M2TSI - SpaceMaster

UE31 Laboratory Report

Spectral Analysis and Sparse Representation

Authors:
Arthur Scharf
Andreas Wenzel

January 24, 2017

1 Introduction

In this report the spectral analysis of irregularly sampled data - in particular line spectra - is performed using the Fourier Transform. Also, the sparse representation of signals is evaluated by applying different classical methods such as "greedy" algorithms and "convex relation" approaches. These methods are applied in particular to irregular sampled data, since this is the most realistic representation of acquired data in astronomy.

2 Spectral analysis with Fourier Transform

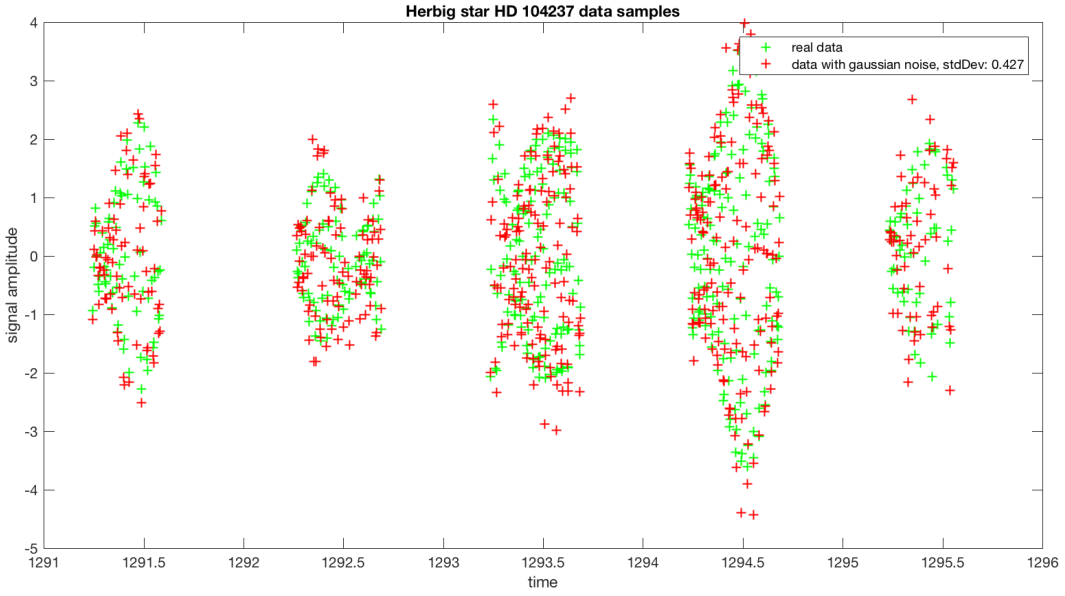

Figure 1: Original data samples and samples with additional noise in time domain

Since working on a real data set might be difficult, in particular when it comes to understanding the underlying techniques of the methods aforementioned, we will simulate our irregular sampled data, for which an initial dataset was given (c.f appendix A). Using this amplitude, phase, time, frequency and the appropriate radial velocity data, the following formula was used to create a realistic data set that is basically a noisy sum of sine functions

$$x(t_n) = \sum_{k=1}^{K} A_k \sin(2\pi\nu_k t_n + \phi_k) + \epsilon_n \tag{1}$$

As for the Signal-to-Noise ratio 20dB in power mean were used, and for the number of sine functions `K = 5`, as our initial data set contained 5 different amplitudes and its according phases

and frequencies (c.f. appendix A). Figure 1 shows the generated data, where the green part corresponds to our simulated data without noise (hereafter seen as "real" or "original" data) and the red part including noise. The noisy data set will be used from here on in all upcoming calculations.

2.1 Irregular sampling case

In the case of irregular sampled data we can not apply the Fast-Fourier-Transform (FFT) as we would have in the regular case. However, the Fourier-Transform can be computed by introducing a Matrix $\boldsymbol{W}$, such as

$$\boldsymbol{W}(l,c) = \exp(2j\pi t_l f_c) \quad , \quad \hat{\boldsymbol{x}} = \boldsymbol{W}^{\dagger}\boldsymbol{x} \tag{2}$$

with $\hat{\boldsymbol{x}}$ being the Fourier-Transform of vector $\boldsymbol{x}$. As stated above, the FFT can not be used here, since the Matrix $\boldsymbol{W}$ must be orthogonal for the FFT - which it is not in the case of irregular sampled data.
Nevertheless, having this tool handy, we can now perform the Fourier Transform on irregular sampled data, e.g. on our data set represented in fig. 1.

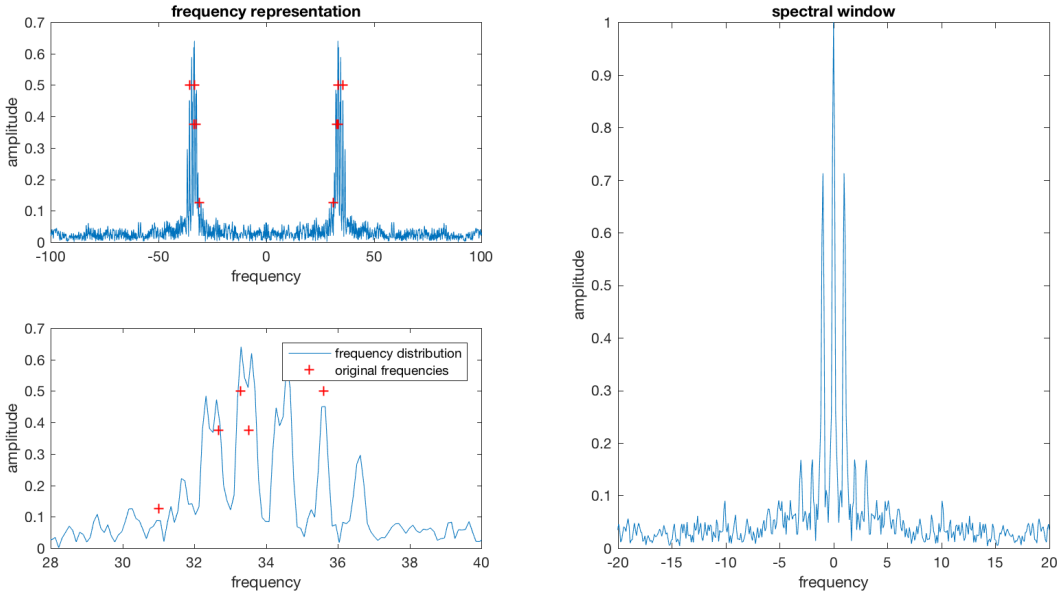

Figure 2: Frequency representation with original frequencies and spectral window

For the beginning we only use one sine-function (in eq. (1), thus `K=1`). In this case it is very easy to determine the frequency and amplitude of the underlying sine-signal in the frequency representation of data, even if there are quite high side lobes. The phase, which essentially is an offset of the signal, can not be directly read out from the amplitude vs. frequency diagram,

but can be calculated from the Fourier-transformed data. The phase will also be affected by the noise intentionally introduced in the data set.
However, if a noisy sum of 5 sine signals is used, the underlying frequencies and amplitudes can not be easily determined anymore. This is due to the fact that the sum of signals and their respective noise adds up, so that some peaks might be indistinguishable from others or side lobes. This can be seen in fig. 2 on the left side, where the red cross-hair markings correspond to the original frequency and amplitude, and the blue waveform to the frequency representation of the sum of signals. It is easy to see that the original frequencies and amplitudes can not be read out - only a rough estimation on where the frequencies are located might be is possible by taking into account the location of the highest peaks.

The spectral window corresponding to the sampling time is shown in fig. 2 on the right. The spectral window as represented in the figure shows the Fourier Transform of a rectangular window (basically consisting of δ-functions). The main peak corresponds to the main frequency peak in the frequency domain of the signal - which can be useful in case of truncating specific frequencies or weighting the signal, since the spectral window (in this case similar to a sinc function) can cut off frequencies above the cutoff frequency when convoluted with the Fourier Transformed signal.

The appropriate MATLAB code used to create the initial data set and calculate the frequency representations can be found in appendix B.

3 Sparse representation with greedy algorithms

The framework of sparse representations and greedy algorithms have been invented in order to perform spectral analysis of irregularly sampled data. The idea of such algorithms is to use iterative techniques for signal analysis, i.e. computing the frequency, which maximizes the frequency representation, followed by the subtraction of the related signal from the data and a further frequency analysis based on the resulting data called residual. In this second part, we investigate different example algorithms for analysing irregular samples, analyse their performance and compare them. These algorithms are applied to the data set $\boldsymbol{x}$ of 5 signals created in the previous part.

3.1 Pre-whitening or Matching Pursuit (MP) algorithm

The general structure of the Matching Pursuit (MP) algorithm is shown in the Pseudo Code below taken from the laboratory paper. After initialization, it searches for the index k, for which the frequency representation calculated via $\boldsymbol{\omega}_k^\dagger \boldsymbol{r}_n$ is maximized. This can be understood as finding the highest peak in the frequency representation of the signal. Note that in this case $\boldsymbol{\omega}_k$ is a column of the spectral window matrix $\boldsymbol{W}$ corresponding to the index k. Related to this, we save the chosen indices by listing them in a matrix Γ_n. The final step in the algorithm

then consists of updating the amplitude of the signal corresponding to the frequency related to the index k, and finally calculating the residual signal by subtracting this detected signal from the data. The resulting data is then used for the following iteration of that algorithm.

Initialisation $k = 0$, $\boldsymbol{r}_0 = \boldsymbol{x}$, $\Gamma_0 = \emptyset$, $\boldsymbol{a} = \boldsymbol{0}$
Iterations $n = 1 \ldots$

a) Select the atom with index $\boldsymbol{k} = \arg\max_k |\boldsymbol{\omega}_k^\dagger \boldsymbol{r}_n|$
Update the list of indices $\boldsymbol{\Gamma}_n = \boldsymbol{\Gamma}_{n-1} \cup \{k\}$

b) Update the amplitude $a_k = a_k + \frac{1}{\boldsymbol{\omega}_k^\dagger \boldsymbol{\omega}_k} \boldsymbol{\omega}_k^\dagger \boldsymbol{r}_n$
and the residuals $\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \frac{1}{\boldsymbol{\omega}_k^\dagger \boldsymbol{\omega}_k} \boldsymbol{\omega}_k^\dagger \boldsymbol{r}_n \boldsymbol{\omega}_k$

One has to choose a stopping rule for that algorithm, as well. We can assume that after a certain amount of iteration steps, the residual $\boldsymbol{r}$ only consists of Gaussian noise, since most of the real signal was detected and reduced from the data at that point. Therefore, the remaining data in $\boldsymbol{r}$ consists of centered Gaussian noise with variance σ^2, which is known. The variable defined by $T = \frac{||\boldsymbol{r}||^2}{\sigma^2}$ has a χ^2 distribution with N degrees of freedom. We stop the algorithm as soon as $T < \tau$ is given for the case that the probability $P(T < \tau) = 0.95$.

Our implementation of that algorithm in MATLAB is presented in the code in the appendix. The resulting diagrams are shown in figure 3. The upper panel shows the frequency representation of the analysed signal, plotting the amplitude for the detected frequency, while the lower panel is the inverse fourier transfor of the upper panel, thus showing the actual result of the analysis in time space. In addition, the upper panel displays the comparison between the analysed frequencies and the actual amplitudes and frequencies within the actual used signal data.

Already in this plot one can observe that this matching point algorithm detects 8 frequencies, although the original signal contained only 5. In addition, the location of those frequencies is detected with small deviations only for the signals with the highest amplitudes. For the other frequencies, the location is not detected correctly. There are also deviations from the original signals concerning the amplitudes of the signals. Table 1 lists the results of the signal analysis using the MP algorithm together with the specification of the original input signal. Note that in the plots and the table we only consider the absolute values of the frequencies and also ignore the false detections in the table, which is only used to illustrate the precision of the algorithm with respect to the original data.

Despite the actual performance regarding results, one also has to consider the performance of the algorithm during the signal analysis. An indicator for this is the number of iteration steps before the algorithm converges and stops, which is in the case of this matching pursuit algorithm implementation 38. During those 38 iteration steps, several indices k reappaeared and were analysed again, which is one main drawback of that method.

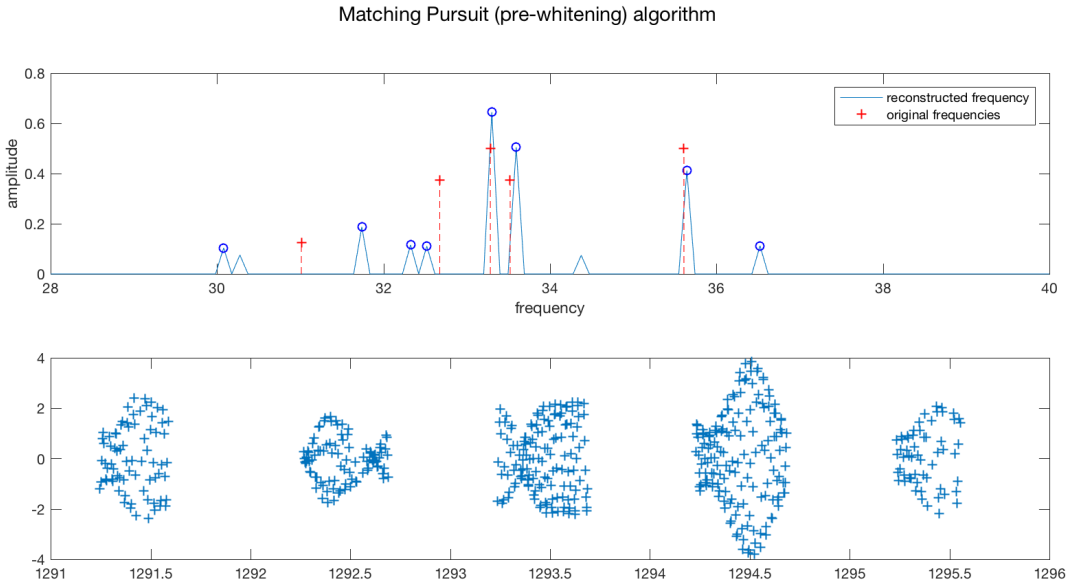

Figure 3: The upper panel shows a comparison between the analysis result using MP-algorithm and the actual signal data in frequency representation, the lower panel displays the signal resulting from the MP algorithm in time domain

		Signal 1	Signal 2	Signal 3	Signal 4	Signal 5
Input signal	Frequency [day^{-1}]	31.012	32.675	33.283	33.521	35.609
	Amplitude	0.125	0.375	0.5	0.375	0.5
MP output	Frequency [day^{-1}]	31.543	32.812	33.301	33.594	35.645
	Amplitude	0.144	0.209	0.563	0.486	0.391

Table 1: Parameter comparison ignoring the additional signals found by the MP algorithm

3.2 Orthogonal Matching Pursuit (OMP) algorithm

The drawback of repeating the analysis of several atoms in the matching pursuit algorithm implemented in the previous step can be overcome by using the orthogonal matching pursuit (OMP) algorithm, which updates the amplitudes of all atoms corresponding to indices k. Therefore, an orthogonal projection of the data onto the atoms is required, given by the following relation:

$$\boldsymbol{a}\left(\Gamma_n\right) = \underset{\boldsymbol{a}(\Gamma_n)}{\arg\min} ||\boldsymbol{x} - \boldsymbol{W}_{\Gamma_n}\boldsymbol{a}\left(\Gamma_n\right)||^2 = \left(\boldsymbol{W}_{\boldsymbol{\Gamma}_n}^{\dagger}\boldsymbol{W}_{\boldsymbol{\Gamma}_n}\right)^{-1}\boldsymbol{W}_{\boldsymbol{\Gamma}_n}^{\dagger}\boldsymbol{x} \tag{3}$$

In this equation, $\boldsymbol{a}\left(\Gamma_n\right)$ are the components of $\boldsymbol{a}$ with index Γ_n, while the matrix $\boldsymbol{W}_{\Gamma_n}$ is a matrix that is extracted from the original matrix $\boldsymbol{W}$ and contains the columns of that matrix with indices Γ_n. Using that formalism, the amplitudes are saved or updated for all indices Γ_n used before or detected in the current step. Thus, the step b) of the matching pursuit algorithm

has to be changed accordingly by replacing the calculation of individual amplitudes α_k with the calculation of a whole vector $\boldsymbol{\alpha}(\Gamma_n)$ via equation 3, as well as replace the calculation of the residuals, which can now be performed using $\boldsymbol{r}_n = \boldsymbol{x} - \boldsymbol{W}_{\Gamma_n}\boldsymbol{a}(\Gamma_n)$.
It is important to understad the difference in the calculation of the residuals compared to the previous matching algorithm, because it is now sufficient to subtract the current analysed signal in each step, which gets updated at each step by keeping track of the already analysed and detected indices Γ_n, from the complete original data set $\boldsymbol{x}$. The final result of an example run of the OMP is presented in figure 4, again comparing the original data and the analysed signal in frequency domain in the upper panel and the result of the analysis algorithm in time space in the lower panel. In addition, the extracted data set is quantitatively compared to the original signals in table 2 ignoring additionally detected signals.

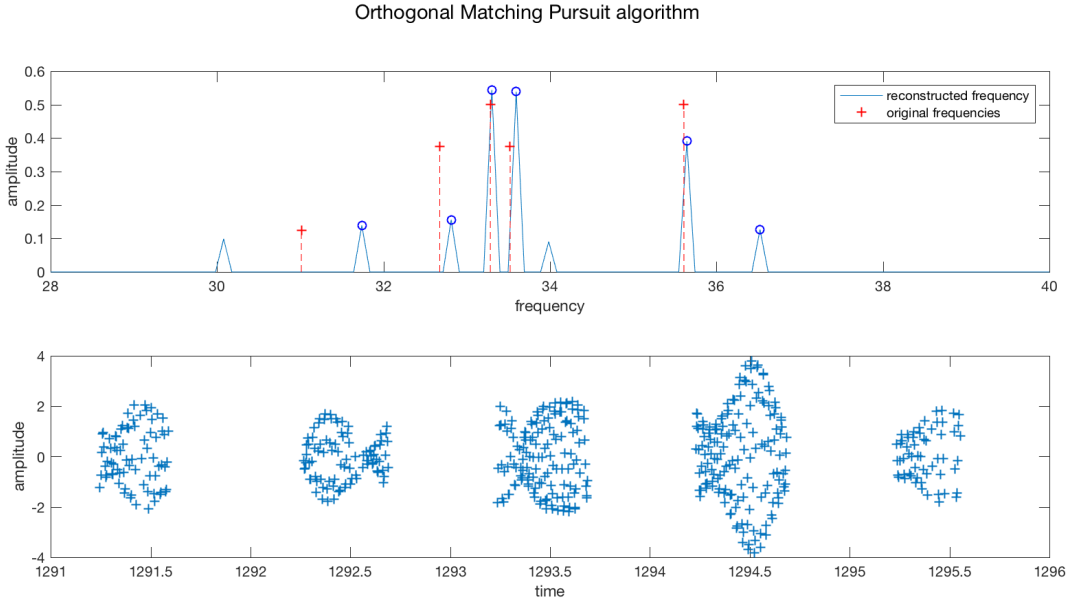

Figure 4: The upper panel shows a comparison between analysis result using OMP-algorithm and the actual signal data in frequency representation, the lower panel displays the signal resulting from the OMP algorithm in time domain

In the case of this OMP, one can see that, in comparison to the MP algorithm, there are less false detected signals, i.e. 8 signals are resulting in total instead of 10, which already is an improvement of the algorithm. Also the estimation of the location of the actual frequencies in the signal has a tendency to be better than before, but still is incorrect for several frequencies, especially the lowest one. So only by comparing the frequency location of the signals in the data compared to the location of the frequencies resulting from the analysis, one can see a deviation. Furthermore, the amplitudes are not correctly estimated for most of the frequencies. However, an advantage of the OMP is the number of necessary iteration steps to achieve those

results, which was reduced by more than a factor of two to 16, since the algorithm keeps track of already used indices, reducing the calculation time.

		Signal 1	Signal 2	Signal 3	Signal 4	Signal 5
Input signal	Frequency [day^{-1}]	31.012	32.675	33.283	33.521	35.609
	Amplitude	0.125	0.375	0.5	0.375	0.5
OMP output	Frequency [day^{-1}]	31.738	32.812	33.301	33.594	35.645
	Amplitude	0.139	0.156	0.544	0.539	0.390

Table 2: Parameter comparison ignoring the additional signals found by the OMP algorithm

3.3 Orthogonal Least Square (OLS)

The previous algorithms made use of the matrix $\boldsymbol{W}$ in order to calculate the Fourier transform using the FFT and also the inverse. However, this requires for the matrix $\boldsymbol{W}$ to be orthogonal with $\boldsymbol{W}^{\dagger}\boldsymbol{W} = N\boldsymbol{I}$. In the case of irregularly sampled data like our data, that property is not given, which was ignored by the previous algorithms leading to the observed inaccuracies. The Orthogonal Least Square algorithm (OLS) corrects the OLP for that by replacing the selection step a) with the selection of index k, for which the approximation error is minimzed when adding the column $\boldsymbol{\omega}_k$ to $\boldsymbol{W}_{\Gamma_n}$. This optimization is formulated via:

$$k = \arg\min_{k} \arg\min_{\boldsymbol{a}(\Gamma_n \cup (k))} ||\boldsymbol{x} - \boldsymbol{W}_{\Gamma_n \cup \{k\}} \boldsymbol{a}\left(\Gamma_n \cup \{k\}\right)||^2$$

Since a non-optimal implementation of such an algorithm can result in a long computing time, we use the matlab function `ols`: `[a, ind]= ols(W,x,Inf,test)`. As test for the stop criterium `test`$= \sigma_b^2 \tau$ is used. This method is implemented in the code shown in the appendix, as well. Also, the comparisons of the signal data to the signal resulting from the analysis using the OLS method are listed in table 3 and displayed in figure 5 analogue to the previous algorithms.

		Signal 1	Signal 2	Signal 3	Signal 4	Signal 5
Input signal	Frequency [day^{-1}]	31.012	32.675	33.283	33.521	35.609
	Amplitude	0.125	0.375	0.5	0.375	0.5
OLS output	Frequency [day^{-1}]	31.055	32.715	33.301	33.496	35.645
	Amplitude	0.114	0.327	0.549	0.293	0.376

Table 3: Parameter comparison ignoring the additional signals found by the OLS algorithm

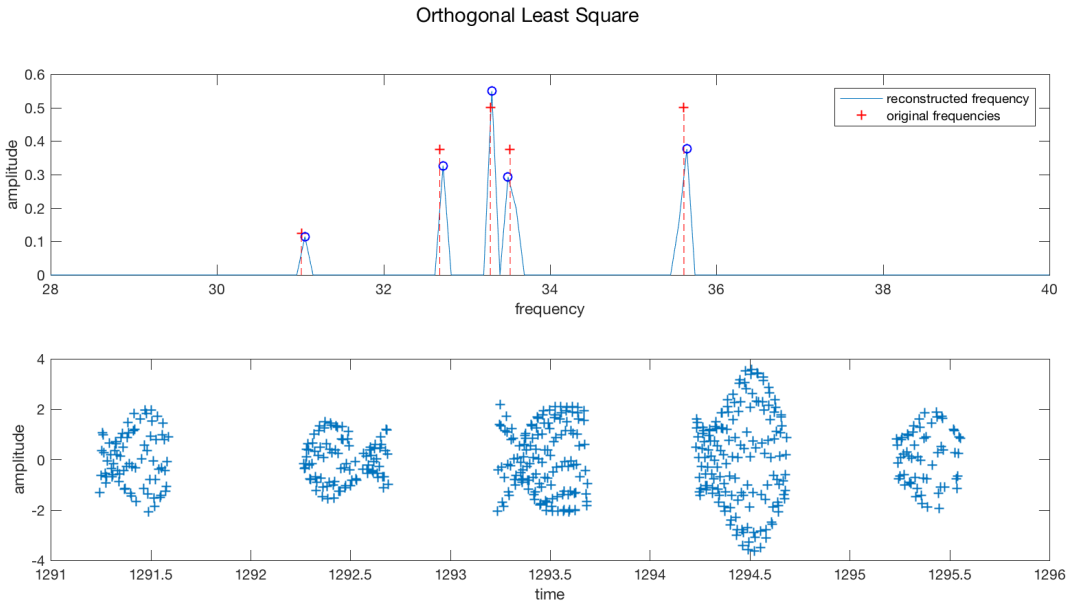

Figure 5: The upper panel shows a comparison between analysis result using OLS-algorithm and the actual signal data in frequency representation, the lower panel displays the signal resulting from the OLS algorithm in time domain

The OLS algorithm does not find any false detections, and also the locations of the 5 frequencies in the original data set are obtained correctly. Also the amplitudes of each of the detected signals are correctly estimated for at least 3 of the 5 signals, while the difference between the estimated amplitude and original amplitude for the remaining two signals is relatively small. Also, the number of iteration steps is relatively low with only 13 required steps. This result shows how the OLS is a strong improvement of the previous two algorithms.

4 Sparse representation with convex relaxation

Instead of using matching pursuit algorithms for obtaining a sparse representation, one can also minimize a convex quadratic cost function including a l^1-norm, as given in equation 4:

$$\boldsymbol{a} = \arg\min_{\boldsymbol{a}} ||\boldsymbol{x} - \boldsymbol{W}\boldsymbol{a}||^2 + \lambda ||\boldsymbol{a}||_l \text{ with } ||\boldsymbol{a}||_l = \sum_k |\boldsymbol{a}_k| \tag{4}$$

As a minimzing algorithm, we make use of the matlab function `min_12_11_0`. The amplitude vector $\boldsymbol{a}$ can be calculated with `min_12_11_0(x,W,lambda,n_it_max`, where the number of iterationsteps for the algorithm `n_it_max)` has to be defined. In general, the solution satisfies the condition $|\boldsymbol{W}^\dagger (\boldsymbol{x} - \boldsymbol{W}\boldsymbol{a})| \leq \lambda$. Because of that λ is the actual parameter describing the

accuracy of the fit by stating that the frequency presentation of the residual in the case that the convergence is smaller than $\frac{\lambda}{N}$. Therefore, λ has to be tuned in order to find a satisfying solution. In our case, λ is chosen to be 6% of the maximum of the frequency representation of the residual, and a maximum number of iteration steps of $n_{it,max} = 100000$. In those cases the algorithm presents a solution within a reasonable computation time. Our implementation of the code is again given in the MATLAB code in the appendix and the results are again shown in figure 6 and listed in table 4.
The convex relaxation algorithm also does not detect any false signals and has a good accuracy in detecting the location of the frequencies. However, it is not able to reconstruct the correct amplitudes for almost all of the signals.

		Signal 1	Signal 2	Signal 3	Signal 4	Signal 5
Input signal	Frequency [day^{-1}]	31.012	32.675	33.283	33.521	35.609
	Amplitude	0.125	0.375	0.5	0.375	0.5
OLS output	Frequency [day^{-1}]	30.957	32.715	33.301	33.496	35.645
	Amplitude	0.032	0.158	0.454	0.193	0.314

Table 4: Parameter comparison ignoring the additional signals found by the convex relaxation algorithm

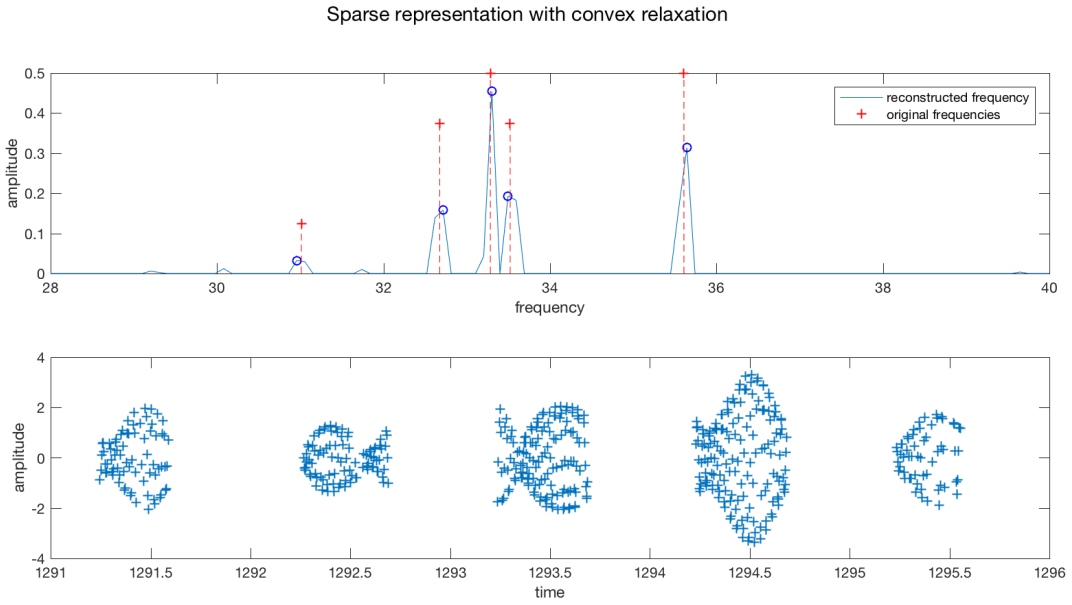

Figure 6: The upper panel shows a comparison between analysis result using the convex relaxation-algorithm and the actual signal data in frequency representation, the lower panel displays the signal resulting from the convex relaxation algorithm in time domain

A Initial Data Set

```
f_th =   31.0120   32.6750   33.2830   33.5210   35.6090
A_th =    0.2500    0.7500    1.0000    0.7500    1.0000
phi_th =  0.3930    0.9960    0.4920    0.2810    0.5960
t =    1291.2      ...  1295.6  [514 x 1]
y =      -0.1648    ...    -0.4816 [514 x 1]
```

B MATLAB Code

```
close all; clc; clear all;
load('data.mat')
SAVEDATA = false;

x = 0;
SNR = 10;

for k=1 : 5
    x = x + (A_th(k) * sin(2*pi*f_th(k)*t + phi_th(k)));
end

% calculate noise amplitude
pms = sumsqr(x)/length(x);
% standard Deviation
sigma = sqrt(pms/SNR);
% add noise to the calculations
noise =  sigma * randn(1,length(x));
x_n = x + transpose(noise);

figure
plot(t,x,'g+')
hold on;
plot(t,x_n,'r+')
legend('real data',sprintf('data with gaussian noise, stdDev: %0.3f'
    ,sigma))
title('Herbig star HD 104237 data samples')
ylabel('signal amplitude')
xlabel('time')
if SAVEDATA
    set(gcf, 'PaperUnits', 'points');
    set(gcf, 'PaperPosition', [0 0 900 450]);
```

```
31      saveas(gcf,'../images/data.png')
32  end
33
34  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35  % 2.2  irregular sampling case
36  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37  fmax = 100;
38  M = 1024;
39  N = length(x_n);
40  freq =(-M:M)/M*fmax;
41  W=exp(2*1i*pi*t*freq);
42  periodogram =  abs(W'*x_n)/N;
43
44  figure
45  subplot(2,2,1)
46  plot(freq,periodogram)
47  % plot real frequencies
48  hold on
49  plot(f_th,A_th/2,'r+')
50  hold on
51  plot((-1.*f_th),A_th/2,'r+')
52  xlabel('frequency')
53  ylabel('amplitude')
54  title('frequency representation')
55
56  subplot(2,2,3)
57  plot(freq,periodogram)
58  hold on;
59  plot(f_th,A_th/2,'r+')
60  xlim([28 40])
61  xlabel('frequency')
62  ylabel('amplitude')
63  legend('frequency distribution','original frequencies')
64
65  % plot spectral window
66  subplot(2,2,[2 4])
67  Win=W'*ones(N,1)/N;
68  plot(freq,abs(Win))
69  xlim([-20 20])
70  title('spectral window')
71  xlabel('frequency')
72  ylabel('amplitude')
```

```
73
74 if SAVEDATA
75     set(gcf, 'PaperUnits', 'points');
76     set(gcf, 'PaperPosition', [0 0 900 450]);
77     saveas(gcf,'../images/data_freq.png')
78 end
79
80 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
81 % 3.1 Matching Pursuit Algorithm
82 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
83 r_n = x_n;
84 Gamma0 = [];
85 a = zeros(2049,1);
86 tau = chisqq(0.95,N);
87 T = tau +1;
88 k = 1;
89
90 while T > tau
91     W_current = W(:,k);
92     [val, k] = max(abs(W'*r_n));
93     Gamma0 = [Gamma0 k];
94     a(k) = a(k) + (1/((W_current')*W_current))*W_current'*r_n;
95     r_n = r_n - (((1/(W_current'*W_current)).*W_current'*r_n).*
           W_current);
96     T = (norm(r_n)^2)/(sigma^2);
97 end
98 MethodOneIterations = size(Gamma0);
99 [MaxMP,MaxIdxMP] = findpeaks(abs(a),'MinPeakHeight',0.1);
100
101 figure
102 subplot(2,1,1)
103 plot(freq,abs(a));
104 hold on;
105 plot(f_th,A_th/2,'r+')
106 hold on;
107 plot((freq(MaxIdxMP)),MaxMP,'bo')
108 for num=1:length(f_th)
109    hold on;
110    line([f_th(num) f_th(num)], [0 A_th(num)/2], 'Color','r','
          LineStyle','--')
111 end
112 xlim([28 40])
```

```
113 legend('reconstructed frequency','original frequencies')
114 xlabel('frequency')
115 ylabel('amplitude')
116 subplot(2,1,2)
117 plot(t,W*a,'+')
118 suptitle('Matching Pursuit (pre-whitening) algorithm')
119 if SAVEDATA
120     set(gcf, 'PaperUnits', 'points');
121     set(gcf, 'PaperPosition', [0 0 900 450]);
122     saveas(gcf,'../images/mp.png')
123 end
124
125
126 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
127 % 3.2 Orthogonal Matching Pursuit Algorithm
128 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
129 r_n = x_n;
130 Gamma0 = [];
131 W_g = [];
132 a = [];
133 tau = chisqq(0.95,N);
134 T = tau +1;
135 k = 1;
136
137 while T > tau
138     [val, k] = max(abs(W'*r_n))
139     Gamma0 = [Gamma0 k];
140
141     W_g = [];
142     for l=1:length(Gamma0)
143         W_g = [W_g W(:,Gamma0(l))];
144     end
145
146     a = ((W_g'*W_g)^(-1))*W_g'*x_n;
147     r_n = x_n - W_g*a;
148     T = (norm(r_n)^2)/(sigma^2);
149 end
150 a_plot = zeros(2049,1);
151 for ind=1:length(Gamma0)
152    a_plot(Gamma0(ind)) = a(ind);
153 end
154 MethodTwoIterations = size(Gamma0);
```

```
155 [MaxOMP,MaxIdxOMP] = findpeaks(abs(a_plot),'MinPeakHeight',0.1);
156
157 figure
158 subplot(2,1,1)
159 plot(freq,abs(a_plot));
160 hold on;
161 plot(f_th,A_th/2,'r+')
162 hold on;
163 plot((freq(MaxIdxOMP)),MaxOMP,'bo')
164 for num=1:length(f_th)
165     hold on;
166    line([f_th(num) f_th(num)], [0 A_th(num)/2], 'Color','r','
        LineStyle','--')
167 end
168 xlim([28 40])
169 legend('reconstructed frequency','original frequencies')
170 xlabel('frequency')
171 ylabel('amplitude')
172 subplot(2,1,2)
173 plot(t,W*a_plot,'+')
174 xlabel('time')
175 ylabel('amplitude')
176 suptitle('Orthogonal Matching Pursuit algorithm');
177 if SAVEDATA
178     set(gcf, 'PaperUnits', 'points');
179     set(gcf, 'PaperPosition', [0 0 900 450]);
180     saveas(gcf,'../images/omp.png')
181 end
182
183
184
185
186 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
187 % 3.3 Orthogonal Least Square
188 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
189 r_n = x_n;
190 Gamma0 = [];
191 W_g = [];
192 a = 0;
193 tau = chisqq(0.95,N);
194 T = tau +1;
195 k = 1;
```

```
196 test = (sigma^2)*tau;
197 a_vec = [];
198 while T > tau
199     [val, k] = ols(W, x_n, Inf, test);
200     Gamma0 = [Gamma0 k];
201
202     W_g = [];
203     for l=1:length(Gamma0)
204         W_g = [W_g W(:,Gamma0(l))];
205     end
206
207     a = ((W_g'*W_g)^(-1))*W_g'*x_n;
208     a_vec = [a_vec a];
209
210     r_n = x_n - W_g*a;
211     T = (norm(r_n)^2)/(sigma^2);
212 end
213
214 a_plot = zeros(2049,1);
215 for ind=1:length(Gamma0)
216     a_plot(Gamma0(ind)) = a_vec(ind);
217 end
218 MethodThrIterations = size(Gamma0);
219 [MaxOLS,MaxIdxOLS] = findpeaks(abs(a_plot),'MinPeakHeight',0.1);
220
221 figure
222 subplot(2,1,1)
223 plot(freq,abs(a_plot));
224 hold on;
225 plot(f_th,A_th/2,'r+')
226 hold on;
227 plot((freq(MaxIdxOLS)),MaxOLS,'bo')
228 for num=1:length(f_th)
229     hold on;
230     line([f_th(num) f_th(num)], [0 A_th(num)/2], 'Color','r','
        LineStyle','--')
231 end
232 xlim([28 40])
233 xlabel('frequency')
234 ylabel('amplitude')
235 legend('reconstructed frequency','original frequencies')
236 subplot(2,1,2)
```

```
237 plot(t,W*a_plot,'+')
238 xlabel('time')
239 ylabel('amplitude')
240 suptitle('Orthogonal Least Square');
241 if SAVEDATA
242     set(gcf, 'PaperUnits', 'points');
243     set(gcf, 'PaperPosition', [0 0 900 450]);
244     saveas(gcf,'../images/ols.png')
245 end
246
247
248 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
249 % 4 Sparse representation with convex relation
250 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
251 lambda_max = max(abs(W'*x_n));
252 lambda = 0.06 * lambda_max;
253 n_it_max = 100000;
254
255 a1 = min_L2_L1_0(x_n,W,lambda,n_it_max);
256
257 [MaxSparse,MaxIdxSparse] = findpeaks(abs(a1),'MinPeakHeight',0.03);
258
259 figure
260 subplot(2,1,1)
261 plot(freq,abs(a1));
262 hold on;
263 plot(f_th,A_th/2,'r+')
264 hold on;
265 plot((freq(MaxIdxSparse)),MaxSparse,'bo')
266 for num=1:length(f_th)
267     hold on;
268     line([f_th(num) f_th(num)], [0 A_th(num)/2], 'Color','r','
        LineStyle','--')
269 end
270 xlim([28 40])
271 xlabel('frequency')
272 ylabel('amplitude')
273 legend('reconstructed frequency','original frequencies')
274 subplot(2,1,2)
275 plot(t,W*a1,'+')
276 xlabel('time')
277 ylabel('amplitude')
```

```
278 suptitle('Sparse representation with convex relaxation');
279 if SAVEDATA
280     set(gcf, 'PaperUnits', 'points');
281     set(gcf, 'PaperPosition', [0 0 900 450]);
282     saveas(gcf,'../images/convex.png')
283 end
284
285 % Save detected data to file:
286 if SAVEDATA
287     fileID = fopen('../images/img_data.txt','w');
288     fprintf(fileID, 'frequency ; amplitude\n ');
289     fprintf(fileID, '# Matching Pursuit, %d iterations\n', (
          MethodOneIterations(2)));
290     fprintf(fileID, [repmat(' %0.3f ; %0.3f \n', 1, length(MaxIdxMP)
          )] , [ freq(MaxIdxMP).' MaxMP].');
291     fprintf(fileID, '# Orthogonal Matching Pursuit, %d iterations\n'
          , (MethodTwoIterations(2)));
292     fprintf(fileID, [repmat(' %0.3f ; %0.3f \n', 1, length(MaxIdxOMP
          ))] , [ freq(MaxIdxOMP).' MaxOMP].');
293     fprintf(fileID, '# Orthogonal Least Square, %d iterations\n', (
          MethodThrIterations(1)));
294     fprintf(fileID, [repmat(' %0.3f ; %0.3f \n', 1, length(MaxIdxOLS
          ))] , [ freq(MaxIdxOLS).' MaxOLS].');
295     fprintf(fileID, '# Convex Relaxation\n');
296     fprintf(fileID, [repmat(' %0.3f ; %0.3f \n', 1, length(
          MaxIdxSparse))] , [ freq(MaxIdxSparse).' MaxSparse].');
297     fclose(fileID);
298 end
```