Normality Test for 2-bit Stream Data in Mark-5 Files

L. V. Benkevitch[1]

[1]MIT Haystack observatory, Westford, MA 01886, USA.

November 1, 2023

Abstract

The radio astronomy data from a single dish are recorded in M5B (Mark-5) or VDIF formatted files. The recorded data are supposed to be Gaussian white noise. A corrupted data segment can be spotted at the output of correlation stage. However, the correlation is expensive and vastly increases the bulk of data to be analyzed for spurious parts. Here we consider a problem of finding non Gaussian (or non-normal) single-dish data segments in M5B files.

The problem may be thought about from a security standpoint. Suppose someone (a malicious character) wishes to hack a system and figures an approach would be to embed some malicious code in. Is it possible to detect the embedded (non-Gaussian) data within a frame or multiple frames before the correlator? Also, even though an algorithm can be determined, is it possible to make its implementation fast enough? The data files are large (up to a hundred gigabytes).

Our solution is based on Pearson's χ^2 test. The null hypothesis states that the data in file are consistent with the normal distribution. If the data in a part of the file do not pass the test at the significance level 0.05, the null hypothesis is rejected, and the data fragment is flagged.

The algorithm runs on the GPU (Graphics Processing Unit), which provides high speed computations.

Contents

1 Signal Quantization and M5B Data Structure

The analog signal from a single antenna/dish is supposed to be Gaussian noise with the expectation μ and standard deviation (rms) σ distributed according to the normal law with the probability density function (PDF):

$$N(\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{1}$$

The signal is centered to ensure zero μ and its band is filtered into 16 frequency channels each of which is passed through the 4-level quantizer with the characteristic shown in Fig. 1. The quantizer has just three switching thresholds, $-\theta$, 0, and θ, so it is characterized with the only parameter, the adjustable quantization threshold $\pm\theta$. Note that the hardware deals with voltage thresholds, $\pm v_0$, but θ is expressed in the signal STDs, thus it is dimensionless, $\theta = v_0/\sigma$. The quantizer output only takes four values as in Tab. 1. These values are saved in the M5B file.

Table 1: Quantizer Characteristic

Input, x	**Output,** $\hat{x}$
$-\infty$ to $-\theta$	0
$-\theta$ to 0	1
0 to θ	2
θ to $+\infty$	3

The M5B file can be considered as an array of 10016-byte frames. Each frame consists of 2504 32-bit words (`unsigned int` or `uint` in C/C++). The first 4 words comprise the frame header, so a frame has 2500 words of pure data. Each 32-bit data word is subdivided into 16 2-bit channels, numbered from 0 to 15. The header contains the sync word `0xABADDEED`, 23-bit long frame number within second (reset to zero each new second), one flag bit to tag the frame invalid, 8-bit long channel ID, BCD Time Code Word 1 ('JJJSSSSS'), BCD Time Code Word 2 ('.SSSS'), and 16-bit CRCC.

Unfortunately, the frame headers do *not* contain the quantization threshold values $\theta = v_0/\sigma$ necessary for the statistical testing. In order to estimate θ, the algorithm has to spend precious time on its 1D search as will be described further.

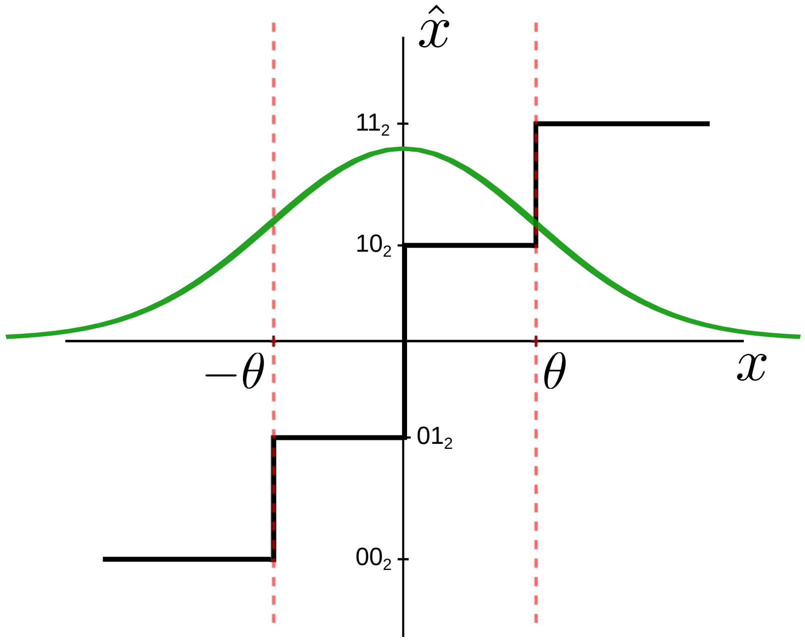


Figure 1: The 4-level quantizer characteristic is shown as a black polygonal line. The input analog signal x, distributed normally, can take any value in $(-\infty\,..+\infty)$. In response to x, the quantizer outputs $\hat{x}$ with only four values, 0, 1, 2, and 3, according to Tab. 1. The normal probability density function (PDF) of x is shown as a green line.

2 Statistical Testing Data Normality Using χ^2

We need to test if the data in each of the 16 channels are sampled from a normally distributed population (the null hypothesis) or not (the alternative hypothesis). In each frame and channel, there are $N = 2500$ data samples divided into $n = 4$ categories by their values, 0, 1, 2, and 3. The counts of data samples in each category form the 4 histogram bins $B_i, i = \overline{0\,..\,3}$, so $\sum_{i=0}^{3} B_i = N$. The Pearson's χ^2 test is based on comparing the observed data counts with the expected (theoretical) counts $E_i, i = \overline{0\,..\,3}$ obtained from the normal PDF (1) as $E_i = Np_i$. Here p_i is the probability that a random value sampled from $N(0, \sigma)$ is within the i-th interval. The intervals are given in Tab. 1, where i is one of the Output values. . . . . · · · . .

For each data frame the value of the test-statistic is calculated as

$$X^2 = \sum_{i=0}^{3} \frac{(B_i - E_i)^2}{E_i}. \tag{2}$$

The X^2 test statistic asymptotically approaches the $\chi^2_{k=3}$ distribution, where $k = n - 1 = 3$ is the number of degrees of freedom. The $\chi^2_{k=3}$ PDF is shown in Fig. 2. The statistic X^2 is a random value showing how close are the observation frequencies B_i to the perfect, theoretical frequencies E_i. The $\chi^2_{k=3}$ PDF has maximum at $X^2 = 1$ and the mean at $X^2 = k = 3$. This means that if X^2 is distributed as $\chi^2_{k=3}$ its most probable random values will be concentrated somewhere around the mean 3 and not much further. But how much further? The area under the $\chi^2_{k=3}$ PDF curve in Fig. 2 over an interval $[a, b]$ equals to the probability that the random value will appear within this interval. If we want to be 95% confident that the data is distributed normally, X^2 cannot exceed the critical value χ^2_{cr} such that the probability for X^2 to appear within the interval $[0\,..\,\chi^2_{cr}]$ is $p = 0.95$. The critical $\chi^2_{k=3}$ value can be calculated as the value of χ^2 Probability Point Function or PPF, the inverse of the χ^2 Cumulative Distribution Function or CDF. In Python this is done as `scipy.stats.chi2.ppf(1-alpha, k)`, where $\alpha = 0.05$ is the level of significance. In our case $\chi^2_{cr} = 7.81$.

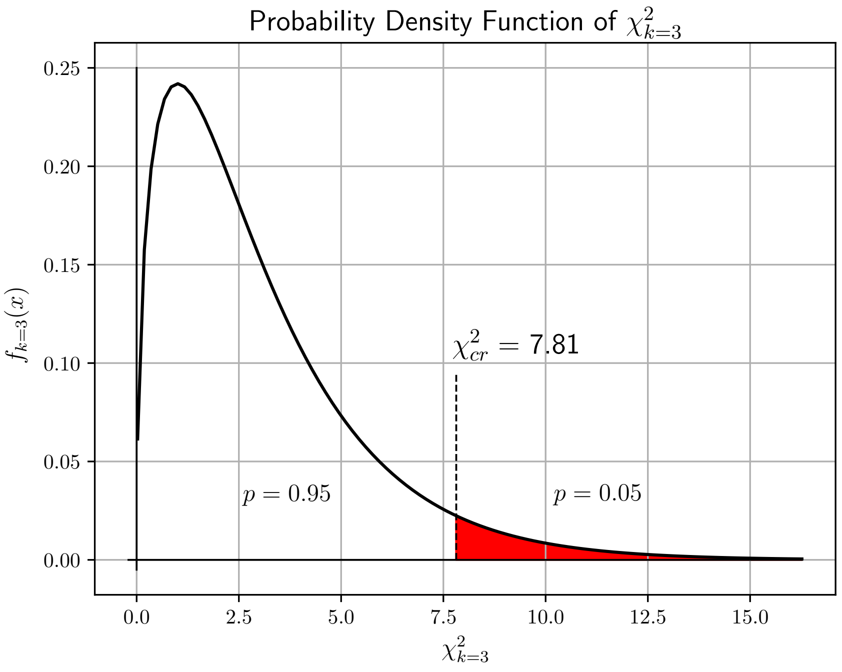


Figure 2: Chi-squared probability density function (PDF). The red area under the curve to the left of $\chi^2_{cr} = 7.81$ equals the probability $p = 0.05$. This is the probability that the X^2 statistic (Eq. (2)) for normal B_i exceeds 7.81.

Thus, our algorithm flags a channel data in frame as non-Gaussian if its $X^2 > 7.81$, the probability of incorrectly rejecting the null hypothesis being $\alpha = 0.05$. In Fig. 2 this is the red-filled area with the probability 0.05 over the range $[\chi^2_{cr}\,..+\infty)$.

3 Estimation of Quantization Threshold Using 1D Search

As mentioned, the M5B files do not provide the values of quantization threshold θ, which is continuously being adjusted and can differ from one frame to the next. Assuming the data are sampled from the normally distributed population we can hypothesize that the θ_{opt} value that provides the best fit of the observed frequencies B_i to the normal frequencies E_i can be a good estimate of the actual θ established at the quantization time.

One can notice that the X^2 statistics in Eq. (2) is a function of a single variable θ:

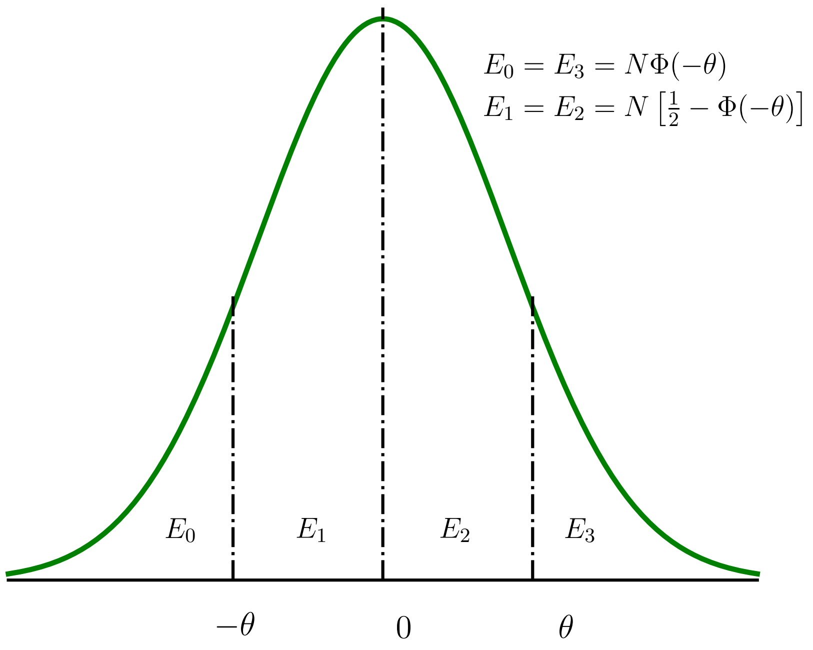
$$X^2(\theta) = \sum_{i=0}^{3} \frac{(B_i - E_i(\theta))^2}{E_i(\theta)}, \tag{3}$$

where the normal frequencies E_i are dependent on the positions of quantization thresholds:

$$E_{0..3}(\theta) = N \cdot \left[\Phi(-\theta); \quad \frac{1}{2} - \Phi(-\theta); \quad \frac{1}{2} - \Phi(-\theta); \quad \Phi(-\theta)\right],$$

as shown in Fig. 3, and $\Phi(x)$ is the normal CDF with $\mu = 0$ and $\sigma = 1$:

$$\Phi(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right]. \tag{4}$$



$E_0 = E_3 = N\Phi(-\theta)$
$E_1 = E_2 = N\left[\frac{1}{2} - \Phi(-\theta)\right]$
E_0
E_1
E_2
E_3
$-\theta$
0
θ

Figure 3: The normal frequencies E_i are dependent on the quantization thresholds $\pm\theta$. They are calculated with the use of normal cumulative distribution function (CDF) given in Eq. (4).

Fig. 4 shows that the curve $y = X^2(\theta)$ has the only minimum. Thus, the best fit of the normal frequencies E_i to the observed frequencies B_i can be reached at the single $\theta = \theta_{\text{opt}}$ value that provides minimum to $X^2(\theta)$.

We have used a fast-converging Brent's algorithm for one-dimensional search. It is a combination of the golden section search at the beginning and the parabolic interpolation when the process is close enough to the minimum.

4 Using Graphics Processing Units (GPU)

The Graphics Processing Units with their hundreds and thousands of processor cores and multiple memory channels allow 1 to 3 orders of magnitude speed-up of the common algorithms. There are two major software frameworks, CUDA and OpenCL. CUDA is developed specifically for the Nvidia GPUs. OpenCL can be used on many parallel architectures. OpenCL is the only option for the AMD GPUs.

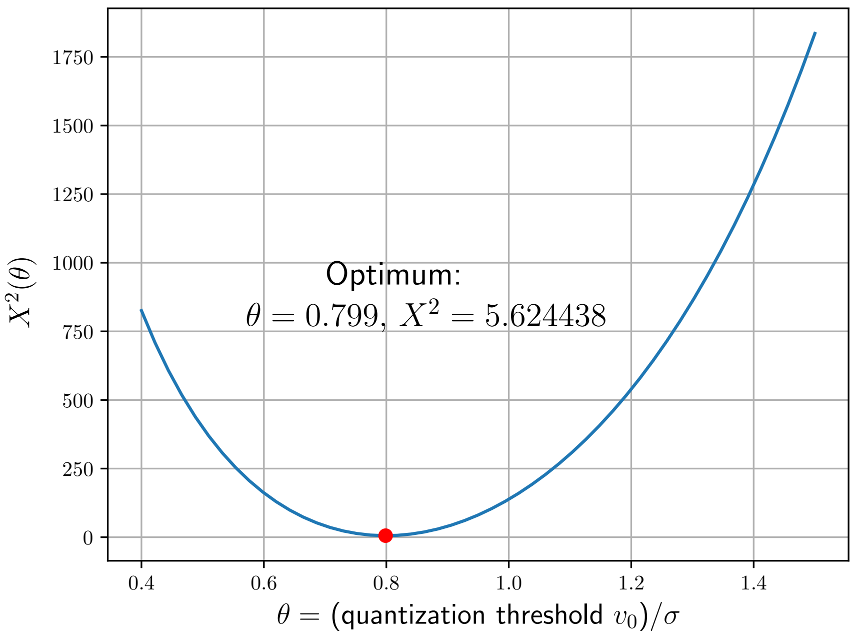


Figure 4: $X^2(\theta)$ from Eq. (3) as the measure of error between the observed, B_i, and normal, E_i frequencies for the range of quantization tresholds $[0.4\,..\,1.5]$. The curve has minimum at $\theta = 0.7988$, which can be used as an estimate for the actual threshold used in the analog signal quantization.

In both frameworks, the application code consists of two parts. One part is executed on the host PC. It reads the M5B files, prepares the GPU, uploads there the codes and the M5B data, starts the computations on the GPU, downloads the results from the GPU to the host memory, and saves them to the disk. The other part of code is executed in parallel on the multiple GPU cores. The programs executed on the GPU are called kernels.

In both frameworks we use single precision floating point arithmetics, which provides the fastest computations.

In the both frameworks, the kernels are written in the C language subsets with some extensions. The host code uses CUDA or OpenCL API. It can be written in C/C++ and some other languages. However, Python is always preferred to make life easier. We use Python wrappers for the both frameworks: PyCUDA and PyOpenCL, and the host part of the NormTest software is written in Python.

The Python module for normality testing, `gpu_m5b.py`, described in the next section, transfers control to the C kernel previously transferred to the GPU. The kernels are different for different frameworks used. In any individual normality test only two C files are used. Their names start with the strings `"ker_m5b_gauss_test"` and `"fminbndf"`:

`ker_m5b_gauss_test*` defines the test kernel `m5b_gauss_test()` and the minimized function `calc_chi2()`, which calculates χ^2 between the four bins of standard normal distribution and the four bins with the observation data counts.

`fminbndf*` implements one-dimensional search for the optimal quantization threshold θ_{opt}

based on Brent's algorithm. It calls the `calc_chi2()` function. The search is performed within the bounds $\theta \in [0.5\sigma \,..\, 1.5\sigma]$ with the absolute precision `atol` $= 10^{-4}$. Also, the number of iterations is limited to `maxiter = 20`.

In the CUDA framework just the two C files are used:

`ker_m5b_gauss_test.cu` - test kernel for CUDA;
`fminbndf.cu` - Brent's algorithm for CUDA.

The OpenCL framework requires more options because of the different implementations of OpenCL for different hardware platforms, AMD and Nvidia. Unfortunately, according to OpenCL Specifications, pointers to functions are not allowed, while CUDA does not have this restriction. So, in the OpenCL code for AMD GPU, instead of using this convenient mechanism to pass to the `fminbndf()` optimizer the pointer to an arbitrary minimized function as its parameter, we have to call `calc_chi2()` from inside `fminbndf()`. This makes `fminbndf()` less universal. On the contrary, Nvidia's OpenCL implementation (probably) has an extension allowing this feature. Therefore, there are four OpenCL C files:

`ker_m5b_gauss_test.cl` - test kernel for Nvidia OpenCL;
`fminbndf.cl` - Brent's algorithm for Nvidia OpenCL,

and

`ker_m5b_gauss_test_amd.cl` - test kernel for AMD OpenCL;
`fminbndf_amd.cl` - Brent's algorithm for AMD OpenCL.

The parallel processing in kernels is organized framewise. As many frames as possible are uploaded and stored un the GPU RAM, and each one data frame is processed in one CUDA thread or in one OpenCL work item in parallel. The frames are independent of one another, so no thread synchronization is required. Within one thread/work item, though, the frame and its 16 channels are processed sequentially. The kernel finds the counts of channel data in each of the four bins, runs the 1D Brent's search for the optimal quantization threshold, calculates χ^2 against the standard normal distribution, and saves the results in the output arrays. This is repeated until all the file chunks are processed.

5 Python Module for Normality Testing gpu_m5b.py

The `gpu_m5b.py` module defines the only class `Normtest` that integrates in itself all the normality testing mechanisms. When imported from the `gpu_m5b` module, `Normtest` class probes the hardware and automatically determines which framework to use. If it detects an AMD GPU, it uses OpenCL. If an Nvidia GPU is detected, the software uses CUDA. In case both frameworks are installed, it prefers CUDA since it is ~1.5 times faster than OpenCL. Thus this class provides "transparent" access to the GPU independent of the software framework used. Current version of the software can use only one GPU, however, in future it is possible to employ multiple GPUs on the same motherboard, even using different frameworks.

The `Normtest` class provides a "class method" `do_m5b()` with one mandatory, positional argument, a string with M5B filename, and an optional keyword argument, `nthreads`:

```
Normtest.do_m5b(<M5B file name> [, nthreads=<# of threads in a block>])
```

It runs the normality test on the available GPU using the software framework selected. If the M5B file is large and it does not fit into either the system RAM or the GPU RAM, it is processed in chunks. The `Normtest` class is not intended to create multiple class instances (although it is surely possible). Instead, the method `do_m5b()` is called directly with the class name or its alias. Below is an example procedure of normality tests on several M5B or M5A files:

```
from gpu_m5b import Normtest as nt
nt.do_m5b("rd1910_wz_268-1811.m5b")
nt.do_m5b("rd1910_ny_269-1413.m5a")
nt.do_m5b("rd1910_ny_269-1404.m5a")
nt.do_m5b("rd1903_ft_100-0950.m5b")
```

The results are saved in 5 binary files. The files have the following names:

`nt_<data>_<framework>_<m5b_basename>_<timestamp>.bin`,

where
`<data>` is one of the result types:
`hist:` `dtype=np.float32, shape=(n_frames,16,4)`, 4 histogram bins for 16 channels;
`chi2:` `dtype=np.float32, shape=(n_frames,16)`, χ^2 for 16 channels;
`thresh:` `dtype=np.float32, shape=(n_frames,16)`, quantization thresholds found for 16 channels;
`flag:` `dtype=np.uint16, shape=(n_frames,16)`, flags for 16 channels;
`niter:` `dtype=np.uint16, shape=(n_frames,16)`, number of iterations of Brent's optimization method used to find the optimal quantization threshold for 16 channels;
`<framework>` is `cuda` or `opencl`.
`<m5b_basename>` is the M5B or M5A file name without extension;
`<timestamp>` is `YYYYMMDD_HHMMSS` followed by milliseconds, like `20231015_113649.078`. The timestamp is intended to make the result filename reasonably unique.

Empirically, it has been found that the best performance is achieved with 8 threads per block (in CUDA terminology), or, which is the same, 8 work items per work group (in OpenCL terms). However, this number can be changed using the `nthreads` parameter. An example of setting 64 threads per block:

```
nt.do_m5b("rd1910_wz_268-1811.m5b", nthreads=64)
```

6 Data Flagging

The binary files `nt_flag*.bin` created by `Normtest.do_m5b()` method contain flags for every channel in every frame. They are actually arrays of C/C++ `unsigned short int` or Python `dtype=np.uint16` with the dimensions `[n_frames,16]`. The flags values mean the following conditions:

0: no errors;

1: bad frame signature; flags for all the channels are filled with 1.

2: 1D search for the optimal quantization threshold did not converge in 20 iterations;

3: $\chi^2 > \chi^2_{cr}$.

Thus the flag value 3 can indicate that the frame contains some extraneous data other than the Gaussian data streams from a dish. The flags can be easily read in IPython into an array. For example:

```
fl = np.fromfile("nt_flag_cuda_rd1910_wz_268-1811_text_20231024_174438.173.bin", \
                 dtype=np.uint16)  # Read the flag file into 1D array fl;
fl = fl.reshape((len(fl)//16,16)) # Convert it into 2D array fl[n_frames,16]
```

7 Python Scripts for Analysis of Results

7.1 inspect_nt.py

This script creates 4x4 plots of 16 histograms for each of the 16 channels. The plots are for one or several (averaged) frames. The histograms are compared with the normal distribution curves showing the vertical borders of the bins at the positions of quantisation threshold estimates, $\pm\theta$. In each subplot both θ and χ^2 values are printed. The χ^2 values exceeding $\chi^2_{cr} = 7.81$ are printed in red. The data are read from the `*.bin` files created with the `gpu_m5b_chi2.py`.

Running:

```
%run inspect_nt.py <m5b_filename> <timestamp> <start_frame_#> <#_of_frames>
```

The `<timestamp>` can be any rightmost part of the timestamp before the file extension, `.bin`. The timestamp is present in order to distinguish the `*.bin` files generated for the same M5B file at different runs. In most cases, providing the last 3 digis (i.e. milliseconds) is enough as long as they refer to the unique group of binary files with normality test results.

The histograms in Fig. 5 are close to the normal histograms with good χ^2 values, i.e. < 7.81. They are generated by

```
%run inspect_nt.py rd1910_wz_268-1811.m5b 575 1000 1.
```

7.2 plot_m5b_hist.py

Plots two histograms of the results from `gpu_m5b.py` for a whole M5B (or M5A) file. Running:

```
%run plot_m5b_hist.py <M5B file name> <timestamp>
```

One histogram is the distribution of χ^2 and a red marker showing position of the critical χ^2 value, $\chi^2_{cr} = 7.81$ at significance level 0.05, as well as the percent of χ^2 exceeding it. The other is

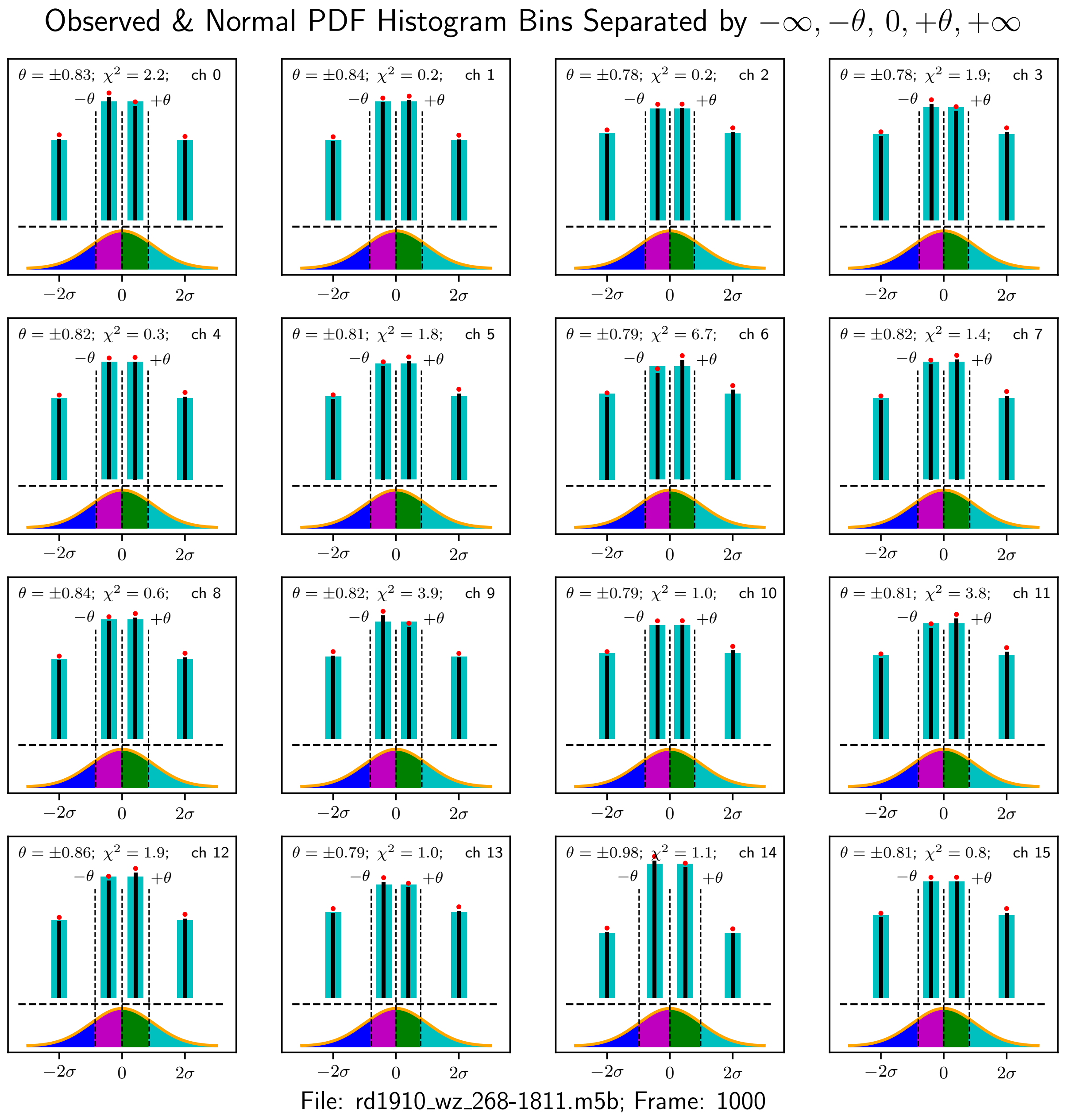


Figure 5: Observed and normal PDF histogram bins separated by $-\infty, -\theta, 0, +\theta, +\infty$. The histograms of observed data from an M5B file frame 1000 are shown as narrow black bars with red dots at the tips. The histograms of normal PDF are wider cyan bars. The normal PDFs with colored areas are given below the histograms. Here the observed distributions are close to the normal.

the distribution of the optimal quantization thresholds and a red marker showing position of the critical threshold value $\theta_{cr} = \pm 0.6745$ rms, as well as the percent of the thresholds that failed to reach it. The critical quantization threshold θ_{cr} is the one that separates 4 bins with equal areas under the normal curve. For $\theta = \theta_{cr}$ the histogram reflects the uniform distribution.

Fig. 6 and Fig. 7 in Addendum show example plots generated by

```
%run plot_m5b_hist.py rd1903_ft_100-0950.m5b 025
```

and

```
%run plot_m5b_hist.py rd1910_wz_268-1811.m5b 793.
```

7.3 Spotting Bad Frames with print_frame_chan_chi2.py

Sometimes a few channels in good data are flagged just because of random fluctuations in the data itself, and not because an alien piece of code was injected. Normally, in cases of fluctuations, χ^2 only slightly exceeds its critical value $\chi^2_{cr} = 7.81$. In contrast, a really foreign piece of code or data damages the whole frame or many frames in a row, and the χ^2 values in such bad frames are many times larger than χ^2_{cr}. In order to spot such frames we should roughen the Pearson's χ^2 criterion by searching for the frames where, say, $\chi^2 > 20$ or even 50. It can be done with the script

```
print_frame_chan_chi2.py <chi2 file name .bin> <chi2_threshold value>
```

For example:

```
%run print_frame_chan_chi2.py \
          nt_chi2_cuda_rd1910_wz_268-1811_bin_code_20231024_125630.067.bin 50.
```

It is also easy to do by hand in `IPython`. For example:

```
import numpy as np
c2 = np.fromfile("nt_chi2_cuda_rd1910_wz_268-1811_text_20231024_174438.173.bin"
                 dtype=np.float32) # Read chi^2 into 1D array c2
c2 = c2.reshape((len(c2)//16,16)) # Convert it into 2D table c2[frame,chan]
ic = np.where(c2 > 50) # Find indices into c2 where chi^2 > 50
# Print a table of frame #, channel #, and chi^2 > 50:
for i in range(len(ic[0])):
    print(ic[0][i], ic[1][i], c2[ic[0][i],ic[1][i]])
```

This will print something like

```
10 0 285.7461
10 1 314.56174
10 2 1594.8091
10 3 2500.0012
10 4 266.89368
10 5 338.24966
10 6 1693.4324
10 7 2500.0012
10 8 337.42603
10 9 342.7479
10 10 1626.329
10 11 2500.0007
10 12 257.20392
10 13 416.293
10 14 1633.9729
10 15 2500.0007
11 0 301.63565
  .  .  .
```

```
   .  .  .
5100 11 547.0831
5100 14 169.2154
5100 15 579.6725
```

We can see frame 10, as many others, is damaged, because its χ^2 values are very high. Inspect it:

```
%run inspect_nt.py rd1910_wz_268-1811_text.m5b 056  10 1
```

It pops up a 4x4-subplot picture quite similar to those shown in Fig. 8 and Fig. 9 in Addendum, where the histograms of the observed data have nothing in common with the normal distributions.

7.4 Miscellaneous

`plot_chi2.py`: Plots the graph in Fig. 2.
`plot_npdf_areas.py`: Plots the graph in Fig. 3.
`plot_m5b_thrange.py`: Plots the graph in Fig. 4.
`plot_various_bins.py` Plots 3x3 subplots showing the histograms of standard normal distribution for several different quantization thresholds $\pm\theta =$[0.2, 0.3, 0.5, 0.6745, 0.8, 1.0, 1.2, 1.5, 2.0]. The thresholds $\theta = \pm 0.6745\,\sigma$ produce histogram of the uniform distribution despite the input analog signal is normal. See Addendum, Fig. 10.
`plot_25pc_npdf_expectations.py`: Plots the area under normal PDF curve divided into 4 equal-area bins by the vertical lines at $-\theta_{cr} = -0.6745$, 0, and $\theta_{cr} = +0.6745$. The mathematical expectations of the bins, $\mu_1 = \pm 0.32\sigma$ and $\mu_2 = \pm 1.27\sigma$ are indicated with red dots. See Addendum, Fig. 11.
`get_dev_info_cuda.py`, `get_dev_info_opencl.py`, `get_cpu_mem.py`: Information scripts using PyCuda and PyOpenCL for the hardware inquiries.

8 Insertion of Extraneous Data into M5B File to Test the Algorithm

In order to see the effectiveness of our normality testing software we replaced some parts of the native data in the `rd1910_wz_268-1811.m5b` file with two types of extraneous data. One piece was an executable code, and the other piece was a UTF-8 coded text, actually, Part 1 of Leo Tolstoy's novel War and Peace in Russian. This insertions were made with the Python script `insert_code_in_m5b.py` on the copies of M5B files:

`rd1910_wz_268-1811_bin_code.m5b`
and
`rd1910_wz_268-1811_text.m5b`.

The script `insert_code_in_m5b.py` accepts two parameters, the M5B file name and the frame number from which the insertion starts:

```
insert_code_in_m5b.py <M5B file name> <starting frame #>
```

Note that the starting frame is filled with uniformly distributed data, and the code or text is inserted starting from the next frame.

The extraneous binary executable code was inserted starting from frame 1000:

```
%run insert_code_in_m5b.py rd1910_wz_268-1811_bin_code.m5b code1.bin 1000
```

and the UTF-8 text was inserted from frame 5000:

```
%run insert_code_in_m5b.py rd1910_wz_268-1811_text.m5b \
                 Tolstoy_War_and_Peace_Part_1_RU.html 5000
```

The corrupted files were then tested for normality with the code:

```
from gpu_m5b import Normtest as nt
nt.do_m5b("rd1910_wz_268-1811_text.m5b")
nt.do_m5b("rd1910_wz_268-1811_bin_code.m5b").
```

The latter normality tests produced binary files of the results including the bins with data counts for histogram plotting in `nt_hist_*.bin`, the quantization thresholds in `nt_thresh_*.bin`, and the χ^2 values in `nt_chi2_*.bin`. The damaged frames were inspected with the script `inspect_nt.py` that reads the binary files of results:

```
%run inspect_nt.py rd1910_wz_268-1811_bin_code.m5b 067  2000 1
%run inspect_nt.py rd1910_wz_268-1811_text.m5b 056  5006 1.
```

The 4x4, 16-channel histogram plots are given in Addendum, in Fig. 8 for the inserted text and in Fig. 9 for the inserted binary code. Obviously, the histograms do not show anything close to the normal distribution. In both Fig. 8 and Fig. 9 the χ^2 values are in the range of hundreds, which is orders of magnitude above the critical value of $\chi^2_{cr} = 7.81$ at the level of significance 0.05.

9 Benchmarks

In order to give a general idea of the time required for the normality tests on different M5A/M5B files, we have made testing on the files available using the PC with the following parameters:

Host CPU: Intel(R) Core(TM) i7 CPU X 980 @ 3.33GHz
Host RAM: 16GB
GPU: NVIDIA GeForce GTX 1060 3GB RAM
GPU CUDA Cores: 1152
GPU Memory Interface: 192-bit
GPU Graphics Clock: 2.0GHz
GPU Memory Transfer Rate: 8.0GHz
GPU Interface: PCI Express x16 Gen2
PCIe Link Speed: 5.0GT/s
Disk: Samsung SSD 870 EVO 2TB
OS: Ubuntu 23.04

With the 3GB GPU RAM, only 201,989 frames fits it at once, so the M5B file chunk size is about 2 GB. This data amount requires 2.3 - 2.4 s to be processed on GPU. However, readig of one file chunk into memory takes 7 - 8 s. Saving the results on disk takes 0.075 - 0.080 s per one

Table 2: Normality Test Timings

File name	File Size	Chunks	Total Time
rd1910_wz_268-1811.m5b	1.3 GB	1	4.4 s
rd1910_ny_269-1402b.m5a	4.6 GB	3	27.9 s
rd1910_ny_269-1413.m5a	5.7 GB	3	30.4 s
rd1910_ny_269-1404.m5a	54 GB	29	4 m 55 s
rd1903_ft_100-0950.m5b	72 GB	38	6 m 39 s

chunk. The benchmark results are given in Tab. 2.

The hardware used is rather outdated. We expect better performance on modern systems.

10 Conclusion

We have designed a highly effective algorithm of finding and locating any extraneous pieces of data in the M5A/M5B files. The algorithm is based on testing the M5B data for normality with the use of Pearson's χ^2 criterion. Our implementation is also highly efficient because it uses the GPU for massive parallelism. It is also well automated: before running the normality tests, at the time of importing the class `Normtest` it checks the hardware and makes a decision which of the GPU frameworks to use, `CUDA` or `OpenCL`, thus allowing usage of both Nvidia and AMD GPUs. Several Python scripts are provided to facilitate analysis of the normality test results.

Addendum

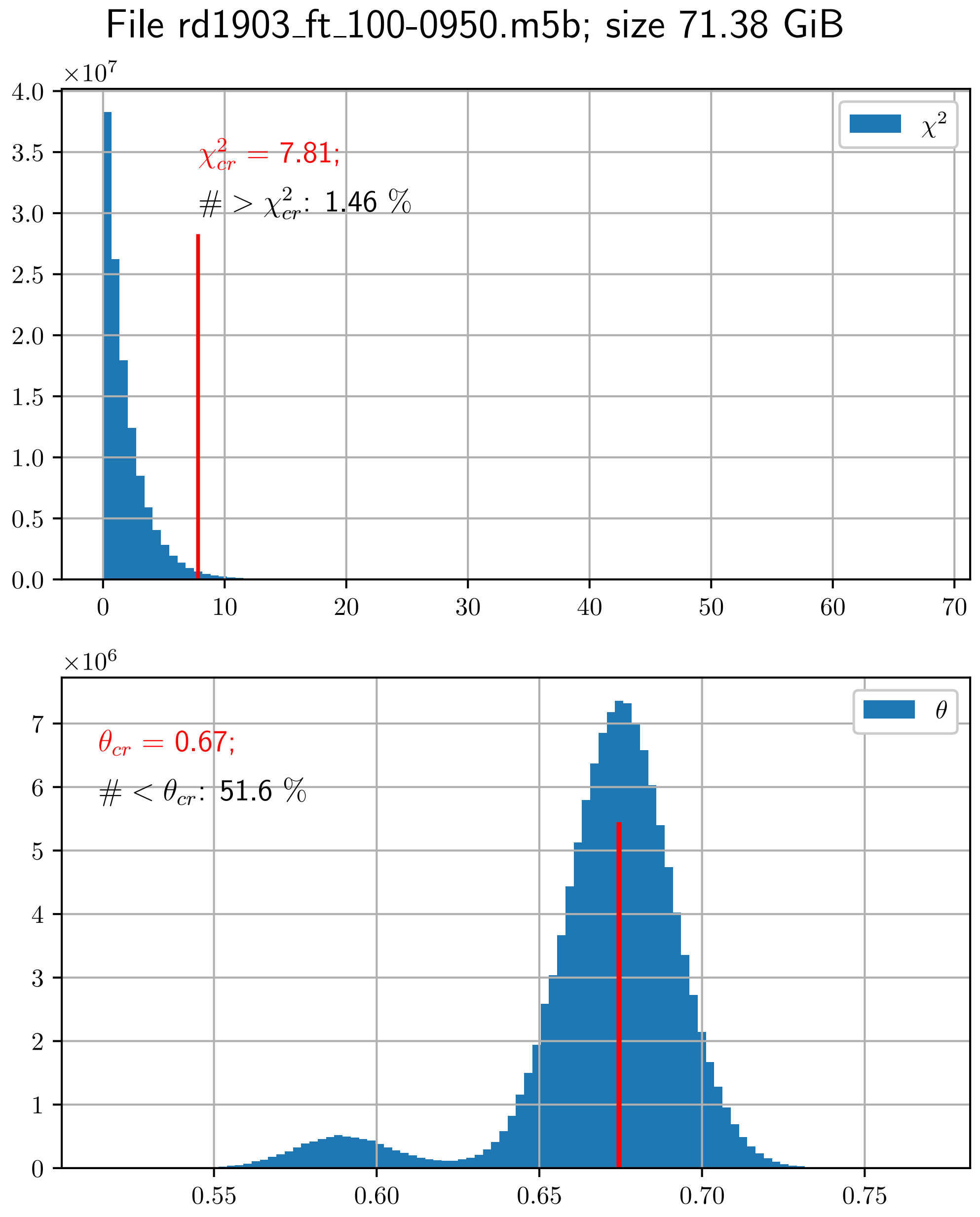


Figure 6: Distributions of χ^2 (upper pane) and θ (lower pane). The plots are based on the results of normality testing the data from file rd1903_ft_100-0950.m5b. The vertical red lines show the positions of χ^2_{cr} and θ_{cr}.

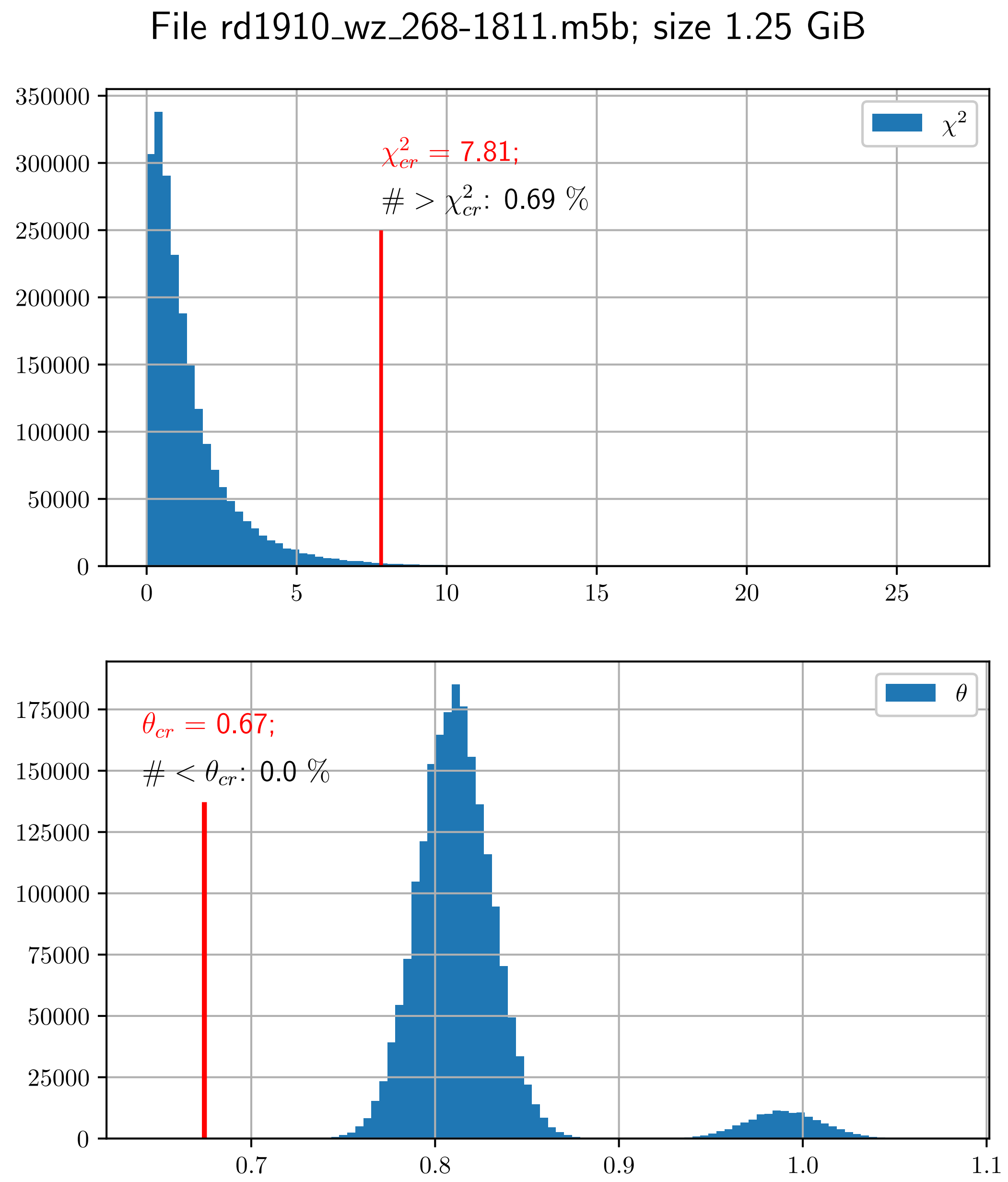


Figure 7: Distributions of χ^2 (upper pane) and θ (lower pane). The plots are based on the results of normality testing the data from file rd1910_wz_268-1811.m5b. The vertical red lines show the positions of χ^2_{cr} and θ_{cr}.

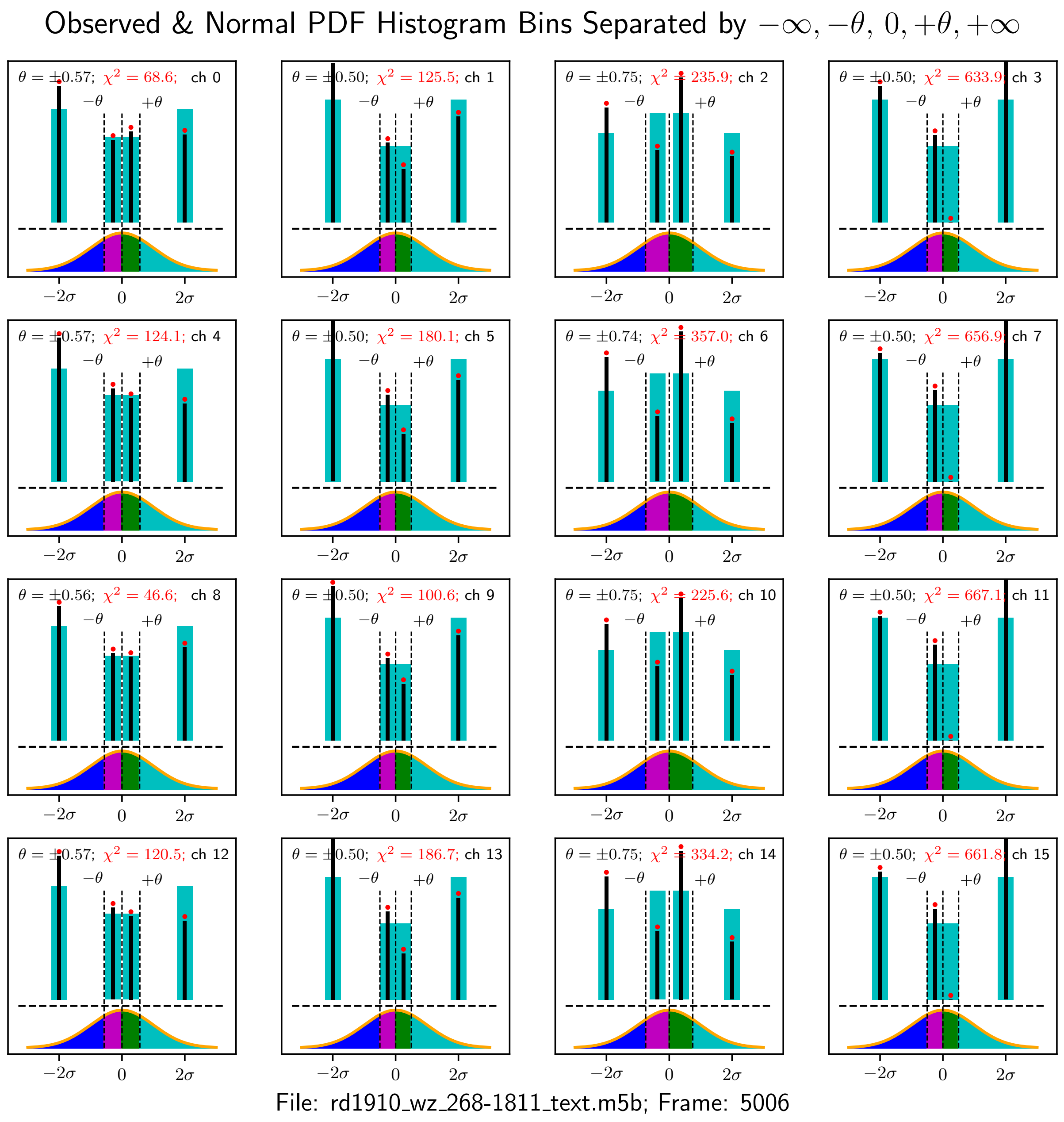


Figure 8: Observed and normal PDF histogram bins separated by $-\infty, -\theta, 0, +\theta, +\infty$. The histograms of observed data from an M5B file frame 5006 are shown as narrow black bars with red dots at the tips. The histograms of normal PDF are wider cyan bars. The normal PDFs with colored areas are given below the histograms. Frame 5006 was filled with some UTF-8-coded text, and one can see its distributions are far from the normal.

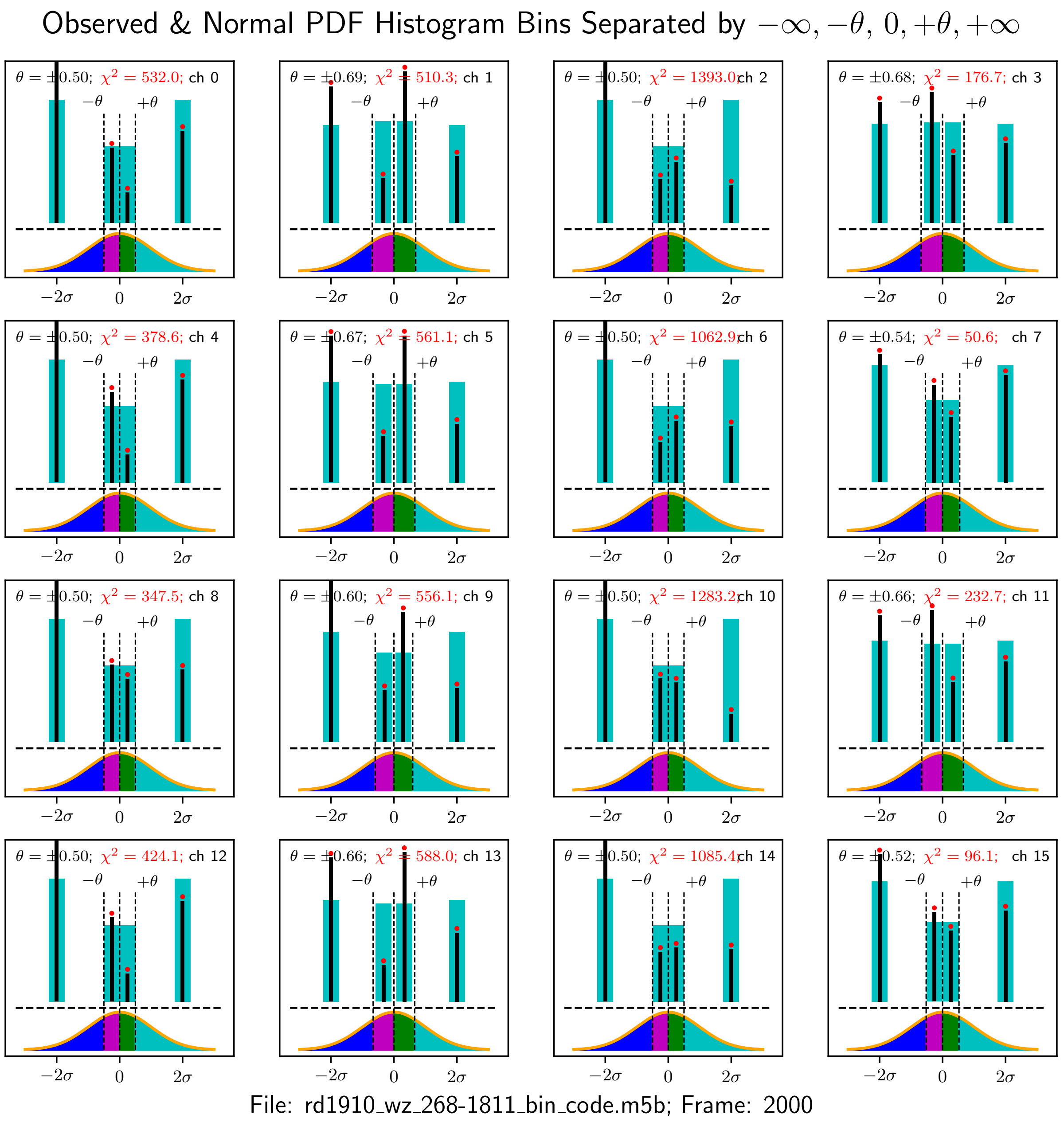
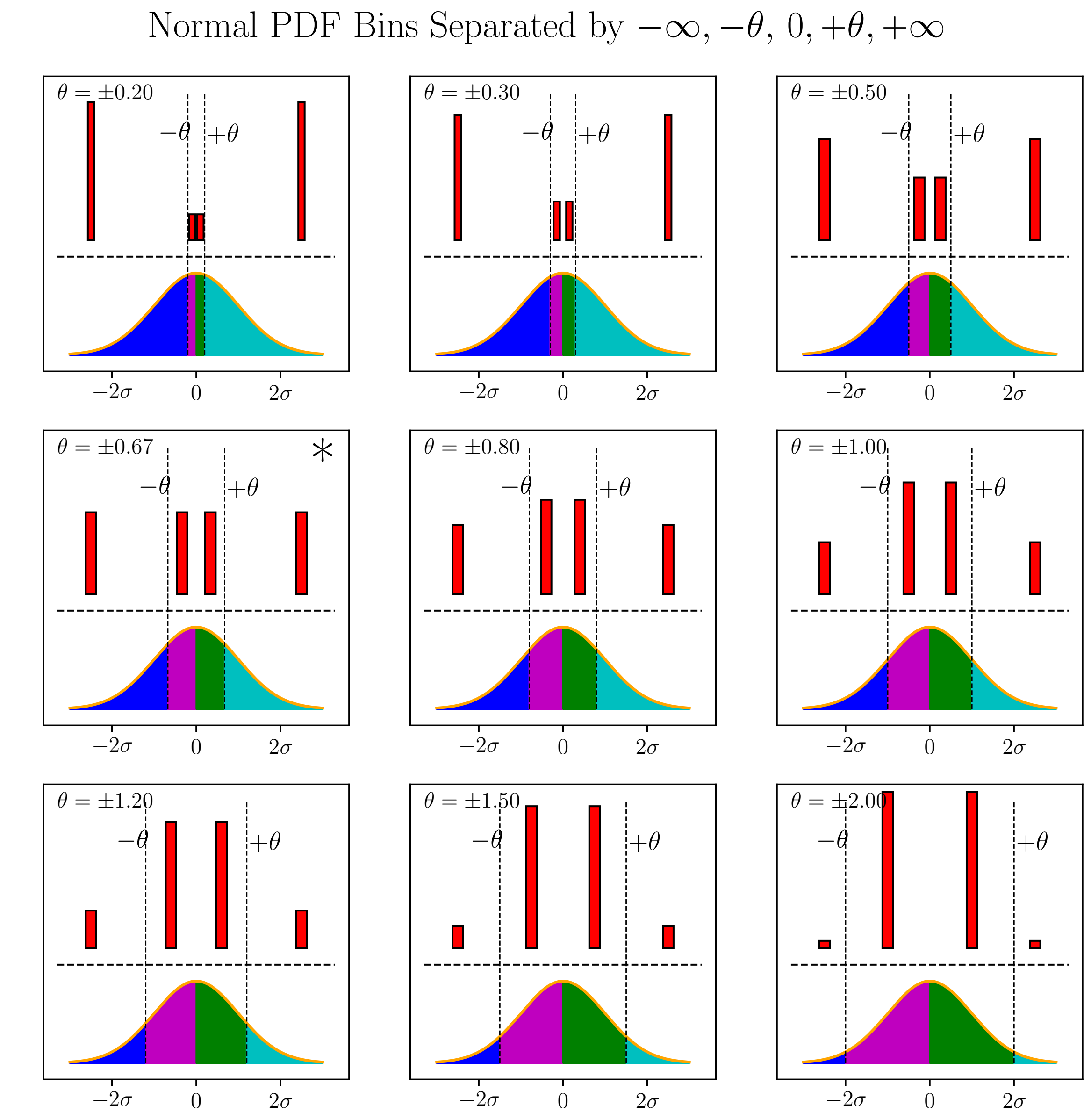


Figure 9: Observed and normal PDF histogram bins separated by $-\infty, -\theta, 0, +\theta, +\infty$. The histograms of observed data from an M5B file frame 2000 are shown as narrow black bars with red dots at the tips. The histograms of normal PDF are wider cyan bars. The normal PDFs with colored areas are given below the histograms. Frame 2000 was filled with some binary code (ELF 64-bit LSB executable), and one can see its distributions are far from the normal.



* Quantization Thresholds $\theta = \pm 0.6745\,\sigma$ produce histogram of uniform distribution.

Figure 10: The histograms of standard normal distribution for several different quantization thresholds $\pm\theta$ =[0.2, 0.3, 0.5, 0.6745, 0.8, 1.0, 1.2, 1.5, 2.0]. The thresholds $\theta = \pm 0.6745\,\sigma$ produce histogram of the uniform distribution despite the input analog signal is normal.

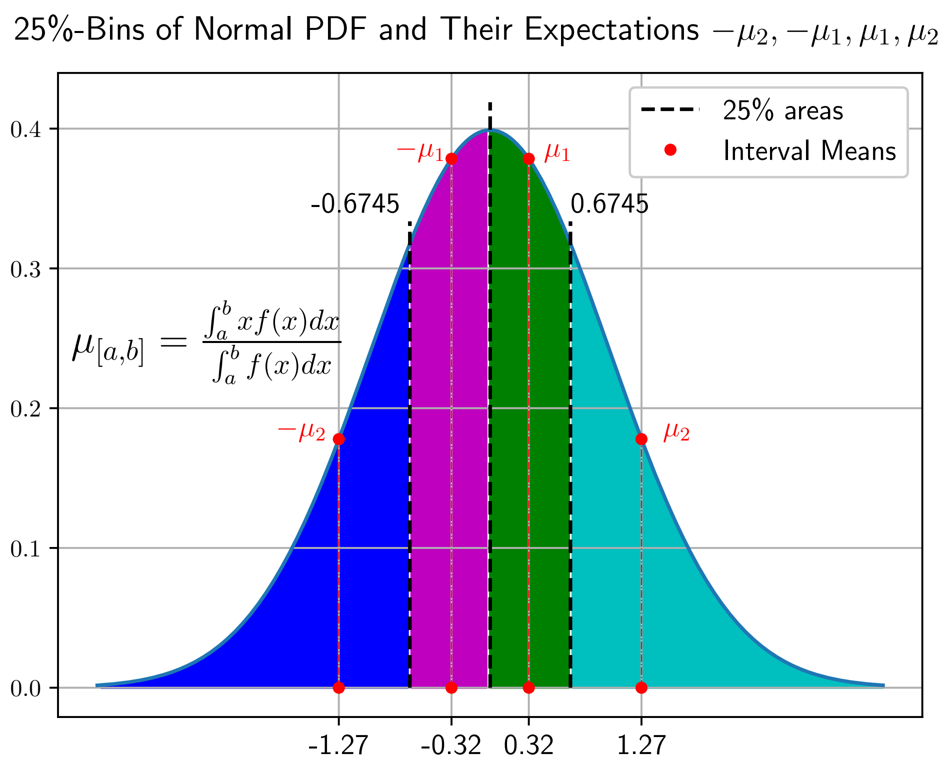


Figure 11: The area under normal PDF curve divided into 4 equal-area bins by the vertical lines at $-\theta_{cr} = -0.6745$, 0, and $\theta_{cr} = +0.6745$. The mathematical expectations of the bins, $\mu_1 = \pm 0.32\sigma$ and $\mu_2 = \pm 1.27\sigma$ are indicated with red dots.