Fall 5-18-2019

Real-time RFI Mitigation in Radio Astronomy

Emily Ramey
Washington University in St. Louis

Nick Joslyn
Simpson College, IA

Richard Prestage
Green Bank Observatory, WV

Michael Lam
West Virginia University, WV

Luke Hawkins
Green Bank Observatory, WV

See next page for additional authors

Author
Emily Ramey, Nick Joslyn, Richard Prestage, Michael Lam, Luke Hawkins, Tim Blattner, and Mark Whitehead

Real-time RFI Mitigation in Radio Astronomy

Emily Ramey[1], Nick Joslyn[2], Richard Prestage[3],
Michael Lam[4], Luke Hawkins[3], Tim Blattner[5], Mark Whitehead[3]

[1]Washington University in St. Louis, St. Louis, MO
[2]Simpson College, Indianola, IA
[3]Green Bank Observatory, Green Bank, WV
[4]West Virginia University, Morgantown, WV
[5]National Institute of Standards and Technology

October 1, 2017

Abstract

As the use of wireless technology has increased around the world, Radio Frequency Interference (RFI) has become more and more of a problem for radio astronomers. Preventative measures exist to limit the presence of RFI, and programs exist to remove it from saved data, but the use of algorithms to detect and remove RFI as an observation is occurring is much less common. Such a method would be incredibly useful for observations in which the data must undergo several rounds of processing before being saved, as in pulsar timing studies. Strategies for real-time mitigation have been discussed and tested with simulated data[2][3], but ideally the results of any approach would be validated by a detailed comparison of the final data products with and without mitigation applied. The goal of this project is to develop an RFI mitigation approach based on strategies suggested by Buch et al.(2016)[2] and to test this program on real data from the observation of pulsar J1713+0747 at the Green Bank Observatory in West Virginia. We use a Median Absolute Deviation (MAD) filter to identify interference in the observation and replace the compromised data with random Gaussian noise to match a characteristic radio signal from space. In order to verify our results, we analyze the pulsar's timing residuals obtained both from the mitigated data and from data processed through offline RFI removal software. Comparing the two, our preliminary findings indicate that our program is able to significantly improve the quality of timing results from the observation.

1 Introduction

Most radio telescopes must detect signals on the order of $10^{-29}\frac{W}{m^2 Hz}$ and below in order to effectively gather data for current projects in astronomy. However, commercial radio signals are traditionally much stronger than even the brightest radio emission from space, and thus they have the potential to significantly impair the collection capabilities of radio telescopes. Any earth-based signal that interferes with an astronomical observation is referred to as RFI, or Radio Frequency Interference.

There are many different types of RFI, but on a broad scale, they can all be classified into two categories. Broadband RFI occurs over a wide range of frequencies but a short duration in time. This can be caused by events such as lightning strikes, short-circuits in electronics, or strong radar signals. Narrowband RFI is the opposite - it occurs over a localized range of frequencies but can be present in an observation for a significant amount of time. Cell phones, radio stations, and even LED displays in close enough proximity to a telescope can all be sources of narrowband RFI. Figure 1 shows an example of broadband RFI in the data set used for this project.

Several strategies are used to limit the presence of RFI in astronomical observations. Buch et al.(2014)[3] discusses many such methods. One way to mitigate RFI is to limit the emission in the vicinity of the radio telescope via regulation of possible sources of RFI, such as in the National Radio Quiet Zone (NRQZ) surrounding the Green Bank Observatory in West Virginia. Another method is to use spatial nulling to cancel out the signal from known emitters of RFI

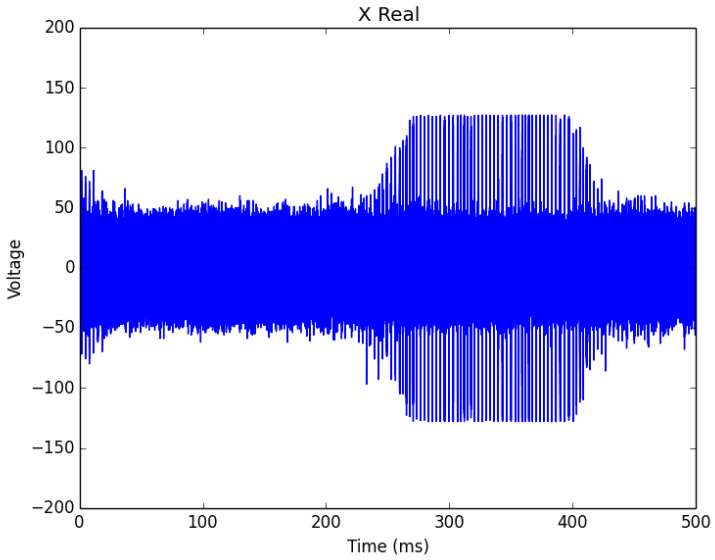


Figure 1: An example of broadband RFI in the observation of pulsar J1713+0747. Likely caused by radar signals from a nearby airfield.

in the environment. However, these methods can only be fully effective if observers know ahead of time the spatial location of every environmental source of RFI for the entire duration of the experiment. Even without factoring in the public usage of radio-emitting technologies, this is a near impossibility due to the unpredictability of natural sources of radio waves, such as thunderstorms and other weather events. In the inevitable case that RFI makes it into observational data, other methods can be used to find and remove it. RFI excision programs can be used to "zap" an observation, removing entirely the sections of data that it identifies as being corrupted with RFI, and researchers sometimes manually sift through a data set in order to remove the more obvious examples of RFI.

Due to the significant increase in the processing speed of computers, new strategies of RFI excision are available that would have been infeasible in decades past. One such strategy, proposed by Buch et al.(2016)[2], suggests filtering RFI out of astronomical data in real time, before it has been saved to disk. This method would be ideal for studies which require data to be reduced and/or processed before it can be stored permanently. The field of pulsar astronomy, in particular, could benefit greatly from the use of such a method, as the signal strength of most pulsars is lower than the noise level of even the most sensitive telescopes. This makes it less likely that usable data will be removed by an RFI-filtering algorithm. In addition, it takes several Terabytes of memory to record even a few hours of raw data, and observations must undergo several rounds of processing before they are saved to disk. With any post-processing mitigation approach, it is much harder to distinguish RFI from usable signals, and more data is lost as a result of any excisions that occur. A real-time mitigation approach would, in theory, solve these problems for most pulsar observations.

Real-time RFI mitigation programs have been tested on simulated data with promising results[2], but, to our knowledge, no evaluation has yet been performed of the method's effects on real observational data. Our goal in undertaking this project is to create a prototype for an RFI mitigation program following the guidelines set by Buch et al.(2016)[2], to test this program on raw data in a manner that mimics real-time data processing, and to compare the results obtained from different methods in order to evaluate the effectiveness of our program.

2 Data Processing

The data used in this research comes from the observation of Pulsar J1713+0747, a binary pulsar with a period of 4.57 milliseconds, which, among other things, is used by NANOGrav to study gravitational waves. This pulsar has a very well-defined pulse pattern due to many years of observations by the astronomical community, and that makes it a good baseline on which to

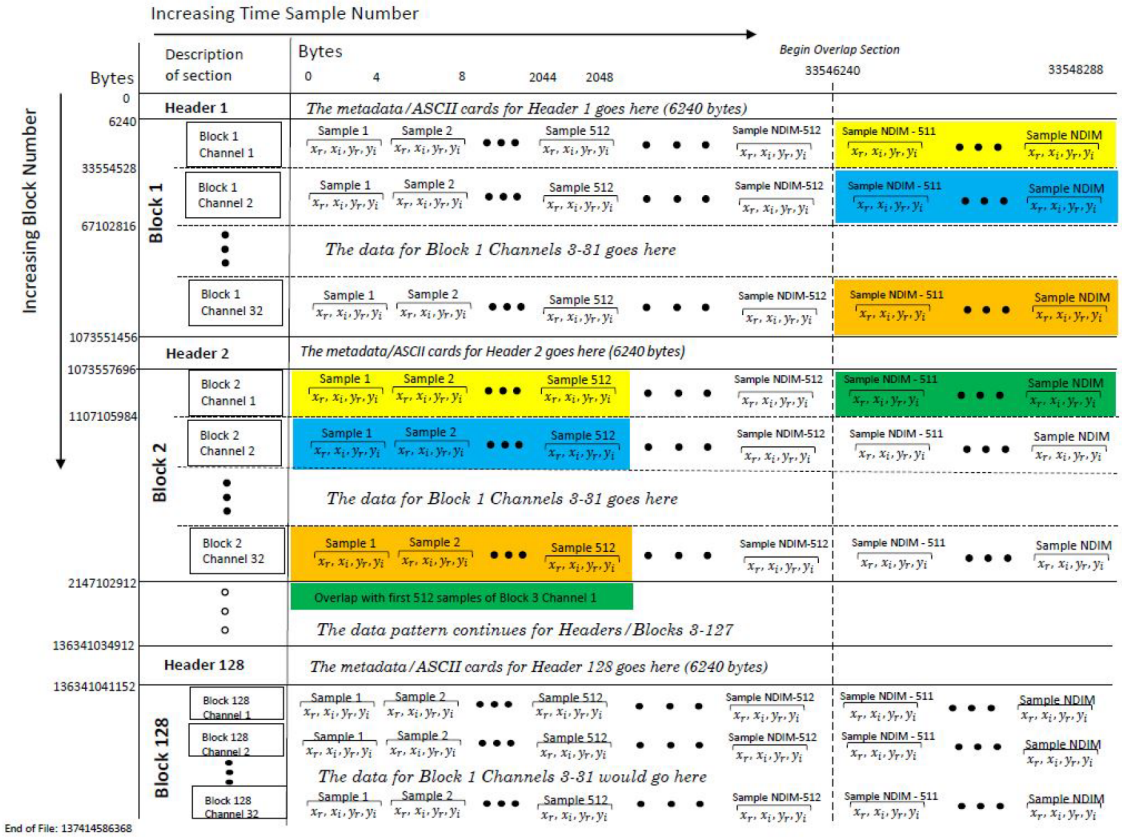


Figure 2: A block diagram representing the data storage scheme of a GUPPI RAW image file. Highlighted areas indicate portions of the data which overlap each other.

conduct this study. The observation was performed with a bandwidth of 200MHz centered on 1378.125 MHz. In total, we have roughly 5 Tb of raw data taken over the course of 24 hours. For this part of the project, we focused our efforts on ten consecutive files, 128 Gb each, taken over a few hours near the beginning of the observation.

2.1 GUPPI Raw data files

The data we have from pulsar J1713+0747 was processed through the Green Bank Ultimate Pulsar Processing Instrument (GUPPI) backend of the Green Bank Telescope (GBT). However, while this backend normally outputs PSRFITS files, which have undergone steps for data reduction, the data stream was modified for use in this project by Keith Omogrosso, a summer student in 2016, in order to write out 5 Tb of GUPPI Raw files. These files have been written from the data stream with minimal processing and are used to simulate real-time acquisition of observational data in our program. The data in GUPPI Raw files is separated into 32 channels, each with a bandwidth of 6.25 MHz, two perpendicular polarizations, and real and imaginary components of each polarization. It is also divided into data blocks, with a separate header for each block. The data for this observation has a resolution of 0.16 μs, and each data point is saved in an 8-bit signed char format. A visual representation of the data storage layout can be found in Figure 2.

2.2 De-dispersion

Astronomical signals from pulsars must travel a significant distance before reaching Earth, and as such are subject to dispersion by the interstellar medium (ISM). The dispersion equation is as follows[5]:

$$(\frac{t}{sec}) \approx 4.149x10^{3}(\frac{DM}{pc, cm^{-3}})(\frac{\nu}{MHz})^{-2}$$

$$DM = \int_0^d n_e dl$$

Where t is the time delay of the signal due to dispersion, d is the distance to the source, and DM is the dispersion measure, which is related to the electron number density between Earth and the source.

As the ISM scatters lower frequencies more strongly than higher ones, individual pulses are broadened, as shown in figure 3. This phenomenon can make it extremely difficult to find new pulsars, as it lowers the signal strength of the pulses, and, in many cases, causes the pulse profile to be obscured entirely. In most pulsar searches, data must be processed with several dispersion measures to determine whether a pulsar is present in a given observation.

Since we are observing a well-observed pulsar, we know the dispersion measure to be roughly 15.97 pc/cm^3. In order to coherently de-disperse the signal, we use a discrete transfer function, which requires data to be repeated at the beginning of each code block in the GUPPI raw files in order to run. Signals are normally de-dispersed in the GUPPI backend as a routine method of processing, but, in this case, we use a program called DSPSR, or Digital Signal Processing Software for pulsar astronomy, to de-disperse the data offline[7].

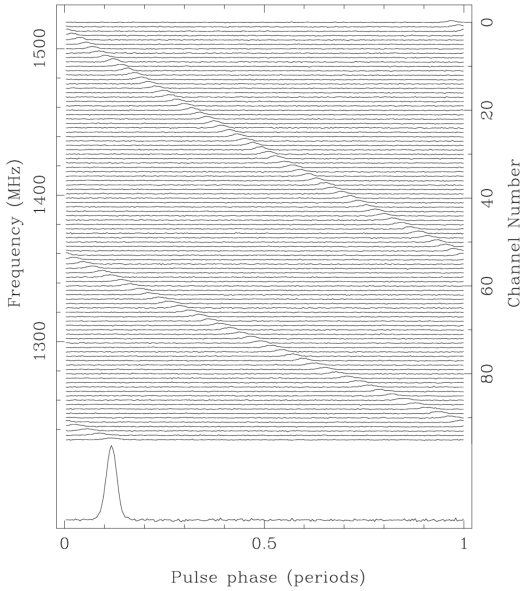


Figure 3: An illustration of the effects of dispersion on a pulsar signal. From Lorimer et al.(2014)[5]

2.3 Folding

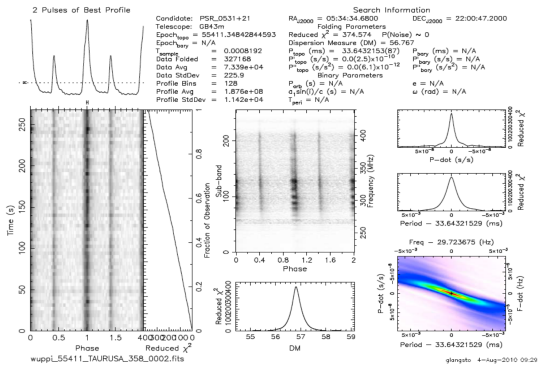


Figure 4: An example of the folding of pulsar data to obtain a pulse profile. This observation is of pulsar J0531+21 taken from a log of example data on the NRAO's website[6].

Even after a pulsar signal is de-dispersed, the signal-to-noise ratio is still usually too low to obtain a good pulse profile. The data must undergo a process called folding in order to increase the signal strength, which averages many pulse periods of the data in order to eliminate the effects of noise. Because noise is random about zero, it cancels out when averaged over several pulse periods, while the signal of the pulsar has a consistent offset and gets stronger when averaged. An example of the folding of a pulsar observation can be found in figure 4. This step is essential to determining the pulse times of arrival, which will be covered in a later section of this paper.

2.4 Characteristics of the data

The signal received by a radio telescope can be characterized as follows[3]:

$$x(t) = x_{signal}(t) + x_{noise}(t) + x_{RFI}(t)$$

where x_{signal} is the signal we are attempting to detect, x_{noise} is the noise from the instrumentation and the background noise from space, and x_{RFI} is the signal from RFI. Without the presence of RFI, the part of the signal that comes from noise dominates the observation. Since the strength of the signal from pulsar J1713+0747 is below the noise level of our observation, individual pulses

are not visible in the data. This means that, without the presence of RFI, the signal that the telescope receives should look exactly like random Gaussian noise. The part of the signal that comes from RFI, however, cannot be approximated by random Gaussian noise, and most emission of RFI is much stronger than that of background radiation and system noise. Therefore, in order to determine if RFI is present in a data sample, we need only remove the data points which do not conform to a Gaussian distribution.

3 Median Absolute Deviation Filter

One of the properties of a random Gaussian noise distribution is that 99.9% of its data is contained within three standard deviations of the mean. As the signal from RFI is much stronger than the background noise, we would expect it to fall outside the standard Gaussian distribution. Therefore, it is reasonable to assume any data point more than three standard deviations away from the mean to be RFI. However, using the mean and the standard deviation as benchmarks for our filter presents a problem in that these statistics give more weight to outliers in the data. The more RFI is present, the larger the upper and lower bounds will be, and this ends up letting a significant amount of RFI slip through the filter unidentified.

A solution to this problem is to change the statistic on which we base our filter from the standard deviation to the Median Absolute Deviation (MAD), as described in Buch et al. (2016)[2]. The MAD of a data set is not based on the mean of the data, but, rather, the median, and is calculated as follows for a given data set, X:

$$M = Median(X)$$
$$MAD = Median|X - M|$$
$$\sigma_r = 1.4826 * MAD$$

where σ_r is the robust standard deviation. This metric is much less susceptible to RFI than one based on the standard deviation, as it allows for up to 50% of the data to be replaced outliers without any change in the filter strength. We now set the bounds of our data set at three median absolute deviations away from the median of the data set, instead of three standard deviations away from the mean. Therefore, we determine a data point, x_i to be RFI if:

$$x_i > M + 3\sigma_r$$

or

$$x_i < M - 3\sigma_r$$

All data points that fit these criteria are replaced with a computer-generated distribution of random Gaussian noise. Figure 5 shows the bounds of the filter for a MAD size of 32,768 bytes.

4 Median Finding Algorithms

As any large-scale implementation of the MAD filter will involve finding the median of a data set nearly constantly, it is essential that we use an optimal median-finding algorithm for our program. One particular criterion is that the algorithm should lend itself to parallelism, as it will eventually be fully implemented on a Graphics Processing Unit (GPU), a system optimized for parallel computing. For a data set of size N, most sorting algorithms can only run in O(Nlog(N)) time due to the need for comparison between elements. However, there are a few types of sorting which use properties of the data to eliminate comparison entirely, and can run in linear (i.e. O(N)) time.

4.1 Radix Sort

One sorting method we consider for the program is the radix sort, which uses bit-wise ordering to sort binary numbers. There are many different ways to implement a radix sort, but a most significant digit (MSD), in-place, and recursive radix sort works best for our purposes. The steps of this algorithm are as follows:

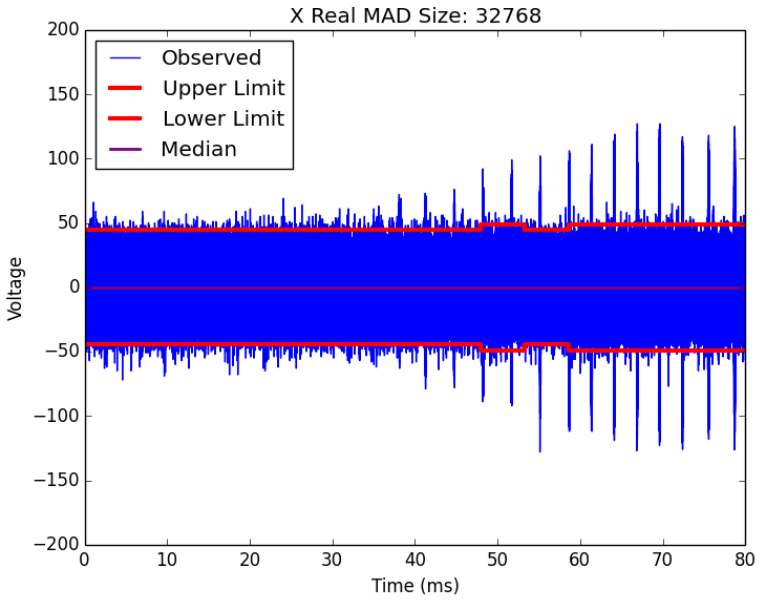


Figure 5: An illustration of the bounds of our MAD filter for the x polarization of channel 32 of our observation.

1. The start of the zeros bin is the beginning of the set and the start of the ones bin is one past the end of the set.
2. k is the place of the digit to evaluate. If k < 0, return the set.
3. If the array to sort contains only one number, return the set.
4. While the zero pointer does not equal the ones pointer:
   (a) Obtain the k^{th} bit of the number at the zeros pointer.
   (b) If the k^{th} bit is a zero, increment the zeros pointer.
   (c) If the k^{th} bit is a one, switch the number to the right of the zeros pointer with the number to the left of the ones pointer and decrement the ones pointer.
5. Recursively run the sort on the zeros bin.
6. Recursively run the sort on the ones bin.

After the sort completes, the number at location N/2 is the median (for an odd number of data points). It must be noted that, in order to sort signed data types, the zeros and ones bins must be switched for the first iteration. If we have a data set of size N, the time that this algorithm takes to run is at most k*N, where k is the number of bits in an element of the data set. When implemented in parallel, it can run in 2N time in its best case. The program also uses a linear amount of memory that corresponds to the input, as a copy of the initial data must be made in order to preserve its ordering for later use.

4.2 Histogram Sort

The histogram sort is ideal for data in which the number of elements in the set exceeds the number of discrete values that can be taken on by any individual element, which is true of our time-domain data. The steps to perform a histogram sort are as follows:

1. Create an array to represent the bins of the histogram - the size depends on the range of the data set and the desired accuracy of the result.
2. For each element of the data set, add one to the bin that corresponds to its value.

3. For each bin in the histogram array:
   (a) Add the value of the bin to a running sum.
   (b) If the sum is greater than N/2 for a data set of size N, stop the loop.
4. The current bin number is the median of the data set.

Since we only have to loop through the data set once, this algorithm will run in linear time. There is also a constant term which is dependent on how long it takes to reach the $N/2_{nd}$ element after the data is sorted, but this is determined by the properties of the distribution and not the size of the initial data set. In this case, there is no need to make a copy of the original data set, as the initial order is preserved. This means that the algorithm uses a constant amount of memory, the value of which is dependent on the range of the data set and the desired accuracy of results. More bins will yield a more accurate result (under the right conditions), but will take up more space in memory, and fewer bins will take up less memory but yield a less accurate result for the median. This algorithm also lends itself very easily to parallelism. Using enough parallel processors, the set could be sorted in near constant time, and the total computation time would only be limited by the need for a sequential summation after the creation of the histogram, which would take roughly N/2 time for normally distributed data.

4.3 Comparison

As previously stated, the time domain data used by our program is stored in 8-bit signed char format. As we are sorting hundreds of thousands of data points at once, and the values of these data points only contain 8 bits (256 discrete values at most), a histogram sort works well for our requirements in the time domain. In order to get frequency domain data, however, the time domain data must be Fourier-transformed and stored as 64-bit floating point data. The range of values increases from 2^8 to roughly 2^{1024}, and the use of a histogram sort would require sacrificing a significant amount of precision in order to keep the time and memory parameters the same. While it is clearly less computationally expensive to use a histogram sort than a radix sort on the time-domain data, this may not necessarily be the case for frequency domain data due to the increased range of the data points. Further study is needed, but if a good balance cannot be reached between the accuracy and speed of a histogram sort, it may be more economical to use an MSD radix sort in the frequency domain.

5 MAD Filtering Algorithm

Our real-time RFI filtering program is laid out as follows:

1. The GUPPI Raw data file is read into the program from memory (or directly from the telescope, in the final version).
2. The header information of the file is extracted.
3. For each set of data points, X, in the observation:
   (a) For each subset of X, x_i
      i. x_i is MAD filtered in the time domain.
      ii. A complex-to-complex FFT (Fast Fourier Transform) is performed on x_i to get frequency data, changing the data type from 8-bit integers to floating point numbers.
      iii. The resulting data is stored in a buffer as $f_1, f_2, ..., f_n$.
   (b) The average spectrum of subsets $f_1, f_2, ..., f_n$ is calculated.
   (c) The MAD algorithm is used to identify RFI in the frequency domain.
   (d) The bins which are identified as containing RFI are replaced with random Gaussian noise in spectra $f_1, f_2, ..., f_n$.
   (e) An IFFT (Inverse Fast Fourier Transform) is performed on spectra $f_1, f_2, ..., f_n$ to get $x_1, x_2, ..., x_n$, casting floating point numbers to 8-bit integers once again.
   (f) The window shifts by half its size in order to process a new subset of data.

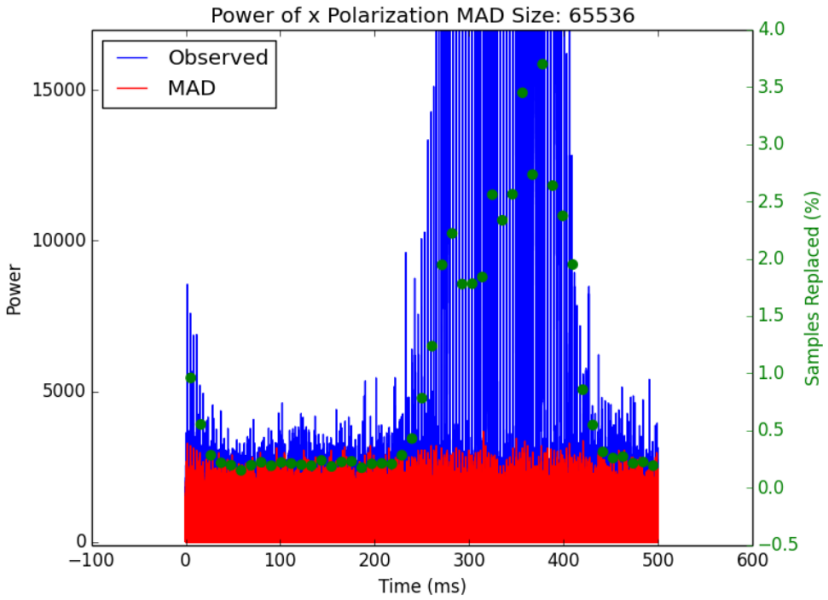


Figure 6: Power of the X polarization of the data set. Data before mitigation is shown in blue while mitigated data is shown in red. The percentage of data samples replaced per MAD calculation is shown on the right hand side in green. The spike of RFI is likely due to a radar pulse from a nearby airfield.

4. Once each subset has been processed, write the data to an output file.

In the frequency domain, normalization is used to ensure that power differentials across each frequency band are accounted for. The normalization strategy used in our program is described in detail in Buch et al. (2016)[2]. The results of this algorithm on a part of the J1713+0747 observation can be seen in figure 6.

5.1 Program controls

It is our intent that the final program will have variable settings for the size of the MAD algorithm (currently set at 2^{16} data points), the amount by which the MAD window shifts for each subset (currently set at half the window size), and the sensitivity of the MAD filter (currently set at 3σ). In addition, we will allow for different replacement strategies to deal with RFI. The methods of replacement under consideration are as follows:

1. **Replace RFI with data from a random Gaussian distribution.** This method is what we are currently using in our program. It is an approach that works well from a data-driven standpoint, but it is more computationally expensive than other methods.

2. **Replace RFI values with zero.** This method requires less time, and does not affect the Gaussian properties of the set, but it also runs the risk of introducing unwanted artifacts into the data.

3. **Replace RFI values with a user-defined value or NaN.** This method would also be computationally inexpensive, but has the same drawbacks as option 2, with the added caveat that it could skew the mean/median of the data set towards the given value. If the value set was NaN, it is unclear how that would be processed in the next stages of the pipeline.

4. **Flag RFI in a metadata file instead of removing it.** This method would not risk introducing artifacts into the data as the others do, but would require additional memory for another header file. It also would not change the fact that it is much more difficult to excise RFI after de-dispersion and folding has occurred on an observation than before it.

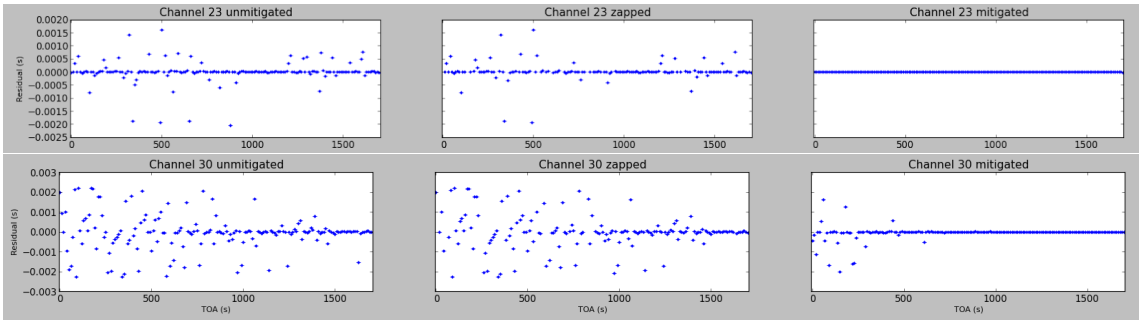


Figure 7: TOA residual comparison for Channel 23 (top) and Channel 30 (bottom), which are the channels that provide the best visual demonstration of our program's capabilities. Going left to right are the unmitigated TOAs, the zapped TOAs, and the MAD filtered TOAs, respectively. The mitigated data from these channels can be visually shown to have fewer outliers than both the unmitigated and the zapped files.

6 Analyzing Raw Data

There are several key processes which must occur in order to derive results from GUPPI raw data. Most of these processes usually happen in the telescope's backend, but, as we are purposefully dealing with raw data in this project, we must instead perform these steps offline. In order to gain usable results from a pulsar observation, we must first de-disperse it, which removes the effects of the interstellar medium, in addition to adjusting for other parameters, such as binary rotation, in order to get an accurate pulse profile. Since the pulse brightness is below the noise threshold of the data, many periods of the pulsar must be averaged in order to discern a noticeable difference in the signal. This means that, while the signal of the pulsar gets stronger after folding, the RFI also remains in the data, and it becomes much more difficult to distinguish interference from the shape of the pulse profile. Since the RFI zapping routine that is currently in use at the Green Bank observatory can only be used after the data has been processed, real-time mitigation is predicted to yield better results, as it acts on the data before these methods can affect the RFI.

After processing the data, we are left with several archive files (.ar extension) containing roughly one pulse profile every ten seconds. These archive files are then compared to the standard profile of the pulsar (.sm extension). This is a pulse profile that has been compiled over many years of observation of the pulsar, and it serves as a way to fit each pulse in our data to a characteristic shape. Once a match to the standard pulse profile is found, the time that the pulse arrived can then be extrapolated based on the time difference between the beginning of the observation and a specifically chosen point on the standard profile. This time is known as the pulse's Time of Arrival, or TOA. The expected TOAs of the pulsar are well known based on several years of study, so in order to examine the accuracy of our data, we take the difference of the TOAs obtained from our data and the expected TOAs to get the pulsar's timing residuals. It is not expected that our TOAs will be a perfect match for those that are expected, so instead we evaluate the success of our results based on the distribution of residuals. If our observation is accurate, the distribution should look more or less flat, with a mean close to zero and a low standard deviation.

7 Results

In order to evaluate the effectiveness of our program compared to existing RFI removal methods, we compare the TOAs from data mitigated using our program to those from data that has been RFI zapped using an offline mitigation method. The offline zapping method is a tool included in the DSPSR package, which is the same software we used to de-disperse and fold the pulsar data[7]. After de-dispersion and folding, TOAs and their residuals are extracted using the program tempo2[4], which compares the standard model of the pulsar to our observations. Figure 7 shows comparisons of channels 23 and 30 between mitigated, zapped, and unmitigated data. In analyzing the two, we find that the weighted standard deviation of the mitigated TOA residuals is roughly 30% lower than that of the zapped TOA residuals. This is by no means irrefutable proof that our program is more effective at mitigating RFI than other methods, but it is nonetheless a strong

indication that this is the case.

8 Conclusion

In order to combat the increasing presence of RFI, radio astronomers must begin looking at new ways of eliminating it from their data. This paper describes one such method, but many more can and should be developed (or improved upon) in order to increase the quality of radio observations around the world.

The mitigation program presented here is intended as a proof-of-concept for a relatively new method of RFI mitigation. While initial results are promising and encourage further development of the program, there is still a lot of work that must be done before it can be used in real-world applications. The program is currently written sequentially in Python and runs roughly 95x slower than the rate at which data is acquired. Our goal is to rewrite the code in CUDA C and parallelize it on a GPU in order to improve the runtime. The use of a Hybrid Task Graph Scheduler (HTGS) API[1] has also been discussed. If it can be made to run in real time, which we believe is feasible, the program is intended to eventually be adapted for and tested on the VEGAS backend of the Green Bank Telescope.

9 Acknowledgements

I would like to thank my thesis advisor, Jim Buckley, and my advisor in the computer science department, Roger Chamberlain, as well as my partner on this project, Nick Joslyn, my summer research advisor, Richard Prestage, and everyone else at the Green Bank Observatory who contributed to this work. Additional thanks to the National Science Foundation, Associated Universities, Inc., and the Green Bank Observatory for funding this research.

References

[1] C. Boeres, A. Lima, and V. E. F. Rebello. Hybrid task scheduling: integrating static and dynamic heuristics. In *Proceedings. 15th Symposium on Computer Architecture and High Performance Computing*, pages 199–206, Nov 2003.

[2] Kaushal D. Buch, Shruti Bhatporia, Yashwant Gupta, Swapnil Nalawade, Aditya Chowdhury, Kishor Naik, Kshitij Aggarwal, and B. Ajithkumar. Towards real-time impulsive rfi mitigation for radio telescopes. *Journal of Astronomical Instrumentation*, 05(04):1641018, 2016.

[3] J. M. Ford and K. D. Buch. Rfi mitigation techniques in radio astronomy. In *2014 IEEE Geoscience and Remote Sensing Symposium*, pages 231–234, July 2014.

[4] G. B. Hobbs, R. T. Edwards, and R. N. Manchester. tempo2, a new pulsar-timing package – i. an overview. *Monthly Notices of the Royal Astronomical Society*, 369(2):655–672, 2006.

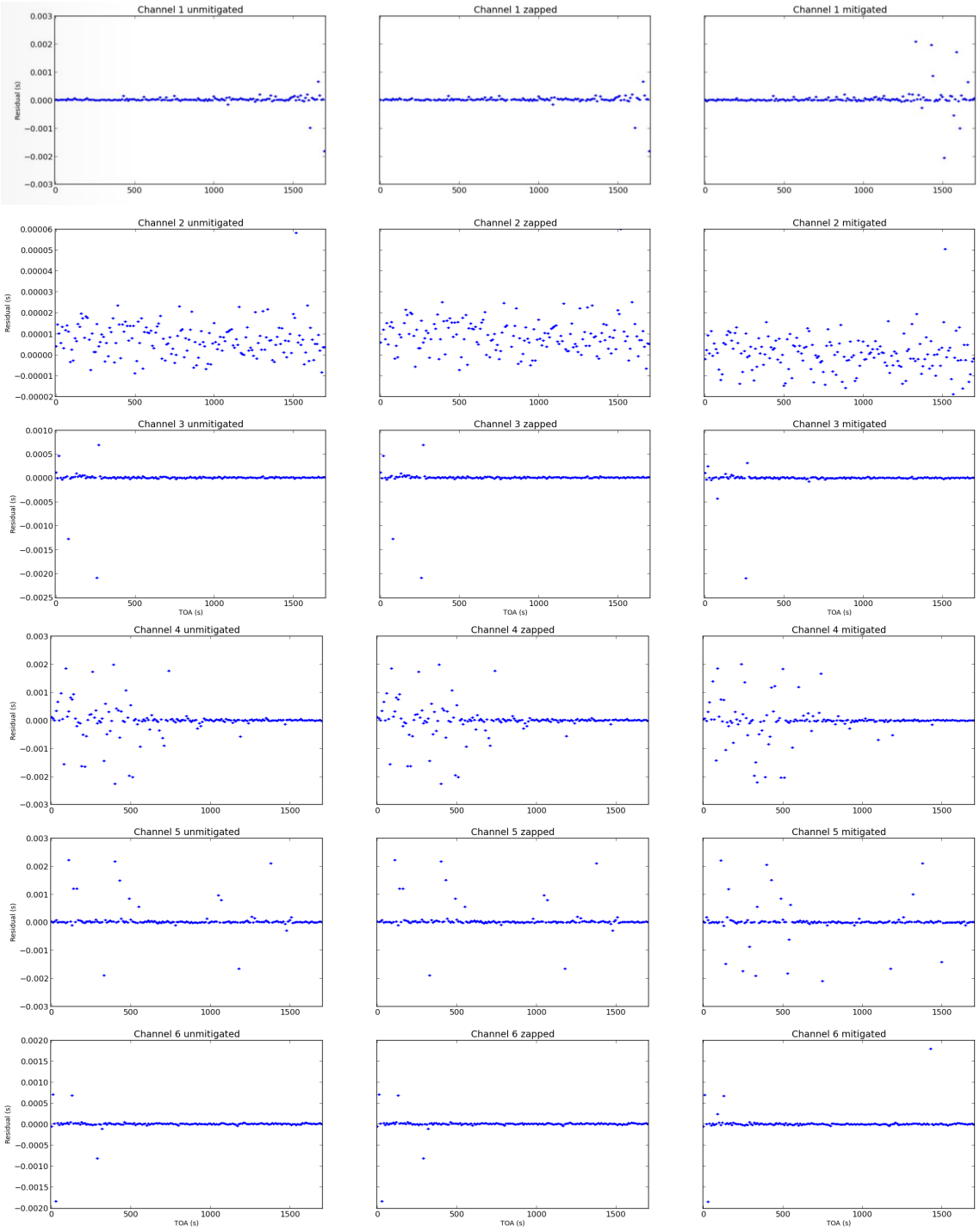
[5] D. R. Lorimer and M. Kramer. *Handbook of Pulsar Astronomy*. December 2004.

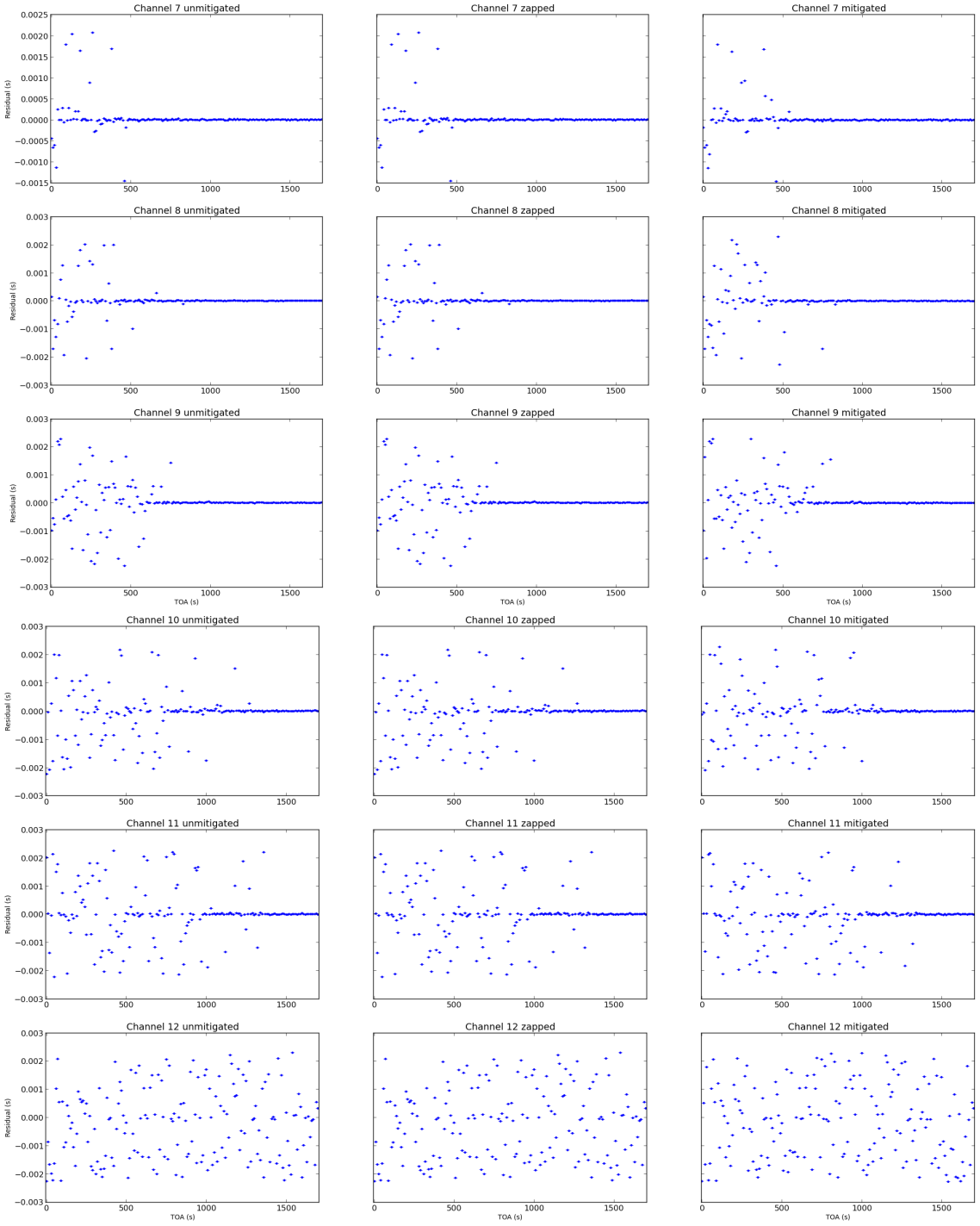
[6] National Radio Astronomy Observatory. Log of wuppi pulsar configurations and example data, 2010.

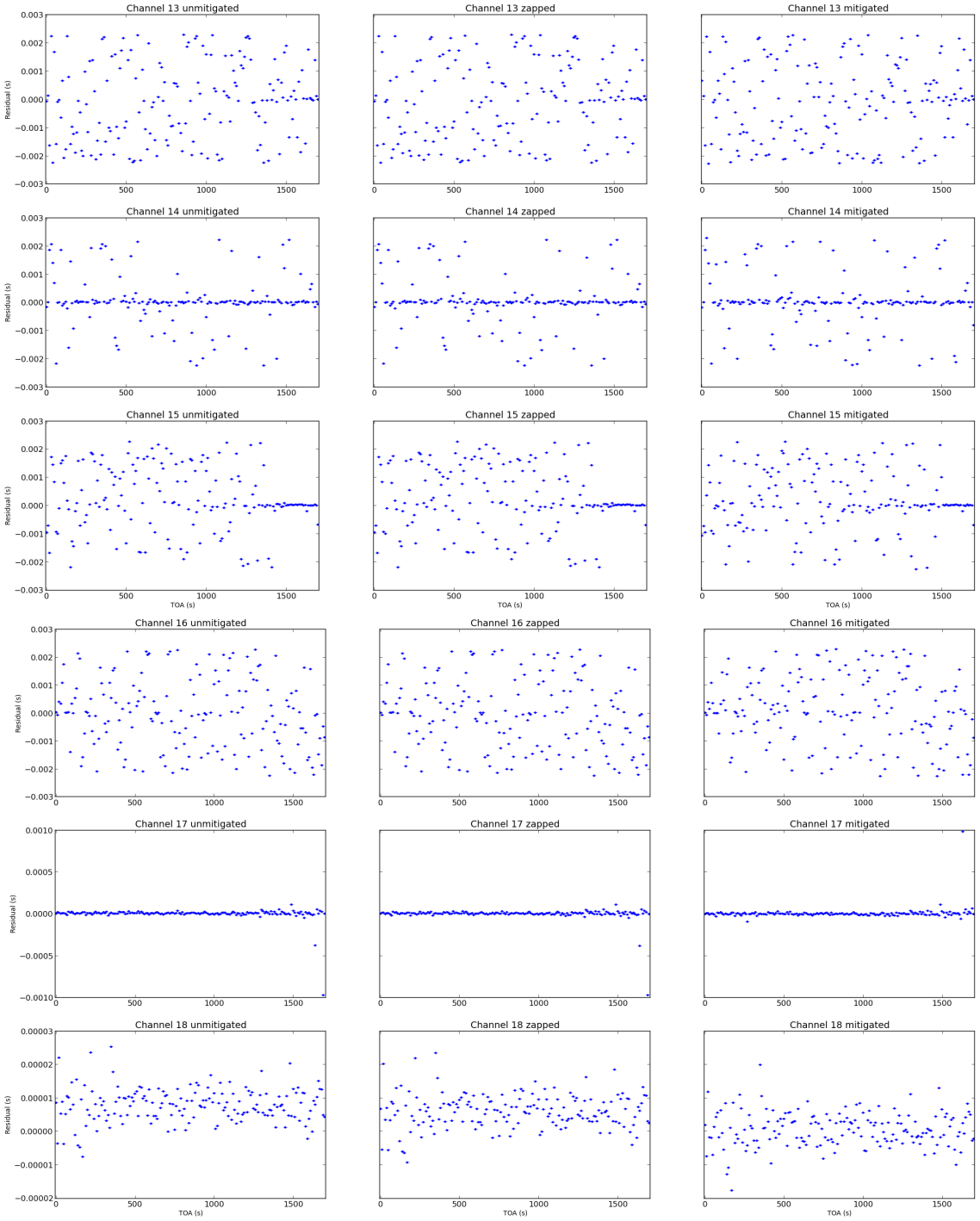
[7] W. van Straten and M. Bailes. Dspsr: Digital signal processing software for pulsar astronomy. *Publications of the Astronomical Society of Australia*, 28(1):1–14, 2011.

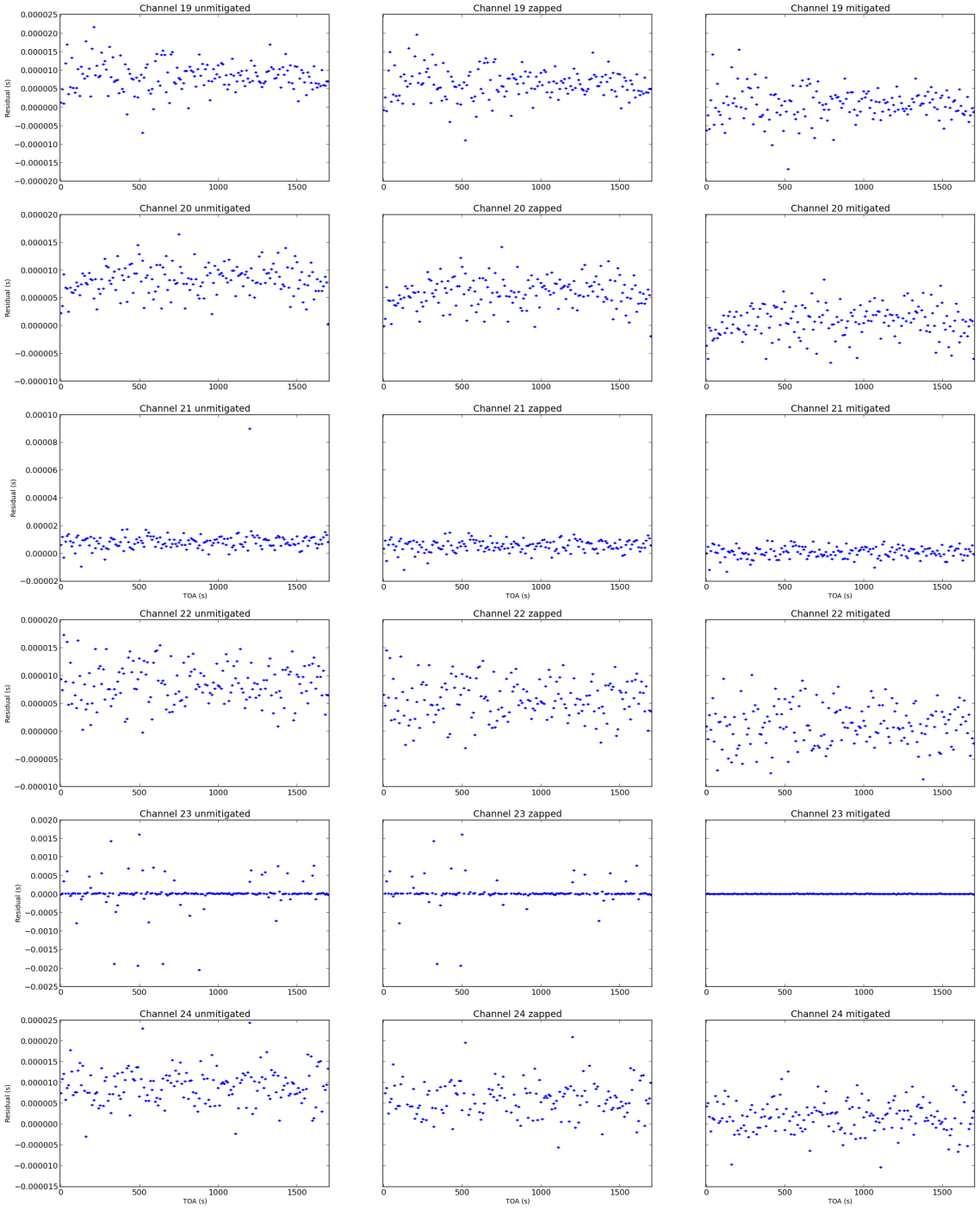
A Residuals for all channels

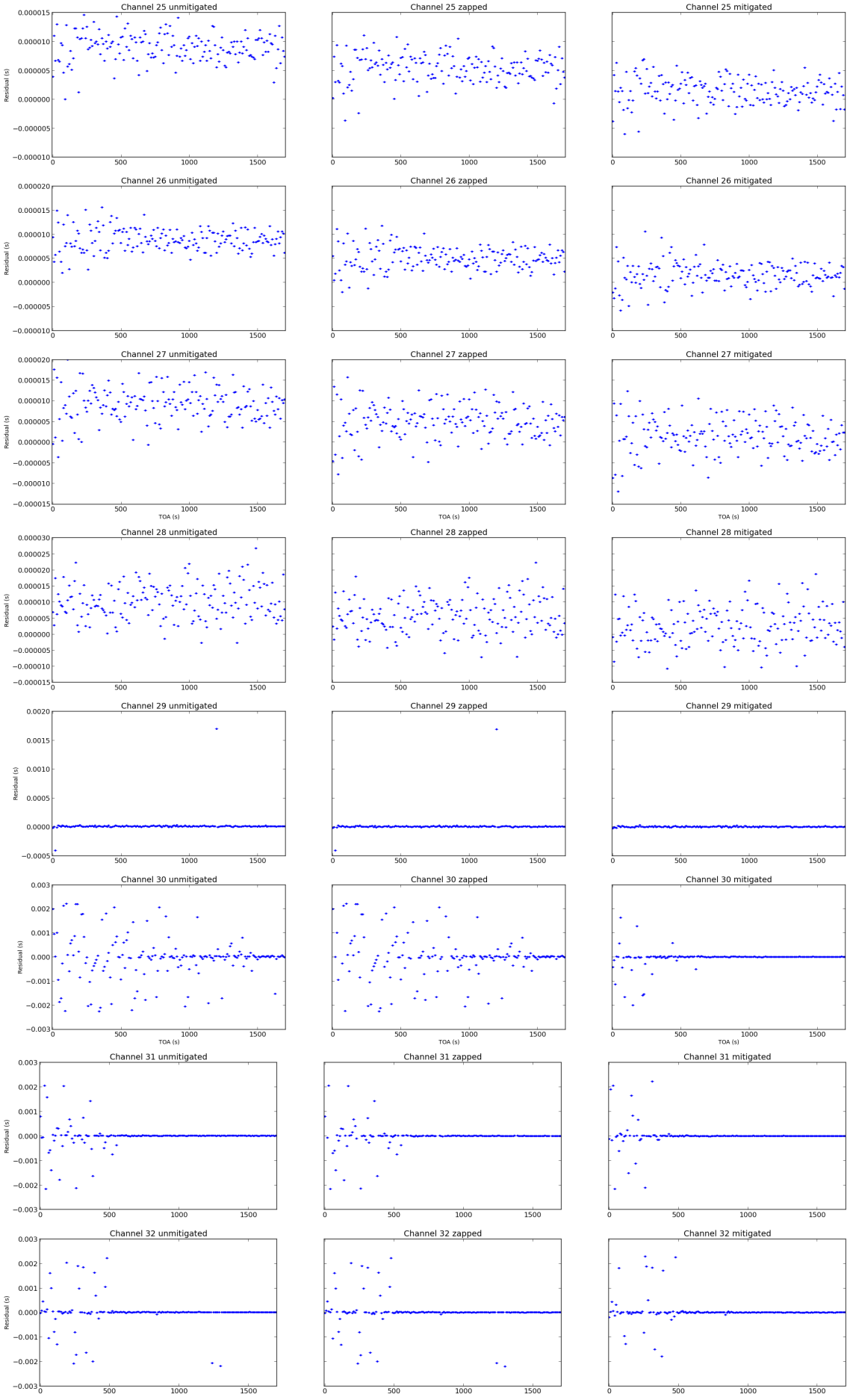
The plots shown below represent residuals from roughly 1700 seconds of observation. All 32 channels are shown, with comparisons between unmitigated, zapped, and mitigated data in the left, center, and right plots, respectively. The y-axes of all the plots are in microseconds and represent the residual values, while the x-axes are in seconds and represent the time during the observation at which each residual occurred.


Channel 1 unmitigated
Channel 1 zapped
Channel 1 mitigated
Channel 2 unmitigated
Channel 2 zapped
Channel 2 mitigated
Channel 3 unmitigated
Channel 3 zapped
Channel 3 mitigated
Channel 4 unmitigated
Channel 4 zapped
Channel 4 mitigated
Channel 5 unmitigated
Channel 5 zapped
Channel 5 mitigated
Channel 6 unmitigated
Channel 6 zapped
Channel 6 mitigated
Residual (s)
TOA (s)


Channel 7 unmitigated
Channel 7 zapped
Channel 7 mitigated
Channel 8 unmitigated
Channel 8 zapped
Channel 8 mitigated
Channel 9 unmitigated
Channel 9 zapped
Channel 9 mitigated
Channel 10 unmitigated
Channel 10 zapped
Channel 10 mitigated
Channel 11 unmitigated
Channel 11 zapped
Channel 11 mitigated
Channel 12 unmitigated
Channel 12 zapped
Channel 12 mitigated
Residual (s)
TOA (s)


Channel 13 unmitigated
Channel 13 zapped
Channel 13 mitigated
Channel 14 unmitigated
Channel 14 zapped
Channel 14 mitigated
Channel 15 unmitigated
Channel 15 zapped
Channel 15 mitigated
Channel 16 unmitigated
Channel 16 zapped
Channel 16 mitigated
Channel 17 unmitigated
Channel 17 zapped
Channel 17 mitigated
Channel 18 unmitigated
Channel 18 zapped
Channel 18 mitigated
Residual (s)
TOA (s)


Channel 19 unmitigated
Channel 19 zapped
Channel 19 mitigated
Channel 20 unmitigated
Channel 20 zapped
Channel 20 mitigated
Channel 21 unmitigated
Channel 21 zapped
Channel 21 mitigated
Channel 22 unmitigated
Channel 22 zapped
Channel 22 mitigated
Channel 23 unmitigated
Channel 23 zapped
Channel 23 mitigated
Channel 24 unmitigated
Channel 24 zapped
Channel 24 mitigated
Residual (s)
TOA (s)


Channel 25 unmitigated
Channel 25 zapped
Channel 25 mitigated
Channel 26 unmitigated
Channel 26 zapped
Channel 26 mitigated
Channel 27 unmitigated
Channel 27 zapped
Channel 27 mitigated
Channel 28 unmitigated
Channel 28 zapped
Channel 28 mitigated
Channel 29 unmitigated
Channel 29 zapped
Channel 29 mitigated
Channel 30 unmitigated
Channel 30 zapped
Channel 30 mitigated
Channel 31 unmitigated
Channel 31 zapped
Channel 31 mitigated
Channel 32 unmitigated
Channel 32 zapped
Channel 32 mitigated
Residual (s)
TOA (s)