performance

Obviously, the more the values are concentrated around zero, the higher compression can be achieved by the entropy coder. In terms of numbers the entropy for the staring dataset (Figure 4.32) was reduced by conditioning from 13.63 bit/sample to 4.95 bits/sample. A few more overheads lead to an actual encoding of 5.03 bits/sample in the end. We remember the compromises made in the scheme – differentiation increases noise – and with an ideal decorrelation we would have been able to further decrease the entropy down to ∼4 bits/sample. In case of the chopped data we see that the original histogram was bimodal (4.33). We could concentrate this in one peak around zero, but still a systematic component contributes to the broadening of the peak. The chopped dataset is shrunk from 14.04 bits/sample to 6.44 bits/sample. The encoding is also not ideal here and in the end we get 6.77 bits/sample.

Possible Improvements

If no chopping would be performed, the second differentiation could be omitted and the first one be better replaced by a running average or even by an integer wavelet. But chopping is an integral part of PACS bolometer observations and the necessary processing steps were included at the cost of staring performance. The biggest improvement

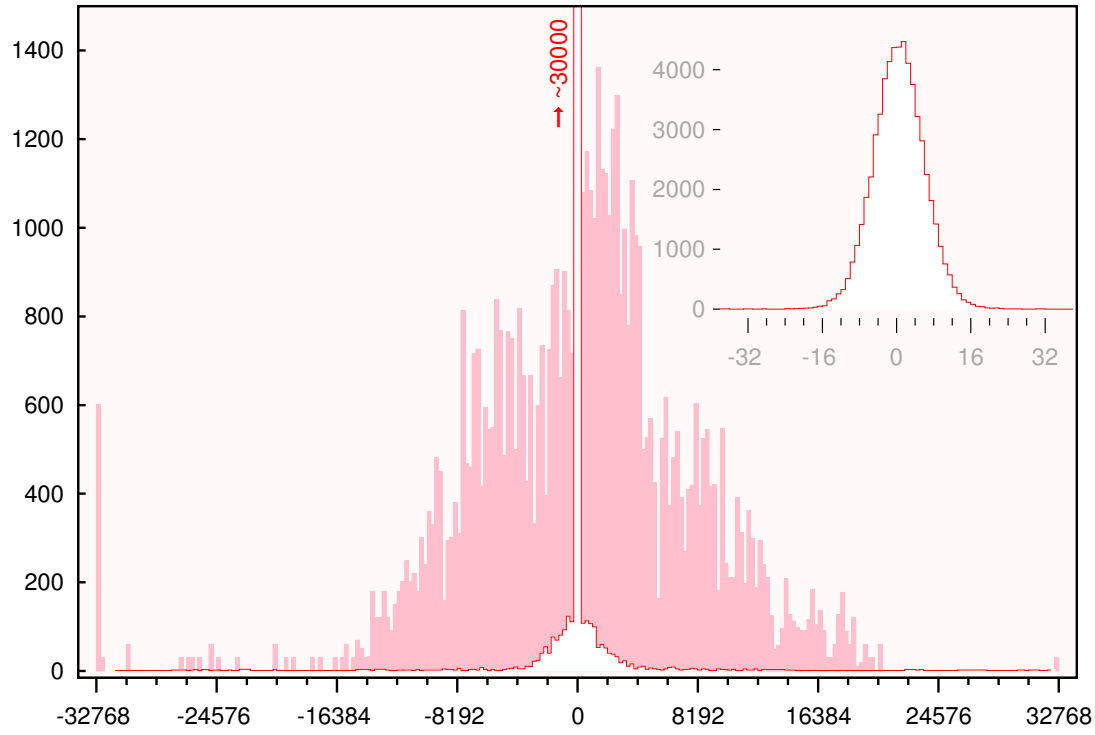


Figure 4.32: **Performance of the decorrelation for staring data.** The histogram of the averaged data (in solid red) fills a wide range of values. After decorrelation the histogram (shown in white) is much more compact. The inset shows the inner bins. Note that the values are shown before the final mapping to positive.

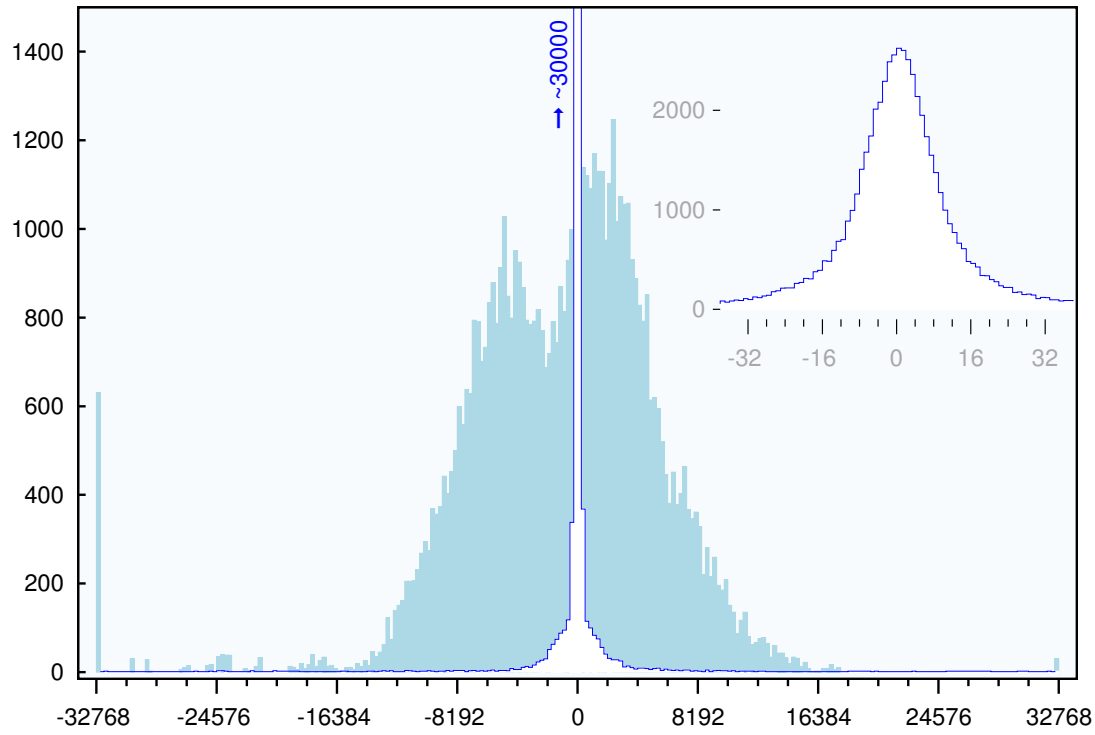


Figure 4.33: **Performance for the chopped observation.** Although the two histograms are compacted into one the result is broader than the staring observation, which in turn leads to a higher data rate.

for chopped data that could be achieved by a single step would be to sort the pixels in the beginning or before the second differentiation by their sensitivity. Unfortunately the sensitivity map depends on the detector setting and so this was never considered practically feasible. To stay compatible with a data rate of 120 kbit/s the entropy would have to be ∼4.7 bit/sample, or 5.5 bit/sample for 140 kbit/s. This limit is exceeded for high gain settings, especially when chopping is done. We see that even with a *perfect* decorrelation, we would be able to save only 1 bit/sample for staring data and probably more for chopped data,[1] but this is certainly not enough. Although the on-board software has ways to further reduce the data rate – pixel masking, adjusting the number of samples/average, decimation – an additional quantisation step had to be included to pay attention to the noise – and this is rounding.

Additional Rounding

The data rate after adaptation of the lossless compression to the FM detectors is still too high for several high gain settings. As the reason for this is that the noise is actually oversampled by such settings, a rounding option was included in the software. That way, even the highest noise setting can fit into the telemetry budget. One rounded bit of noise for 2560 pixels (red and blue together) at 10 Hz (averages) saves 25600 bit/s in photometry. The cost of this is additional quantisation noise, which amounts to a few percent of total noise increase. Remember the tricky thing about this is, that statistically correct rounding involves random number generation. The resulting combined averaging/rounding algorithm that I developed is given in Section 1.6. The algorithm tries to save CPU cycles and also ensures that the digitisation noise from the nominal averaging process and the extra rounding do not cumulate sporadically. The best way to envision how integer rounding works is to imagine that the values are projected onto a coarser grid. In 1-bit rounding, no odd numbers are left, in 2-bit rounding, the numbers are divisible by four etc. In Figure 4.34 the direct effect of additional rounding is shown. Note that the quantisation error is more or less uniformly distributed. For the shown dataset, the readout noise $\sigma = 4.5$ is increased by the 2-bit rounding to $\sigma = \sqrt{\sigma^2 + 4^2/12} = 4.65$, i.e. by ∼3 percent. Compared with the savings of the data rate, this additional quantisation noise is a small price to pay.

Benefits of Rounding

Rounding away noise bits helps in reducing the entropy of a dataset as long as the noise distribution is not degraded. In addition to that, the numerical range of the signal is reduced, because the rounded bit is always zero and the significant bits can be right-shifted.[2] This has no impact on the entropy, but it makes decorrelation easier. For each rounded bit we expect to save 25 kibit/s for both arrays together and even slightly more if chopping is performed, depending on frequency and signal range. Figure 4.35 reveals that this prediction is fine, but the more bits are rounded the less the savings are. The reason for this is that with too high quantisation the noise is rounded away.

[1] It is hard to estimate the contribution of the signal to the entropy. Unless a signal is *perfectly* predictable it contributes to the entropy. In our case it is legitimate to say that the chopped signal has up to 1.5 bits of entropy.

[2] This needs to be undone during inflation before the values can be used.

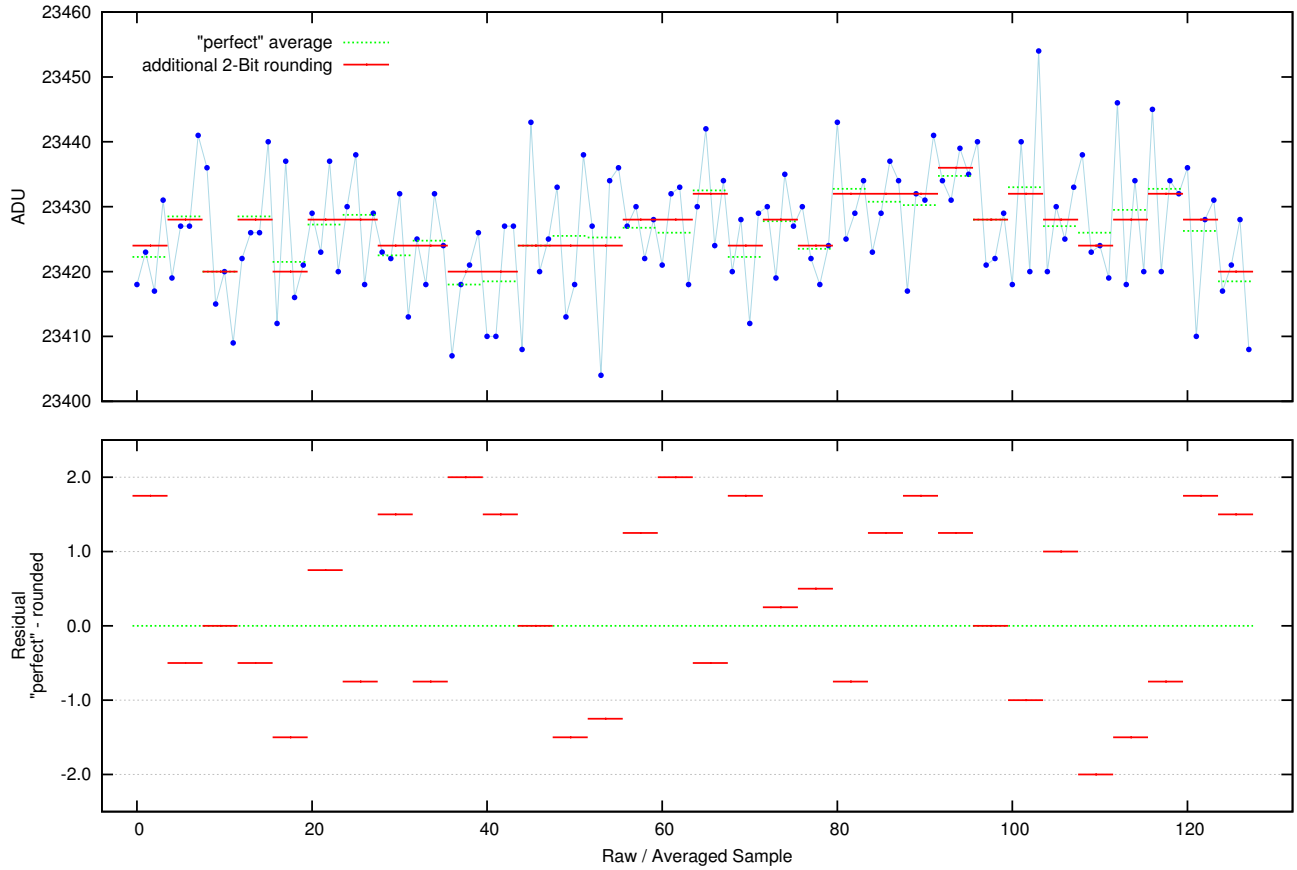


Figure 4.34: **The effect of additional rounding.** (Top) The spiky line with the blue dots represents the pure raw data from a pixel. Every four samples averaged in float give the *perfect* averages, shown by dotted green lines. In PACS the averages are integers, so a quantisation of the *perfect* average already occurs in the default reduction anyway. Additional rounding as the 2-bit rounding indicated by the red dashes still deviates from these values depending on the quantisation interval, which is determined by the number of bits that are rounded. Compared with the original noise the rounding operation has a hardly noticeable effect. (Bottom) Plot of the deviation from the *perfect* average due to the rounding. The errors follow approximately a uniform distribution.

Conclusion

The FM compression scheme for the bolometers is based on averaging, simple decorrelation steps and entropy coding. A number of tradeoffs lead to a data rate that is up to 20% higher than what would be theoretically achievable, but even an ideal decorrelation would not meet the 120 kbit/s for high gain measurements. The countermeasure against this is rounding, which drastically reduces the data rate at little noise increase.

What if...

If we could turn back time three or more years into the past, would I come up with a different compression scheme? Probably not. All the steps involved are well justified and there are no compression artefacts to be feared in this scheme. The various lossy steps allow for a control of the data rate. If more processing resources would be available, a fully adaptive model for the arithmetic compression could save another 5%. I have mentioned several times already that the biggest savings can still be made by another decorrelation stage, which would have to be able to cope with chopped data. There are

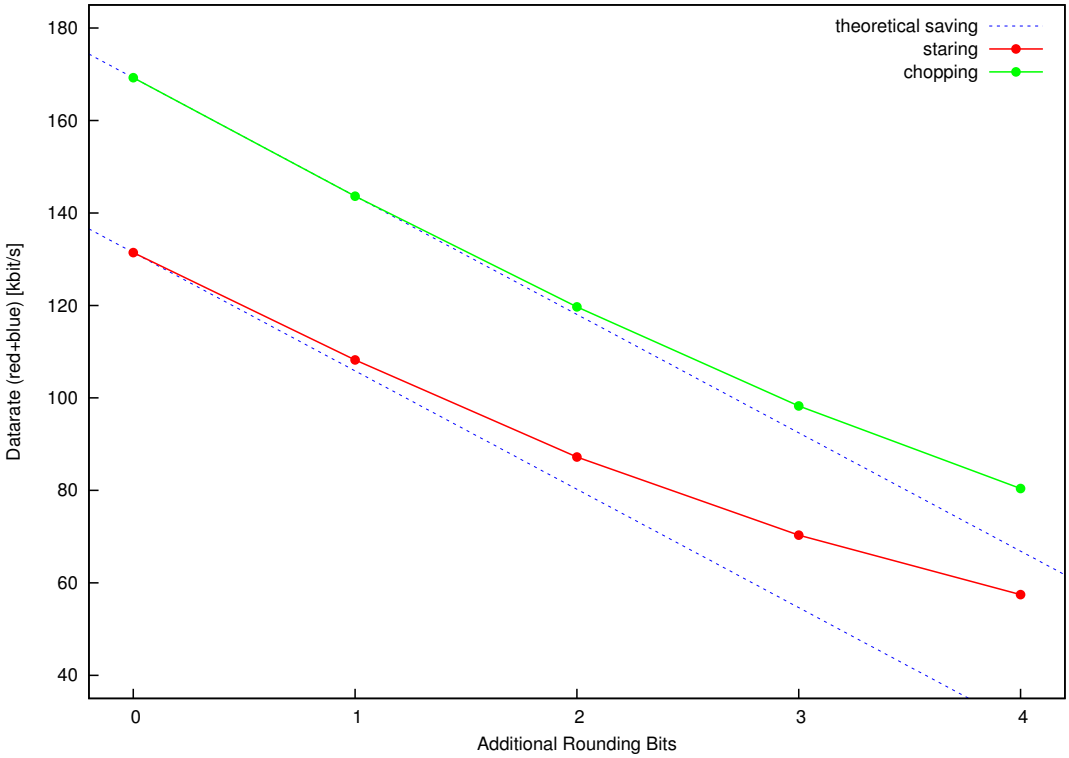


Figure 4.35: **Data rate savings by rounding.** Staring and chopped data were processed with different bit roundings to see if the predicted savings are achieved. Towards higher roundings we find less savings than predicted. This is in part due to pixels with less noise that reach a constant level for higher bit roundings and in part due to the lossless compression, which is optimised for 4 bits of noise.

open points in photometry, especially for fast scan observations, where the averaging step smears the sources in scan direction. Even worse, most scans are made in the so-called *parallel mode*, where PACS and SPIRE observe simultaneously and share the bandwidth. In this case, we have to average 8 frames to stay within the telemetry budget and the smearing is very unfavourable.

Decimation

One option that helps is already included in the software, it is *decimation*, i.e. frame dropping. It works like *average only 4 frames out of 8*, but although it decreases the smearing it also increases the noise, because it doesn't make use of all available data. A completely different approach is made by *Compressed Sensing*. This is a compression mode we are currently developing for the PACS parallel mode based on a new sampling theorem. First experiments are very promising [Bob08] and a dedicated research project has been started.

5

PACS FM Spectrometer Software

When PACS is operated as a spectrometer, the discrepancy between the allowed data rate and the amount of generated raw science data is even greater than in photometry. The two Ge:Ga arrays have less pixels, but are read at higher rates and a reduction of a factor of ∼40 needs to be achieved. The basic idea behind the on-board data processing is similar to the photometer, but the decorrelation stage is different and the lossy part, which contributes most to the reduction factor, either consists of averaging *sub-ramps* or of slope fitting. The detector technology of the spectrometer is very different to the photometer, yet the noise after decorrelation is also in the range of 3–7 bits. What is shared with photometry is the compression back-end, because the same entropy coder is used, but with a different initial noise model.

> **ramp**, n: the act or an instance of ramping; any of various alliums used for food; a short bend, slope, or curve usually in the vertical plane where a handrail or coping changes its direction; a sloping way; a sloping floor, walk, or roadway leading from one level to another; a stairway for entering or leaving an airplane; a slope for launching boats; apron 2h
>
> —*Merriam Webster's Collegiate Dictionary*

In this chapter we will follow the same structure as in photometry – we analyse the input data, see which kind of processing is necessary, learn what was actually implemented and discuss its performance and possible improvements. The latter part deals with implementation details that concern both photometry and spectroscopy, with emphasis on the back-end encoder.

1 Imaging Spectroscopy

The particular thing about the two spectrometer arrays [Pog08] (a stressed one for the *red* part of the wavelength range and an unstressed one for the *blue*) is that they are read very quickly at 256 Hz in a non-destructive way to get a fine description of

The chapter is divided into the following sections:

the intrinsic nonlinearity, the high reset noise[1] as well as transient effects like glitches [Eck99, Ste07]. Thus, each sample will provide a larger signal unless the sampling is reset. This is called sampling *up-the-ramp*. In a non-destructive readout the measure for the intensity is the slope of the achieved ramp. Assessing the raw data properties, especially to separate the entropy of the readout noise from the signal content is a tricky task, but this is our first step to establish a basis for judging possible strategies for the compression of these data.

Image and Signal

Very similar to the bolometer, a single frame of a spectrometer array as shown in Figure 5.1 has a quite inhomogeneous appearance. Each column is an integrated module of 16 photoconductor pixels, stacked to form a 25×16 array. Two so-called *dummy channels*, a so-called *open channel* and a *resistor channel* plus an additional *ghost column*[2] finally lead to a dataset of 26×18 pixels for each of the two arrays. Remember that the data flow of the spectrometer is also illustrated in Figure 4.1. The two spectrometer arrays have identical dimensions and so they contribute equally to the data rate.

The signal from an individual pixel is substantially different to the photometer, as it is read non-destructively. Only after a number of samples are gathered, the capacitance is reset and the integration starts anew. Some features of photoconductors can be found in the plots of Figure 5.2. In this sequence of raw data from a single pixel a reset occurs every 64 samples. Of course you would expect an ascending line, but the PACS electronics output a descending line – something we will easily get used

[1] Every ramp is offset by a considerable random value.
[2] There is no official designation, and whenever I used this term people would willingly accept it.

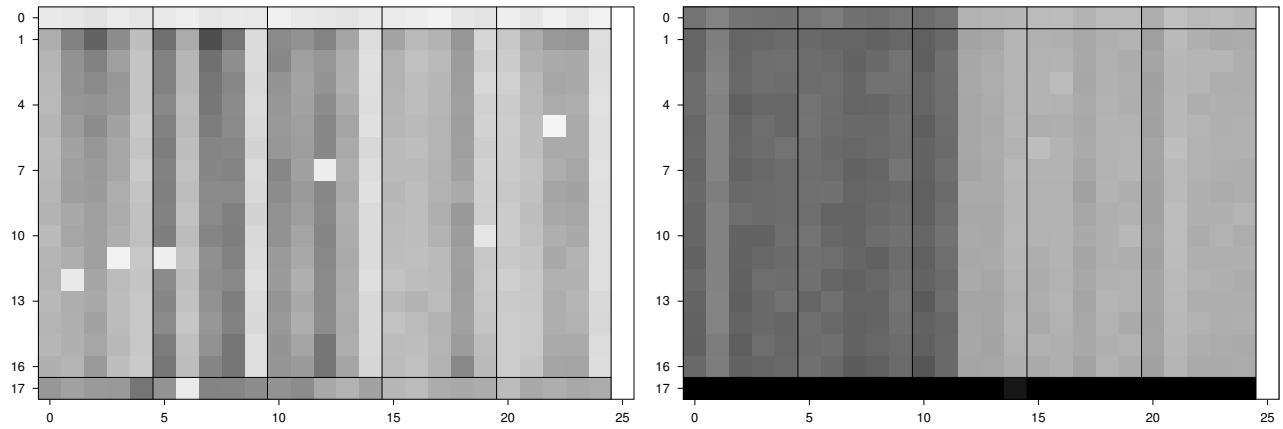


Figure 5.1: **Cosmetic aspect of the spectrometer arrays.** (Left) A frame from the red array is shown and (Right) one from the blue spectrometer. The line on top and on the bottom are dummy pixels that could be used for calibration and are usually included in the telemetry. The *ghost column* is plotted to the right side of each array, although it has no physical pixels. It is normally excluded from telemetry by the detector selection mechanism.

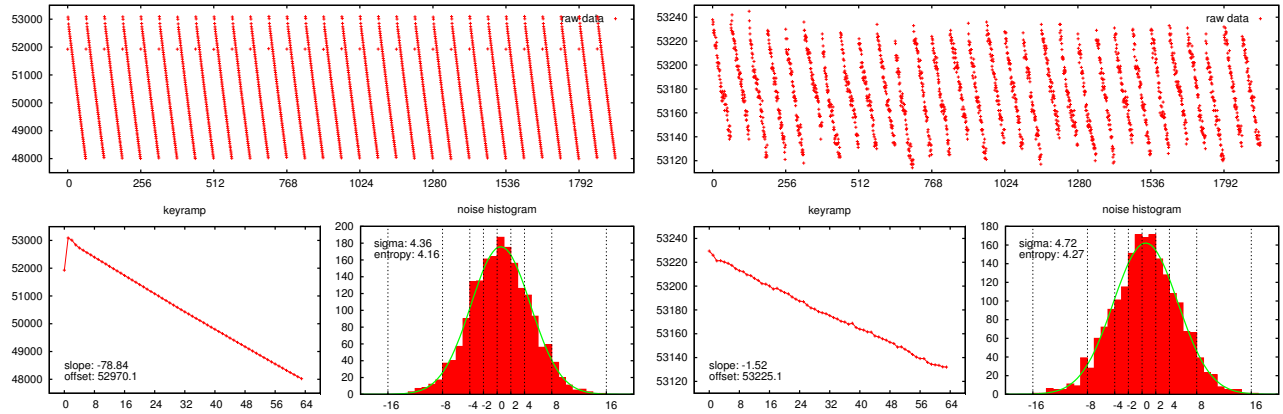


Figure 5.2: **Signal of the spectrometer, easily identifiable by the ramps.** The hook is nicely represented in the left plot (pixel from red array) of the keyramp. The reset noise can be seen in the plot of the blue pixel (right side). In both examples the readout noise is between 4–5 bits. The ordinates of the plots are in ADU except for the histograms.

to. The key in extracting the readout noise in spectroscopy is to subtract a mean *keyramp* from each single ramp. It accounts for most of the nonlinearities including the so-called *photoconductor hook* [Hae01], but some additional measures still need to be taken against sensitivity drift and the characteristic reset noise, before we get an estimate of the data entropy for consideration in the reduction and compression scheme.

The FM Dataset

Similar to the different gains in photometry, different integrating capacitances can be selected to be able to adjust to faint and bright sources. By increasing the dynamic range, the noise is amplified as well. For the spectrometer we have 3–7 bits of noise per readout. Different settings of the filter wheel and the grating order have no effect on the readout noise and therefore only affect the data reduction by influencing the actual slope.

The following analysis is based on raw data from FM instrument level tests on ground, which were considered *what we will likely see in space.*[1] We will continue with these test data, because the developed reduction and compression strategies are based upon them. We will see that the red and the blue array differ in this dataset in that the red one has a much larger signal as well as a considerable hook, whereas for the blue array the SNR seems to be the main concern. For this reason we need to analyse both datasets for now.

> Hi Roland.
>
> Sorry for the late answer – I was in a bit of a summer-hibernation.
>
> I think the most appropriate data can be found in the buffer transmission data we took in the loops over the different capacitances:
>
> ⋮
>
> Cheers,
> Bart
>
> —*email received on the 14th of September in 2007*

The test procedure that was conducted is a loop over integrating capacitances (0.1 pF, 0.2 pF, 0.4 pF, 1 pF) in *buffer transmission mode* (a few seconds of raw data are recorded for all pixels) at 3 different grating positions. It takes about 50 minutes to complete. The blackbody temperatures were at 22.6 and 25.2 K for the analysed dataset. That file contains 72 compressed entities for the red as well as the blue channel, each compressed entity having 512 raw frames and therefore adding up to 36864 frames per channel. Both channels will be treated hereafter.

Spatial and Temporal Features

To get an impression of the spatial and temporal characteristics of the data, I picked a few frames for displaying the main characteristics and chose three pixels to represent *bright*, *medium* and *dark* classes. Figures 5.5 to 5.8 show how the image evolves during ramp integration. The first image is just like a bias frame and as expected, contrast grows with integration time. Obviously the columns have more in common than rows. Exact calculations (as can be found next to these figures) underline this statement. σ_h is the average of all 16 standard deviations that were derived from calculating the mean of a 25-pixel line, σ_v is the same for all 25 16-pixel rows. However, two things can be derived from that: a pixel does not *know* much about its horizontal neighbour and is also quite independent in vertical direction. Secondly, this becomes even clearer as the contrast grows during integration. This finding more or less disqualifies spatial redundancy reduction steps, as the uncertainties in the temporal axis are smaller (this will be uncovered soon).

[1] In the meantime, real observations from the performance verification phase are available and the best instrument setup seems to be the one with the highest readout noise.

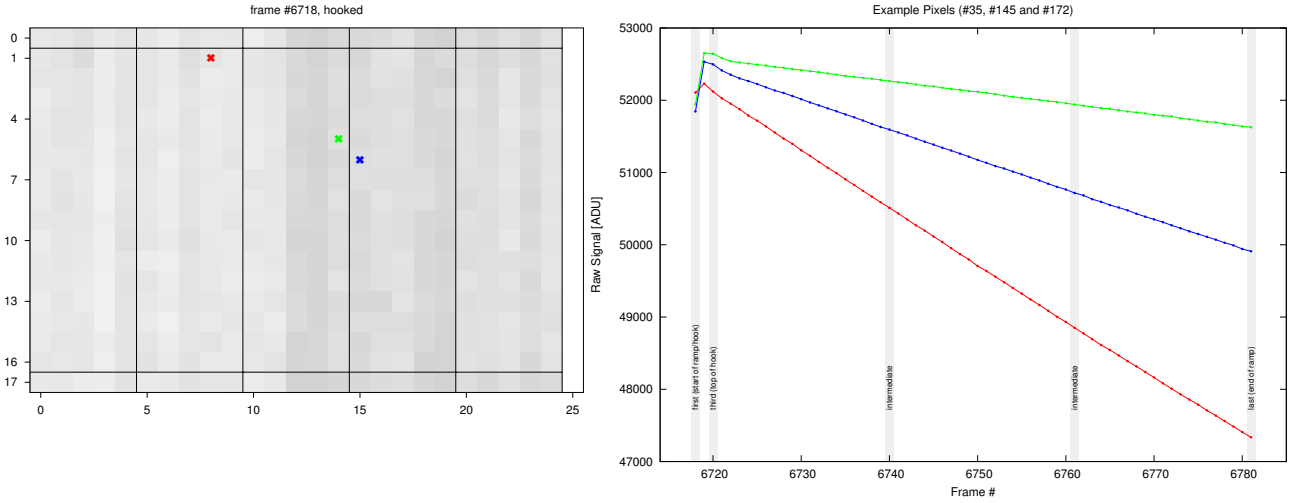


Figure 5.3: **Red example data.** The image to the left is an example frame for the red array. In the top line we find the *open channel*, the bottom line is made up of the *resistor pixels*. On the right end of the array is the *ghost column*, which actually belongs to the left supply group and would be in the middle of the array if real pixels were attached. The interesting part is the inner 16×25 pixel area. (Right) Example ramps for the three marked pixels are given. The black and white colour scale of the image frame is taken from this plot.

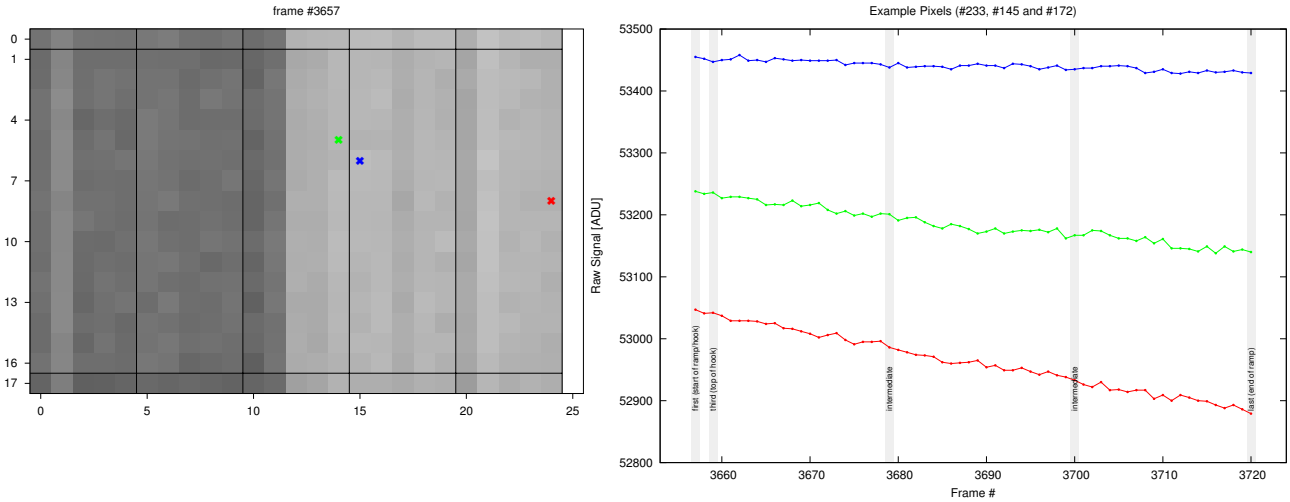


Figure 5.4: **Blue example data.** In the blue array the two supply groups are easily discernable. By looking at the ramps we get a feeling of the different behaviour in linearity.

The Three Pixels

A threesome of pixels have been chosen for each array to represent the rest when we focus on the temporal properties. Their locations are indicated in Figures 5.3 and 5.4, where example ramps can be found as well. In Figures 5.9 and 5.10 the test procedure – test the four capacitances at three different levels of input signal – is well recognised. We will now take a closer look at the data during different stages of the performed test procedure with special interest on the readout noise.[1]

[1] I prepared the analysis steps necessary to separate different sources of noise after discussion with Rainer Hönle from MPE, an experienced FIFI LS [Ros02] developer.

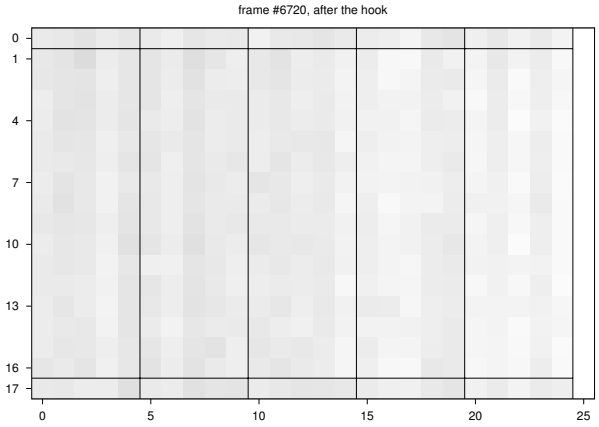


Figure 5.5: **Start of integration.** The first 6 frames are still affected by the *hook*. Contrast is low, as the exposure has just started. σ_v=78.2 and σ_h=186.6.

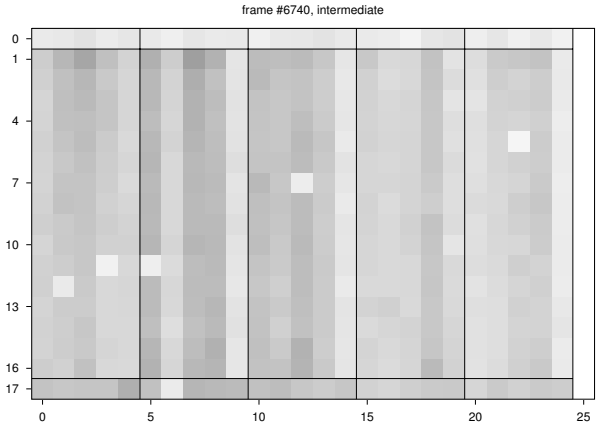


Figure 5.6: **Early ramp.** After the initial *hook* the signal is as linear as the individual pixel allows it to be. σ_v=170.1 and σ_h=441.7.

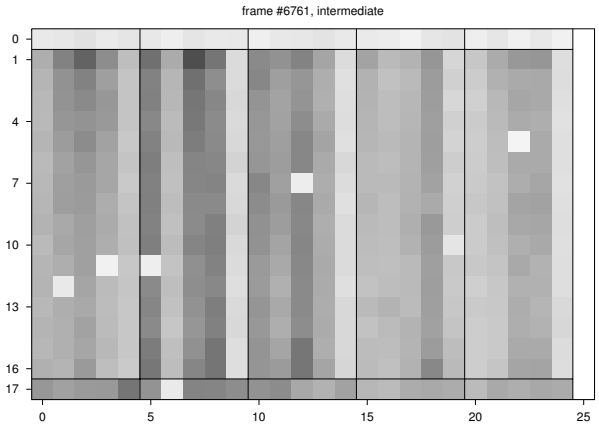


Figure 5.7: **Towards the end the contrast has grown.** This is best seen by comparing the open and resistor pixels. σ_v=285.4 and σ_h=788.0.

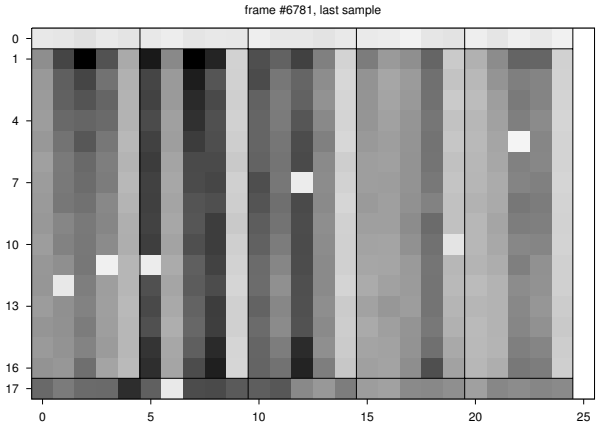


Figure 5.8: **The last frame of the ramp has the best contrast.** As a consequence, the pixel to pixel variation is at a maximum here. σ_v=397.1 and σ_h=1111.6.

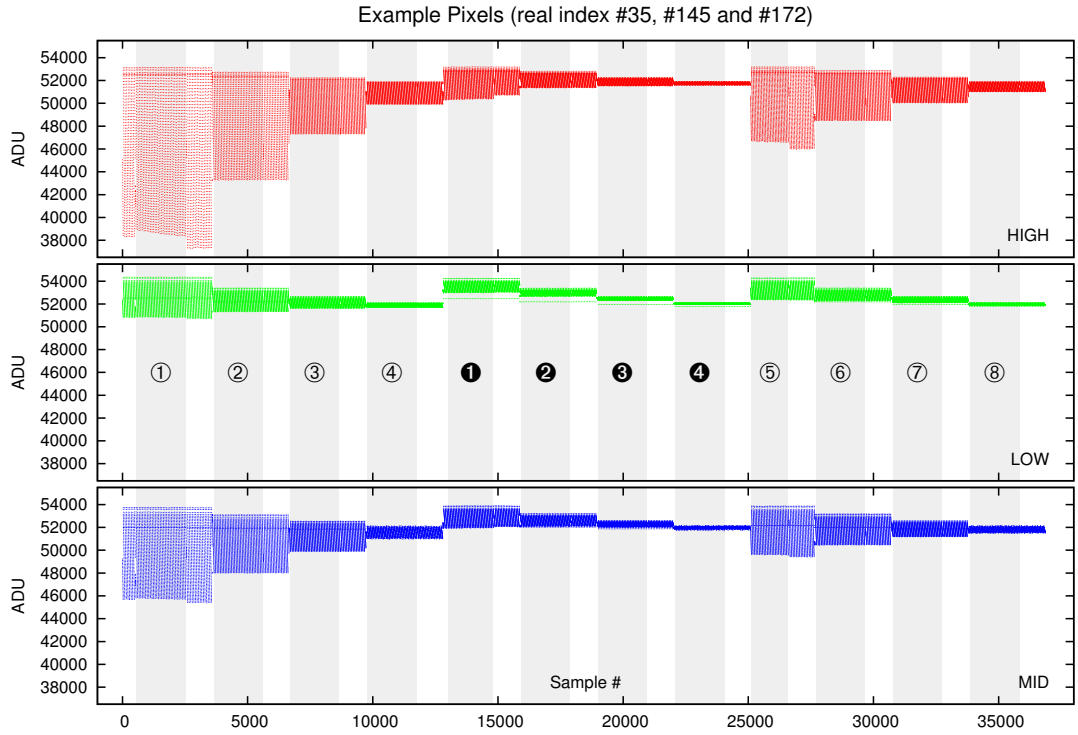


Figure 5.9: **An overview of the entire red test file for the three example pixels.** The file is split in 12 sections of different size, only the contiguous parts shaded in grey have been used for analysis.

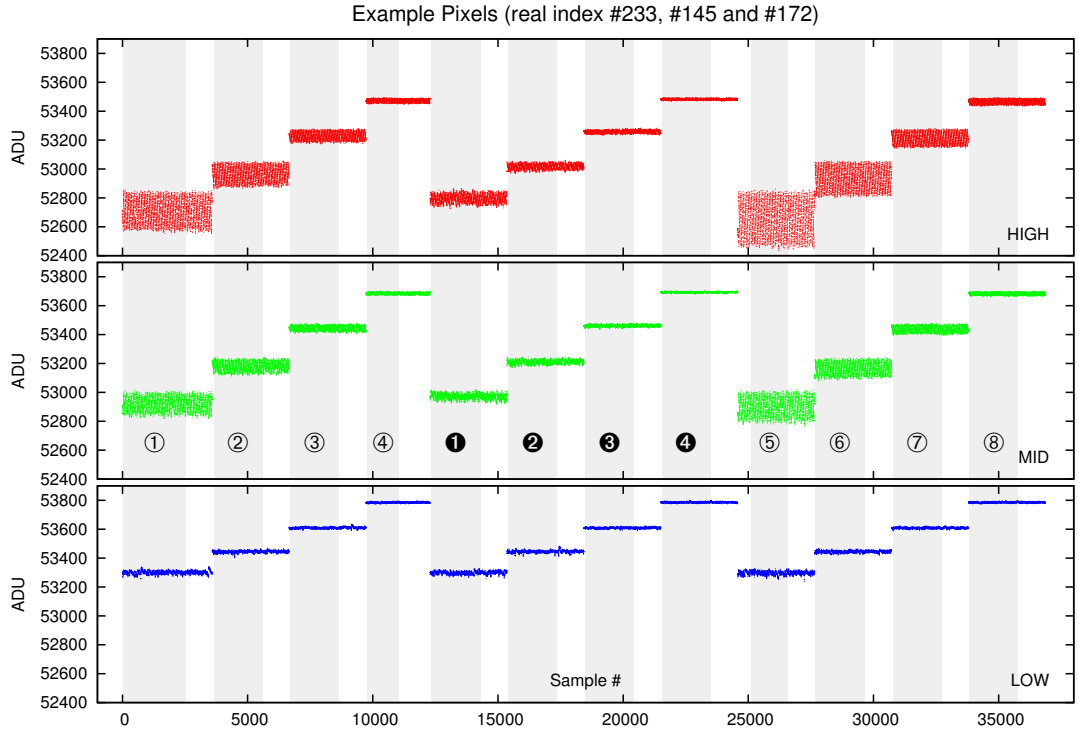


Figure 5.10: **Overview for the blue data.** Note the much smaller dynamic range.

2 Separation of Signal and Noise

As mentioned before, it is not as easy as in photometry to determine the readout noise. To distill the noise, we have to remove the systematic components – the ramp, the hook, the nonlinearities. But how hard we may try, we will not succeed completely, because due to the sampling up-the-ramp the noise is a compound of random walk and the readout noise. Yet for the purpose of compression it is not necessary to fully disentangle these two.[1] By the way, in case you wonder about the ramp length – it is something that needs to be optimised with respect to SNR. If the ramp is too short, then it is dominated by Poisson noise, but increased length brings us closer to saturation, it means a higher chance of being hit by a glitch and don't forget that a random walk is divergent.

Now let's start to find the best way to isolate the noise. What follows now is a description of different steps that would work as a decorrelation. Several plots illustrate the immediate implications and most importantly, the sample entropy H_S is given to get an objective measure of compressibility. In the explanation of the different steps I use yet another example pixel from section ② of the red array. This one is located several pixels to the right of the *red* pixel (at $x, y = 16, 1$). It was chosen for aesthetic reasons.

From Ramps to Noise

The central point to start with is the *keyramp* (or *mean ramp*) of a pixel, providing the typical integration ramp. It is derived by averaging all samples with the same index on the ramp. The average of all first samples of the ramps is the first sample of the keyramp and so on. In Figure 5.11 the raw data are shown with the derived keyramp next to the plot. Such a keyramp contains already the main systematic components of the data. Its shape is not influenced by the reset noise and if the slope would not vary, the residuals were just noise. Yet Figure 5.12 tells us that there is still some way to go.

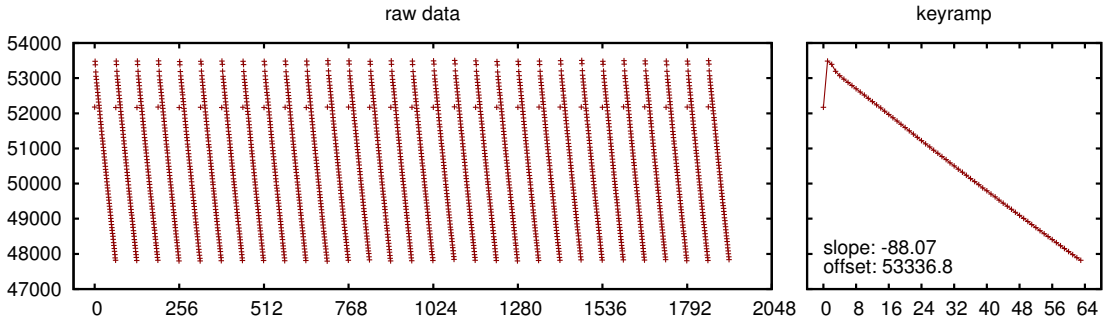


Figure 5.11: **Raw data of red pixel 43 on section ②.** The abscissa has the sample number, the ordinate is in ADU. The units stay the same for all plots of that kind. (Right) The keyramp derived from these data is pretty linear, well, apart from the hook. Note that the ordinate is in the same scale.

[1] In Chapter 1 we learned that random walk noise is the result of the integration of white noise, hence we could *decorrelate* it by differentiation, which in turn increases the uncorrelated readout noise.

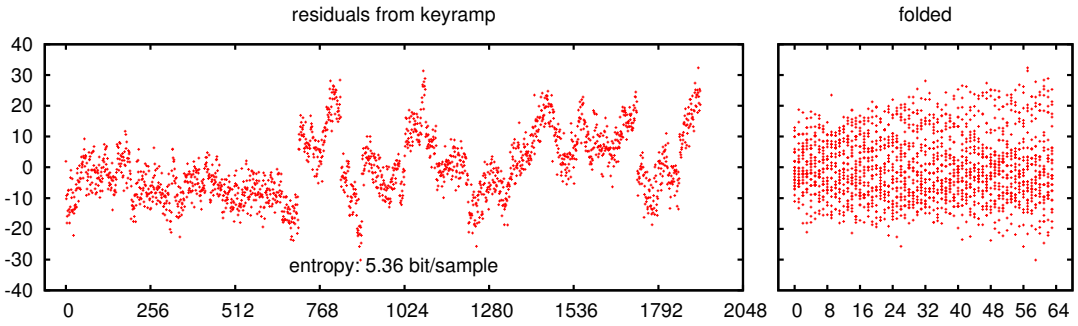


Figure 5.12: **The residuals from the keyramp (same units as before).** The range of the ordinate was adjusted to ±40. (Right) The *folded* plot is an over-plot of the ramp residuals, showing a cone-like structure. It is however broadened by the reset noise.

Residuals from the Keyramp

The difference between data and keyramp contain systematic effects that still need to be separated. First of all, the reset noise makes up for a *shredded* appearance in Figure 5.12. At a first glance this operation may seem disappointing, because we can still identify features that have the length of a ramp. Nonetheless virtually all residuals are already in a range between ±30, or, in terms of entropy, around 5 or 6 bits.

Reset Noise

Simply subtracting the keyramp leaves many structures in the residuals, but we are so convinced about the usefulness of this operation, that we go ahead with little adjustments. The residuals have not been corrected for the reset noise, something which can be easily done by subtracting the first sample of a (residual) ramp from the rest. That way all residuals start from zero when a new ramp is begun. We can confirm this by taking a close look at the *folded* plot in Figure 5.13, where the first sample from each residual ramp starts at zero. The second sample is then already found somewhere in the scatter field.

What we can see now is that the residuals have improved in that the ramp pieces have been shifted a bit together. The entropy decreased considerably, which is a sign that we are improving. Still, we have an unwanted zig-zag-pattern in our data. We can credit a changing slope due to a sensitivity drift (given that the source signal is constant) for that, but also the random walk is a candidate. In any case this plot is handicapped by the fact that the first sample is not only affected by reset noise, but of course also by readout noise and thus the scatter in the folded plot is higher than it should be (by one time the readout noise).

We can avoid this by subtracting the intercept instead of the first sample from each residual ramp. Now the folded plot looks as if it has been focused (compare Figures 5.13 and 5.14). Still, the cone-like shape is present and we cannot tell whether it is due to drift or random walk. To do so, we would have to plot the slopes and see whether they are systematically changing. An unsystematic change would be due to random walk. You see, whichever of the two components – drift or random walk – stands more above the readout noise is the one that can be credited for the cone shape in the intercept corrected plot.

In any case the intercept is a better corrector for the residuals as it leads to a smaller entropy. Yet for real-time compression resources can be tight and so it may not be possible to do a slope fit. Instead, the idea is to correct each residual ramp by its

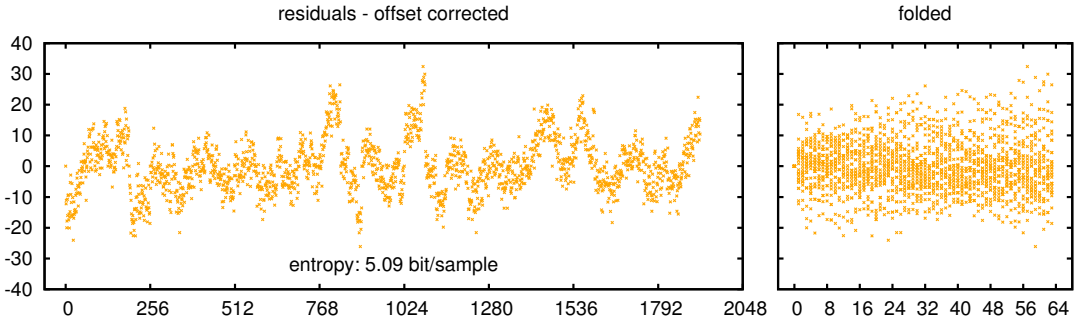


Figure 5.13: **The residuals corrected for the reset noise by subtracting the first sample of the residual slope.** All ramps in the folded plot thus start with 0.

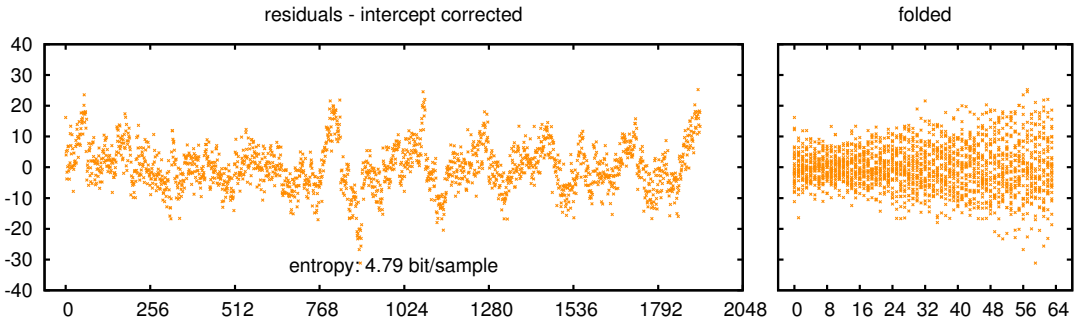


Figure 5.14: **The residuals corrected for the intercept.** The cone-like shape in the folded plot is now as clearly visible as it can get.

mean. The result, as shown in Figure 5.15 is yet another decrease in entropy and an interesting hourglass-like shape in the folded plot. This is because of a systematic effect – especially if linear – in the residual ramp, which led to the cone shape in the folded plot before. So the residuals will now be re-centred in the middle of the ramp. Through this exercise we don't learn more about our data, but we see that this is a powerful preprocessing for the purpose of compression. You are now invited to speculate how we could use this to disentangle random walk and slope drift. In any case we still have a zig-zag in the data, even if it's much reduced by now.

Readout Noise

So what else can we do? Both, the cone and the hourglass are caused by mostly linear systematics. Again, starting from the untouched residuals, the last word is now spoken by fitting the (residual) ramps again and correcting with the derived lines. That way the reset noise is removed by the intercept, the sensitivity (signal) fluctuation and the linear part of the random walk is removed by the slope. What is left is mostly the readout noise. In Figure 5.16 the residuals appear almost like white Gaussian. A large random walk would still be visible by a $\sqrt{n}$-like cavity in the folded plot, but in this dataset the readout noise is of a comparable intensity.

Are the reduction steps also good as a decorrelation? Well, they are not, because they are not revertible unless additional values – the offset, intercept and slope – are saved for each ramp to do the reconstruction. Remember our three example pixels? I will now give you a gallery of useful plots for them on different data sections and you are again welcome to speculate about your own reduction system, especially concerning the reversible decorrelation stage.

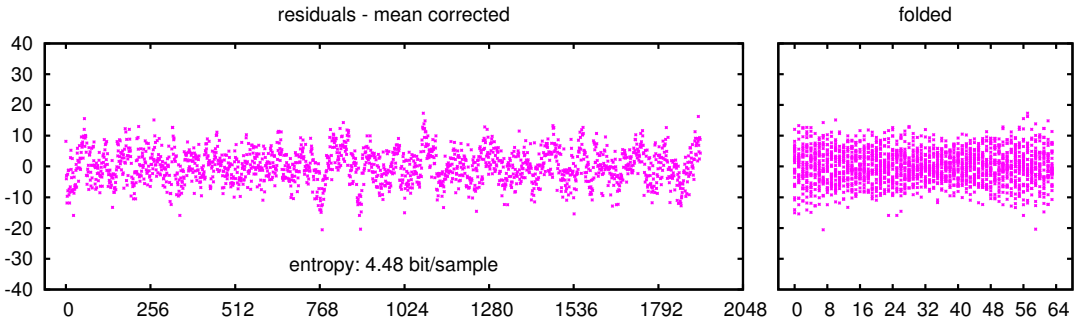


Figure 5.15: **The residuals corrected for their mean.** This leads to a re-centring and a further reduction in entropy. The hourglass-like shape in the folded plot is the consequence.

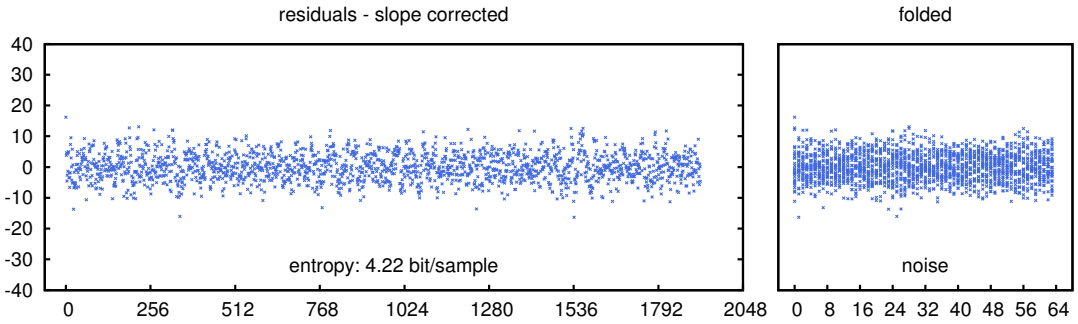


Figure 5.16: **The remaining noise after all the correction steps.** The folded plot reveals that the remaining noise is the same throughout the whole ramp. Only slight traces of the zig-zag pattern are left in the data. This is as close as we can get to the readout noise.

Details for Selected Pixels

The steps above are now applied to the three pixels for selected sections from the data file and presented in graphical form. A few additional plots, especially the comparison between intercept and slope and a histogram of the readout noise will complete the view on the facing page. The given entropy should be read like *this pixel on this section with that kind of processing has an entropy of H_S bits per sample*. *Red*, *Green* and *Blue* pixel are again referring to their origin (refresh your memory at Figure 5.3).

After the plots that concentrate on single pixels additional information is given in Figures 5.25, 5.26 and 5.27 about the character of the whole array. Knowledge of these data properties will be useful for the next section, where the implemented decorrelation will be described in detail.

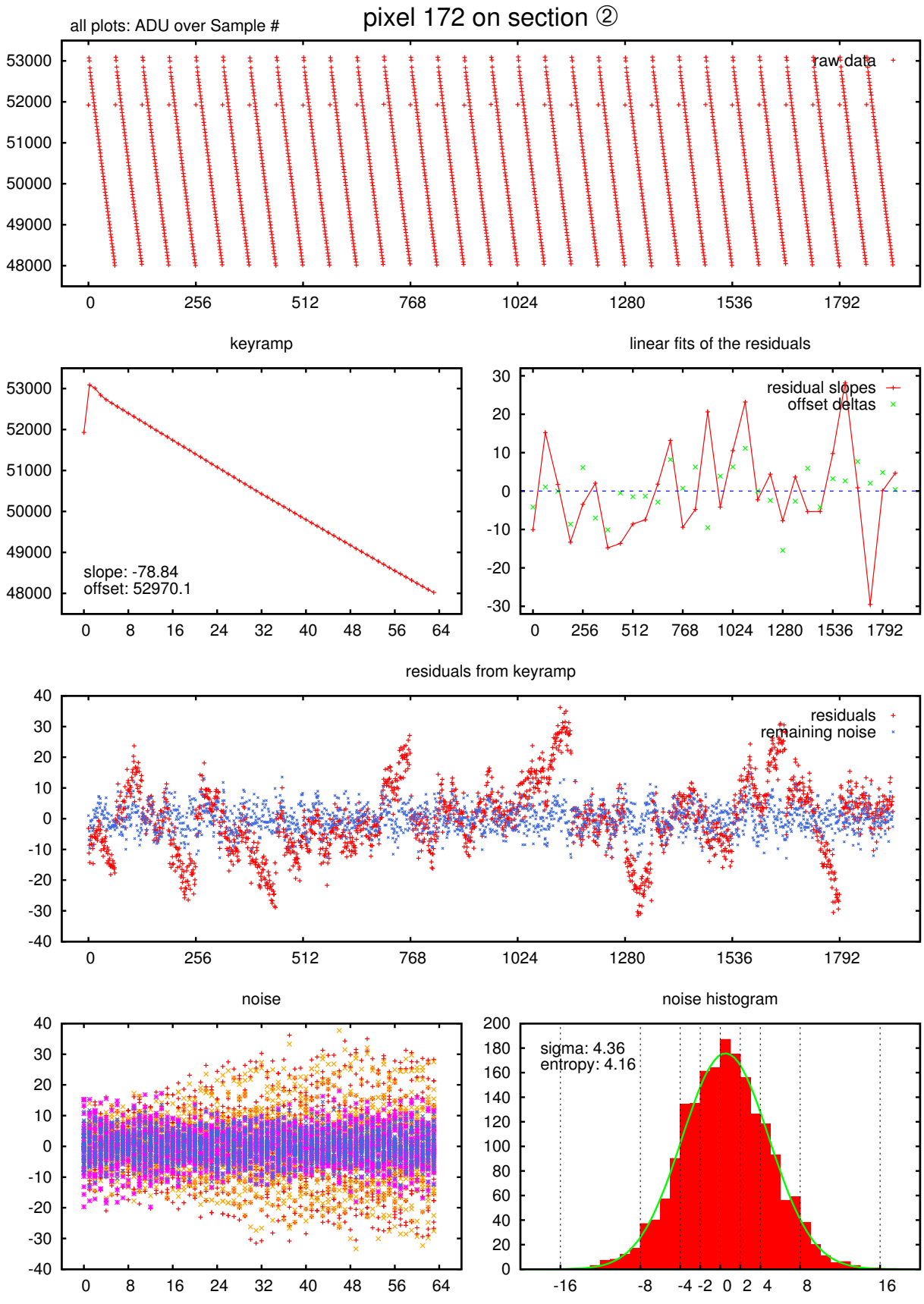


Figure 5.17: **Red array, blue pixel, section ②.** In the plot to the right of the keyramp we see that the slopes are not correlated with the offset noise. A histogram of the readout noise [quantity over ADU] shows a Gaussian profile. The bottom left plot is an overlay of all folded plots.

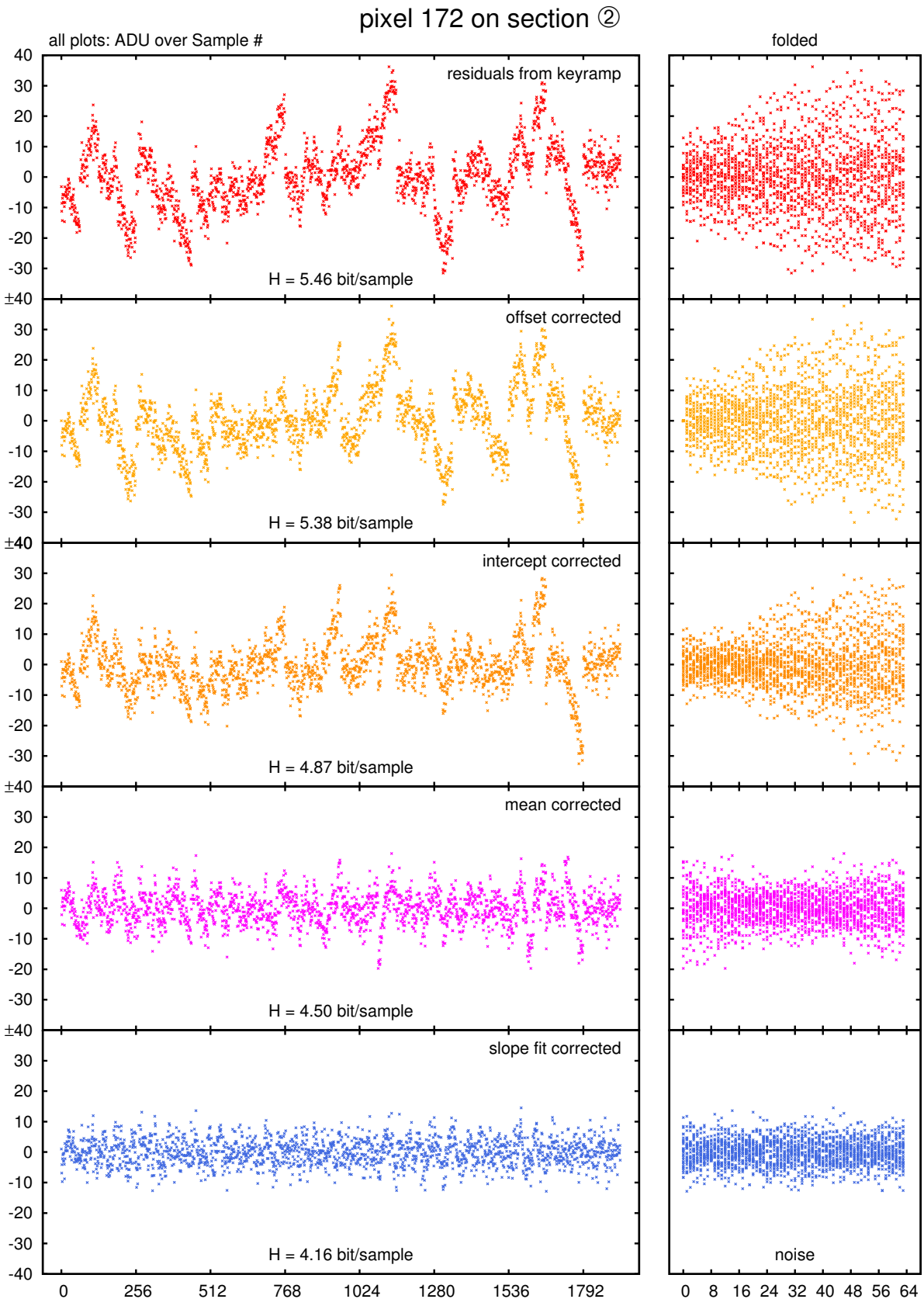


Figure 5.18: **Various plots of the residuals.** This pixel nicely demonstrates how the different reduction steps extract the noise. It is also worth noting that the initial hook leaves no traces in the residuals. Yet we still see some spikes in the slope fit corrected residuals (Bottom).

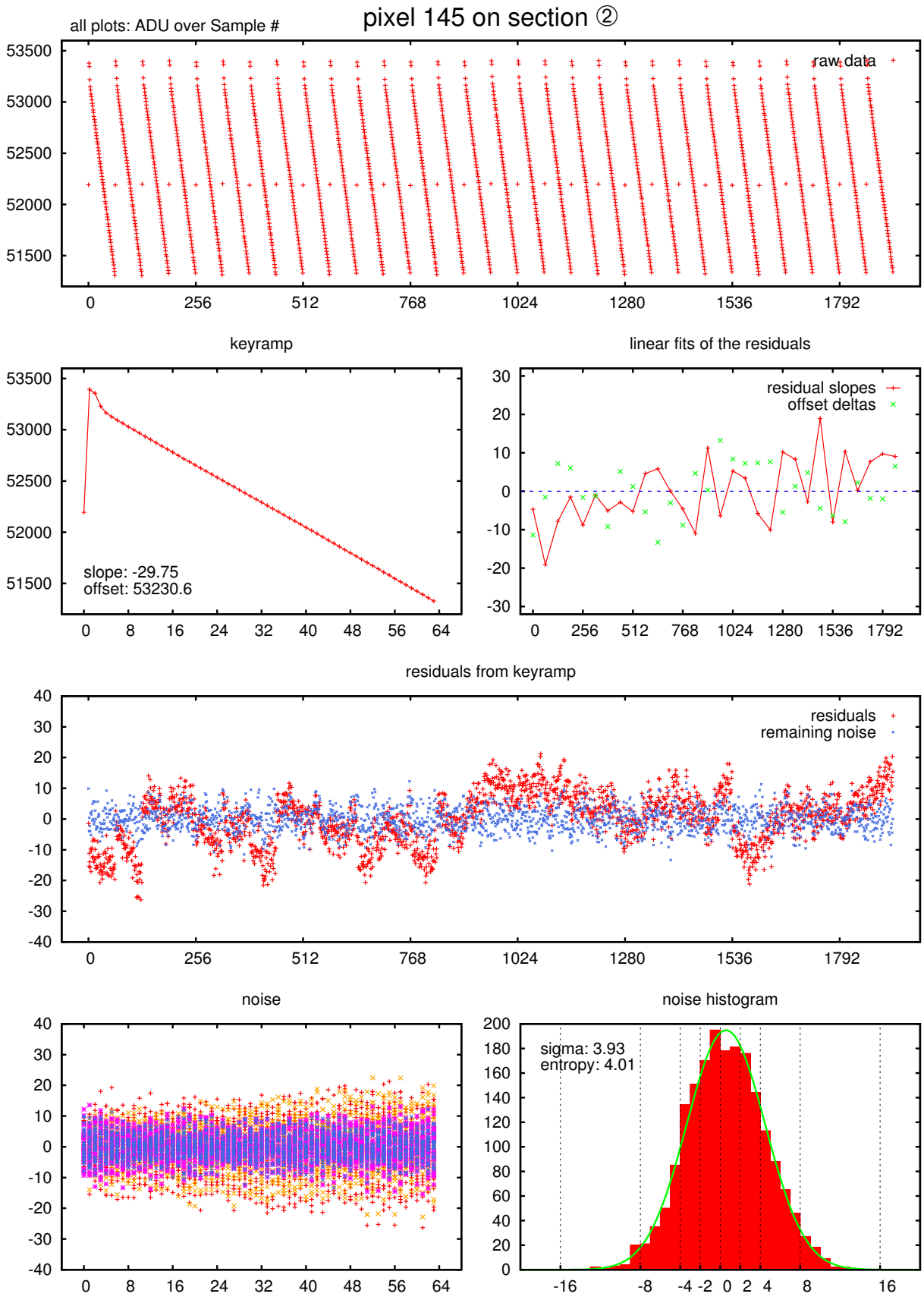


Figure 5.19: **Red array, green pixel, section ②.** This one has an even larger hook, but the reason why I picked it is that we see a drift of the slope. In this case we cannot credit the cone shape to the random walk.

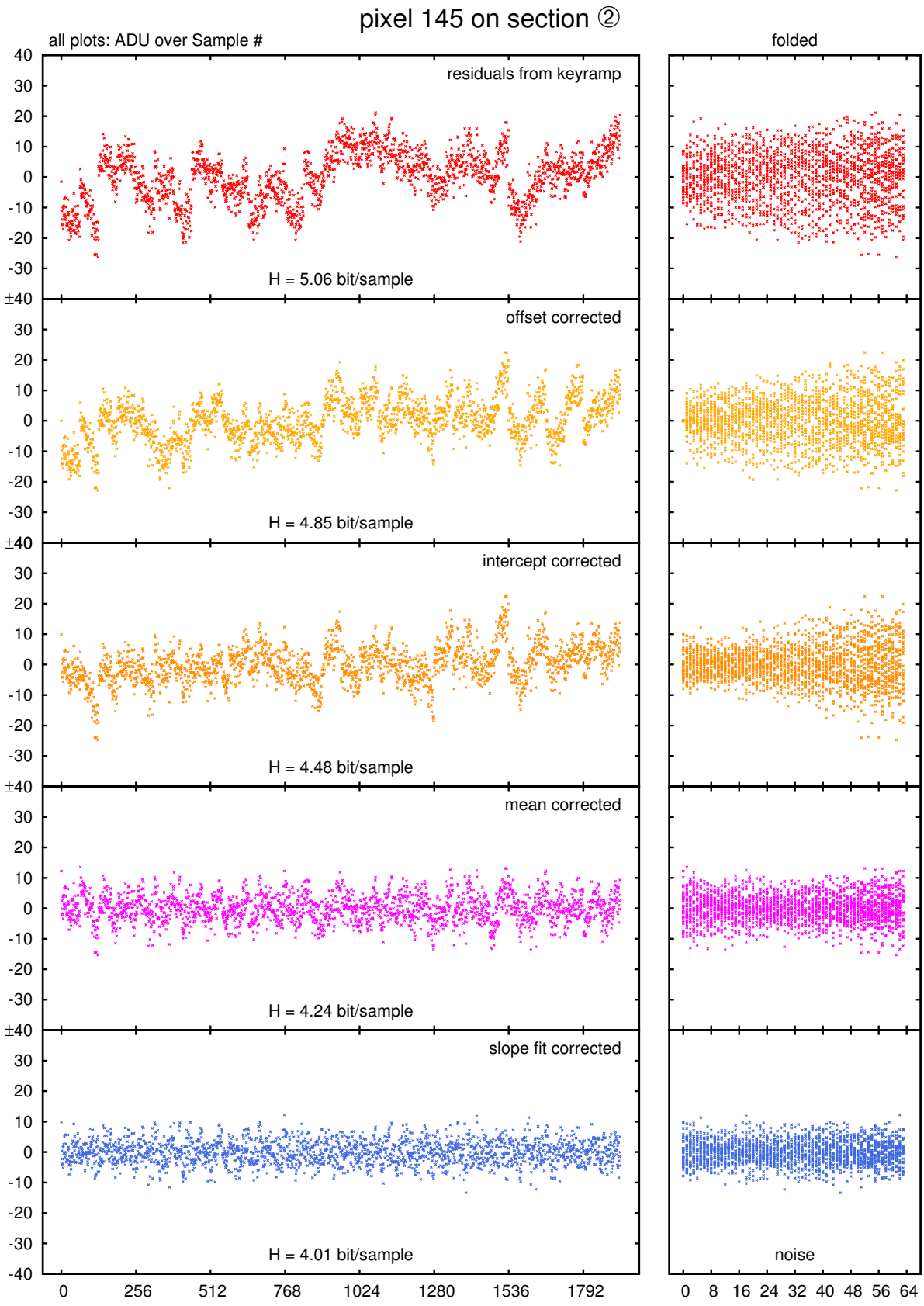


Figure 5.20: **Residuals.** Most of the cone shape in the intercept corrected plot comes from the drift. The difference in the entropy between the uncorrected residuals and the fitted ones is 1 bit, but a large reduction is already achieved by the mean correction.

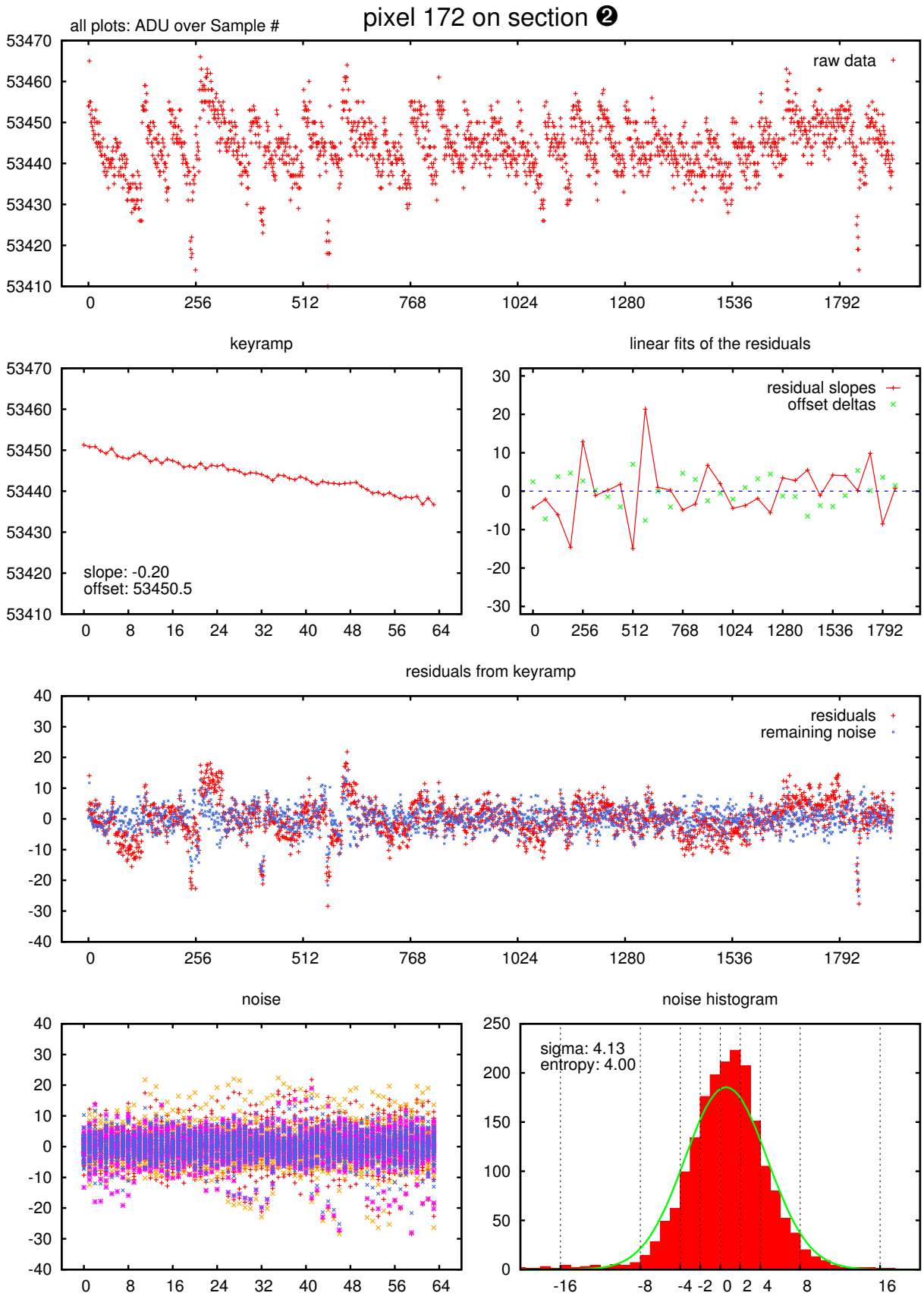


Figure 5.21: **Blue array, blue pixel, section ❷.** Here we have no hook and the slopes are in general much smaller. The dataset shown contains four glitches. They disturb the slope and as they rip apart the ramp they stand out in the residual as well. Even the histogram is spoiled by that.

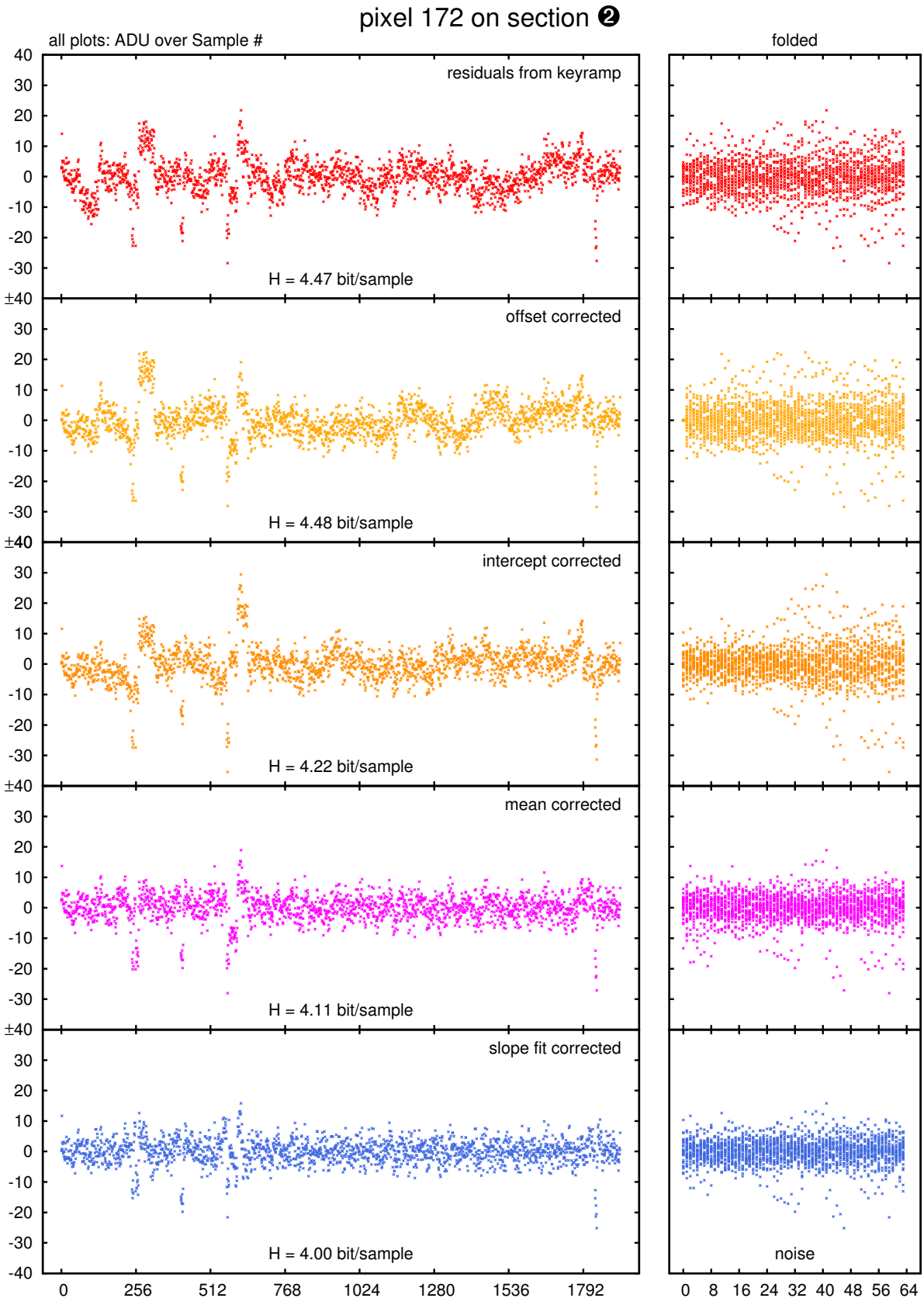


Figure 5.22: **Residuals.** The effect of the glitches is not corrected by any one of the different methods. We also see that the entropy savings are only little here.

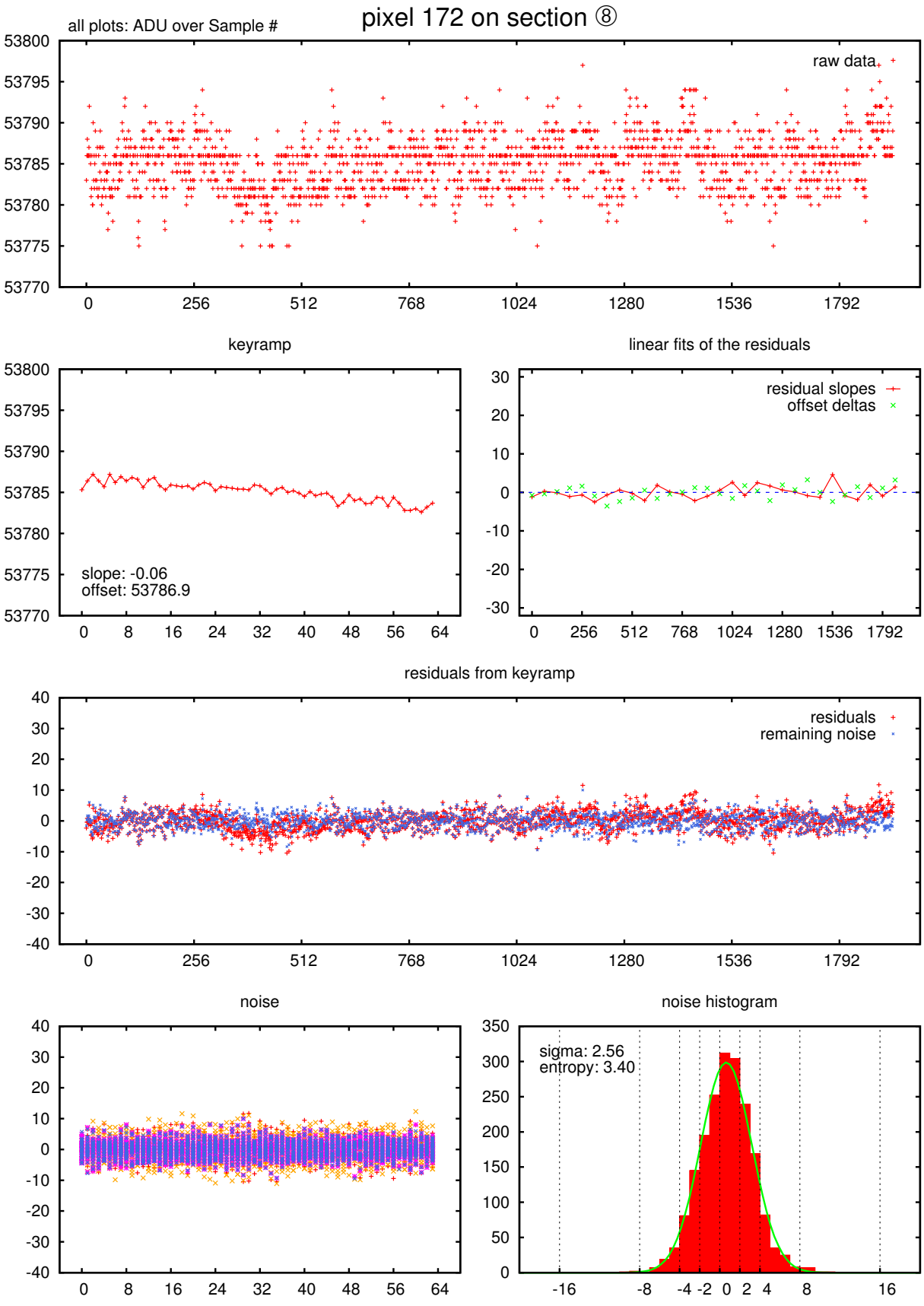


Figure 5.23: **Blue array, blue pixel, section ⑧.** This is the same pixel as before, but due to the different capacitor now with with little signal. The ramps are no longer easily visible in the raw data.

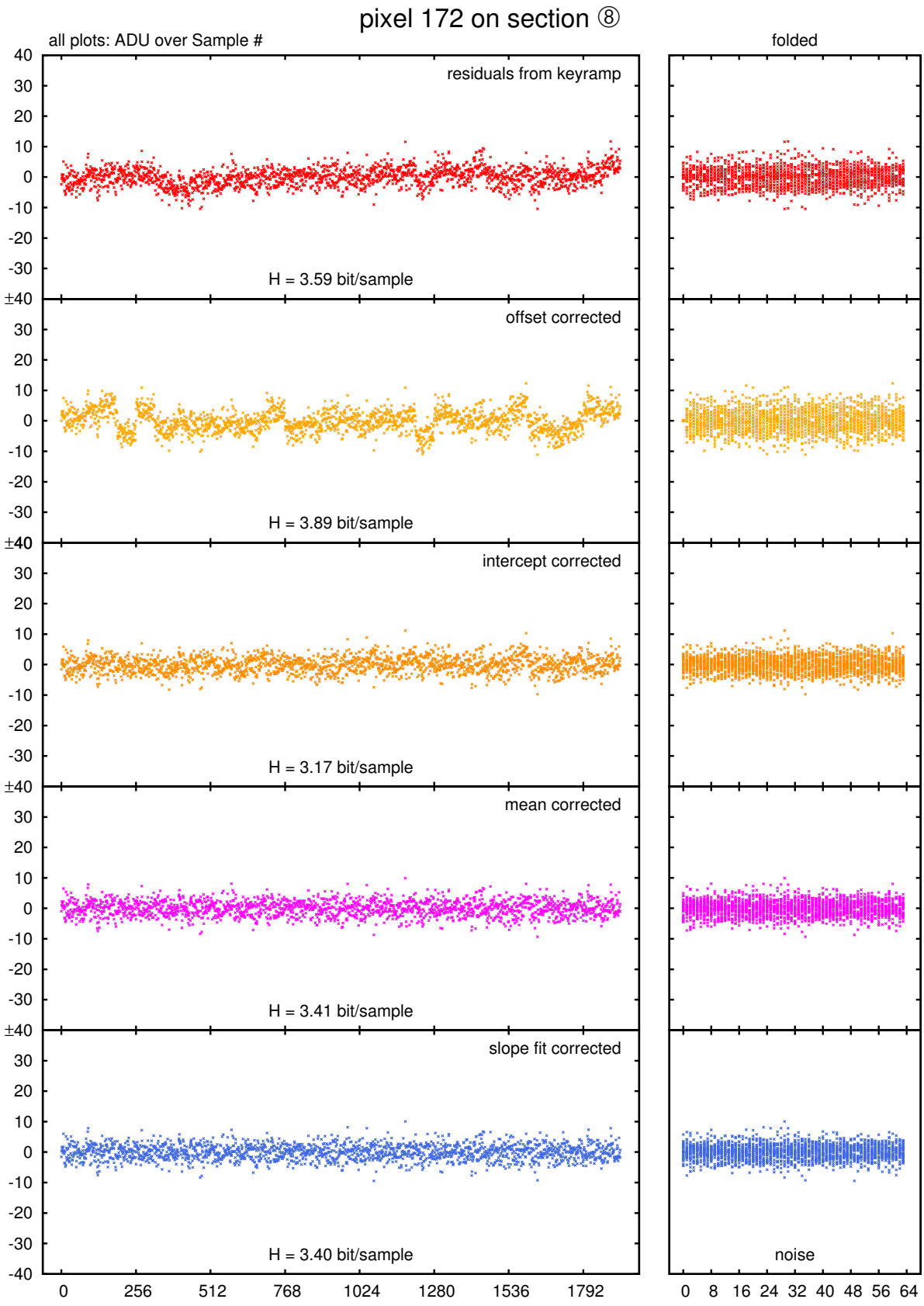


Figure 5.24: **Residuals.** We are running into troubles with the entropies now. This is an indication that the ramp is no longer well sampled. There is no cone although we clearly see a pattern in the offset corrected residuals, which is a sign that we are now dominated by readout noise.

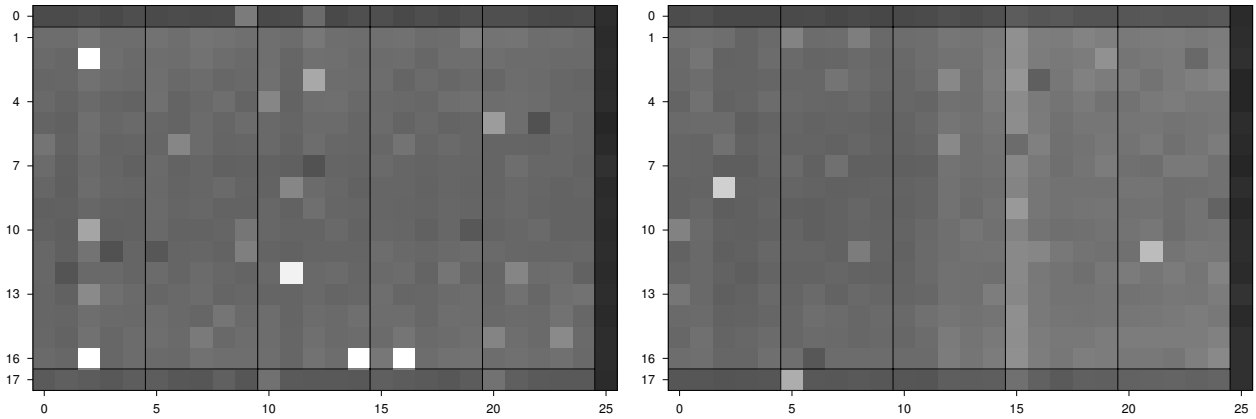


Figure 5.25: **Noise map of the red (Left) and blue (Right) array for section ❷.** By noise the standard deviation of the slope fit corrected residuals from the keyramp is meant, which is as close as we got to the readout noise. A few isolated pixels are much noisier than others and some produce less noise. The colour scale is σ =0 ADU for black and 15 ADU for white.

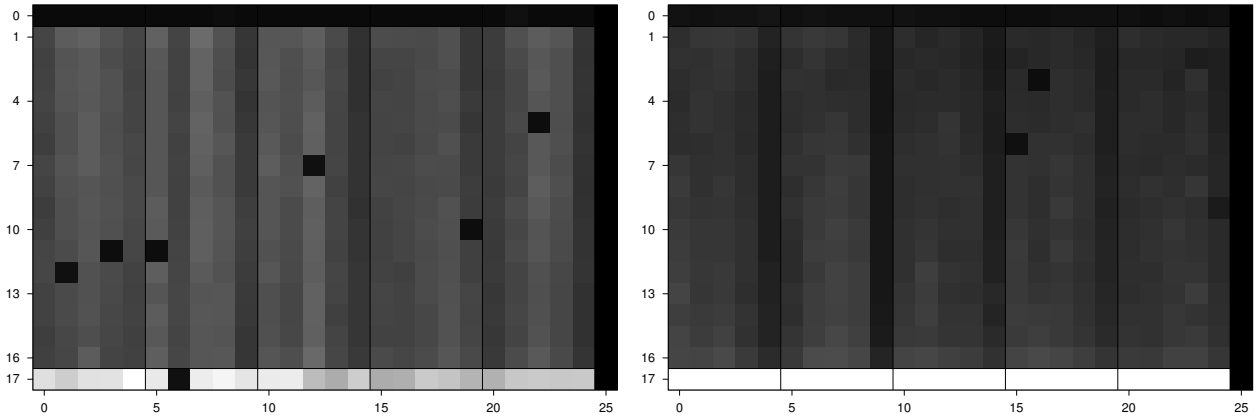


Figure 5.26: **Responsivity maps, showing how the dynamic range is used by the slope.** This is to be interpreted as a flatfield. Especially in the red we see that pixels of the same column have a similar slope. The scaling in here is 0 ADU/256 Hz for black and -127 (red), resp. -15 (blue) for white colour. The open channel has no slope and is thus black. Bad pixels have no or almost no slope and show up in black as well. They correspond well to what we thought were the low noise pixels in the noise map.

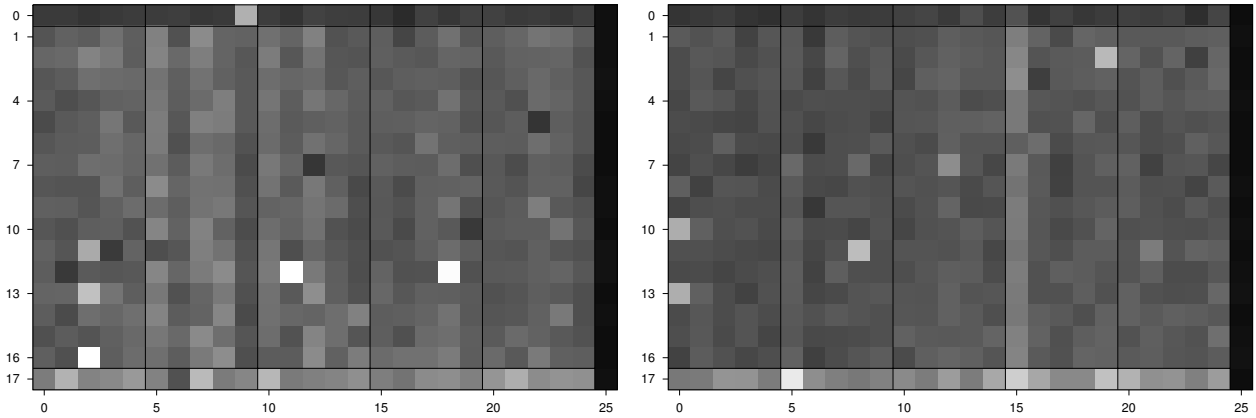


Figure 5.27: **Fluctuation of the slope as measured by its standard deviation.** Black stands for 0 and white for 0.5 ADU/256 Hz. Such a plot can be used to weight individual pixels.

Summary of the Analysis

The most important information from all the plots made is the entropy of the noise. Apart from that, the following statements seem to be justifiable now:

- There is no obvious correlation between offset noise and signal. The magnitude of the reset noise is similar to the readout noise, depending on the capacitance used. There is also no obvious correlation between the size of the hook and the offset noise.
- The readout noise is the same over the whole ramp.
- The nonlinearity and especially the hook are sufficiently well covered by the keyramp.
- Random walk and variation of sensitivity cannot be sufficiently well separated by the presented methods, which is mostly because random walk, drift and readout noise have much the same magnitude for this dataset.
- Different capacitors have a large influence on signal and noise. Through the entropy this drives the data rate.
- The entropy of the readout noise ranges between 3 and 6 bits per sample, but mostly being around or below 4 bits.

3 The Devised Reduction Scheme

Before we have a look at the implemented reduction scheme we need to recall that the raw data need to be reduced to meet the large compression factor of ∼40, either by slope fitting or by averaging. In the latter case, not the whole ramp is averaged – this would remove the meaningful signal – but partitions of the ramp are. A meaningful setup is to create four so-called sub-means for a ramp of 64 samples. To get a better temporal resolution of the ramp a setup with 8 sub-means on a ramp is preferable, but we will see that in this case the data rates are normally above the limit.

Rampdiff

During spring 2007 I came up with what I called the *rampdiff* compression scheme, which aimed at keeping as much temporal resolution of the ramp as possible.[1] The goal was to make a setup of 8 sub-means per 64-sample ramp compatible with the downlink limit. From the analysis of the FM test data we remember that the entropy per sample is ∼4 bits, so a perfect lossless compression would be able to achieve a data rate of 450 (pixels per array) × 2 (arrays) × 256 (Hz) × 4 (bit entropy) = 921.6 kbit/s. In case of averaging 8 samples, we reduce the 4 bit readout noise ($\sigma = 13.2$) by a factor $\sqrt{8} = 2.83$ (13.2/2.83=4.67, or $H_S = 2.96$), so we get 450×2×256/8×2.96 =85.2 kbit/s. Even taking some overheads into account this should stay well below 120 kbit/s – in other words, it should be possible.

After ramp fitting or sub-mean averaging has been performed, the data need to be decorrelated again before they can be passed to the lossless compression. This is done

[1] This has become the default FM compression scheme, because it works also well for different other scenarios such as slope fitting.

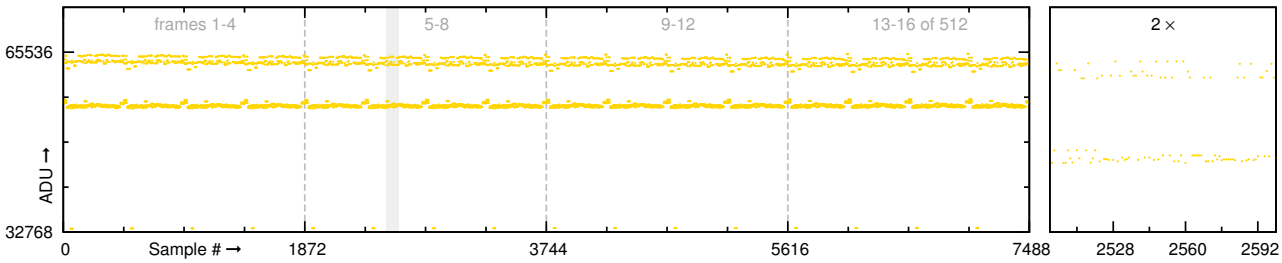


Figure 5.28: **Raw input data for spectroscopy.** This is a 1D-plot of the data as they are received by the SPU from the DMC. The double line is due to the two different supply groups.

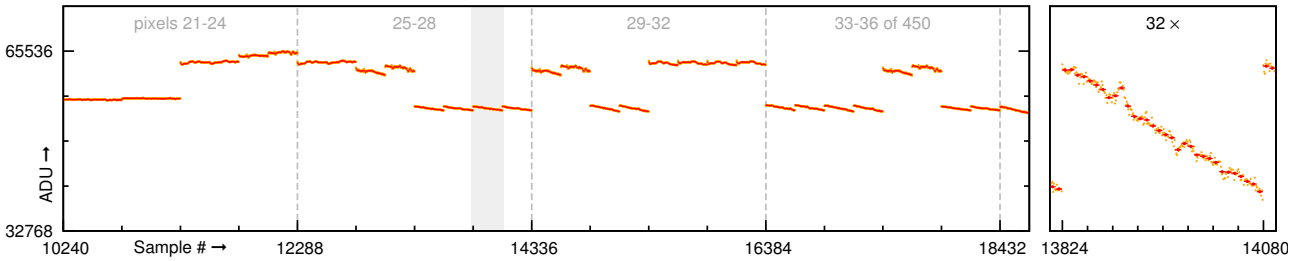


Figure 5.29: **Ghost column removed, pixelised and averaged.** The ramps can be distinguished now. In the magnification we see how the sub-mean averaging reduces the temporal resolution.

in a different way as in photometry and the details follow now. For better visibility in the plots I chose a dataset with longer reset intervals. The ramps used to create Figures 5.28 to 5.35 have a length of 256 samples.

1, 2: Preprocessing and Averaging

The first processing step is again to discard data from pixels that were deselected. Even if the whole frame is selected, the *ghost column* is deactivated by default. After that, the data are rearranged to *pixel order* just as in photometry, to group the samples of the same pixel together. This is shown in Figure 5.29, where the averaging step is also already included. In spectroscopy the SPU synchronises to the start of a ramp as indicated by dedicated header parameters. As you can imagine, unsynchronised slope fitting or averaging would be disastrous to the ramps. The averaging routine that is used in spectroscopy is the same as in photometry. This means, that additional bit rounding can be commanded as well.

3: Ramp Differentiation

Differentiation is also an important ingredient in the *rampdiff* scheme. First of all, each ramp is differentiated so that the first value of a ramp is left untouched and the remaining ones become small negative values. This is done in place by applying the `Delta` function (the same as in photometry) to each ramp, but it also increases the uncorrelated noise by $\sqrt{2}$ as a side-effect. Figure 5.30 shows the data after this step.

4: Keyramp Subtraction

The analysis of the FM test data in the previous section was based on the keyramp, but our processing steps contain nothing like that so far. As keyramp we used the mean of all ramps, but this would not be profitable in our case where we have only a few ramps per buffer, because the calculated keyramp needs to be transmitted as well. Instead, we use the first ramp of the buffer and subtract it from the others. That way the uncertainties of the first ramp are reflected in the residuals and if the first ramp is hit by a glitch this will create spikes in the residuals, but this is yet another tradeoff we have to make. In Figure 5.31 the result of this operation is shown.

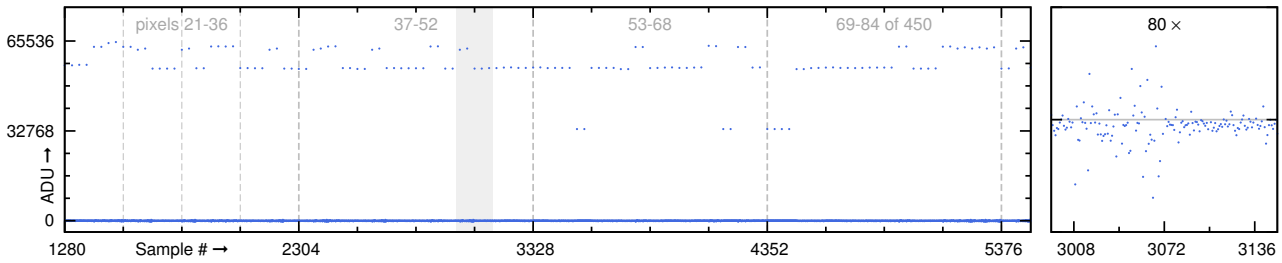


Figure 5.30: **Ramp Differentiation.** The start samples of the ramps have not been altered. In the magnification two neighbouring pixels with very different noise are shown.

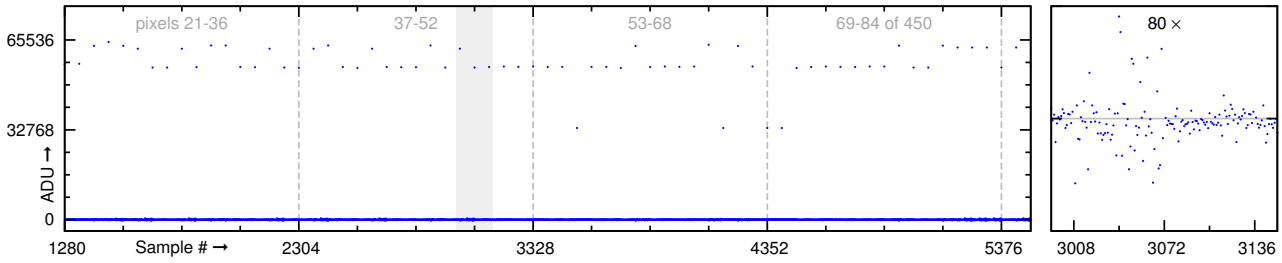


Figure 5.31: **The data after keyramp subtraction.** Now only one unaltered high value remains for every pixel. In the magnification we see that the residuals are unchanged.

5: Differentiation

Although the residuals are close to zero now they will be systematically offset by a constant value if chopping is performed for instance. This makes it necessary to differentiate the residuals another time. By comparing the magnified plot in Figure 5.32 with the previous one in 5.31 we see that this step is justified.

6: Offset Removal

Now is the time to get rid of the remaining large offset values. There is one such value for each pixel which is made up of bias, offset noise and readout noise. Ideally, we would make a bias correction now, but there is no bias frame at hand. In case of sub-mean averaging of 8 samples we get one such value in a group of 64, so there is not much compression to be gained by bringing these values down to zero. However, we try by calculating a mean integer value and subtracting it from these start values. From their distribution we can tell that this will not bring them down to zero, but at least closer to it so that less bits will be needed for encoding. This mean value needs to be included in the compressed data stream otherwise this step cannot be undone.[1] Find the result in Figure 5.33.

7: Mapping

We owe one last step to the back-end encoder, this is to map the signed values to unsigned ones. For this purpose the same subroutine as in photometry is used. Though we are sceptical about the spikes still contained in the dataset, we continue to the next plot showing the overall performance of the decorrelation that was knocked together in here.

[1] I agree that it would be better to use the median instead, but keep in mind that the data rate savings are minimal.

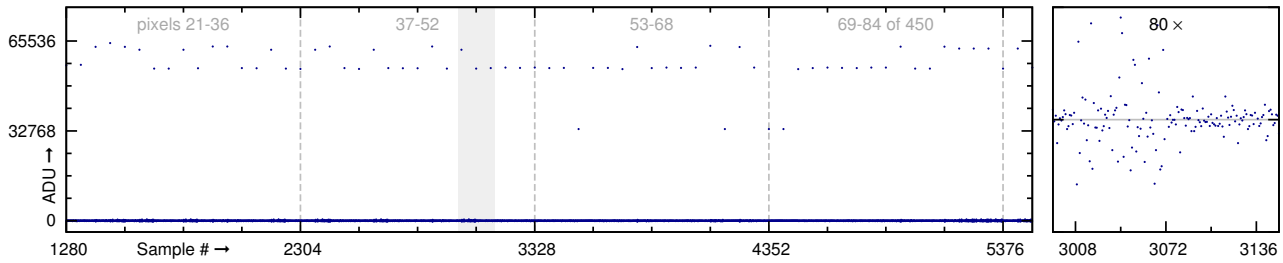


Figure 5.32: **Another differentiation.** The residuals are now really centred, but the uncorrelated noise has been increased.

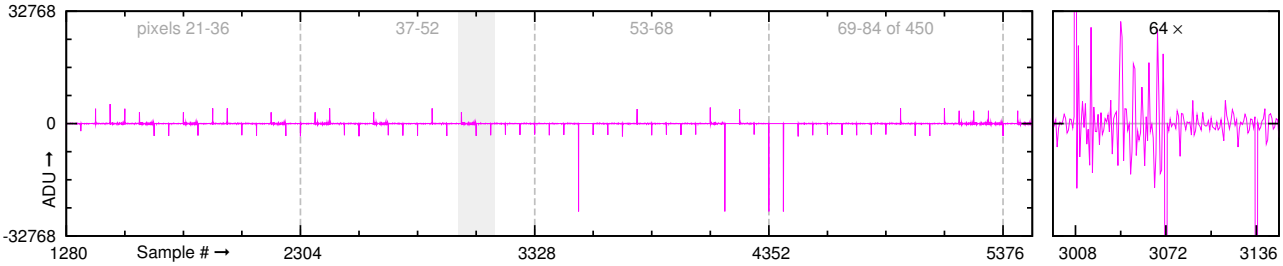


Figure 5.33: **Offset shrinking.** We were able to minimise the spikes yet they are still clearly visible. The other residuals were not changed.

Performance

By reading the histogram in Figure 5.35 we see that the *rampdiff* scheme managed to concentrate the samples that were originally distributed around ∼60000 ADU to a ∼5 bit distribution. The remaining spikes we had in Figure 5.34 can be found as three little heaps around 8000 ADU. They don't matter much, as they are ∼500 values around 13 bits, which is 8 bits more than the rest, so we have a penalty of 4 kbit/s for both arrays together. Yet the distribution is not as concentrated as it could be, mostly owing to the two differentiations.

Data Rate Tests

In Table 5.1 the results are given for all the sections in the FM data file we analysed previously. Three different settings – sub-mean averaging of 16 samples and sub-mean averaging of 8 samples with and without additional 2-bit rounding – were used.

For each section a new input data file for on-board compression tests at MPE was generated. Each test file included the first 512 frames of the respective section. They were further used as input for the software simulator as well as for my own algorithm framework that I use to develop and test experimental compression schemes. Table 5.1

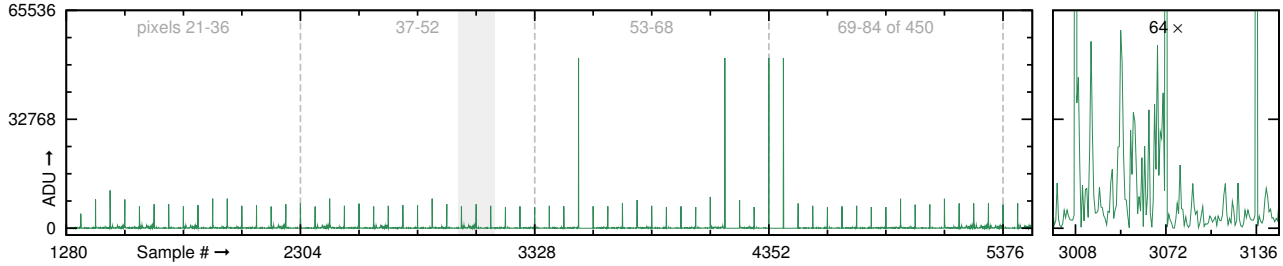


Figure 5.34: **The result of the rampdiff scheme.** The data are unsigned and ready for entropy coding.

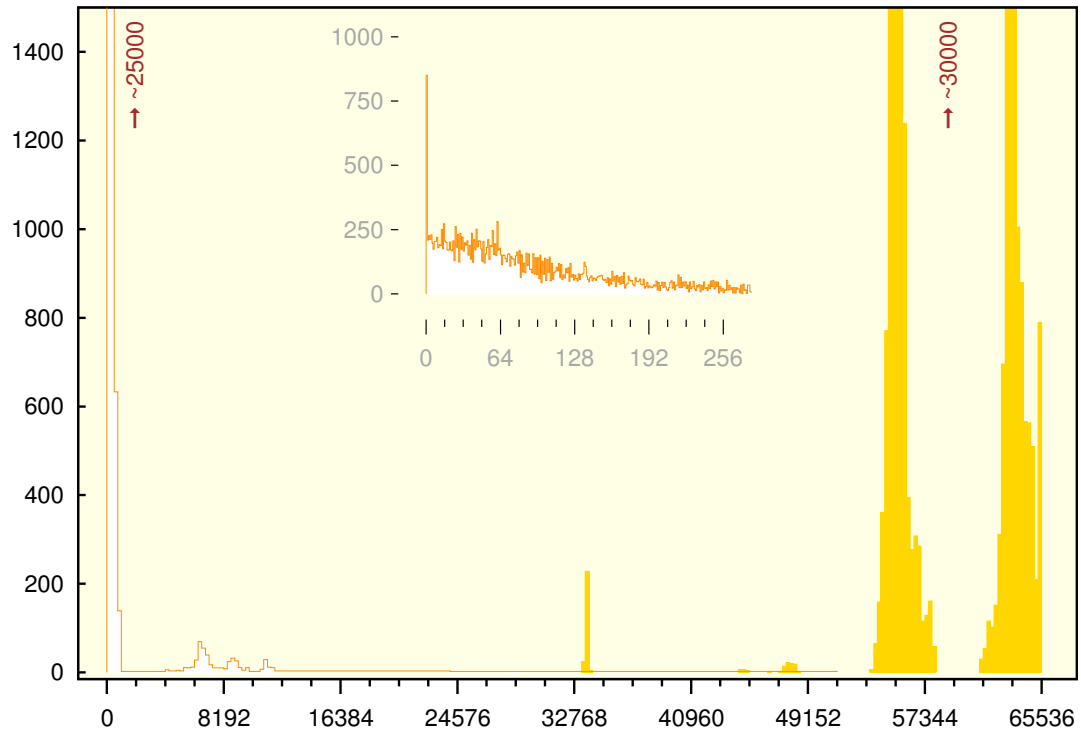


Figure 5.35: **Performance of the rampdiff scheme.** The yellow histogram shows the original bimodal distribution that has been converted to the white distribution by the *rampdiff* scheme. In the inset the inner 8 bits are resolved. This probability distribution must now be considered for entropy coding.

compares data rates (for both arrays) for three different reduction settings. The numbers provided can be compared with Figures 5.9 and 5.10.

file name	segment	data rate 16 [B/s (bit/s)]	data rate 8	8 with 2-bit rounding
`spec_seg01.dat`	①	14540 (116320)	24088 (192704)	15504 (124032)
`spec_seg02.dat`	②	12276 (98208)	20484 (163872)	12752 (102016)
`spec_seg03.dat`	③	10108 (80864)	17296 (138368)	10840 (86720)
`spec_seg04.dat`	④	7828 (62624)	14104 (112832)	9232 (73856)
`spec_seg05.dat`	❶	15012 (120096)	24188 (193504)	15588 (124704)
`spec_seg06.dat`	❷	12548 (100384)	20780 (166240)	13128 (105024)
`spec_seg07.dat`	❸	10332 (82656)	17592 (140736)	11252 (90016)
`spec_seg08.dat`	❹	8432 (67456)	15252 (122016)	10248 (81984)
`spec_seg09.dat`	⑤	14012 (112096)	23304 (186432)	14704 (117632)
`spec_seg10.dat`	⑥	12360 (98880)	20600 (164800)	12912 (103296)
`spec_seg11.dat`	⑦	10276 (82208)	17500 (140000)	10968 (87744)
`spec_seg12.dat`	⑧	7856 (62848)	14132 (113056)	9320 (74560)

Table 5.1: **Data rates using the rampdiff decorrelation.** For small capacitances (leading to large signals) the data rates are exceeded if only 8 samples are averaged. This is well compensated by additional 2-bit rounding.

Rounding in Spectroscopy

In photometry we use bit rounding to further reduce the data rate when a high gain is used. As in spectroscopy the same averaging algorithm is used to calculate the sub-means, the question is whether we can take advantage of its rounding capability or if that would degrade our data.

In case of a linear fit the precision of the slope is a function of the precision of the individual sample. According to [Hig91], for a linear regression of the form $y = a + b \cdot x$ the variance of the slope is

$$\mathrm{var}(b) = \frac{\sigma_y^2}{n \cdot \mathrm{var}(x)}$$

with σ_y as the standard deviation of the samples (the readout noise) and $\mathrm{var}(x)$ for the distribution of the sampling. For now we assume a discrete, errorless, uniform sampling and we use the uniform distribution where $\mathrm{var}(x) = \Delta^2/12$.

So if we want to know the precision of a slope fit of 64 samples we use $\mathrm{var}(x) = (64-1)^2/12$, considering the indexing of the samples with 1..64. This leads to:

$$\sigma_{\mathrm{slope}} = \sqrt{\frac{\sigma_{\mathrm{y}}^2}{64 \cdot (64-1)^2/12}} = 1/145.5\sqrt{\sigma_{\mathrm{y}}^2} = \frac{\sigma_{\mathrm{y}}}{145.5}$$

However, the slopes calculated in the SPU are not in float but in integer, multiplied with the number of samples that were fit, leading to a digitisation noise of $\sqrt{1/(12 \cdot (\text{subramp length}))}$, which is negligible given the fact that our slopes are anyway disturbed to a larger extent by nonlinearities.

Averaging 4 vs 8

What is better, 4 sub-means à 16 samples or 8 sub-means à 8 samples, and what if the 8 are used with rounding? The variance of the sub-mean is $\sigma_m^2 = \sigma_y^2/n + 1/12$ (1/12 due to the representation of the sub-mean by an integer). So the derived slope fit from the sub-means in comparison looks like:

$$\begin{aligned}
\sigma_{\mathrm{fit4}}^2 &= \frac{\sigma_y^2/16 + 1/12}{4(3)^2 12} \\
\sigma_{\mathrm{fit8}}^2 &= \frac{\sigma_y^2/8 + 1/12}{8(7)^2 12} \\
\sigma_{\mathrm{fit4}}^2/\sigma_{\mathrm{fit8}}^2 &\sim 98/18 \;\Rightarrow\; \sigma_{\mathrm{fit4}}/\sigma_{\mathrm{fit8}} \sim 2.3
\end{aligned}$$

If the 1/12 is negligible, we see that the precision of the fit from 8 sub-means is 2.3 times better than the precision of a fit from 4 sub-means. Now let us consider a 2-bit rounding ($\Delta = 2^{\mathrm{bits}} = 4$) for the 8 sub-means:

$$\sigma_{\mathrm{fit8}R2}^2 = \frac{\sigma_y^2/8 + 16/12}{8(7)^2 12}$$

We see that the quantisation term is no longer negligible and wonder what this means to 4 bit Gaussian readout noise ($\sigma = 13.2$ ADU). A fit made by fitting 8 sub-means that were derived by averaging 8 samples will yield the following precision: (Setting $\sigma = 13.2/\sqrt{8} = 4.7$, $\Delta = 4$ and using $\mathrm{var}(x) = (8-1)^2/12 = 4.1$)

$$\sigma_{\mathrm{slope}} = \sqrt{\frac{4.7^2}{8 \cdot 4.1}} = 0.67\mathrm{ADU}$$

The same with the digitisation noise that is introduced with 2-bit rounding:

$$\sigma_{\text{slopeR2}} = \sqrt{\frac{4.7^2 + 16/12}{8 \cdot 4.1}} = 0.85\text{ADU}$$

That is, by taking $b = 2$ bits of rounding in a case where the input noise is just 4 bits, the noise of the slope derived from 8 rounded sub-means is 27% higher than if no rounding would have been made. However, given the result from above that the sub-mean fit of 8 is 2.3 times better we are still very much in favour of 8 sub-means $(2.3 - 27\% = 1.68)$.[1]

What happens to the glitches?

No doubt that a higher temporal resolution helps in the detection and correction of glitches, but the question is, how much easier or harder is it to find a glitch (by only looking at the outlier, not considering the change in sensitivity affecting subsequent samples) if a sub-mean is either 16 samples or 8 samples long and if rounding is used. It is trivial, that a glitch increases the average by $1/n$ times its equivalent signal, whereas the standard deviation of the sub-mean is decreased by $1/\sqrt{n}$. Thus, if twice the samples are used, the precision is $\sigma/\sqrt{2}$, whereas the glitch (of size 1) is reduced to 1/2. We see that the more we average, the harder it is to find a glitch, with the glitch detection ability following a $\sqrt{n}$ trend. A 2-bit rounding affects the sub-mean in that it increases its noise ($\sigma_{mr}^2 = \sigma_m^2 + (2^{bits})^2/12$). Due to the triangle inequality the rounding follows a trend *less* than $\sqrt{\Delta^2}$. If we take the example from above, where $\sigma_{m8} = 4.7$ and apply a 2-bit rounding there we get $\sigma_{m8R2} = 4.84$, which is a 3% decrease of glitch detection capability (which goes linear with the standard deviation).

Conclusion

The *rampdiff* scheme is not ideal, essentially it has two shortcomings that could be improved. The bigger problem is of course that differentiation is used which increases the entropy. The other one is that the bias is not well subtracted. In the end all the overheads add up to 20–30 kbit/s. On the other hand the cases where the data rate is problematic with 8 sample sub-mean averaging is when the smallest capacitor is used and large variable slopes are in the observation, such as for a grating scan or a chopped observation. In such cases we end up around 180 kbit/s and we need anyway to decrease this by means of lossy reduction. Then the rounding option comes in handy, which allows us to get high temporal resolution with an acceptable noise increase.

Would I implement *rampdiff* in the same way again? Let me answer this by telling you that I had several candidate schemes for the FM decorrelation. As in photometry, other strategies were better for staring observations, others for chopped data, but the one which worked best for all the scenarios was *rampdiff*.

[1] When the spectrometer is operated with the smallest capacitor with a higher readout noise we only get a 10% increase.

4 Back-end Lossless Compression

The next step for both spectroscopy and photometry after decorrelation is entropy coding. The FM implementation is based on arithmetic compression, but with a few modifications to account for the little CPU resources and the few large data values that are caused by glitches. Due to the noisy nature of our data, compression algorithms that aim at recurring patterns are not useful, because we have hardly any repeating segments. The best way to deal with noise is by not trying to compress it but by efficiently encoding it to the entropy limit and for that we need an entropy coder, such as the ones described in Section 3.2.

Both reduction schemes produce a data stream of 3–6 bits noise around 0 with certain peaks that take the full 16 bit range. This had to be considered in designing the back-end encoding algorithm. I decided to implement arithmetic compression (see Chapter 3) to have an encoding whose efficiency depends on the data model, but is relatively stable when the model does not fit well. The limited CPU resources excluded a fully adaptive (i.e. that the whole probability table is updated after every encoding symbol) model. On the other hand, static models are not flexible enough for the variety of effects that impact the probabilities. Another driver in the design of the model was that the full 16 bit range had to be encoded, yet 90–95% of the encoding symbols are within the 8 bit region. The trick was to handle values 0–254 with a classical cumulative probability table and use the index 255 for values 255–65535, indicating that they are put aside for separate compression with a variable block word length encoder (also in Chapter 3). The data are processed in chunks of 8192 decorrelated samples and the *semi-adaptivity* of the model stems from the probability table, which is replaced after the encoding of a chunk by its statistics. That way there is no need to transmit the probability table as well, yet the model is sufficiently well related with the data. The encoding starts with a fixed initial model, which I derived by averaging a number of histograms from test data and edited by hand to represent the statistics of the first chunk as good as possible. Look at Figure 5.36 to see what it looks like.

Performance

In terms of CPU power, the lossless compression uses up to 50% of what is available. This is of course the biggest consumer of available resources, yet this is well justified: the more efficient the lossless compression works, the less drastic reduction has to be made. The fact that the model is not an accurate description of the data, but only an approximation leads to an encoding overhead of 5%. This is very acceptable given the resource savings that are achieved by this implementation.

5 On-board Implementation

What follows next is a short outline of the implementation of the reduction and compression software. On various pages I have given detailed information of the lossy and lossless steps that are performed on board. The short treatment hereafter shall explain how these algorithms, which are the core of the on-board processing, fit into the data flow and how they are operated. An in-depth treatment of the on-board reduction/compression software and related matters can be found in [Ott04a].

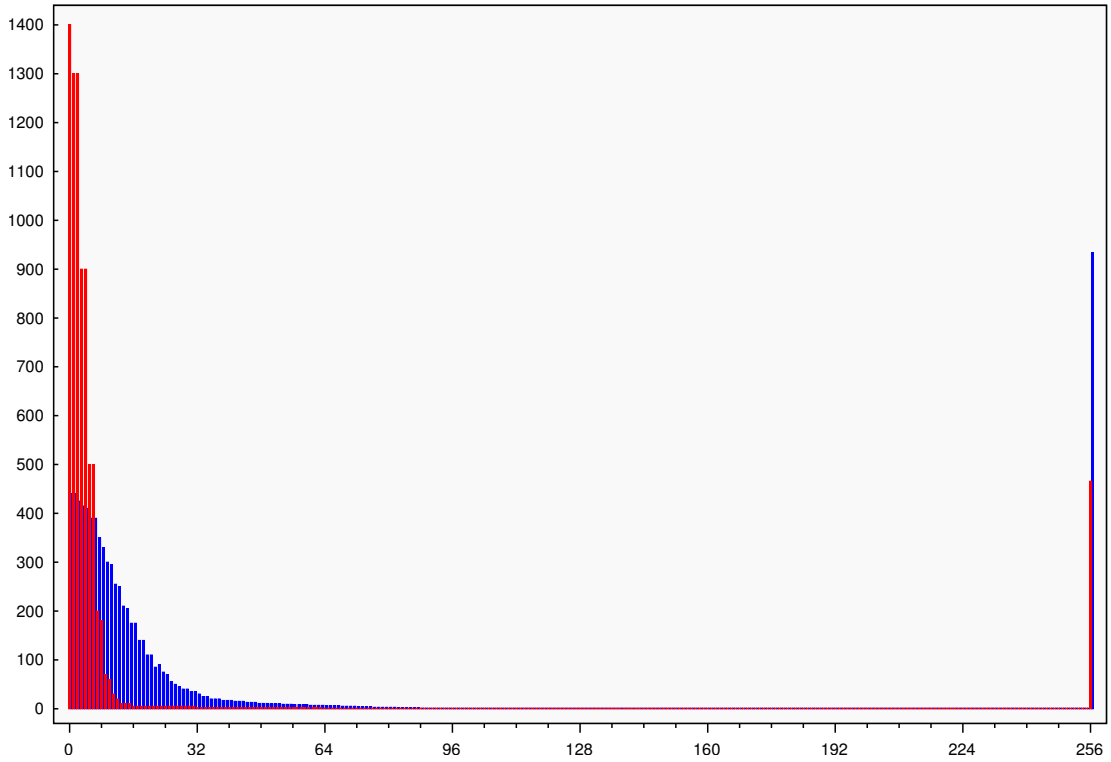


Figure 5.36: **Initial noise models.** The FM arithmetic compression starts with fixed probability models for the first 8192 decorrelated samples. The model in photometry (in blue colour) is a wider distribution and it also assumes that there will be almost 1000 outliers. In spectroscopy (red colour) a narrower distribution is used for the first chunk, which is – apart from the probability reserved for the outliers – similar to a 4-bit Gaussian noise.

Remember that PACS has four independent SPU computers, each one coming with startup software and low-level drivers. For all the data processing 4 MiB of data memory and 3 MiB of program memory are available. This is sufficient to buffer a few seconds of incoming data and still have space for processing. Due to the noisy character of the data there is not much redundancy to be expected within a few seconds and due to the drift in the detectors no longer periodicities can be exploited as well.

SW Infrastructure

To handle the task management – especially with all the simultaneously operating high speed communication links – the compression software runs under the Virtuoso™ real-time operating system from WindRiver (previously developed by EONIC). It is linked with the executable during compilation. The drivers to access the SMCS interface chip and other hardware are provided by the hardware manufacturer. The operating system supplies the necessary functionality for multitasking, interrupt handling, signalling, semaphores and FIFOs. A number of tasks are running in parallel, most notably the input and output data buffering tasks, the compression task which processes the data in chops of a few seconds, the housekeeping task which reports the software status every two seconds and the so-called watch process, which monitors the command link with respect to the high priority commands to start and stop processing. The command link is also used for initialisation of software relevant tables such as principal compression parameters or the detector selection tables used for masking bad pixels. It is also possible to upload a new software to the EEPROM via that link.

One SPU communicates via three links: raw data are permanently received from DMC, compressed data are sent to the DPU and commands can also be received any time from the DPU. The warm electronics units use a Scalable Multichannel Communications Subsystem (SMCS) 332 chip which implements the *Spacewire* standard. It is nominally configured to handle up to 10 Mibit/s on each link. To meet the real-time requirements it was necessary to implement the input and output buffers as circular buffers, that are continuously filled and drained by independent tasks.

SPU Software Structure

PACS operates either as a photometer or as a spectrometer, and each detector type comes with two independent detector arrays for the red and the blue channel. Normally, we concentrate on reaching the necessary compression factors yet retaining as much original data as possible. For some observations, temporal resolution is preferred over spatial resolution, for others the raw data of a selected sub-area are required. Other restrictions apply if PACS and SPIRE are operated in *parallel mode*, sharing the available bandwidth. There is also *burst mode*, which more than triples the available bandwidth if continuous operation is not scheduled. All these things have to be considered during on-board data processing leading to a variety of sequences and algorithms, commandable and/or adaptable to the data input and uploadable sets of parameters and tables to ensure a maximum of flexibility.

Each SPU has a full copy of the data processing/compression software, no matter if it is installed in the red or in the blue unit. A SPU is not configured to its channel, the distinction is made only by looking at the metadata in the frame headers. In general, once the SPU software is booted and has established connection with the other warm electronics units, it is waiting to receive a *start* command from the DPU. Once this has been acknowledged, the compression threads are signalled to be started and data from the input link are buffered. The data are identified by their headers and the software branches accordingly into the necessary processing steps.

Compression Modes

The full nominal sequence as illustrated in Figure 5.37 is not the only mode of operation. All processing steps are optional – it's even possible not to do anything at all with the data, though that case is most unfavourable, as the price to be paid for a few seconds of raw data from all arrays is a 1-minute time-out, because the data cannot be written fast enough to the spacecraft and need to be buffered in the warm electronics units. What the compression software is meant to do is determined by:

- the header parameters, indicating detector type, channel and compression mode;
- direct commands, used for starting and stopping the data flow, for updating software and tables and for switching to test and diagnostic modes;
- the on-board tables, controlling the bad pixel masks as well as various parameters for fine-tuning, such as choosing the lossless compression algorithm.

If all parameters are left at their initialisation values, the software operates in *default mode*, which has the minimum of lossy operations for transmitting the data of all pixels. This mode has been used during most of the instrument level tests and is also the standard in space. Default mode in photometry means averaging frames by 4 and lossless compression of the results. In spectroscopy we have either generation of

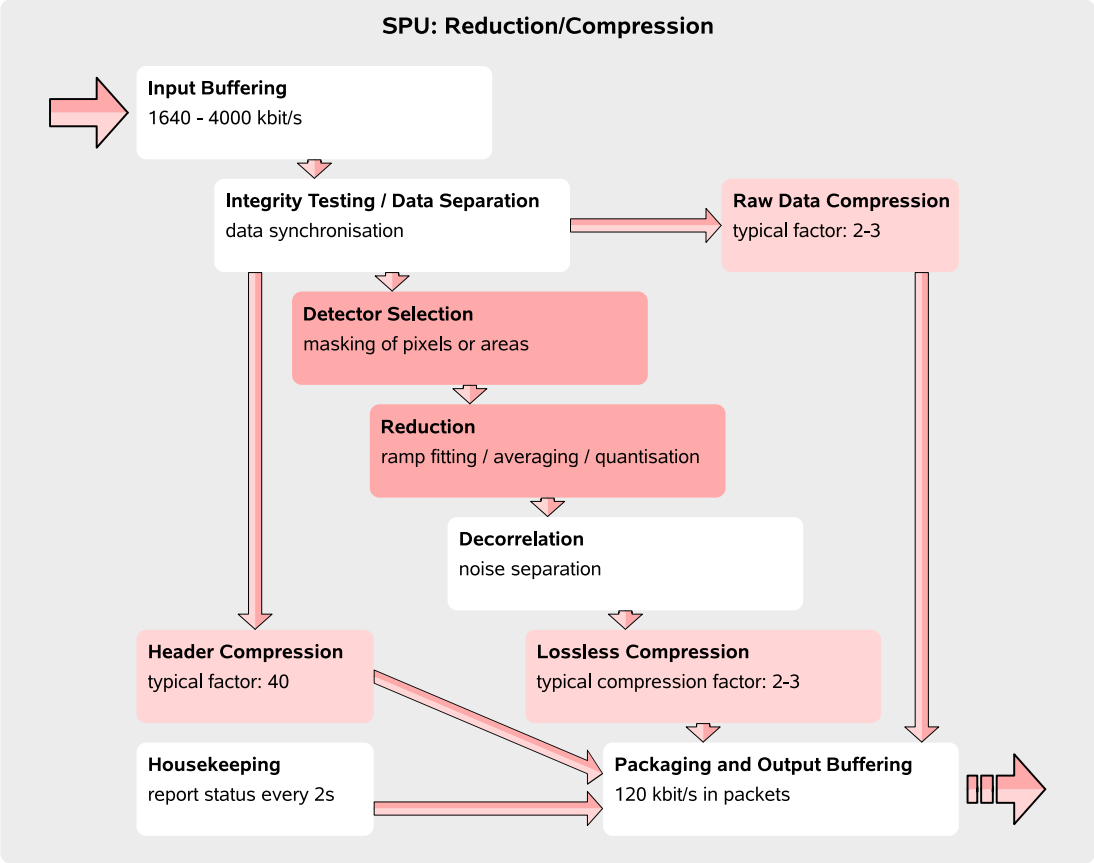


Figure 5.37: **Default reduction and compression steps.** Modules where raw data are irreversibly reduced are depicted with darker colour. From top to bottom: incoming data are tested for their integrity, the headers are separated from the data and passed to the header compression subroutine. The science data are filtered with the detector selection table mask and passed on to the reduction module. A few pixels are routed outside the normal data processing for inclusion of additional raw data in the downlink stream. The reduction module is where averaging or ramp fitting are performed, optionally including an additional rounding step. After that, the data are prepared in the decorrelation stage for the lossless compression. At the bottom of the scheme, the data streams that went different paths are reunited and sent to the DPU in packets ready for transmission.

sub-means of 16 (8) samples length or slope fitting using a length of 32 and lossless compression. However, if the arrays are driven in very noisy settings, it is necessary to include an optional rounding step. Figure 5.37 gives an overview of the modules involved in the standard data processing of the SPU.

Metadata and Header Compression

The origin and type of the data are reported in the frame header, as is the compression mode. This can resemble something like *photometry, red channel, default compression*. Some of the 16 header parameters are important for on-board data reduction, others – such as the position of the filter wheel – are needed on ground for pipeline processing and data product generation.

As the header parameters have the same structure for all observations, the header compression is the same for all of them. Although the headers have to be compressed

in a lossless way and in the worst case their data rate already exceeds the available downlink rate by a factor of 2, we manage to yield a typical factor of 40 owing to the structured content of the headers, which contains parameters that stay the same for several frames or counters which increase or decrease in a steady manner. Only a few parameters, such as the chopper position or the grating position, are affected by a 2-bit noise. The principal algorithm *rzip* used for header compression has been published [Ott04b, Ott04a] and is one of the few examples for a subroutine that has survived from the very beginning of the development until the flight model of the software without major modification needed.

The frame headers have another very important purpose. In photometry, the data frames are synchronised with the movement of the chopper via a flag in the header, signalling the start and stop of a sequence. A start of a sequence marks the first frame of a number of frames to be averaged. That way it is guaranteed that averaging does not occur during a chopper transition. In spectroscopy, the thing is a little bit more tricky. Two header parameters are used for synchronisation. One indicates the length of a ramp, the other one counts the sample index on the ramp. That way the compression software can synchronise onto the start of a ramp.

On-Board Tables

The on-board software has a compression parameter table for spectroscopy and one for photometry, plus detector selection tables for each detector array which are used to exclude any individual pixel. Most of the compression parameters deal with choosing different options for the decorrelation and lossless compression modules.

By default, all pixels from the arrays are transmitted,[1] but there are cases where it may not be needed to use the full array for observation. In addition to that, the detector will degrade over time and individual pixels will become defunct. So it is most reasonable to apply a pixel selection mask on the detector arrays. Depending on the number of selected pixels, the raw data could even be transmitted without reduction.

Detector selection is handled by the SPU with uploadable tables. These tables have a unique identification number that allows to reconstruct the array geometry on ground. Detector selection tables are stored within the ground segment software, where the tools for preparation and upload are available.

In the end, the data rate has to be met, no matter what parameters are tuned:

- (De)Selecting pixels. If raw data are to be transmitted, the compression factors of 16 (photometry) and 40 (spectroscopy) have to be achieved by selecting only that fraction of total pixels from the arrays.
- Changing the number of frames to be averaged: instead of the default number of 4, any other number can be commanded, such as 8 in the photometer parallel mode.
- In addition to averaging, the results obtained can be rounded by 1, 2 or even 3 bits to discard oversampled noise.

[1] The read-out electronics of the spectrometer generate 26 columns of 18 pixels although only 25 columns are equipped with Ge:Ga crystals. That 26th *ghost column* is always deselected by the pixel mask, as it contains no useful information.

- Decimation. As one of the last additions to the software before launch a configurable frame-dropping mechanism was added.

In addition to the nominal data stream, a few pixels' raw data can always be added to the downlink stream, mostly for verification purposes. Lossless compression of these so-called *additional raw data* typically yields a factor of 2 here.

Ground Software

What is compressed in space needs to be unpacked on ground again. This *decompression* framework is a part of the PACS Common Science System PCSS, which in turn is a part of the Herschel Common Science System HCSS [Wie04]. This huge framework (exceeding 100000 files) is written in Java™, which is very beneficial for collaboration of the many dozen of developers spread all over the globe. Right before launch it was renamed *Herschel Interactive Processing Environment* – HIPE. For de-compression backwards compatibility to old on-board software versions had to be assured. Although it is not anticipated to revert to any older version any more, the test and calibration measurements that were made with outdated on-board software still have to be readable.

Aside from the features of the operating FM software version – which has been used during the ILT phase with no undesired effects and proved its reliability in the flight phase up to now – a few more improvements are considered for preparation, such as a dedicated *compressed sensing* mode for photometry. If any unexpected instrumental effect or even damage gives rise to a bigger software update, this can be done due to the flexibility of the setup. Thanks to the compact format of the executable it is possible to upload a new software version to the SPU within a few minutes.

> Assez. Quand la science a prononcé, il n'y a plus qu'à se taire.
> —*Jules Verne, Voyage au centre de la terre, 1864*

6

New Challenges with SAFARI

In a project like PACS hundreds of people are involved and many of them for more than a decade. Our on-board software contribution is in itself a substantial project with some 30 man-years. Ten years ago we started from scratch, we were mostly unfamiliar with the procedures of space projects and had to learn the language of the business first. Among the lessons we learned is of course that the compression techniques we normally use in daily routine are not feasible for science data. Much of the experience that I have gained throughout the project is written down in this thesis so that it should be interesting material for everybody in this field. Almost a decade later I can tell what the relevant components in the design and implementation of a tailored real-time compression system for science data are and thus it is reasonable to further involve ourselves in upcoming missions. In this spirit we have joined the consortia of SPICA/SAFARI and PLATO, two missions that are candidates for selection as M-class missions in ESA's *Cosmic Vision* programme [Big05]. For these and other next-generation missions our experience in on-board processing is a valuable ingredient to make them *intelligent detectors*.

> Hello Roland
>
> world is small indeed, especially the astronomical one ...
>
> Ciao
> Stefano
>
> —*email reply received on the 25th of January, 2008*

This short chapter deals with the SAFARI instrument [Swi08] for the SPICA mission[1] [Nak08, Swi09], which can be seen as a successor to both Herschel/PACS and Herschel/SPIRE. For this mission, I have made an assessment of on-board processing for the *Phase-A1 Study Report* [Gri09a]. In the chapters on the PACS FM software I have described the last stage of development in a space project, here I show how a good start is made.

[1] The acronyms are short for SpicA FAR-infrared Instrument and Space Infrared Telescope for Cosmology and Astrophysics.

The chapter is divided into the following sections:

1 SAFARI for SPICA

Herschel was still on the ground, but we were already selling its technological heritage by planning the next big infrared observatory. The Japanese space agency JAXA plans a *Cryo-Herschel* mission for 2018 with major European contributions, such as the telescope and one of the prime instruments SAFARI.[1] With its actively cooled telescope SPICA will cover wider spectral ranges from the near to the far-infrared at a much higher sensitivity.

SAFARI is a spectroscopic camera for the 35–210 micron range. The UK, then Netherlands-led PI instrument is based on the technological heritage from both PACS and SPIRE. The instrument is an imaging Fourier transform spectrometer, its optical design has thus similarities with SPIRE. SAFARI's wavelength range is almost the same as for PACS, with similar detector technology and an extension to the short. The detector arrays are bigger and generate a much higher amount of data than PACS did. Although there is also a technology advance in telecommunication, it cannot compensate for the large amount of pixels. Well, we know the solution is to carry out data reduction and compression steps on board. So I carried out a study to shed light on this issue for the SPICA/SAFARI mission, giving an outline of possible measures to be taken, including the required resources for implementation on board.

Be aware that at such an early stage there is no way to give precise numbers concerning on-board compression because of the various uncertainties. First of all, the downlink rate is not contained in the instrument control specification, because a second ground station was discussed. A number to use as a guideline is 30 GB/day, which assumes a daily downlink rate of 10 Mbit/s for 8 hours. For lossless compression, signal and noise related issues are the main points of interest. At this time four different detector options are still being discussed and even the readout rate is not yet fixed. With all these obstacles my priority for the study was still to use representative data as input to derive plausible quantities. Once a set of simulated data was at hand, I knew that in light of the large raw data rate and the limited hardware resources, it would be central to find a very simple and fast decorrelation and encoding pair.

[1] In the beginning the instrument was called ESI (European SPICA Instrument), but we decided in the consortium to pay credit to the extra-European contributions by finding a geographically unrestrictive acronym.

2 On-Board Compression for SAFARI

SAFARI is planned as an imaging Mach-Zehnder [Mac92] Fourier transform spectrometer (FTS). This means that while the FTS mechanism scans over the optical path differences, the image of the source changes its intensity according to the phase shift. Four different detector technologies are competing for being selected for SAFARI. Whichever technology will succeed, three arrays for the different wavelength ranges will be needed for the FTS output channels. Spectral resolution, sensitivity and scan speed are given as scientific requirements. They determine the readout rate, which is at least 40 Hz, as the 20 Hz scanning resolution of the FTS mirror have to be Nyquist sampled. For the following figures I have used this number, even if a higher readout rate of 60 Hz is used in the final study.

Detector Characteristics

We have become familiar with two of the four detector options that are candidates for the instrument very well from PACS. These are the bolometers and the Ge:Ga photoconductors. In the previous chapters we have dealt with data from them and have a feeling for their signal and noise properties. Just note that the detectors for SAFARI need to be more sensitive by two magnitudes, so systematic and transient effects will be much more important.

Bolometers

The silicon bolometers that were used in PACS can be made more sensitive by further cooling. Their biggest problem though is their time constant – at present they seem to be too slow for the required scan speed. Bolometers are susceptible to transient effects, such as cosmic rays and solar protons. Their read-out circuits comprise a number of amplification stages which are by themselves noise contributors. As their fabrication is difficult, the pixels vary in sensitivity with little correlation.

TES

Transition Edge Sensors (TES) are bolometers with superconducting thermometers [Mau08]. In contrast to semiconductor devices, saturation in a TES means no output signal, rather than a nonlinear but finite response. The pixels are read and multiplexed using SQUID (superconducting quantum interface device) amplifiers. As these detectors are very sensitive, they are also quickly saturated.

KID

Kinetic Inductance Detectors (KID) are fundamentally different to the other detector options [Bas08]. When absorbed in the superconductor, the incident radiation will break Cooper pairs, which result in an excess number of quasi-particles above the thermal ones. They are read by feeding a high frequency into the channel, and the resulting resonances are packed closely together in frequency space. The phase change and/or amplitude change for each resonance feature will be read by a digital readout circuit. One of the main challenges of the KIDs is that readout process.

Photoconductors

The photoconductor (PC) option is completely different, as in this case SAFARI will not be a FTS, but use a grating just as in PACS to derive the spectrum. It would require a significantly longer measurement to derive a spectrum with the same resolution and sensitivity as with the FTS. The photoconductor option combines two

Ge:Ga arrays with a BIB (Blocked Impurity Band) array. The Ge:Ga detectors are read non-destructively at a much faster rate to measure accurately the accumulation on the integrating capacitor. In Chapter 5 we have seen what this means, we know that they saturate quickly and learned about nonlinearities and memory effects. This option generates the highest data rate and some sort of on-board data reduction like in PACS is then mandatory.

Processing Scenarios

Available Data Rate

According to present planning SPICA will have a daily telecommunication period (DTCP) of 8 hours, with a telemetry budget of ∼30 GB/day. Various uncertainties, such as the precise downlink rate, the telemetry portion of housekeeping (status values sent independently of the science data) and other non-scientific data – or whether there will be observations during downlink or not – lead to only a rough estimate for the average sustainable data rate. Even the metrology[1] will have a significant share as well. To derive a number we can work with we use 30 (GB/day) / 86400 (s/day) = 2.78 Mbit/s.

Instrument Raw Data Rate

In the instrument definition document [Gri09b] the array dimensions, their readout rates and data types are specified. Three of the four proposed detector technologies (TES, KID and bolometers) have similar dimensions, which are 64×64, 38×38 and 20×20 pixels. The table below gives the data rates for different ADC bit depths. Note that in the definition only 12 bit analogue/digital conversion is discussed, but in practice a 16 bit data type is likely to be taken to have better control over gain and bias – such as in the case of PACS.

			sample rate	data rate @40 Hz		
μm	array pixels	samples	@40Hz	12 bit ADC	14 bit ADC	16 bit ADC
35–60	64×64	4096	163840	1.97 Mbit/s	2.29 Mbit/s	2.62 Mbit/s
60–110	38×38	1444	57760	0.69 Mbit/s	0.81 Mbit/s	0.92 Mbit/s
110–210	20×20	400	16000	0.19 Mbit/s	0.22 Mbit/s	0.25 Mbit/s
		5940	237600	2.85 Mbit/s	3.33 Mbit/s	3.80 Mbit/s

Table 6.1: **Baseline data rates.** The three distinctive ADC scenarios result in different data rates.

In case of the KIDs, it is possible to transmit twice the data, because the signal can be derived by the amplitude or by the phase shift of the resonance feature. In case this kind of redundancy is required, the data rate doubles.

PC data rates

The fourth detector option is to use a grating and Ge:Ga photoconductors, which need to be read in a non-destructive way at a much higher rate. This also increases the data rate as shown in the table beneath. Note that for the short wavelengths, a BIB array is used.

[1] Timing and positional parameters needed to interpret the detector data. If these are attached to the science frames we would call them *headers*.

PC μm	array pixels	samples	sample rate @256Hz	data rate @256 Hz 12 bit ADC	14 bit ADC	16 bit ADC
42–110	32×32	1024	262144	3.15 Mbit/s	3.67 Mbit/s	4.19 Mbit/s
110–210	32×32	1024	262144	3.15 Mbit/s	3.67 Mbit/s	4.19 Mbit/s
		2048	524288	6.29 Mbit/s	7.34 Mbit/s	8.39 Mbit/s
BIB μm			@10Hz	@10 Hz, 12 bit		@40 Hz, 16 bit
35–40	256×256	65536	655360	7.86 Mbit/s		41.94 Mbit/s

Table 6.2: **Data rates for the Photoconductors.** The grating with photoconductors option has considerably higher data rates due to the higher readout rate. In here, the numbers for the Ge:Ga arrays and for the BIB array are given separately.

Obviously, the amount of raw data generated by the photoconductors in combination with the BIB array is much higher than for the other detector technologies and reduction steps have to be included as the typical lossless compression factor of 2 (which is found in PACS for this detector type) is not sufficient.

Imaging

SAFARI shall also have the possibility of direct imaging. In this case, the main difference is that the temporal signal of a pixel is not an *interferogram* (IFGM), but a direct measure of brightness, such as in case of PACS photometry.

Need for On-Board Processing

Given the numbers in the tables above – which can at this point only be rough estimates – and the fact that the latest trend is even towards a higher readout rate of 60 Hz, a form of data preprocessing with successive entropy encoding that yields a compression factor of 2 is mandatory for proper instrument operation. We know that the two detector types (photoconductors and bolometers) that are used in PACS generate data with very little mutual information at a low signal to noise ratio. The TES and the KIDs have signals that are presumably similar to the bolometers and I also expect very little spatial correlation with these detector options.[1]

For these types of detectors I have presented lossy and lossless reduction and compression steps in Chapter 4 and 5. However, special attention must be paid to the fact that we have a FTS here and the data are samples in Fourier space. The effect of nonlinearities and memory effects must be studied well before lossy steps such as quantisation or decimation can be used.

Available Resources

The design of the warm electronics foresees an instrument control unit (ICU) which is responsible for on-board data handling. Its current design is based on a Leon3[2] SoC (System on Chip), which can be implemented as a FPGA or as an ASIC. In the FPGA implementation it has 20 MIPS, which is comparable with the DSP used in Herschel/PACS. Given the fact that the ICU also has the task of commanding and is the interface to the spacecraft, it cannot dedicate all resources to compression. In case floating-point operations need to be executed, an FPU must be included in the SoC design, but for now I'll show how to achieve a lossless factor of 2 with that little processing power for SAFARI in integer arithmetics. As a basis for this it is crucial to have a solid number for the entropy of the data to encode.

[1] At the moment only few pixels have been fabricated and I have not received real test data yet.
[2] Leon is a high performance 32-bit SPARC V8 processor platform for space applications.

Simulation of Representative Data

In general, the achievable limit of lossless data compression is the entropy of the decorrelated data. If the compression factor gained by such techniques is not sufficient, then lossy reduction steps have to be included in the data processing.

To get as representative test data for SAFARI as possible, I have combined an ISO/LWS spectrum with real data from Herschel/PACS and design parameters from SAFARI. For parameters that were uncertain I chose values that would lead to data that are harder to compress. So an array of 64×64 pixels was simulated, each pixel was given a modified ISO/LWS spectrum that would agree with SAFARI parameters. The data cube was multiplied with a galactic input sky map and photon noise was added. For the FTS we can consider the total intensity at a given optical path difference (OPD) x [Gri83]:

$$I(x) = \int_{-\infty}^{\infty} D(k)I(k)\cos(2\pi kx)dk$$

with k being the wavenumber $1/\lambda$ and $D(k)$ is a window function being 1 within the bandwidth and 0 outside. The scan of the FTS was simulated according to SAFARI design parameters (scan length and speed). Intensity is lost at larger OPDs due to imperfections in the alignment of the optical elements and their surfaces, leading to a drift of the absolute signal during the scan, so an artificial intensity drift was applied to achieve the typical look of Herschel/SPIRE interferograms. In the end, detector characteristics, such as *bias*, *flatfield*, the *relative spectral responsivity function* (RSRF), *readout noise*, *dynamic range*, and a *sampling rate* of 40 Hz were used to modify the IFGMs.

The result is a dataset with 8240 frames for each of the 64×64 pixels, where each pixel sees the same original input spectrum. This is not a problem, as we will concentrate on a single pixel now. Figure 6.1 shows such a typical IFGM.

Decorrelation

My experience with the detector technologies used for the far-infrared is that it is not useful to go for a 2D decorrelation of the image frame, because neighbouring pixels have too little correlation that could be exploited. Instead, the best strategy is to concentrate on the 1D signals of the individual pixels – the IFGMs.

The fastest reversible operation that will bring the IFGM down to noise around zero is a differentiation. This has of course the negative side-effect of increasing the uncorrelated readout noise by $\sqrt{2}$, i.e. half a bit per sample. The next best decorrelation would be a running average, but this one suffers from the oscillations of the signal component. LPC is too CPU-intense to be considered for SAFARI unless a dedicated coprocessor is used, as are transform-based decorrelations such as DCT and the popular CDF 9/7 wavelet. In any case, these sophisticated techniques can save at maximum half a bit per sample with respect to differentiation. Another reason against sophisticated techniques is that the buffer sizes of the on-board electronics will only be able to store a few seconds of data, i.e. only short fractions of a full IFGM which takes more than 200 seconds according to the current instrument design. Figure 6.2 shows how a differentiated IFGM looks like. This works sufficiently well as a starting point for entropy encoding. To get a better impression of the statistics, I draw the histogram of the decorrelated

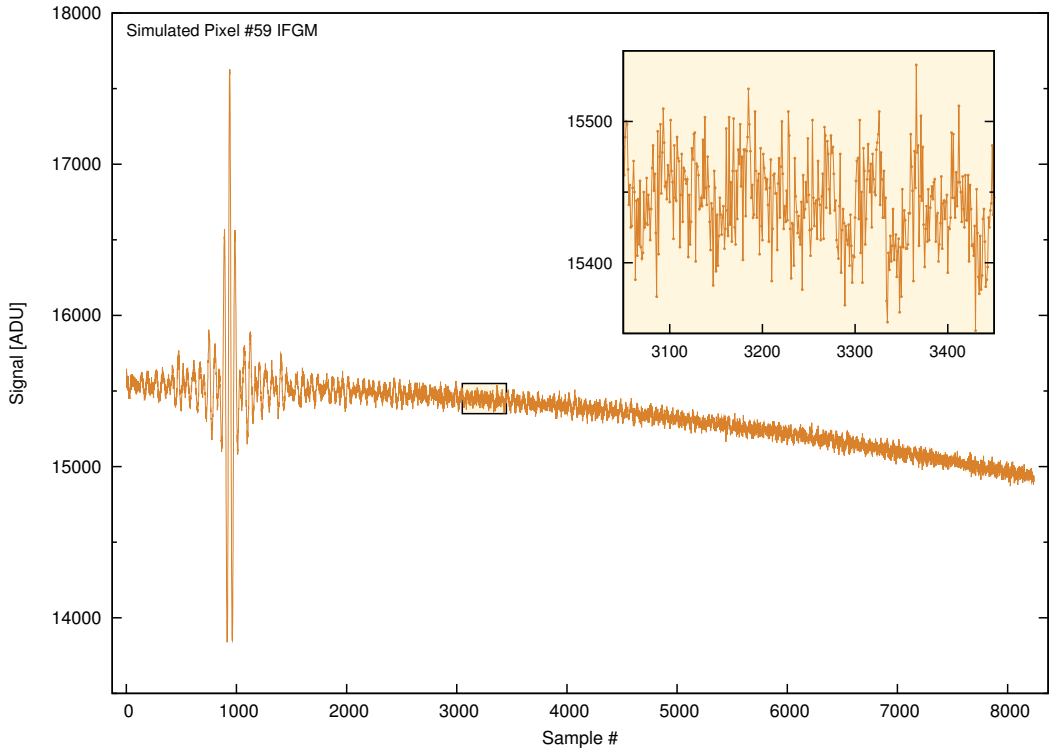


Figure 6.1: **Typical pixel IFGM.** (Left) An example IFGM from a pixel with H=7.32 bits/sample.

example pixel in Figure 6.3. For an overview of the entire decorrelated dataset another histogram is also given, showing the entropies that make up the dataset.

The average entropy for the simulated dataset is 7.32 bits/sample. With this number we can update the table of data rates for the interferometer options.

μm	array pixels	samples	sample rate @40Hz	data rate @40 Hz, H=7.32 bits/sample
35–60	64×64	4096	163840	1.20 Mbit/s
60–110	38×38	1444	57760	0.42 Mbit/s
110–210	20×20	400	16000	0.12 Mbit/s
		5940	237600	1.74 Mbit/s

Table 6.3: **Compressed data rates.** Here are estimated data rates using differentiation and entropy coding. Note that with more sophisticated decorrelations this can be brought down to 1.62 Mbit/s.

Encoding

The actual entropy coding is another design driver for the warm electronics of the instrument, at least, whether a dedicated coprocessor is required or not. Ideally, we would use arithmetic compression, which lets us encode the data down to the entropy limit.

Arithmetic Compression

To stay within the Leon3 resources, we would propose a static encoding model derived from the histograms of differentiated IFGMs. A fully adaptive model for the 5 bit range [-16:15] requires on average 16 updates of the cumulative frequency table

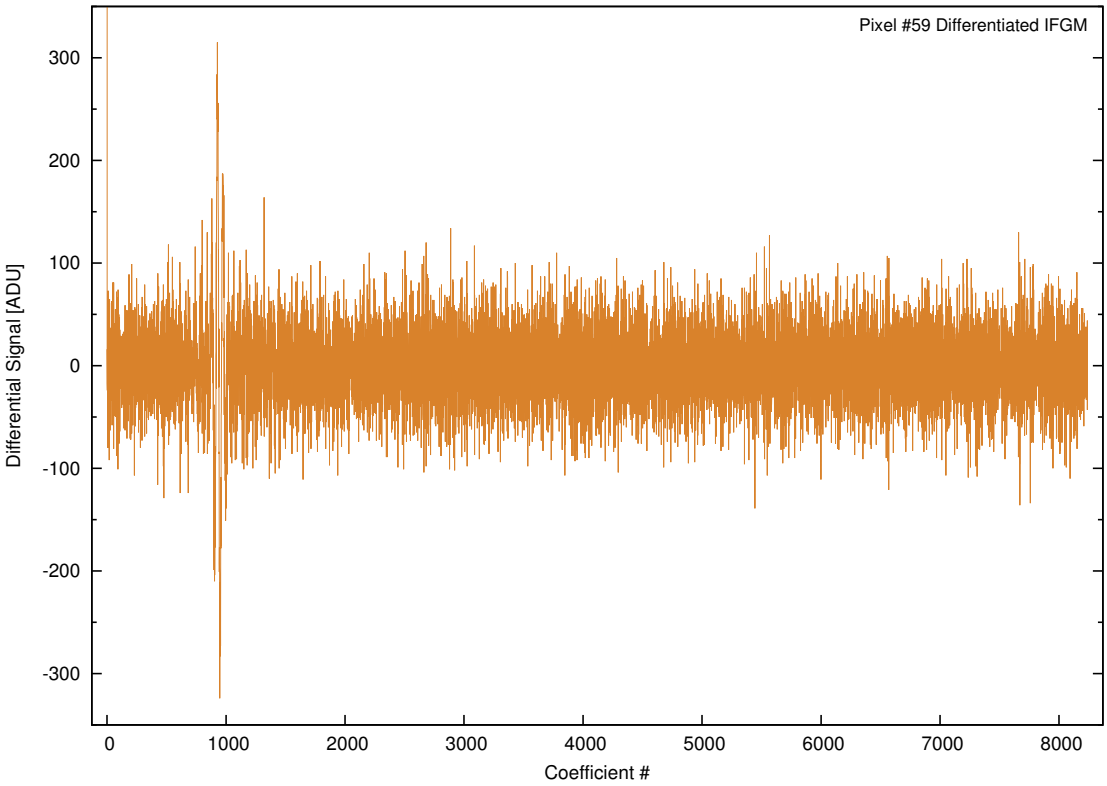


Figure 6.2: **Decorrelated IFGM.** The example IFGM has been differentiated. That way the intensity drift and the oversampled features of the IFGM are taken out and what remains is mainly the read-out noise.

per sample, leading to ∼4 MIPS. A static model requires only a look-up table and thus stays far below 1 MIPS. The actual arithmetic encoding is independent of the used model. It takes about 10 operations per sample without writing the output bits into the compressed bit stream. This in turn takes 20 operations per sample. So, encoding needs ∼30 operations/sample, leading to ∼7 MIPS. All in all, we end up around 10 MIPS for arithmetic compression, which is already 50% of the Leon3 resources. But still, as the entropy of 7.32 bit/s is considerably larger than 5 bits, the encoding would not be very efficient. An increased adaptive model would already require ∼100 updates per sample and exceed the available CPU resources.

Fixed-Size Codes

The fastest way of entropy encoding is to use a look up table of fixed codewords. For this purpose we could use fixed Huffman or Golomb codes. They come close to the entropy, but a penalty of another half a bit must be foreseen and we end up with 1.86 Mbit/s for our simulated dataset. The advantage is that the whole encoding procedure consists of a look up in a table and writing the codeword into the bit stream. As mentioned before, this encoding takes about 20 operations/codeword if no dedicated CPU instructions are available. The disadvantage is that the used fixed code can become inefficient if the properties of the noise change.

If I have to give some sort of *least demanding still meaningful encoding* I can think of a code which does not need the bit stream encoding, but collects whole bytes. Then every decorrelated value between −127 and 127 would be directly coded as a byte and −128 would be used to escape a larger 16-bit value. For the simulated dataset this leads

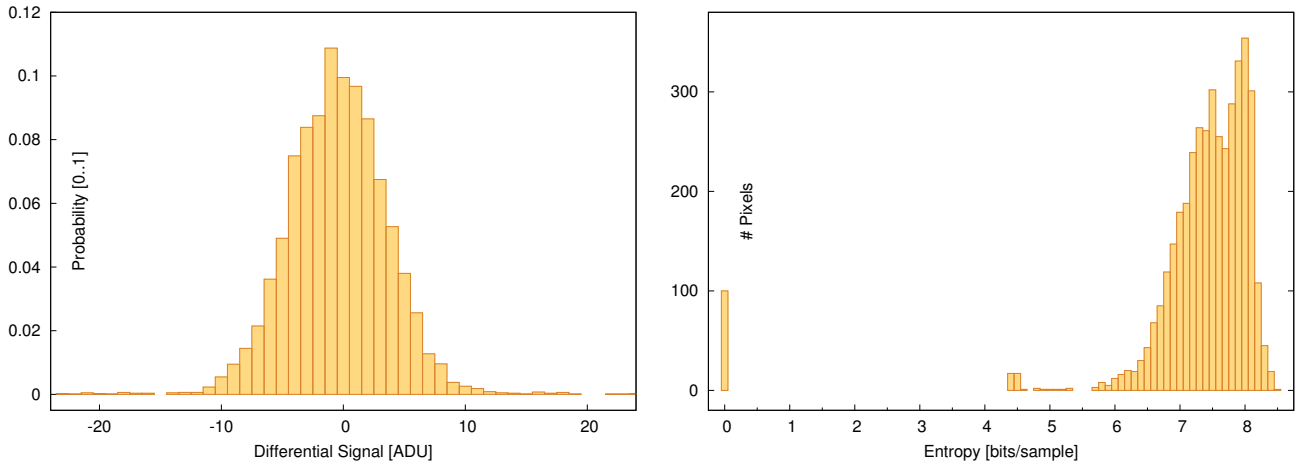


Figure 6.3: **Histograms of decorrelated IFGMs.** (Left) The histogram of the decorrelated example pixel has a distribution around zero. It is slightly over-balanced on the left side due to the downward intensity drift of the IFGM. (Right) The entropies of all 4096 simulated pixels, shown as a histogram. Apart from a few (∼1%) dead pixels with H_S=0 most pixels have an entropy around 7–8 bits/sample.

to 1.93 Mbit/s. This however gets quickly inefficient as soon as the average entropy is above 8 bit/sample as in the case of glitches, but the CPU requirements would be around 2 MIPS.

Reduction for the PC and Optional Steps

The *grating with photoconductors* option requires lossy reduction steps of factors around – let's say 8. This strongly depends on the readout frequency, especially for the BIB array with its many pixels. A simple reduction like averaging of 8 frames for the two Ge:Ga arrays would require 10 operations for 8 samples and consume only 0.6 MIPS, followed by decorrelation and encoding similar to the IFGM data. As test data for this option we used input from the PACS spectrometer. Slope fitting, where the slope $s = (\sum x \sum y - n \sum xy)/(\sum x \sum x - n \sum xx)$ requires ∼200 operations for a 64 sample ramp and may require floating point addition and multiplication. In any case, less than 1 MIPS for the two Ge:Ga arrays. To further speed this up, one can also use the faster $\sum xy - (n+1)/2 \sum y$ as a measure for the slope, which works in integer.

Dealing with the BIB array, for example through binning by the OBSW is equally possible if the sampling rate stays below 10 Hz. But even at this rate a similar lossy reduction factor as for the Ge:Ga arrays must be accomplished, such as by 2×4 binning or averaging of 8 frames. This is still possible with the resources of the Leon3, but for a sampling rate of 10 Hz the resources get tight.

Optional Lossy Reduction Steps

One simple strategy for lossy data reduction is to discard data from pixels that are not needed or whole frames that have no use on ground. Pixel masking can be done by on-board tables that would be uploaded and used by dedicated control procedures. Another strategy is to reduce the entropy by quantisation. If combined with transform coding, this will degrade the linearity of the data. If applied directly to the measures, additional digitisation noise is added.

Implementation

Here are some early considerations of what will play a role for the actual implementation. These are relevant for the design of the actual ICU hardware.

- Decorrelation. Differencing can be done in place, but the values need to be mapped from signed to unsigned data type. This works in place as well and the combined processing resources are about 5 operations per sample, or roughly 1 MIPS.
- Encoding. Glitches and other outliers exceeding the intended range (5 bit) of the entropy coder shall be encoded with an escape character followed by a fixed-size codeword depending on the valid range of values, e.g. 16 bits. Since SPICA will operate in a similar environment as Herschel, we can make an estimate based on the glitch rates, which are about 1 event in 5–10 seconds per pixel. In a differentiation/encoding scheme this increases the data rate by only a few percent (2.2% for the one described above). As SAFARI will have more sensitivity, much more *weak* glitches are likely to be seen, but the overall increase should be still in the order of 10%.
- Buffer size. In order to operate on the data, each sample must be stored in a 32 bit data word. This leads to a typical buffer size of ∼1 MB/s (950400 bytes). A positive feature of the differentiation is that it can be done in place, thus no work buffer is required. Also, arithmetic compression can be made memoryless depending on the type of prediction model used.
- Codeword encoding. The encoding options discussed above need some form of bit stream encoding. The advantage is, that the bit stream needs not be seekable and can be implemented in the form of a *cache*. So, essentially the task is to *attach these n bits to the already cached bits*, where n is a number from 1 to 32. Typical assembler instructions needed to do so are *shift*, *add*, *subtract*, *and*, *or*.

Here is an implementation of a bit stream encoding function in C. For PACS, I rewrote this function in assembler to save CPU power, as it is heavily used there.

Listing 6.1: PutNBits

```
# define u4 unsigned int

/* needs an initialised array of bit masks, create it once with: */
for(i=0; i < 32; i++) { WordMaskRight[i] = 0xfffffffe << (31-i); }

void putnbits (u4 value, u4 bitOffset, u4 nbits, u4 *dest)
{
  u4 A, A1, A2, A3, offs = bitOffset >> 5, *dest1;
  dest1 = dest + offs;

  /* end of the bit string with local offset, overflowed bits: A-32*/
  A1 = (bitOffset & 0x1f);
  A = A1 + nbits;

  if (A <= 32) // the pattern fits into one word
  {
    *(dest1) &= WordMaskRight[A1];  /* clear destination */
    *(dest1) |= (value<<(32-A));    /* assign value */
  }
```

```
21    else // if the pattern to write falls on two words
22    {
23      A2 = 32 - A1;
24      A3 = A - 32;
25      putnbits (value >> A3, bitOffset, A2, dest); /* first part*/
26      putnbits (value, bitOffset+A2, A3, dest);    /* second part */
27    }
28    return;
29  }
```

Recommendation for SAFARI

A lossless compression factor of 2 can be realised for the interferometer options of SAFARI without the need for extra hardware, although a Leon3 is typically 50% busy with this task, where most of this comes from the bit stream encoding. This factor is achievable with simple techniques. The implementation would have to be in assembler, otherwise the risk of exceeding the CPU resources is too high. The memory requirements are low, compression can even be carried out with memoryless algorithms, but for real-time buffering I suggest to work on data chunks of a few seconds, leading to a required DRAM of 8–16 MiB.

For the photoconductors option lossy reduction steps have to be considered. Depending on the readout rate of the BIB array, a coprocessor may be required. Also, for optional on-board deglitching a dedicated chip would be needed. The CPU and memory requirements of this scenario are 2–4 times higher than for the IFGM options and floating point operations would be favourable for addition and multiplication. There is also a need to synchronise the reduction steps with the data (the start of a ramp).

Further Involvement

SPICA/SAFARI is a challenging new project in many ways – also when it comes to on-board compression. We can base our contribution on a decade of software development for PACS and thereby provide a wealth of experience. On the other hand working on a FTS is a chance to acquire new skills, because any lossy steps would have to be studied extensively with respect to their effects on the Fourier reconstruction. If SAFARI is selected to be built and our institute stays involved I will gladly accept the challenge, even if this means 10 more years in the business ☺.

> Hi Roland,
>
> I just am editing the information from your note into the instrument study. Well done, it is a good report.
>
> Thanks
>
> Doug
>
> *—email received on the 19th of August, 2009*

Bibliography

[Ans48] F. J. Anscombe. The Transformation of Poisson, Binomial and Negative-Binomial Data. *Biometrika*, 35:246–254, 1948.

[Bas08] J. Baselmans, S. J. C. Yates, R. Barends, Y. J. Y. Lankwarden, J. R. Gao, H. Hoevers, and T. M. Klapwijk. Noise and Sensitivity of Aluminum Kinetic Inductance Detectors for Sub-mm Astronomy. *Journal of Low Temperature Physics*, 151:524–529, April 2008.

[Bel04] A. N. Belbachir. *On-board Processing for an Infrared Observatory*. PhD thesis, Technische Universität Wien, 2004.

[Bel05] A. N. Belbachir, H. Bischof, R. Ottensamer, F. Kerschbaum, and C. Reimers. On-board data processing to lower bandwidth requirements on an infrared astronomy satellite: case of Herschel-PACS camera. *EURASIP J. Appl. Signal Process.*, 2005:2585–2594, 2005.

[Ben86] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei. A Locally Adaptive Data Compression Scheme. *Communications of the ACM*, 29(4):320–330, April 1986.

[Ber06] E. Bertin. *SExtractor v2.5 User's Manual*. Institut d'Astrophysique & Observatoire de Paris, 2006.

[Big05] G. Bignami, P. Cargill, B. Schutz, and C. Turon. Cosmic Vision. ESA Publications Division, ESTEC, October 2005. BR-247.

[Bil06] N. Billot, P. Agnèse, J.-L. Auguères, A. Béguin, A. Bouère, O. Boulade, C. Cara, C. Cloué, E. Doumayrou, L. Duband, B. Horeau, I. le Mer, J. Lepennec, J. Martignac, K. Okumura, V. Revéret, M. Sauvage, F. Simoens, and L. Vigroux. The Herschel/PACS 2560 bolometers imaging camera. In *Space Telescopes and Instrumentation I: Optical, Infrared, and Millimeter. Edited by Mather, John C.; MacEwen, Howard A.; de Graauw, Mattheus W. M.*, volume 6265 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, July 2006.

[Bis00] H. Bischof, A. N. Belbachir, D. Hönigmann, and F. Kerschbaum. Data reduction concept for FIRST/PACS. In J. B. Breckinridge and P. Jakobsen, editors, *Proc. SPIE Vol. 4013, p. 244-252, UV, Optical, and IR Space Telescopes and Instruments*, volume 4013 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 244–252, July 2000.

[Bob08] J. Bobin, J.-L. Starck, and R. Ottensamer. Compressed Sensing in Astronomy. *IEEE Journal on Selected Topics in Signal Processing: Signal Processing for Astronomical and Space Research Applications*, October 2008.

[Box58] G. E. P. Box and E. Muller. A Note on the Generation of Random Normal Deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[Bra99a] R. Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill, third edition, 1999.

[Bra99b] K. Brandenburg. MP3 and AAC explained. In *High quality audio coding*, Audio Engineering Society International conference No 17, pages 99–110, 1999.

[Bur94] M. Burrows and D. J. Wheeler. A Block-sorting Lossless Data Compression Algorithm. Technical Report Research Report 124, Digital Systems Research Center, May 1994.

[CCS97] Consultative Committee for Space Data Systems. Lossless Data Compression. Blue Book 121.0-B-1, CCSDS Secretariat, NASA, 1997. Issue 1, Cor. 2, 2007.

[CCS05] Consultative Committee for Space Data Systems. Image Data Compression. Green Book 122.0-B-1, CCSDS Secretariat, NASA, 2005. Issue 1, Cor. 2, 2008.

[CCS06] Consultative Committee for Space Data Systems. Lossless Data Compression. Green Book 120.0-G-2, CCSDS Secretariat, NASA, 2006.

[CCS07] Consultative Committee for Space Data Systems. Image Data Compression. Green Book 120.1-G-1, CCSDS Secretariat, NASA, 2007. Issue 1, July 2007.

[Cle84] J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[Coo65] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, April 1965.

[Cov06] T. M. Cover and A. T. Joy. *Elements of Information Theory*. John Wiley & Sons, second edition, 2006.

[Dau92] I. Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992.

[Dev86] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.

[Eck99] S. Eckardt. *Wirkung Kosmischer Strahlung auf Infrarot-Detektoren in Astronomie-Satelliten und Charakterisierung einer Kamera für den Satelliten FIRST*. PhD thesis, Ruprecht-Karls Universität Heidelberg, 1999.

[Eva03] H. Evangelaras, C. Koukouvinos, and J. Seberry. Applications of Hadamard matrices. *Journal of Telecommunications and Information Technology*, February 2003.

[Gam04] M. N. Gamito and M. Salles Dias. Lossless Coding of Floating Point Data with JPEG 2000 Part 10. In A.G. Tescher, editor, *Proceedings of the SPIE. Applications of Digital Image Processing XXVII*, volume 5558, pages 276–287, 2004.

[Ghi04] F. Ghido. An Efficient Algorithm for Lossless Compression of IEEE Float Audio. In *Proceedings of the 2004 IEEE Data Compression Conference*, volume 1068-0314, 2004.

[Gol66] S. W. Golomb. Run-Length Encodings. *IEEE Transactions on Information Theory*, IT-12(3):399–401, 1966.

[Gra09] T. de Graauw, N. Whyborn, E. Caux, T. Phillips, J. Stutzki, A. Tielens, R. Güsten, F. Helmich, W. Luinge, J. Martin-Pintado, J. Pearson, P. Planesas, P. Roelfsema, P. Saraceno, R. Schieder, K. Wildeman, and K. Wafelbakker. The Herschel-Heterodyne Instrument for the Far-Infrared (HIFI). In L. Pagani & M. Gerin, editor, *Astronomy in the Submillimeter and Far Infrared Domains with the Herschel Space Observatory*, volume 34 of *EAS Publications Series*, pages 3–20, 2009.

[Gri83] P. R. Griffiths. Fourier transform infrared spectrometry. *Science*, 222:297–302, 1983.

[Gri08] M. Griffin, B. Swinyard, L. Vigroux, A. Abergel, P. Ade, P. André, J.-P. Baluteau, J. Bock, A. Franceschini, W. Gear, J. Glenn, M. Huang, D. Griffin, K. King, E. Lellouch, D. Naylor, S. Oliver, G. Olofsson, I. Perez-Fournon, M. Page, M. Rowan-Robinson, P. Saraceno, E. Sawyer, G. Wright, A. Zavagno, A. Abreu, G. Bendo, A. Dowell, D. Dowell, M. Ferlet, T. Fulton, P. Hargrave, G. Laurent, S. Leeks, T. Lim, N. Lu, H. Nguyen, A. Pearce, E. Polehampton, D. Rizzo, B. Schulz, S. Sidher, D. Smith, L. Spencer, I. Valtchanov, A. Woodcraft, K. Xu, and L. Zhang. Herschel-SPIRE: design, ground test results, and predicted performance. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7010 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, August 2008.

[Gri09a] D. Griffin. SAFARI Phase-A1 Study Report. Technical Report SAF-RAL-REP-0038, RAL, 2009.

[Gri09b] D. Griffin. SPICA/SAFARI Instrument Definition Document. Technical Report SAF-RAL-REP-0024, RAL, 2009.

[Haa10] A. Haar. Zur Theorie der orthogonalen Funktionen-Systeme. *Math. Ann.*, 69:331–371, 1910.

[Had93] J. Hadamard. Résolution d'une question relative aux déterminants. *Bull. Sci. Math.*, (17):240–246, 1893.

[Hae01] N. M. Haegel, W. R. Schwartz, J. Zinter, A. M. White, and J. W. Beeman. Origin of the Hook Effect in Extrinsic Photoconductors. *Applied Optics*, 40:5748–5754, November 2001.

[Her96] Herreros, Y. Lossless Compression of Telemetry Data. Technical Report ASCO-TN-3000-013-AAR, Alcatel Espace, 1996.

[Hig91] J. Higbie. Uncertainty in the linear regression slope. *Am. J. Phys.*, 59(2), 1991.

[Hoi07] D. H. Hoiberg, A. Wolff, and T. Pappas, editors. *The New Encyclopaedia Britannica*. Encyclopaedia Britannica, Inc., fifteenth edition, 2007.

[Hol89] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian. A Real-Time Algorithm for Signal Analysis with the Help of the Wavelet Transform. In J.-M. Combes, A. Grossmann, and P. Tchamitchian, editors, *Wavelets. Time-Frequency Methods and Phase Space*, pages 286–, 1989.

[Huf52] D. Huffman. A Method for the Construction of Minimum Redundancy Codes. In *Proceedings of the IRE*, volume 40, pages 1098–1101, 1952.

[IEE87] IEEE 754. International Standard. Reprinted in SIGPLAN Notices Vol. 22 Nr. 2, February 1987.

[ISO04] ISO/IEC 15444-1:2004. Information Technology – JPEG 2000 Image Coding System: Core Coding System. International Standard, 2004. 2nd ed. Geneva: ISO, 2004.

[Iwa04] M. Iwahashi and M. Ohnishi. Analytical evaluation of integer DCT. In *IEEE International Symposium on Communications and Information Technology, 2004*, volume 2, pages 1065–1068, October 2004.

[Jay84] N. S. Jayant and P. Noll. *Digital Coding of Waveforms*. Prentice-Hall, 1984.

[Kam98] K. D. Kammeyer and K. Kroschel. *Digitale Signalverarbeitung*. Verlag B.G. Teubner Stuttgart, 1998.

[Ker00] F. Kerschbaum, H. Bischof, A. N. Belbachir, T. Lebzelter, and D. Höenigmann. Evaluation of FIRST/PACS data compression on ISO data. In J. B. Breckinridge and P. Jakobsen, editors, *UV, Optical, and IR Space Telescopes and Instruments*, volume 4013 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 253–262, July 2000.

[Kol65] A. N. Kolmogorov. Three Approaches to the quantitive definition of Information. *Problems of Information Transmission*, 1:1–7, 1965.

[Kor90] Kordes, F. L. G. and Hogendoorn, R. A. and Marchand, J. L. Handbook of data compression algorithms. Technical Report TM-06, ESA, 1990.

[Kri51] D. G. Krige. A statistical approach to some basic mine valuation problems on the Witwatersrand. *J. of the Chem., Metal. and Mining Soc. of South Africa*, 52(6):119–139, 1951.

[Lin06] P. Lindstrom and Isenburg M. Fast and Efficient Compression of Floating-Point Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), September/October 2006.

[Lit05] T. E. Litwin and J. N. Maki. Imaging Services Flight Software on the Mars Exploration Rovers. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, October 2005.

[Mac92] L. Mach. Über einen Interferenzrefraktor. *Zeitschrift für Instrumentenkunde*, 12:89–93, 1892.

[Mac94] Claudio Maccone. *Telecommunications, KLT and Relativity.* IPI Press, 1994.

[Mak05] J. N. Maki, T. Litwin, M. Schwochert, and K. Herkenhoff. Operation and performance of the Mars Exploration Rover imaging system on the Martian surface. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 1, October 2005.

[Mat98] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[Mau08] P. Mauskopf, D. Morozov, D. Glowacka, D. Goldie, S. Withington, M. Bruijn, P. DeKorte, H. Hoevers, M. Ridder, J. Van Der Kuur, and J.-R. Gao. Development of transition edge superconducting bolometers for the SAFARI far-infrared spectrometer on the SPICA space-borne telescope. In *Millimeter and Submillimeter Detectors and Instrumentation for Astronomy IV*, volume 7020 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, August 2008.

[Mof98] A. Moffat, R. M. Neal, and I. H. Witten. Arithmetic coding revisited. *ACM Trans. Inf. Syst.*, 16(3):256–294, 1998.

[Nak08] T. Nakagawa. SPICA mission for mid- and far-infrared astronomy. In *Space Telescopes and Instrumentation 2008: Optical, Infrared, and Millimeter*, volume 7010 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, August 2008.

[Nel70] E. D. Nelson and M. L. Fredman. Hadamard Spectroscopy. *Journal of the Optical Society of America*, 60(12), December 1970.

[Nyq24] H. Nyquist. Certain factors affecting telegraph speed. *Bell Syst. Tech. J.*, 3:324, 1924.

[Ott02a] R. Ottensamer, F. Kerschbaum, C. Reimers, A. N. Belbachir, and H. Bischof. The Austrian HERSCHEL/PACS On-Board Reduction Work Package. *Hvar Observatory Bulletin*, 26:77–80, 2002.

[Ott02b] R. Ottensamer, F. Kerschbaum, C. Reimers, E. Wieprecht, and H. Feuchtgruber. The HERSCHEL/PACS On-Board Data Reduction Concept. In D. A. Bohlender, D. Durand, and T. H. Handley, editors, *Astronomical Data Analysis Software and Systems XI*, volume 281 of *Astronomical Society of the Pacific Conference Series*, pages 303–, 2002.

[Ott04a] R. Ottensamer. *HERSCHEL/PACS On-Board Processing.* Master thesis, Universität Wien, 2004.

[Ott04b] R. Ottensamer, A. N. Belbachir, H. Bischof, H. Feuchtgruber, F. Kerschbaum, A. Poglitsch, and C. Reimers. Herschel/PACS on-board reduction/compression software implementation. In J. C. Mather, editor, *Optical, Infrared, and*

Millimeter Space Telescopes. Proceedings of the SPIE, Volume 5487, pp. 481-490 (2004)., volume 5487 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 481–490, October 2004.

[Ott05] R. Ottensamer, A. N. Belbachir, and F. Kerschbaum. Intelligent detectors – On-board data reduction for future missions. *Astronomische Nachrichten*, 326:582–583, August 2005.

[Ott08] R. Ottensamer and F. Kerschbaum. HERSCHEL/PACS on-board reduction flight software. In *Advanced Software and Control for Astronomy II*, volume 7019 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, August 2008.

[Pen93] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[Pen99] W. D. Pence. CFITSIO, v2.0: A New Full-Featured Data Interface. In D. M. Mehringer, R. L. Plante, and Roberts D. A., editors, *Astronomical Data Analysis Software and Systems VIII*, volume 172. ASP Conf. Ser., 1999.

[Pen06] B. Penna, T. Tillo, E. Magli, and G. Olmo. A New Low Complexity KLT for Lossy Hyperspectral Data Compression. In *IEEE International Conference on Geoscience and Remote Sensing Symposium, 2006.*, pages 3525–3528, August 2006.

[Pil08] G. L. Pilbratt. Herschel mission overview and key programmes. In *Space Telescopes and Instrumentation 2008: Optical, Infrared, and Millimeter*, volume 7010 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, August 2008.

[Plo04] G. Plonka. A global method for invertible integer DCT and integer wavelet algorithms. *Applied and Computational Harmonic Analysis*, 16:90–110, 2004.

[Pog06] A. Poglitsch, C. Waelkens, O. H. Bauer, J. Cepa, H. Feuchtgruber, T. Henning, C. van Hoof, F. Kerschbaum, D. Lemke, E. Renotte, L. Rodriguez, P. Saraceno, and B. Vandenbussche. The photodetector array camera and spectrometer (PACS) for the Herschel Space Observatory. In *Space Telescopes and Instrumentation I: Optical, Infrared, and Millimeter. Edited by Mather, John C.; MacEwen, Howard A.; de Graauw, Mattheus W. M.*, volume 6265 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, July 2006.

[Pog08] A. Poglitsch, C. Waelkens, O. H. Bauer, J. Cepa, H. Feuchtgruber, T. Henning, C. van Hoof, F. Kerschbaum, O. Krause, E. Renotte, L. Rodriguez, P. Saraceno, and B. Vandenbussche. The Photodetector Array Camera and Spectrometer (PACS) for the Herschel Space Observatory. In *Space Telescopes and Instrumentation 2008: Optical, Infrared, and Millimeter*, volume 7010 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, August 2008.

[Pre07] W. H. Press. *Numerical recipes: the art of scientific computing*. Cambridge University Press, third edition, September 2007.

[Rab78] L. R. Rabiner and Schafer R. W. *Digital Processing of Speech Signals*. Prentice-Hall, 1978.

[Rei04] C. Reimers, A. N. Belbachir, H. Bischof, R. Ottensamer, D. A. Cesarsky, H. Feuchtgruber, F. Kerschbaum, and A. Poglitsch. A Feasibility study of On-Board Data Compression for Infrared Cameras of Space Observatories. In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04)*, volume 1, pages 524–527, Washington, DC, USA, 2004. IEEE Computer Society.

[Rob04] W. Robinson et al. *CASSINI/RPWS Instrument Users Guide and Software Operations Manual*. University of Iowa, March 2004.

[Ros02] D. Rosenthal, J. W. Beeman, N. Geis, U. Grözinger, R. Hönle, R. O. Katterloher, S. Kraft, L. W. Looney, A. Poglitsch, W. Raab, and H. Richter. Stressed Ge:Ga detector arrays for PACS and FIFI LS. In *Proc. of NASA/Ames USRA/SOFIA Far-IR, Submm, & mm Detector Technology Workshop*, April 2002.

[Rüf92] P. Rüffer, F. Rabe, and F. Gliem. A DCT Image Data Processor for use on the Huygens Titan Probe. In *IEEE-IGRASS*, volume 92, 1992.

[Sal07] D. Salomon. *Data Compression*. Springer-Verlag London, fourth edition, 2007.

[Sau09] M. Sauvage. *PACS FM Photometer Focal Plane Unit User's Manual*. SAp CEA Saclay, first edition, February 2009. SAp-PACS-MS-0616-06.

[Say03] K. Sayood, editor. *Lossless Compression Handbook*. Academic Press, 2003.

[Say06] K. Sayood, editor. *Introduction to Data Compression*. Morgan Kaufmann, 2006.

[Sch85] M. R. Schroeder and B. S. Atal. Code-excited linear prediction (CELP): high-quality speech at very low bit rates. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 10, pages 937–940, 1985.

[Sch96] B. Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, 1996.

[Sch97] M. Schindler. A Fast Block-Sorting Algorithm for Lossless Data Compression. *Data Compression Conference*, page 469, 1997.

[Sch98] M. Schindler. A fast renormalisation for arithmetic coding. In Storer J.A. and M. Cohn, editors, *Proceedings of the 1998 IEEE Data Compression Conference*, volume 572, Los Alamitos, CA, USA, 1998. IEEE Computer Society.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27:379–423, 623–656, 1948.

[Sha49] C. E. Shannon. Communication in the presence of noise. In *Proc. IRE*, volume 37, pages 10–21, January 1949.

[Sta98a] J.-L. Starck, F. Murtagh, and A. Bijaoui. *Image Processing and Data Analysis*. Cambridge University Press, first edition, 1998.

[Sta98b] J.-L. Starck, F. Murtagh, and R. Gastaud. A New Entropy Measure Based on the Wavelet Transform and Noise Modeling. *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 45(8):1118–1124, August 1998.

[Sta06] J.-L. Starck and F. Murtagh. *Astronomical Image and Data Analysis*. Springer, second edition, 2006.

[Ste07] J. M. Stegmaier, S. M. Birkmann, U. Grözinger, N. Haegel, D. Lemke, and O. Krause. Transients in stressed Ge:Ga photoconductors under high background for PACS. In *Infrared Spaceborne Remote Sensing and Instrumentation XV*, volume 6678 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, October 2007.

[Str05] T. Strutz. *Bilddatenkompression*. Friedr. Vieweg & Sohn Verlag, third edition, 2005.

[Swi08] B. Swinyard, T. Nakagawa, H. Matsuhara, D. Griffin, M. Ferlet, P. Eccleston, A. di Giorgio, J. Baselmans, J. Goicoechea, K. Isaak, P. Mauskopf, L. Rodriguez, F. Pinsard, W. Raab, L. Duband, N. Luchier, N. Rando, A. Heras, T. Jagemann, N. Geis, and S. Vives. The European contribution to the SPICA mission. In *Space Telescopes and Instrumentation 2008: Optical, Infrared, and Millimeter*, volume 7010 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, August 2008.

[Swi09] B. Swinyard, T. Nakagawa, P. Merken, P. Royer, T. Souverijns, B. Vandenbussche, C. Waelkens, P. Davis, J. Di Francesco, M. Halpern, M. Houde, D. Johnstone, G. Joncas, D. Naylor, R. Plume, D. Scott, A. Abergel, S. Bensammar, J. Braine, V. Buat, D. Burgarella, P. Cais, H. Dole, L. Duband, D. Elbaz, M. Gerin, M. Giard, J. Goicoechea, C. Joblin, A. Jones, J. P. Kneib, G. Lagache, S. Madden, R. Pons, F. Pajot, D. Rambaud, L. Ravera, I. Ristorcelli, L. Rodriguez, S. Vives, A. Zavagno, N. Geis, O. Krause, D. Lutz, A. Poglitsch, W. Raab, J. Stegmaier, E. Sturm, R. Tuffs, H. M. Lee, B.-C. Koo, M. Im, S. Pak, W. Han, J.-H. Park, U.-W. Nam, H. Jin, D.-H. Lee, I.-S. Yuk, S. Lee, Y. Aikawa, N. Arimoto, Y. Doi, K. Enya, M. Fukagawa, R. Furusho, S. Hasegawa, M. Hayashi, M. Honda Kanagawa, S. Ida, Imanishi, Masatoshi, S.-i. Inutsuka, H. Izumiura, H. Kamaya, H. Kaneda, T. Kasuga, H. Kataza, K. Kawabata, M. Kawada, H. Kawakita, T. Kii, J. Koda, T. Kodama, E. Kokubo, Keiji Komatsu, H. Matsuhara, T. Matsumoto, S. Matsuura, T. Miyata, M. H. Miyata, H. Nagata, T. Nagata, T. Nakajima, K. Naoto, R. Nishi, A. Noda, A. Okamoto, Y. K. Okamoto, K. Omukai, T. Onaka, T. Ootsubo, M. Ouchi, H. Saito, Y. Sato, S. Sako, T. Sekiguchi, H. Shibai, H. Sugita, K. Sugitani, H. Susa, T.-s. Pyo, M. Tamura, Y. Ueda, M. Ueno, T. Wada, J. Watanabe, T. Yamada, I. Yamamura, N. Yoshida, K. Yoshimi, Y. Yui, M. Benedettini, R. Cerulli, A. Di Giorgio, S. Molinari, R. Orfei, S. Pezzuto, L. Piazzo, P. Saraceno, L. Spinoglio, T. de Graauw, P. de Korte, F. Helmich, H. Hoevers, R. Huisman, R. Shipman, F. van der Tak, P. van der Werf, W. Wild, J. Acosta-Pulido, J. Cernicharo, J. Herreros, J. Martin-Pintado, F. Najarro, I. Perez-Fourmon, J. Ramon Pardo, F. Gomez, N. Castro Rodriguez, P. Ade, M. Barlow, D. Clements, M. Ferlet, H. Fraser, D. Griffin, M. Griffin, P. Hargrave, K. Isaak, R. Ivison, M. Mansour, J. Laniesse,

P. Mauskopf, D. Morozov, S. Oliver, A. Orlando, M. Page, C. Popescu, S. Serjeant, R. Sudiwala, D. Rigopoulou, I. Walker, G. White, S. Viti, B. Winter, J. Bock, M. Bradford, M. Harwit, and W. Holmes. The space infrared telescope for cosmology and astrophysics: SPICA A joint mission between JAXA and ESA. *Experimental Astronomy*, 23:193–219, March 2009.

[Syl67] J. J. Sylvester. Thoughts on Orthogonal Matrices, Simultaneous Sign-Successions, and Tessellated Pavements in Two or More Colours, with Applications to Newton's Rule, Ornamental Tile-Work, and the Theory of Numbers. *Phil. Mag.*, (34):461–475, 1867.

[Tho86] A. R. Thompson, J. M. Moran, and G. W. Jr. Swenson. *Interferometry and Synthesis in Radio Astronomy*. Wiley, 1986.

[Wat01] J. Watkinson. *The MPEG Handbook*. Focal Press, 2001.

[Wei98] M. Weinberger, G. Seroussi, and Sapiro G. The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS. Technical Report HPL-98-193, Hewlett Packard Computer Systems Laboratory, 1998.

[Wei01] Changjiang Wei, Pengwei Hao, and Qingyun Shi. Integer DCT-based Image Coding. In *Picture Coding Symposium (PCS)*, pages 175–178, April 2001.

[Whi92] R. White, M. Postman, and M. Lattanzi. *Digitized Optical Sky Survey*, pages 167–175. Kluwer, 1992.

[Wie04] E. Wieprecht, J. Brumfit, J. Bakker, N. de Candussio, S. Guest, R. Huygen, A. de Jonge, J. J. Matthieu, S. Osterhage, S. Ott, H. Siddiqui, B. Vandenbussche, W. de Meester, M. Wetzstein, E. Wiezorrek, and P. Zaal. The HERSCHEL/PACS Common Software System as Data Reduction System. In F. Ochsenbein, M. G. Allen, and D. Egret, editors, *Astronomical Data Analysis Software and Systems (ADASS) XIII*, volume 314 of *Astronomical Society of the Pacific Conference Series*, pages 376–, July 2004.

[Wit87] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[Yar03] L. Yaroslavsky. *Digital Holography and Digital Image Processing: Principles, Methods, Algorithms*. Springer, 2003.

[Yeh93] P.-S. Yeh, R. F. Rice, and W. H. Miller. On the Optimality of a Universal Noiseless Coder. In *Proc. of the AIAA Computing in Aerospace 9 Conference*, pages 19–21, October 1993.

[Yu09] G. Yu, T. Vladimirova, and M. N. Sweeting. Image compression systems on board satellites. *Acta Astronautica*, 64:988–1005, 2009.

[Ziv77] J. Ziv and A. Lempel. A universal algorithm for data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–343, 1977.

[Ziv78] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, IT-24(5):530–536, 1978.

Index

Epilogue

So, PACS is working, I'm full of ideas of how to further improve the instrument and the next planned missions are already on the horizon. I have joined the SAFARI instrument consortium for the Japanese SPICA mission and I am in contact with PIs from other missions that are currently being planned. Just recently Anna Di Giorgio, the warm electronics developer of SPICA/SAFARI surprised me with her words, *"Roland, we can learn so much from you"*. This encourages me to continue my work and take on the new challenges in the European space programme.

Compressed Sensing

In the prologue I have named inventions that revolutionised astronomy. Maybe when reading this it came to your mind that it seems we would have everything now – the best telescopes with the best detectors at the best places to observe. So you might wonder what the next revolution might be, given these achievements. Well, if I have to place a bet – it's the way we measure and analyse the data and I can already name a candidate invention that has the potential of changing the way we measure, it's called *Compressed Sensing*. Together with French colleagues I am currently researching how this new conception can be used to improve the performance of PACS and if this is the next big leap forward.

> They say a technology needs to be obsolete on ground to be qualified for space.
> —*Dinner conversation with Marc Sauvage*

Acknowledgments

Alex, Jürgen, Paul, Robert, Stefan, Verena. The events of 2009 made me believe that *there is no time to lose*.

PACS Project

We have a working instrument thanks to the work horses that were involved during all those years. There are so many brilliant people in the PACS consortium that I don't want to start naming them here. They know anyway that they are meant! Some people in the consortium use to introduce me to others with the words, *"He is our on-board software"*. Although it is easy to get a conversation started from that, I have to stress out that our software is a common effort by quite a big team. Here are their names in alphabetical order with their most notable contribution in parentheses. Colleagues that spent more than 3 full years on the software are printed in Roman.

- University of Vienna: *Angela Baier (deglitching), Cornelia Diethart (detector selection editor), Josef Hron (science),* Franz Kerschbaum (project lead), *Thomas Lebzelter (test data),* Roland Ottensamer (OBSW development, ICC) *Thomas Posch (science), Peter Reegen (SW design),* Christian Reimers (documentation), *Werner W. Zeilinger (science)*

- TU Vienna: *Horst Bischof (OBSW work package lead),* Ahmed Nabil Belbachir (OBSW development), *Dieter Hönigmann (SW development), Florian Pezzei (SW design), Caterina Saraceno (SW design)*

- Joanneum Research: *Eveline Greschitz (PA/QA), Christian Kropiunig (PA/QA)*

Nabil, wir sind nicht *in die*, sondern nur *durch* die Hölle gegangen! ☺

This work was supported by the Austrian Federal Ministry of Transport, Innovation and Technology within the project FIRST/PACS Phase I and the ASAP project of the FFG/ALR.

Colophon

This thesis has been set in LaTeX, edited in emacs, plots were done in gnuplot, I used a little metapost and lots and lots of self-made tools. The complete PACS on-board software in its current version 13.96 has 16000 lines of C code and 1000 lines of assembler code. The de-compression module that I provided for the Herschel ground segment infrastructure has almost the same number of lines, written in Java™. All the plots in this thesis were made with 8000 lines of gnuplot scripts. The C codes for the various manipulations of *Mona Roli* add up to 12000 lines. Another 3000 lines are composed of shell scripts and awk scripts. Finally, all the LaTeX code for this thesis including styles and the plain texts are 10000 lines. If I would include all the sources, this thesis would have 1200 pages!

> Der Anblick war bittersüß; Das Ende eines Weges ist immer so.
> —*Bentley in Sly 3*

Only open source software was used to create this thesis and its contents.

Zusammenfassung

Wenn ein Satellit aus gutem Grund 50 Messwerte produziert, aber nur 5 zur Erde übertragen kann, stellt sich die Frage: *Wie können die Daten gesendet werden, ohne dass wichtige Informationen verloren gehen?* Probleme wie dieses kommen bei astronomischen Forschungssatelliten vor, da deren Detektoren vor allem in den letzten Jahren enorm gewachsen sind. Natürlich gibt es auch Fortschritte bei der Übertragung der Daten, die jedoch bei weitem nicht ausreichen, um mit dem Mengenzuwachs fertig zu werden. Die naheliegende Lösung des Problems besteht in der Datenkompression. Das allein muss nicht ausreichen, da wissenschaftliche Daten verrauscht sind und sich nur in geringem Maß komprimieren lassen. Also geht man einen Schritt weiter und baut Reduktionsschritte ein, die sonst Astronomen am Boden vornehmen würden. Ein Messgerät, welches in der Lage ist, in den oben genannten 5 Werten das Maximum an wissenschaftlicher Information unterzubringen, nenne ich einen *intelligenten Detektor*.

Die vorliegende Arbeit stellt eine Basis zur Entwicklung von On-Board Software für astronomische Satelliten dar. Sie dient als Anleitung und Nachschlagewerk und zeigt anhand der Projekte Herschel/PACS und SPICA/SAFARI, wie aus den Grundlagen weltraumtaugliche Flugsoftware entsteht. Dazu gehören das Verstehen des wissenschaftlichen Zwecks, also *was* soll *wie* gemessen werden und *wofür* ist das gut, sowie die Kenntnis der physikalischen Eigenschaften des Detektors, das Beherrschen der mathematischen Operationen zur Verarbeitung der Daten und natürlich auch die Berücksichtigung der Umstände, unter welchen der Detektor zum Einsatz kommt.

Diese Dissertation basiert auf Entwicklungen für Herschel/PACS. Sie beschreibt detailliert, wie die Messdaten des Instruments an Bord verarbeitet werden und enthält Programmteile, welche nun in 1,5 Millionen Kilometer Entfernung eingesetzt werden. Da das Problem der Kommunikation im Weltraum gegenwärtige und zukünftige Weltraummissionen zu Reduktionsschritten zwingt, wie sie bisher nur am Boden durgeführt worden sind, bedeutet das zugleich, dass den Astronomen nicht mehr Rohdaten zur Verfügung stehen, sondern zumindest teilweise reduzierte Datenprodukte.

Nach all den durchwegs schwierigen Jahren wurde Herschel am 14. Mai 2009 gestartet. In den ersten Monaten im Weltraum wurde das Instrument auf seine Leistungsfähigkeit überprüft und man kann mit Sicherheit sagen, dass die nun folgenden drei bis vier Jahre fantastische Entdeckungen im *kalten Universum* bringen werden.

Lebenslauf

Roland Klaus Johannes Ottensamer

1130 Wien · Geb. am 30.11.1975 in Freistadt, Oberösterreich

Schule und Studium

- 1982–1986 Volksschule Freistadt
- 1986–1994 Bundesgymnasium Freistadt, Matura mit gutem Erfolg
- 1996–2004 Astronomie an der Universität Wien
- 2004–2009 Doktoratsstudium Astronomie an der Universität Wien

Beruflicher Werdegang

- 01.2001–02.2002 Forschungsstipendium vom IfA: *Herschel/PACS*
- 03.2002–01.2006 Softwareentwickler am IfA: *Herschel/PACS on-board software*
- 02.2006–09.2009 Chefentwickler am IfA: *Herschel/PACS on-board software*
- 10.2009– Astronet/FFG Proj. CEA/IfA/TUG: *Compressed Sensing for Herschel*

Die wichtigste Publikation

- R. Ottensamer, F. Kerschbaum. HERSCHEL/PACS on-board reduction flight software. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7019, August 2008.

Sonstiges

- 1995–1996 Zivildienst an der Volkshilfe Braunau/Inn
- 2001–2007 Lehrtätigkeit am IfA (Tutor): Instrumentenkunde, Observatoriumspr.
- Gründungsmitglied der ÖGAA

Wien, am 12. November 2009