**C++ Beamformer Library with RFI Mitigation  
Version 0.1.0 – GNU GPL 3.0**

Max-Planck Institute for Radio Astronomy  
Jan Wagner, 29. August 2011  
(DOCMENT UNDER WORK)  
  
Developed for FP7 ALBiUS WP6 D6.3.1, RFI Mitigation  
This work received funding under EC Contract #????

# General

This is a general purpose C++ library that implements beamforming and RFI mitigation in the sense of interferer suppression and signal recovery. Library functions can be applied to short time integrated cross-correlation data of multi-pixel receivers, focal plane or phased arrays and interferometers.

The library was developed primarily for radio astronomy applications under FP7 ALBiUS. It is publicly released in the hope that it may prove useful in other applications as well. Accelerated BLAS/LAPACK linear algebra routines are used for performance. Single-core throughput on a Xeon E5430 with 64 antenna element data, double precision and complex arithmetic ranges from 100 to 9000 channels per second, depending on the type or RFI mitigation and beamforming.

Contents

[General 1](#_Toc303100174)

[Introduction 2](#_Toc303100175)

[C++ Library Requirements and Compiling 2](#_Toc303100176)

[C++ Library Class Overview 3](#_Toc303100177)

[C++ Library and Multithreading 5](#_Toc303100178)

[C++ Library Performance 5](#_Toc303100179)

[Matlab Script Overview 7](#_Toc303100180)

[Requirements on the Antenna Array data 8](#_Toc303100181)

[Details on Subspace Methods 9](#_Toc303100182)

[Details on Adaptive Beamforming 15](#_Toc303100183)

[Details on Reference Signal Subtraction 15](#_Toc303100184)

[Future work 16](#_Toc303100185)

[References 17](#_Toc303100186)

# Introduction

Radio frequency interference in low frequency bands is a growing concern in radio astronomy. Research in Digital Signal Processing and Advanced Radio Communications has produced a wealth of interference mitigation methods that are widely used in military and commercial communication technology and are usually tailored for certain scenarios and radio environments.

Mitigation methods have found their way into radio astronomy, too. Thanks to advances in the performance of FPGAs and computers, digital instead of analog processing approaches have gained entrance into the early layers of radio astronomic data capture and data preprocessing. Here they improve the data quality at varying degrees of success.

For cases where signal recovery is not thought to be possible, there exist statistical methods (flaggers) that analyze time series of multi-channel data and identify parts affected by interference. Data of identified parts is then discarded during post-processing.

On the other hand, there exist also methods that aim to recover as much of the desired signal as possible. Promising methods in radio astronomy include real-time adaptive filtering, adaptive beamforming, spatial filtering, subtraction of actual or reconstructed interfering signals with the help of reference antennas, and filtering of complex visibilities amongst others.

This C++ library together with Matlab reference source code is intended for certain observation setups that provide the required additional information which allows recovery of the desired signal to a higher degree, while not harming the data in those bands that are free of interference.

General requirements and a software details as well as results are given below.

# C++ Library Requirements and Compiling

The C++ library has been tested to compile at least under **Linux** and **GCC 4.4.4** and GCC 4.5.1. You also need **autoconf, automake, libtool**.

For good performance, linear algebra operations use standard accelerated linear algebra. You need to install **Armadillo C++ Linear Algebra Library version 2.2.1 or later** (<http://arma.sourceforge.net>). Armadillo supports OS X, Windows and Linux and provides compile-time arithmetic expression optimizations. It also interfaces to any underlying library that provides standard BLAS/LAPACK interfaces. These may be ATLAS, Intel MKL or AMD ACML. Under Linux the Armadillo library requires **cmake, blas-devel, lapack-devel, atlas-devel, boost-devel**. You might want to install ATLAS packages that are specific for your system, e.g. **atlas-sse3** and **atlas-sse3-devel** instead of the generic atlas-devel. To install Armadillo under OS X or Windows please read the Armadillo web site.

Unpack the C++ beamformer source code package to some directory. You can compile in single instead of double precision by editing *./src/BeamformerTypeDefs.h* and defining *USE\_SINGLE\_PRECISION*.

To build and install the Beamformer library including example programs:  
$ aclocal ; autoconf ; autoheader ; automake –a  
$ ./configure --prefix=/usr/local  
$ make ; sudo make install

# C++ Library Class Overview

The library is written in C++ and documented with doxygen tags. All classes and member functions are documented both in the source code as well as the doxygen-generated PDF Reference Manual. These are useful for lower level details about the library.

Higher level details of the architecture and the algorithms are described in the current document. A brief summary of classes and what they do is found in Table 1.

The following types of processing are supported, with variations:

|  |
| --- |
| Beamforming |
| Create an *ArrayElements* object to describe antennas and their positions. Create one *Beams\_t* structure with electrical pointing angles of all desired beams. |
| For each new multi-channel *Covariance* data loaded from file or memory, use a *BeamformerWeights* object to compute new element weights using classic beamformer, MVDR or Cox RB-MVDR. |
| Computed weights can be loaded into e.g. an external GPU beamformer. If raw input data that formed covariances was buffered, weights can be applied to their own data. This reduces error. |

|  |
| --- |
| Nulling without RFI reference antennas |
| Load *Covariance* data, pass to a *Decomposition*, run RFI detection and nulling using a *DecompositionModifier*. This modifies data in the Decomposition object. The recompose() methods allow to generate a final cleaned Covariance. |
| Clean covariances can be useful as the input into external UV plane imaging software. |

|  |
| --- |
| RFI Templating with reference antennas |
| Load *Covariance* data, pass it to a *CovarianceModifier* to run RFI Template generation and subtraction. |
| Clean covariances can be useful as the input into external UV plane imaging software. |

Example source code is in the *./examples* directory. The *analysis* program is mainly intended for debugging and comparing data to Matlab. The *benchmark* program executes different processing steps on a single CPU core and reports the performance.

Table - Overview of library classes

|  |  |
| --- | --- |
| Class | Description |
| *ArrayElements* | Use this class to describe your focal plane array or antenna array. The class stores information on element positions (X,Y,Z) and the properties of each element (LCP, RCP polarizations) and its dedicated use (astronomy signals, or RFI reference antenna for RFI signals). Can pre-generate positions for uniform linear and uniform grid array layouts.  Required by:  *Beamformer* (RB-MVDR weight calculation)  *BeamformerWeights* (conversion of beam angles into steering vectors)  *CovarianceModifier* (RFI templating and subtraction) |
| typedef *Beams\_t* (BeamformerData.h) | Used to list the desired electrical beam pointing angle(s). Also the output storage of computed (or “manually” edited) steering vectors and beamformer weights matching these input beam pointing angles.  Required by:  *BeamformerWeights* (conversion of beam angles into steering vectors)  *BeamformerWeights* (weight calculation, joint with Covariance input) |
| *BeamformerWeights* | Two use cases. First, helps to convert *Beams\_t* electrical beam angles into steering vectors.  Second, provides functions to convert steering vectors and covariance matrices or their decompositions into beamformer weights (CBF, MVDR, Cox WNGC MVDR, other methods). Weights are stored back into *Beams\_t*. |
| *Covariance* | Stores time-integrated covariance matrix data. Data can be single or multi-channel. It can be cross-correlation data (in which case it needs to be frequency domain) or covariance data (in which case it needs to be time-domain with contributing signals X having expectation value E<X>=0).  Required by:  *CovarianceModifier* (changes to covariance data) *Decomposition* (decompositions or recompositions of covariance data) |
| *CovarianceModifier* | Applies non-toxic RFI mitigation algorithms to a Covariance object. Currently it implements two types of RFI Template subtraction. See van der Veen [VE04] and Briggs [BRI00].  Requires that: 1) covariance data was observed with Nref ≥1 reference antennas  2) must have Nref ≥ NRFI/channel, otherwise system underdetermined  3) RFI ≥10dB stronger in reference antennas; mean of RFI autocorrelations 10 larger than times mean of other autocorrelations. |
| *DecompositionAnalyzer* | Extracts features from a covariance Decomposition. Currently returns number of RFI signals in a channel, estimated from eigenvalues with MDL or AIC information criteria or 3-sigma thresholding. (Direction of arrival DOA estimation with MUSIC in C++ is TODO.) |
| *Decomposition* | Base class for decomposing a 2D covariance matrix or a 3D multi-channel Covariance object. Can also generate a new Covariance from (possibly modified) decomposition data.  Child classes: *SVDecomposition, EVDecomposition, QRDecomposition*  Required by:  *DecompositionAnalyzer* (feature extraction, number of RFI, RFI DOA) *DecompositionModifier* (nulling) |
| *DecompositionModifier* | Applies automated changes to a Decomposition object. Currently editing steps are RFI interferer estimation and nulling (subspace method). |

# C++ Library and Multithreading

Multi-threading is not directly implemented in this C++ library. Basic time-division parallelism is of course possible, if you handle Covariances of different short time integration intervals on different CPU cores.

However, data channel-division parallelism is also possible. All data processing in the library is memory-in-place. Channels are processed one at a time. Those library functions that have high arithmetic cost, such as covariance data decomposition, can be invoked for just a sub-range of frequency channels.

You can thus use Parallel For on for example *CovarianceDecomposition::decompose()* and loop it over non-overlapping channel ranges, to utilize all CPU cores.

Parallel For can be found for example in the Intel Thread Building Blocks (*parallel\_for*), OS X Grand Central Dispatch (*dispatch\_apply*), OpenMP (*#pragma omp parallel for*), or *Boost.Thread* parallel for.

# C++ Library Performance

The individual RFI mitigation and beamforming functions were tested in a sequence typical for normal usage in a real-time or off-line astronomic signal processing pipeline. There library source code comes together with a program called *benchmark* under the *examples*. This program was run on a single core of an Intel E5430 2.66 GHz CPU. The double-precision performance is in Table 2 below.

Table – Throughput of full processing using double precision complex arithmetic, 64 phased array elements, synthetic and APERTIF 71-channel 64x64-size covariance data in memory, running on 1 core of a dual-processor Intel Xeon E5430 system (12MB L2, 2.66 GHz, quad core).

|  |  |
| --- | --- |
| # Armadillo with ATLAS, Beamformer compiled ‘-g –O3 -Wall’ for double precision (default)  numactl –physcpubind=0 ./benchmark | |
| Integrate 64-elem vector into Covariance | 80300 channels/sec (better use FPGA or GPU!) |
| Decomposition -> recomposition (average) | 230 channels/sec |
| SVD -> RFI detect -> null -> recomposition | 150 channels/sec |
| EVD -> RFI detect -> null -> recomposition | 230 channels/sec |
| 1-RFI/ch, 2-reference Template subtraction | 5420 channels/sec |
| 2-RFI/ch, 2-reference Template subtraction | 9100 channels/sec |
| 64-beam classical beamformer | 3600 channels/sec |
| 64-beam MVDR (Cox b=1.0) | 290 channels/sec |
| 64-beam RB-MVDR (Cox b=1.0+1e-4) | 290 channels/sec |

Computation can be spread across available CPU cores using a ParallelFor loop. Performance of the processing steps will scale linearly. It is also possible to recompile the C++ library to use 32-bit single precision floating point in all vector and matrix arithmetic. The single precision performance is shown in Table 3 below.

Table - Throughput of full processing using single precision complex arithmetic, 64 phased array elements, synthetic and APERTIF 71-channel 64x64-size covariance data in memory, running on 1 core of a dual-processor Intel Xeon E5430 system (12MB L2, 2.66 GHz, quad core).

|  |  |
| --- | --- |
| # Armadillo with ATLAS, Beamformer compiled ‘-g –O3 –Wall -DUSE\_SINGLE\_PRECISION=1’  numactl –physcpubind=0 ./benchmark | |
| Integrate 64-elem vector into Covariance | 156000 channels/sec (better use FPGA or GPU!) |
| Decomposition -> recomposition (average) | 270 channels/sec |
| SVD -> RFI detect -> null -> recomposition | 190 channels/sec |
| EVD -> RFI detect -> null -> recomposition | 260 channels/sec |
| 1-RFI/ch, 2-reference Template subtraction | 8700 channels/sec |
| 2-RFI/ch, 2-reference Template subtraction | 21900 channels/sec |
| 64-beam classical beamformer | 5850 channels/sec |
| 64-beam MVDR (Cox b=1.0) | 390 channels/sec |
| 64-beam RB-MVDR (Cox b=1.0+1e-4) | 360 channels/sec |

# Matlab Script Overview

The source code package includes MathWorks Matlab scripts in addition to C++ source code. The Matlab scripts are essentially the reference for the numerical parts of the C++ library. The scripts are included to help you test new algorithms visually.

|  |  |
| --- | --- |
| Matlab file(s) | Function |
| subspcrfi | Test program that calls some of the functions below |
| subspcrfi\_A | Generates array steering matrix in one direction for all channels. |
| subspcrfi\_AICrank subspcrfi\_MDLrank | Estimate the number of independent components from a list of matrix eigenvalues; estimates the rank of the original covariance matrix. See information theory text books or Wax-Kailath [WK85]. |
| subspcrfi\_MVDR | Beamformer weights from covariance data and a set of beams. Classical, MVDR and RB-MVDR Cox Projection beamforming. For Cox see [CZO87]. |
| subspcrfi\_RtoUV | Basic UV gridding. Converts covariance matrix and antenna element positions into UV plane matrix. |
| subspcrfi\_SNR | Beamformer weights from Ronsource – Roffsource calibration covariances, Maximum SNR weights into steering direction, or conjugate field match. |
| subspcrfi\_doa\_MUSIC | Attempts RFI 3D DOA estimation from decomposed covariance, antenna element positions using the MUSIC algorithm. |
| subspcrfi\_elemXYZ | Generates antenna positions (x,y,z) in uniform grid array in (x,y) plane. |
| subspcrfi\_getEV | Eigenvalue decomposition of covariance matrices for all channels. |
| subspcrfi\_getNoises | Estimates antenna noise from covariance matrices. Uses estimator described for example in Ippoliti [IPP05]. |
| subspcrfi\_loadRxxFile | Read multi-channel complex covariance data from a file that the C++ Beamformer library Covariance::store() function can generate. |
| subspcrfi\_modelgen | Synthetic covariance data generator. Single-channel, takes a spatial array layout, noise powers, list of signals (their powers and angles) and outputs covariance. Signals are assumed to be orthogonal and multipathing delays longer than integration time. |
| subspcrfi\_modelgen2 | Synthetic covariance data generator that uses RFI reference antennas. Identical to subspcrfi\_modelgen, but specified antennas see RFI signals at higher gain and celestial sources at low to zero gain. Signals are assumed to be orthogonal and multipathing delays longer than integration time. |
| subspcrfi\_nulling | Interferer nulling. Takes EVD decomposed multi-channel covariance data, estimates interferers, replaces dominant eigenvalues with mean of noise-space eigenvalues. Assembles “cleaned” output covariance. Uses standard methods, for gentle introduction see Briggs-Kocz [BK05]. |
| subspcrfi\_plotArrayResponse | Beamformer weights are converted into an array response over -90..90deg phi/theta angles. The array radiation pattern is plotted in 3D. |
| subspcrfi\_plotEVspec | Plot multi-channel eigenvalue spectrum. Overlays N most dominant eigenvalues into the same plot. |
| subspcrfi\_steer | Similar to subspcrfi\_A but computes steering for only one frequency. |
| subspcrfi\_subtraction | Reference antenna method. Corrects array covariance data by subtracting RFI signal contributions, estimated by covariance between reference antennas and array elements. Uses methods from van der Veen [VE04] and Briggs [BRI00]. |
| subspcrfi\_test\_subtraction | Test program for the subtraction methods. |
| subspcrfi\_writeRxxFile | Write multi-channel complex valued covariance data into a file that the C++ Beamformer library can import. |

# Requirements on the Antenna Array data

Below are the (reasonable) requirements the input data must be meet for proper operation of the RFI mitigation and analysis algorithms. The Matlab sources are quite flexible. The C++ library however expects certain additional Covariance matrix layout constraints to work efficiently.

Here are the points to consider while forming the Covariance input and earlier, while planning the technical aspects of an astronomic observation:

1. “Covariances” between array elements should be formed and time-integrated over short time intervals (STI: 1ms<Tint<10ms). The choice of Tint is a balance between increased noise at very short Tint on one hand, and less effective RFI mitigation at the other due to multipathing and RFI variability at very long Tint.
2. Time-integrated covariances should best be formed on FPGA (fixed-point) or GPU (single precision floating point) for CPU speed and I/O limit reasons.
3. If “Covariances” are cross-correlations, the matrices must contain frequency domain data.
4. If “Covariances” are time-domain covariances, the matrices must contain time domain. Signals of all contributing antennas need to have zero mean (E[X] = 0).
5. RFI reference antennas may be used. In this case, full covariances between reference antennas and array antennas need to be formed. While giving indices out to all the antennas, the reference antennas should have the lowest indices. That is, reference antenna data should be in the top left corner of the covariance matrices.
6. Channelizing antenna signals into a large number of narrow frequency channels may be desirable from an RFI perspective, if narrower channels reduce the likelihood that any given channel will contain more than just one RFI source.

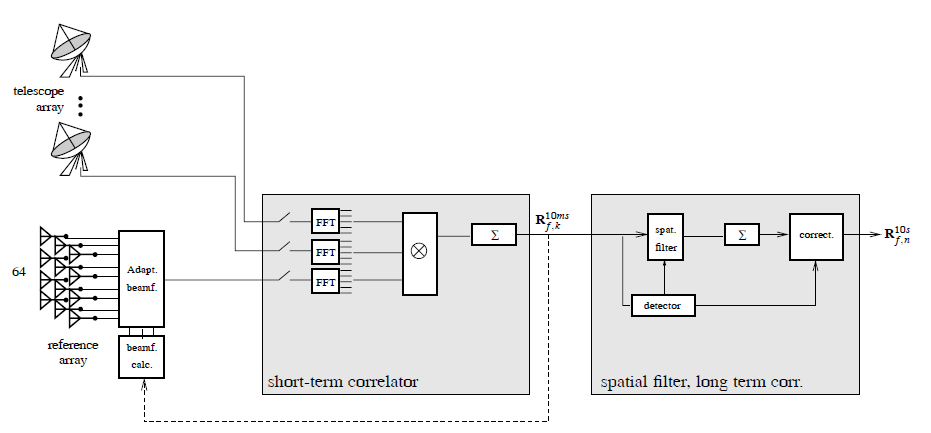


Figure – Suggested processing configuration. Figure is re-used from van der Veen et al. [VE04].  
Mid block: All signal sources are channelized (FFT), channel covariances (X) are short-term integrated (Σ) into matrices Rf,k. Right block: matrices may be processed in the C++ library (RFI reference subtraction, spatial filtering e.g. Nulling) before long term integration, or C++ library can update RFI-nulling beam weights (beamf. calc). Left: signal sources and RFI reference signals.

An example telescope configuration can be seen in Figure 1 by van der Veen et al. They used the phased array as an RFI reference “antenna”, with beams steered towards RFI signals. The telescope array

Data processing in Figure 1 consists of cross-correlation matrices (“covariances”) from several FFT channels (subbands) being formed in FX correlator style. Cross-correlations are time integrated over some short time interval (STI). These STI covariance estimates can be further processed with for example this C++ Beamformer library, outputting new beamformer weights or modified covariance data that went through RFI mitigation steps. The cleaned STI covariance estimates are then further time-integrated according to observer wishes.

# Details on Subspace Methods

The C++ library and Matlab code provide decompositions of the Hermitian array covariance matrix estimate () and has methods for interferer nulling. Nulling is based on partitioning of the eigenspaces of the matrix decomposition into interferer and noise subspaces. Below is a very condensed summary of the method. We start with a receiver array that has Nant elements and a single narrow-band frequency channel. The time snapshot of all Nant real-valued or complex-valued signals from the array is combined into a signal vector,

(1)

The true array covariance or cross-correlation matrix is estimated by which forms the average of signal cross-correlations over a short time range that consists of M signal vector snapshots,

(2)

where x\* denotes the complex conjugate transpose. is a Hermitian conjugate symmetric (Nant x Nant) matrix and is non-singular when enough snapshots are integrated (M > Nant). Covariance data passed to the C++ library must be data integrated by M > Nant. This may be problematic in pulsar observations at very short timescales with a large number of antennas.

For short integration times, with no RFI present, and with a celestial source signal power less than the array element noise power , the (t) estimate is close to the cross-correlation of an independent identically distributed (i.i.d.) random variables process that has Nant variables. In this case is the cross-correlation of white noise and has full rank and is well-conditioned.

To introduce RFI interferers, we first make the reasonable assumption that the astronomic, interferer and array noise signals are mutually orthogonal and not correlated during the M-snapshot averaging time (assumed to be shorter than any significant multipathing effects of the interferer signal). Now can be expressed as the linear sum of individual contributions,

(3)

where is the diagonal (Nant x Nant) matrix of true auto-correlated noise power estimated by . Note that covariances and cannot be reliably and separately estimated a priori, thus we can’t simply subtract to yield an RFI-free version of . However, when the number of RFI interferers *q* is less than the total number of antenna signals (*q* < Nant), the interferer covariance estimate becomes rank deficient (rank is *q*) and is ill-conditioned. An eigendecomposition of will have only *q* non-zero eigenvalues, the remaining Nant – q eigenvalues are zero. This can be used for RFI mitigation.

To begin “nulling” the RFI-contaminated Hermitian covariance matrix estimate , it is first transformed to its singular value decomposition (SVD) or its eigenvalue decomposition (EVD):

; with properties and S, diagonal (4)

The square matrices S and contain the eigenvalues of on their diagonal. The values are usually sorted along the diagonal in non-increasing order with the largest value in the top left (0,0) of the matrix. Matrices U and V contain SVD left-hand and right-hand eigenvectors, while EVD eigenvectors are in the matrix W.

The SVD and EVD decompositions are very closely related. Below we treat only nulling using the EVD decomposition for simplicity. As a practical note, the EVD has numerical problems for ill-conditioned Ĉxx. The SVD may be preferable over EVD as it tends to be more stable and accurate when Ĉxx is ill-conditioned.

The eigenvalue decomposing of in (3) is, by linearity,

(5)

with q-interferer submatrix sized and noise . Note all eigenvalues in are non-negative, and real-valued for Hermitian .

Applying the earlier assumption that astronomic source noise power is much lower than antenna noise ( and that each antenna sees an interferer power then the diagonal submatrices (from interferer space) and (from noise space) with eigenvalues of are

with interferer noise powers and

with only antenna noise powers (6)

To get an RFI-free estimate from the decomposition of the entire interferer eigenvalue submatrix could be set to (“Nulling”). However, this ignores the contribution in (6) and biases . In practice a better approach is to “null” using an estimate of the noise space eigenvalues that are in and fill the diagonal of using either

or (7)

Using the median is more robust. The same method above can be applied in the SVD decomposition, too, to create a “nulled” singular value matrix to get an RFI-free estimate .

The number of largest eigenvalues or singular values to null, the number of interferers , can be estimated using Minimum Descriptor Length (Rissanen 1978) or other information criteria applied to a list of eigenvalues. For log likelihood we use the ratio of geometric and arithmetic means.

(8)

(9)

The estimated by MDL is reliable when the interferer to noise ratio (INR) is large. The eigenvalue list then has clear outliers. Other options to classify eigenvalues to RFI interferer space are 1) find values that exceed three standard deviations from the mean or median, 2) traverse the sorted eigenvalue list from the smallest to largest value and find sufficiently large change in for example a moving average, 3) use heuristic knee point detection methods such as “Kneedle”. An example of the typical shape of eigenvalue plots without RFI and with two RFI interferers are shown in Figure 2 and Figure 3, respectively.

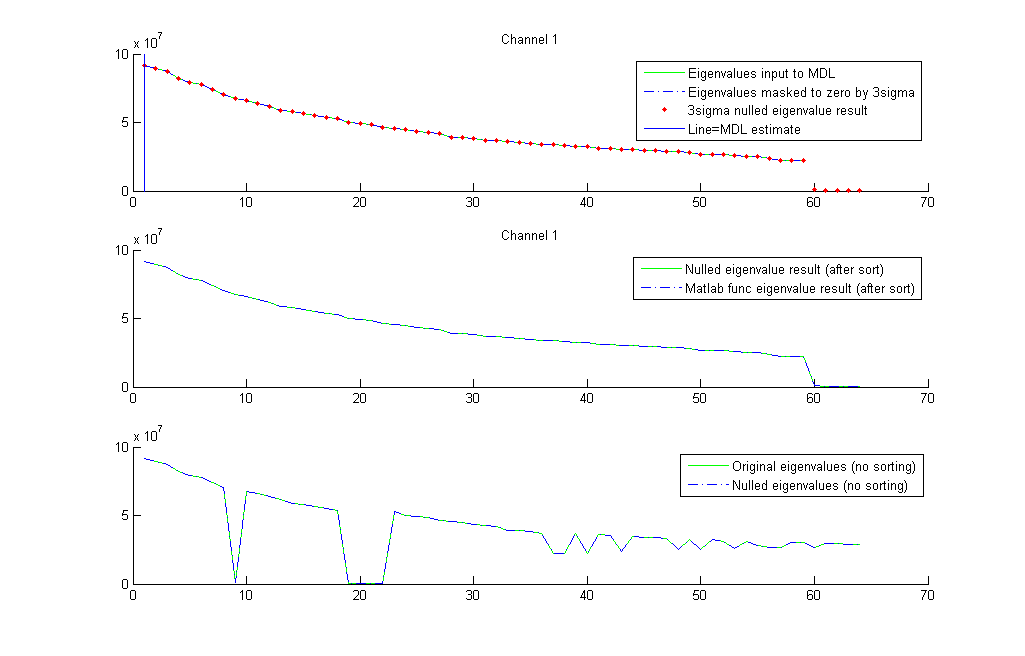


Figure - Eigenvalues of APERTIF covariance data for Virgo A, channel 3 of 71, no RFI present. Array elements 1-59 connected, 60-64 disconnected, resulting in 5 near zero values. These are removed from the sorted list passed to MDL detection (solid green). Points left of vertical line are MDL-detected RFI. Three-sigma thresholding (dashed blue) finds no outlier eigenvalues. The final output after median replacement is identical to the input and has replaced no eigenvalues (red dots).

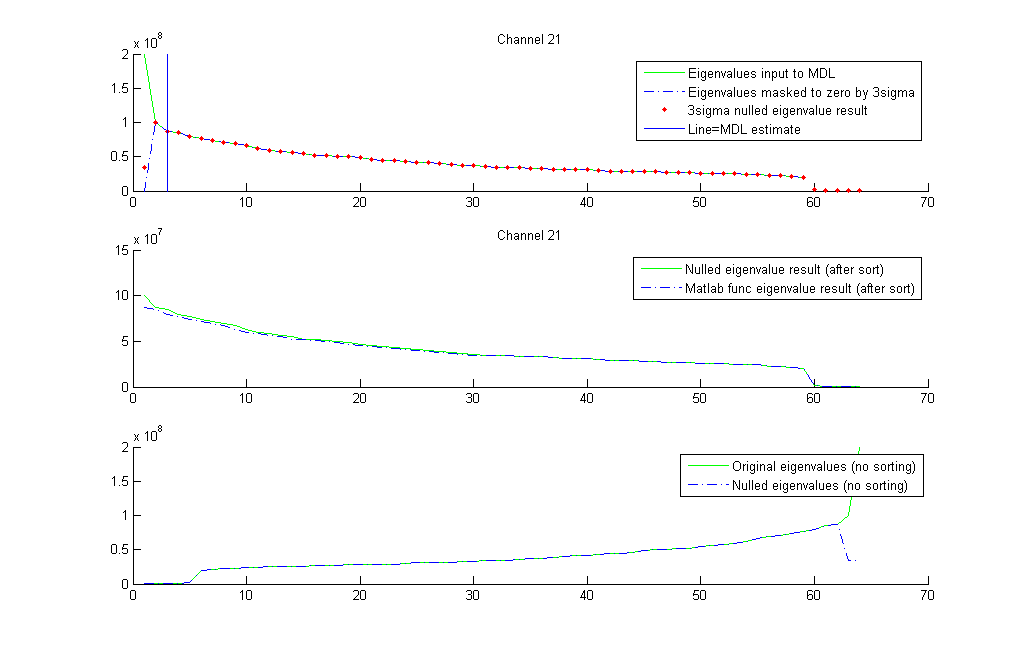


Figure - Eigenvalues of APERTIF covariance data for Virgo A, channel 18 of 71, one RFI source. Input to MDL detection (solid green) shows strong knee point at 2nd eigenvalue. Points left of vertical line are MDL-detected RFI. Three-sigma thresholding (dashed blue) flags the 1st largest eigenvalue. Median replacement assigns median of non-flagged eigenvalues to all flagged eigenvalues, here only 1st flagged. Final result of processed eigenvalues (red dots) is later used to reconstruct a clean covariance matrix for the frequency channel.

After estimating q, nulling and reconstructing a clean covariance matrix, the time series of nulled covariance matrices can be fed into long-term time integration for imaging and spectral line detection or into pulsar data dedispersion processing.

Strongly detected pulsars may need special care in the processing. Pulsars can contribute dominant eigenvalues to the EVD or SVD decomposition and they may be identified as RFI and get erased.

To demonstrate the algorithm, a result of nulling applied with Matlab and also the C++ Beamformer library to data from APERTIF is shown in Figure 4.

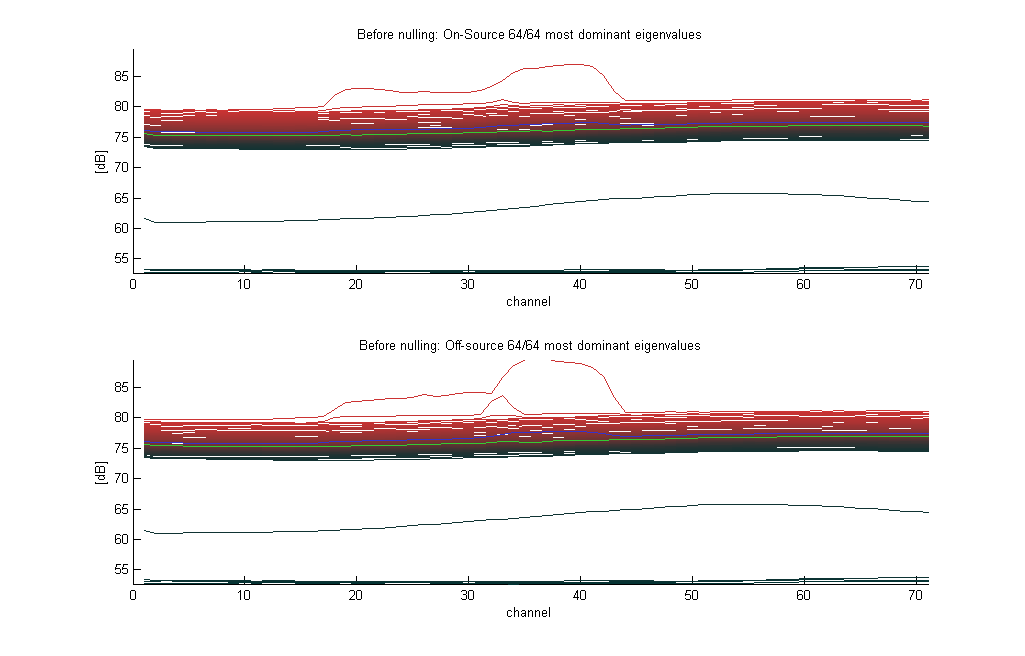


Figure – Eigenspectrum of Virgo A covariances. Derived from 64-element 71-channel raw APERTIF covariance data (Wim van Cappellen). AfriStar digital satellite radio RFI around center channels. This EuroStar-family satellite signal adheres to ETSI EN 302 550-1-3. Horizontal axis: frequency channels 1 to 71 (1.4830-1.4967 GHz). Vertical axis: overlay of all 64 eigenvalue powers, colored red-to-black from largest to smallest eigenvalue. Green curve: median. Blue curve: mean.

The number of interferers q that should be nulled via median eigenvalue replacement is estimated with MDL and with a 3-sigma threshold. The figure shows one interferer detected in the central channels of the 1.4830 - 1.4967 GHz band. It is completely nulled in the processing output show in Figure 5.

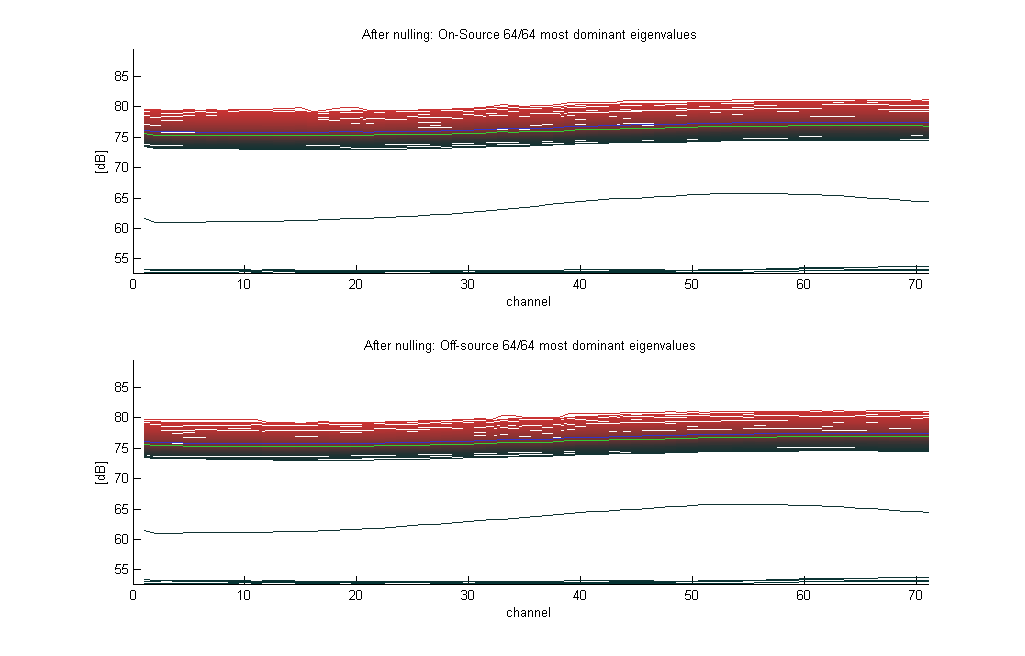


Figure - Eigenspectrum of nulled Virgo A covariances. Original 71x64x64 matrices first SVD or EVD decomposed. Interferer number per channel auto-estimated with MDL and limited by 3-sigma thresholding. Dominant eigenvalues replaced by median of noise space eigenvalues (“nulled”). Nulled matrix decompositions are reconstructed, resulting in clean covariance matrices. Cleaned covariance matrices EVD-decomposed a second time to get eigenspectrum for this figure. Vertical axis: overlay of all 64 eigenvalue powers, colored red-to-black from largest to smallest eigenvalue. Green curve: median. Blue curve: mean.

The effect of MDL + 3sigma RFI nulling on the autocorrelations is shown in Figure 6. On-source and off-source data were available. Their difference was used to remove standing wave effects of the WSRT dish. Autocorrelations of all 64 elements are plotted along 71 channels. Nulling is apparently highly effective in reducing the level of RFI in all elements. Nulling also prominently brings out Virgo A continuum seen by element array 31. There is a clear processing artifact, however.

The nulled off-source data has zeroes in element 11 near low channels. This causes two spikes after subtraction from on-source data. Clearly nulling is not always 100% effective and a last step of data flagging will sometimes be necessary.

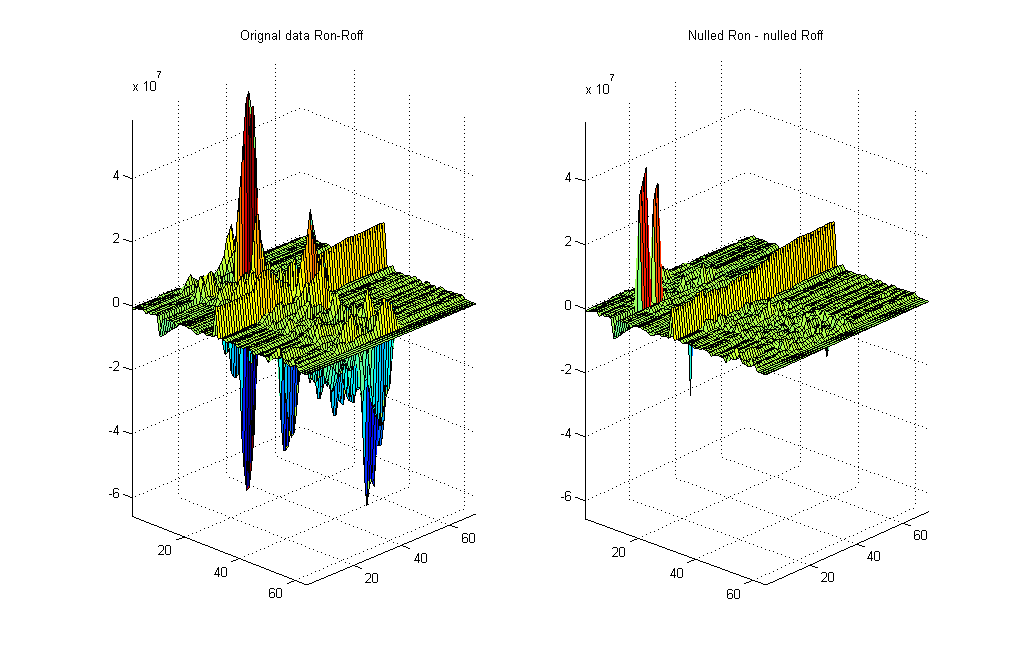


Figure – Autocorrelations from APERTIF Virgo A covariances. Autocorrelations of 64 elements plotted over 71 channels. Left: ON-source minus OFF-source of original data. Right: same for data nulled using MDL and 3sigma thresholding.

The simple Matlab UV gridder was run on the APERTIF data, gridding the difference of nulled ON-source and nulled OFF-source covariance matrices, excluding autocorrelations. A final crude image was constructed by summing the Fourier-transformed UV data sets of all 71 APERTIF data channels. This is shown together with the original covariance data imaged in the same fashion. In autocorrelation data, Virgo A is on an off-center element (31). APERTIF element beams are non-overlapping and only one element sees Virgo A, hence it can not be imaged. The nulled data in the right panel of Figure 7 reveals a point source that is also present in the original data but was masked by RFI in channels 20-45. The point

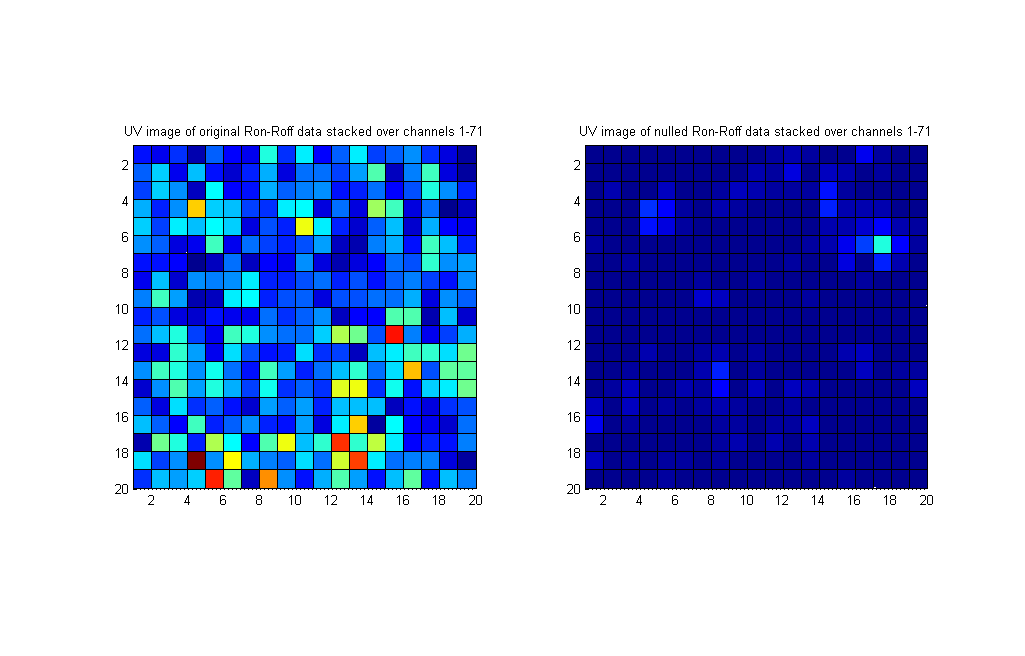


Figure – Imaged covariances after UV gridding, 2D FFT transform and stacking channels 1 to 71. Image FOV approximately ±1 rad. Left: ON-source minus OFF-source difference imaged from original data. Right: same for nulled data. Nulling reveals point source at right image edge. This is likely wide-band RFI entering APERTIF from around the WSRT dish edge. Main dish subtends ±0.96 rad, WSRT has f/D=0.35, D=25m. The point source is seen only in the difference, not in separately imaged ON-source or OFF-source data.

The result of nulling for synthetic data is shown in three figures. Figure 10 shows the raw RFI-contaminated covariance matrix of a single-channel 64-element array that sees two point source interferers and one very weak astronomic point source. A simple UV gridder that you can find in the Matlab source code of C++ Beamformer library was used to make an image from this covariance data. An image of the RFI-contaminated data is shown in Figure 8. EVD-based nulling was used to estimate the cleaned covariance which after UV gridding and Fourier transform is shown in Figure 9. The two interferers have been completely nulled while the astronomic point source (seen together with array interference pattern) has been retained at same magnitude as in a completely RFI-free model.

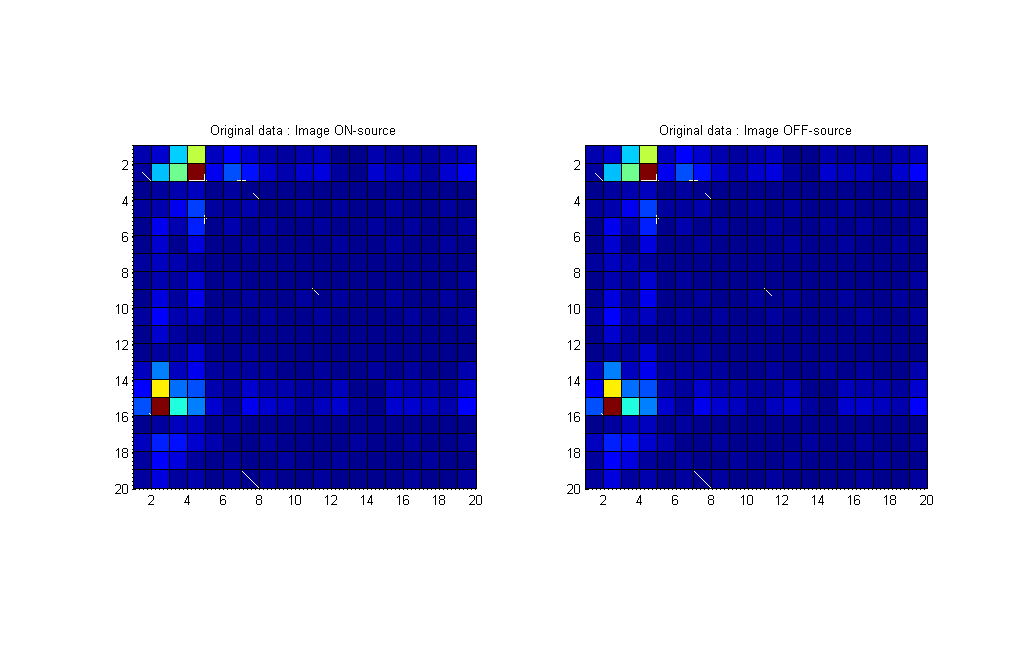


Figure – Image of UV plane of synthetic data. Array is a 64-antenna uniform grid array.   
Element covariances contain two strong RFI point sources and one weak point source (INR=103).

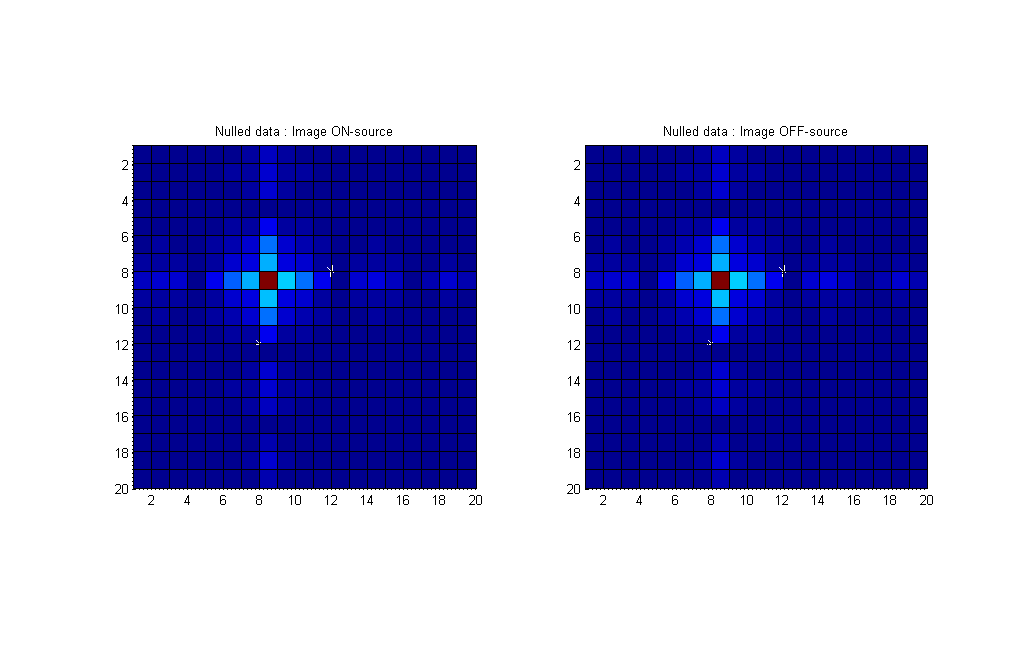


Figure – Image of UV plane, after nulling has been applied to the underlying   
covariance data of . Both RFI sources have been fully mitigated.

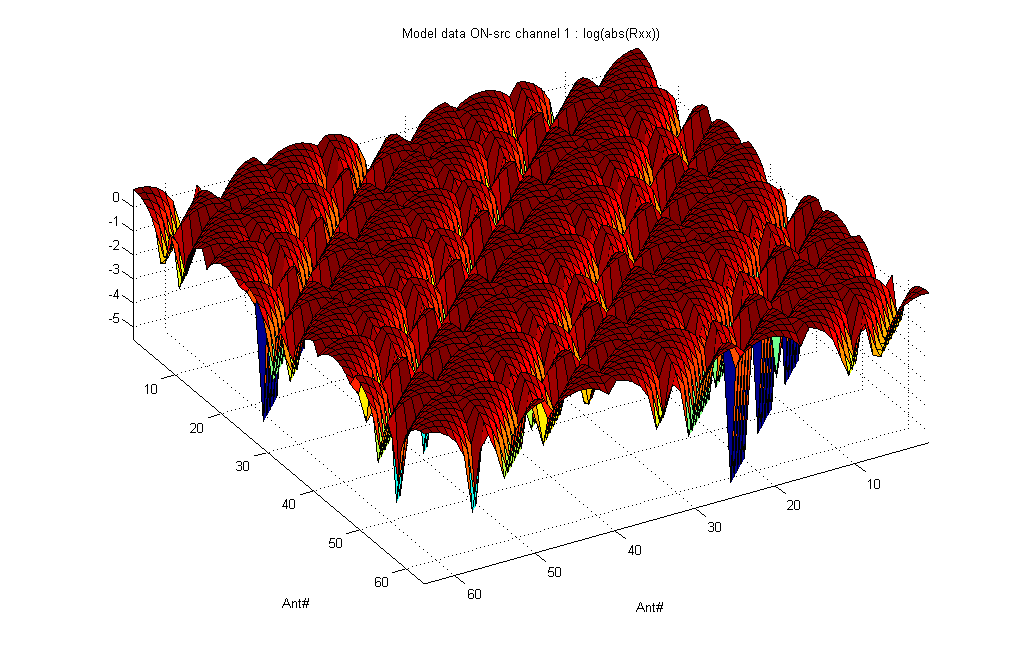


Figure – Magnitude plot of synthetic data from 64x64–element covariance matrix. Array is a 64-element uniform grid array organized as 8\*8. Element covariances contain two strong RFI point sources and one weak point source (INR=103). Periodicity stems from the square 8\*8 grid layout being plotted linearly.

# Details on Adaptive Beamforming

CBF, MVDR, RB-MVDR

# Details on Reference Signal Subtraction

It is possible to remove certain RFI as a post-correlation step. A standard method employs one or more low-gain local reference antennas that do not see the observed celestial object. The general case is described in van der Veen et al. (2004), while a special case with one interferer per channel and two reference antennas is described in Briggs et al. (2000).

|  |  |
| --- | --- |
| C:\Users\jwagner\Dropbox\apertif\showcase\1RFI_model2_contaminated.png  Figure – Image of UV plane of synthetic data. Phased array is 8\*8 (64-element uniform grid) and 2 reference antennas are located at diagonal corners of array with 103 extra gain towards RFI. Final covariance matrix size is 66x66. Signals: 1 strong RFI, 1 weak sky point source. | C:\Users\jwagner\Dropbox\apertif\showcase\1RFI_model2_nulled.png  Figure – Image of UV plane after RFI Nulling. Intended only for comparison with RFI Templating result. Covariance matrix underlying was reduced to 64x64 (exclusion of RFI reference antennas), decomposed, nulled, reconstructed and UV-gridded. |
| C:\Users\jwagner\Dropbox\apertif\showcase\1RFI_model2_subtracted.png  Figure – Image of UV plane after RFI Templating. Covariance underlying was cleaned by Briggs’ and Kesteven’s method (special case of 1 RFI per channel, 2 reference antennas). Similar clean outcome with 101 … 103 relative gains towards RFI. Result comparable with Nulling result of . | C:\Users\jwagner\Dropbox\apertif\showcase\1RFI_model2_subtracted_Rxx_delta_INR_1e1.png  C:\Users\jwagner\Dropbox\apertif\showcase\1RFI_model2_subtracted_Rxx_delta.png  Figure – Abs difference between cleaned and clean model UV images (covariance of model with 1 RFI and Template subtracted, vs. covariance from model without RFI). Top: relative RFI gain 101, vertical range 0 to 9·10-11. Bottom: relative RFI gain 103, vertical range 0 to 13·10-15. |

# Future work

It may be of interest to consider the Jeffs and Warnick [JW09] findings on spectral bias (“spectral scooping”) for narrowband interference. Spectral bias is caused by beamformer weights that are calculated from a covariance matrix that is not the exact covariance matrix but an estimate. Spectral bias may be of interest mainly for PSD estimation and correction.

A further possibility is to integrate functions of this library into AIPS or CASA. For CASA it would be reasonably straight forward. With AIPS not so, one would need to pointers to instances of the involved C++ library classes and make extern “C” wrapper functions (*extern “C” void create\_​CovarianceModifier(​void\*\* classPtr, [CovarianceModifier c’stor arguments])*) to handle the details of Fortran calling C++.

# References

**[BK05]** F. H. Briggs, J. Kocz; Overview of Technical Approaches to RFI Mitigation, e-print arXiv:astro-ph/0502310, 2005, <http://arxiv.org/pdf/astro-ph/0502310>   
**[BRI00]** F. H. Briggs et al.; *Removing radio interference from contaminated astronomical spectra using an independent reference signal and closure relations*, The Astronomical Journal Vol 120 No 6, 2000, [http://​arxiv.​org​/abs/astro-ph/0006222v2](http://arxiv.org/abs/astro-ph/0006222v2)  
**[CZO87]** H. Cox, R. Zeskind, M. Owen; *Robust adaptive beamforming*, IEEE Trans ASSP, Vol ASSP-35, 1365-1377, 1987, <http://ieeexplore.ieee.org/iel6/29/26204/01165054.pdf>  **[IPP05]** Ippoliti, Romagnoli, Fontanella; *A noise estimation method for corrupted data*, Statistical Methods and Applications, Vol 14, 343-356, 2005, [http://www.springerlink.com/​content/​74x042n77330115t/](http://www.springerlink.com/content/74x042n77330115t/) **[JW09]** Jeffs, Warnick; *Spectral Bias in Adaptive Beamforming With Narrowband Interference*, IEEE Trans SP, Vol 57, No 4, April 2009, <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04731746>  
**[VE04]** Van der Veen et al.; Spatial filtering of RF interference in Radio Astronomy using a reference antenna, Proc. IEEE ICASSP, pp. II. 189-193, 2004, [*http://*ieeexplore.ieee.org/​iel5/​9248/​29344/​01326226.pdf](http://ieeexplore.ieee.org/iel5/9248/29344/01326226.pdf) **[WK85]** M. Wax, T. Kailath; *Detection of Signals by Information Theoretic Criteria*, IEEE Trans ASSP, Vol ASSP-33, No 2, April, 1985, [*http://*ieeexplore.ieee.org/iel6/29/26190/01164557.pdf](http://ieeexplore.ieee.org/iel6/29/26190/01164557.pdf)