Defining $b := R^t x$, one first forward solves

$$Rb = A^t r$$

After this, one backward solves

$$R^t x = b$$

This yields x. Knowing that the matrix $B = A^t A$ is always positive definite makes it possible to use the Cholesky decomposition instead of Gauss elimination and eliminates the need for handling the numerical stability which is a common problem for Gauss elimination. See [5] p. 145 for a detailed description.

6 Measurements

The following algorithms have been compared:

1. Rescaled Range
2. Standard method
3. Optimal method with wavelets D4, D8, D16 and D64 (Described in [4]).
4. Wavelet method with wavelets D2, D4, D8, D16, D64, C6 and C12 (DN: Daubechies wavelet with N coefficients, CN: Coiflet with N coefficients)
5. Wavelet method using stationary wavelet decomposition with wavelets D2, D4, D16, D64 and C12

Artificial data has been created with Hurst exponents of $H = 0.0, 0.1, \ldots 1.0$ and with trace lengths of 2^8 to 2^{20}. Thirty-two test sets for each combination of H and length have been generated, average and standard deviation have been computed. The relevant criteria were the accuracy of the estimator as well as the standard deviation of the results.

The tests were run with the data as is as well as with various levels of noise added in a way which will be described in the next section.

7 Results

Tests have been run on simulated data for all combinations of input length $2^6, 2^8, 2^{12}, 2^{16}, 2^{20}$, $H = 0.0, 0.1, \ldots 1.0$ and hurst estimators on thirty-two test sets for each combination to compute the standard error yielding more than 1500 test results. Only a small subset of all the data is shown in the following graphics. The error bars displayed on all images mark the standard deviation which was computed over all thirty-two test sets.

7.1 Bias

Some initial testing with a "naive" implementations of "Rescaled Range" and "Standard" has shown some significant bias. After some discussion, two main reasons for the bias were identified:

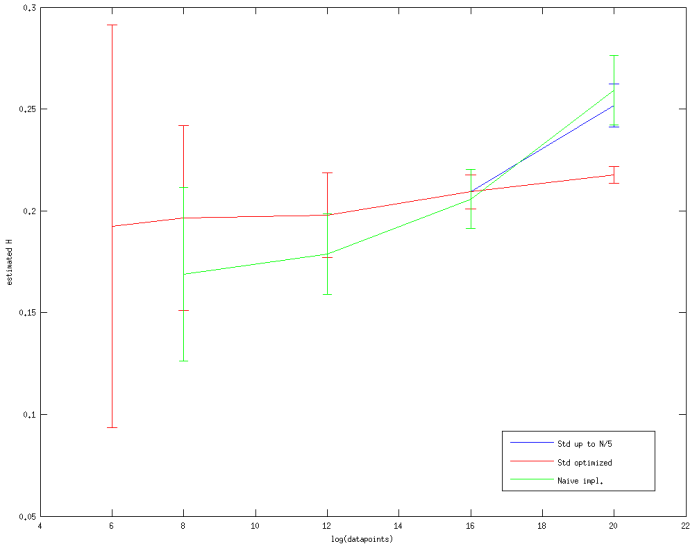

Figure 3: Comparison of three different implementations of the Standard algorithm for $H = 0.2$ and different input sizes. Green: Naive implementation, Blue: Taking into account that window sizes should not be larger than $\frac{N}{5}$, Red: Additionally limiting any window size to a maximum of $\frac{2^{16}}{5}$. Red and blue are identical up to 2^{16}.

- Taking into account window sizes which are too large compared to the length of the time series. As a general rule of thumb, one should not use windows which are larger than $\frac{N}{5}$ where N is the length of the time series.
- For very large time series (larger than 2^{16}), one should additionally limit the upper size of any window to about $\frac{2^{16}}{5}$.

Figure 3 shows a comparison between the naive implementation, the limitation to $\frac{N}{5}$ and the second limitation to $\frac{2^{16}}{5}$ for large inputs for a hurst estimator of $H = 0.2$. From the data, it's clear that these two steps will significantly reduce bias of these two algorithms.

After optimizing the maximum window size for the two window-based algorithms, all of the measurements have been run with the optimized versions. Raw data for the non-optimized versions is delivered as add-on.

7.2 Minimum input length

Results clearly show that most of the tested algorithms are not suited for small inputs (e.g. 2^6 data points). The wavelet-based algorithms need to fill their filter banks in order to produce some useful output. A large percentage of the data worked on will consist of padding when running the algorithms on short inputs. Also the rescaled range algorithm is very unstable on too short inputs and should therefore not be used. Generally, the following rules of thumb hold:

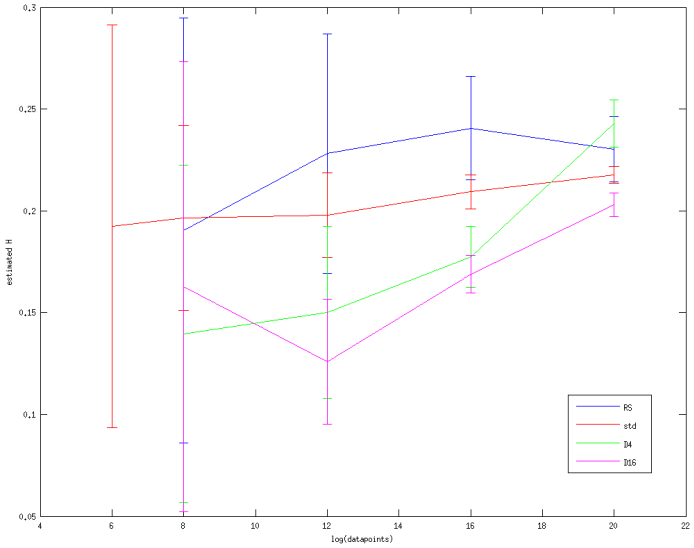

Figure 4: Results for synthetic data with $H = 0.2$

Up to input sizes of 256, only the standard algorithm can be used. After this, wavelet-based algorithms using wavelets with few wavelet coefficients (up to D16) as well as rescaled range can be used as well. Wavelet-based algorithms using bigger wavelets such as D32 or D64 should only be used on very large data sets as the requirement for padding grows with the number of wavelet coefficients in a wavelet.

Generally, larger input sets will also reduce the standard error of the estimation, which is evident on all subsequently shown graphs in this section. Especially estimations on very short input data sets should be treated with care, as the standard error is significant! All of the graphs in this section only contain information for input sets of the size 2^6 for the standard algorithm. All of the other algorithms either cannot produce any result because they need some minimum input size, or have that little accuracy or a that large standard error, that they are completely unusable. Based on the results of these tests I have set some minimum sizes for the input data length in the code of the delivered software, so that users do not run unreasonable combinations of input data length and algorithms. Of course this can be easily changed in the code if really needed, but this should be done with care.

7.3 Comparison of RS, standard, D4 and D16

The first series of graphs shows a comparison of the Hurst estimators “Rescaled Range”, “Standard”, “Wavelet” (using D4) and “Wavelet” (using D16). Results are shown for a synthetic traces with H chosen as 0.2, 0.5 and 0.8.

It’s obvious from the graphs that the wavelet-based methods start with a huge standard error for small datasets compared to the classical algorithms but

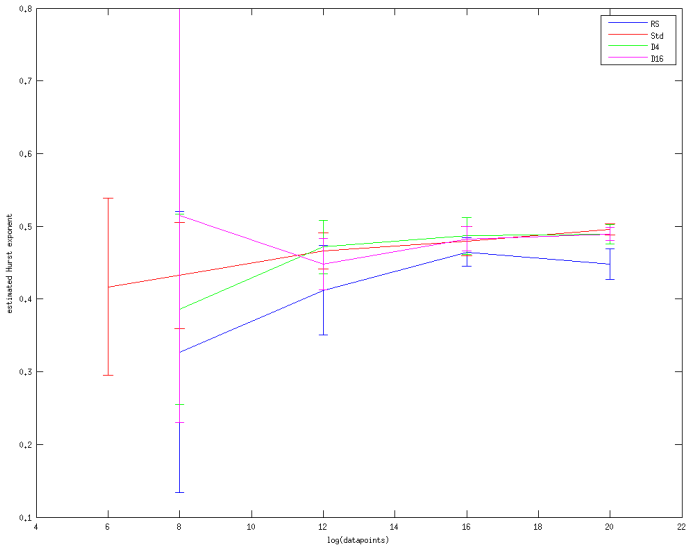

Figure 5: Results for synthetic data with $H = 0.5$

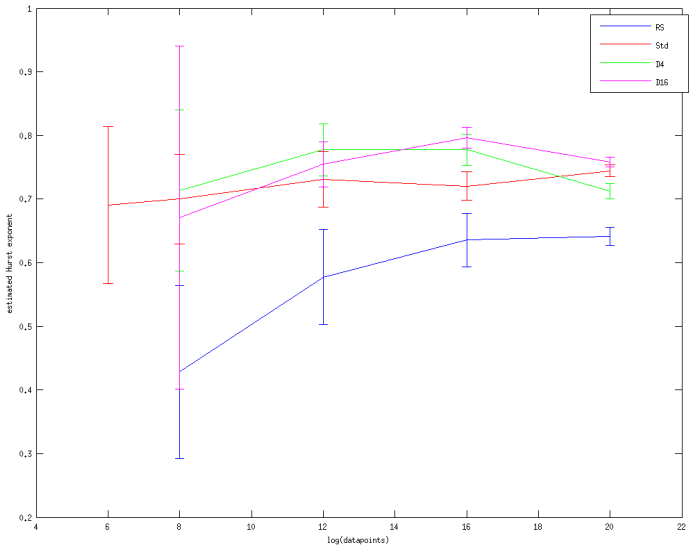

Figure 6: Results for synthetic data with $H = 0.8$

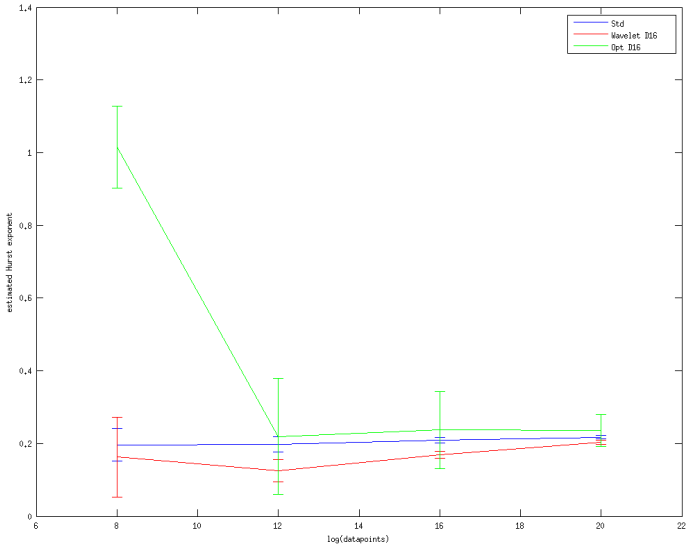

Figure 7: Comparison of the optimal algorithm to others with $H = 0.2$

will eventually outperform all other algorithms for long inputs. By looking at the results, the best way to work with estimators would be to choose the classical algorithms for small inputs and the wavelet-based algorithms for large inputs. This even gets clearer when one takes the optimal algorithm by Gloter and Hoffmann into account (called "opt" in the image labels) (see Figure 7). Although only a graph for $H = 0.2$ is shown, this holds for all other tested values as well.

7.4 Errors depending on H

Another observation is that the accuracy depend on H for the classical algorithms ("Rescaled Range", "standard"). It tends to be better around $H \approx 0.5$ and gets worse at the extremes. Both algorithms seem to have a bias towards 0.5 while the wavelet-based algorithms are bias-free. Figure 8 illustrates this.

7.5 Noise

In contrast to synthetic traces, most real-world data will contain noise, as measurements cannot be carried out with arbitrary precision. The authors of [4] claim their algorithm treats noise best. I have run all algorithms with the same test sets with noise added to it. Noise has been modeled as random normal distributed factor with average 1 and standard deviation l to each data point:

$$y_i := x_i R \tag{15}$$

Where $R = \mathcal{N}(1, l)$. I have run noise tests for $l \in \{0.05, 0.1, 0.2\}$. On the one hand, it's clear that multiplying random distributed noise will influence

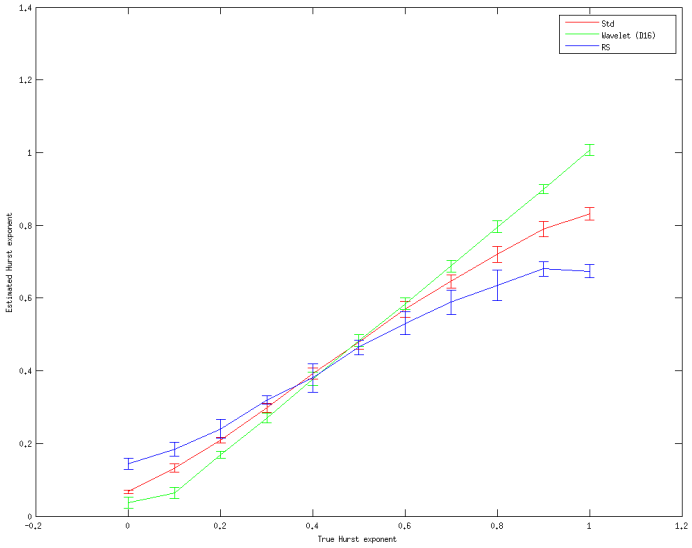

Figure 8: Plot of true H against estimated H with 2^{16} data points as input

the Hurst exponent of a time series. However, it's very interesting to see that already for $l = 0.05$, most of the algorithms show very poor performance (see Figure 9). As can be seen from the figures, RS and Std are already unusable for $l = 0.05$. The standard wavelet algorithm starts to give completely useless results with $l = 0.1$ (see Figure 10). Only the optimal algorithm can handle this level of noise. However, keep in mind that the optimal algorithm needs large input sets (size 2^{16} or larger, see figure 7) to get to a reasonable level of standard error. If only small input sets are available, the only thing to do is to reduce noise in measurements as far as possible. Otherwise, there is no way to estimate the Hurst exponent from the data as there is no suitable algorithm for small data sets which can cope with a significant amount of noise.

After these results, I have run more tests only with the optimal algorithm for $l = 0.4$ and $l = 0.6$ to explore the limitations of this algorithm. Figure 11 shows the results. The results show that the optimal algorithm even at higher noise level gives some useful results. However, the standard errors keep on growing, so large input data sets well beyond 2^{16} are needed to estimate the Hurst exponent with a certain confidence.

The conclusion from the noise measurements is, that for short input sets (such as 64 data points), where the standard algorithm is the only one which is available, measurement accuracy is crucial, as even small amounts of noise will distort the results. Algorithms which cope well with noise need very large input sets to reach a certain precision.

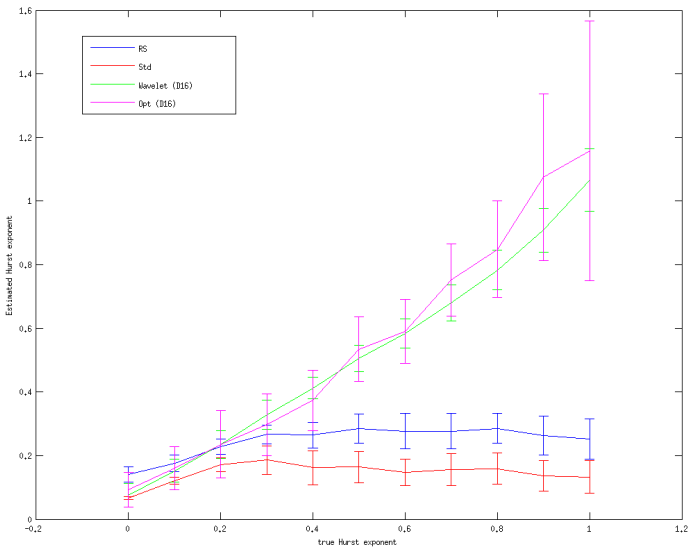

Figure 9: Plot of different algorithms on a data sets of size 2^{16} with multiplicative noise normally distributed with $\mathcal{N}(1, 0.05)$

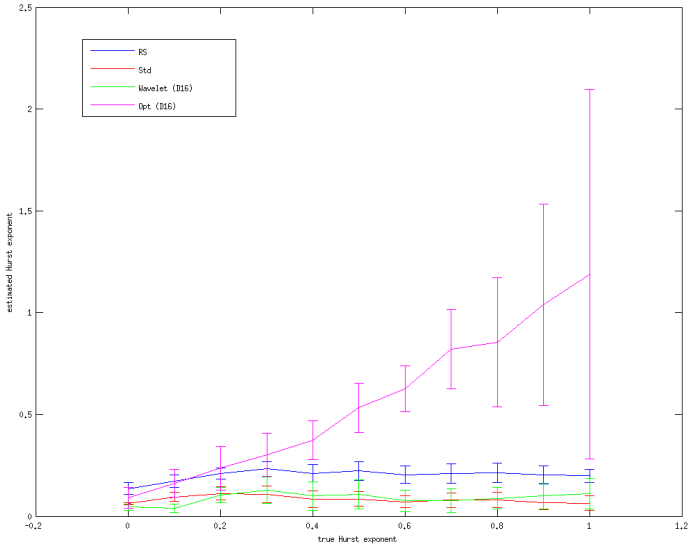

Figure 10: Plot of different algorithms on a data sets of size 2^{16} with multiplicative noise normally distributed with $\mathcal{N}(1, 0.1)$

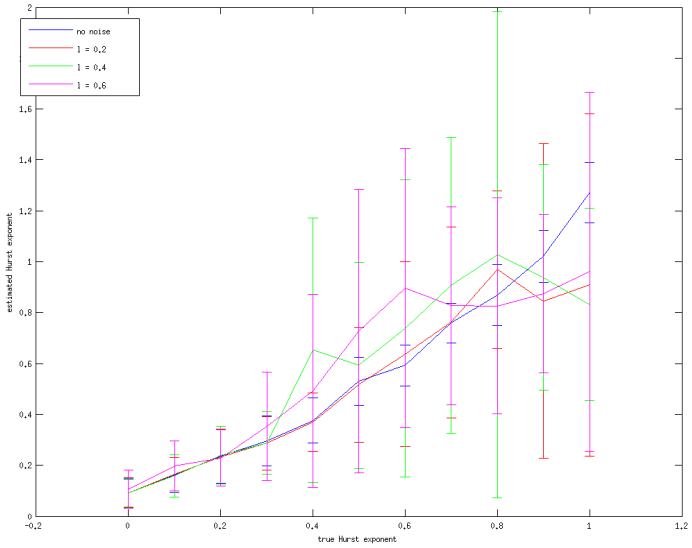

Figure 11: Plot of the optimal algorithm (using D16) on a data sets of size 2^{16} with multiplicative noise normally distributed with standard deviation l as in the legend.

7.6 Runtime

Run time measurements have been made for inputs between 2^8 and 2^{20}. The measurements have been made on a machine with an Intel Core Duo Quad CPU and 4GB RAM. The complete test bench is written single-threaded. The given values are pure CPU time in the maximum resolution available. Figure 12 shows the run-time behaviour of the different algorithms. It's clearly visible that the standard algorithm is in $O(n^2)$, the Optimal algorithm in $O(nlog(n))$ while the others are in $O(n)$.

Another metric is the run time needed to achieve a certain precision. This is based on the following consideration: The confidence interval for a series of n measurements of a random variable X is computed as follows:

$$ci = t_{p,n}\frac{\sigma}{\sqrt{n}} \tag{16}$$

where for any X holds

$$P(\bar{x} - ci \leq X \leq \bar{x} + ci) \geq p \tag{17}$$

where σ is the standard deviation of the random variable X and $\bar{x}$ its average and $t_{p,n}$ the corresponding value of the t-distribution for a certain confidence level and a certain n. (See [8], chapters 6 and 7.)

This means that quadrupling the input will make the confidence intervals shrink by a factor of two. Depending on the inherent precision of the algorithm in question (i.e. how much standard error it produces), more or less input data is

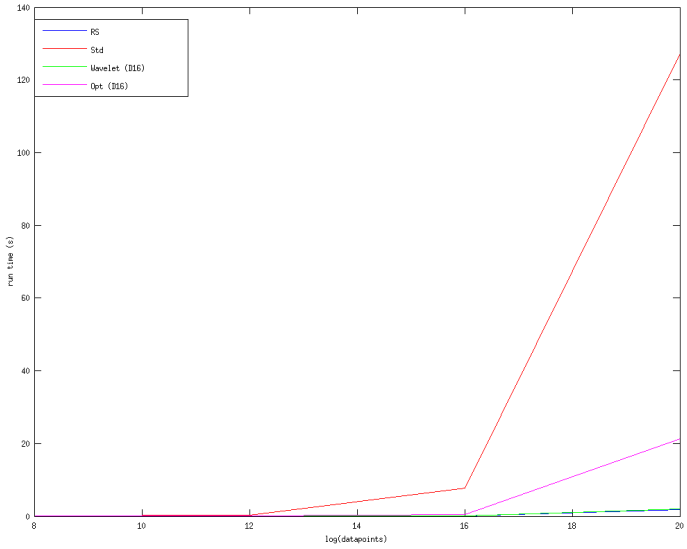

Figure 12: Runtime comparison of the different algorithms.

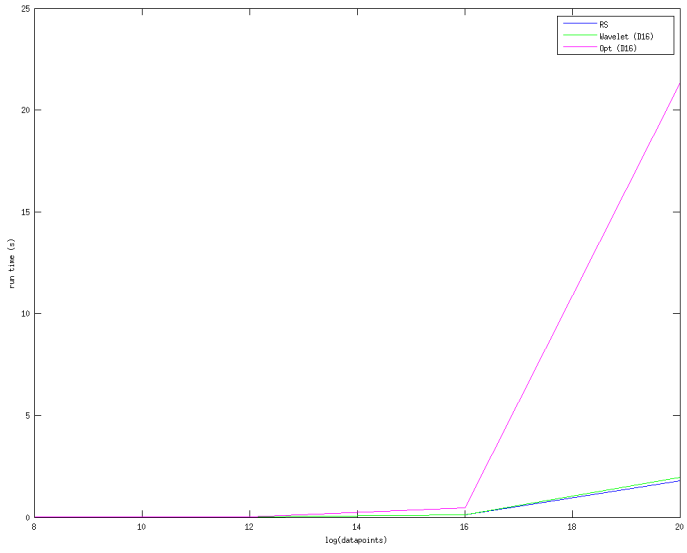

Figure 13: Runtime comparison of the different algorithms without std algorithm for better overview.

needed to meet a certain target. If the desired confidence interval is $ci_{des} = 0.05$ and the measured confidence interval with n input data is c, then the number n_{des} of input data points needed to reach ci_{des} is

$$n_{des} = n\left(\frac{c}{ci_{des}})\right)^2 \tag{18}$$

Table 1 first shows the raw values for the amount of input data needed to reach a confidence interval of at most 0.05 around the average value. These values are to interpreted like this: For a given algorithm, measuring as many time series with the run length given on the top so that the total number of data points (number of time series times length of each run) equals the value in the table will guarantee that in 95% of all cases doing this the resulting average will be within ± 0.05 around the average that a certain method estimates. However, this is just a guarantee for the precision (being within a certain range around an average), it does not guarantee the accuracy of the result (being within a certain range around the true value). As has been shown in the previous sections, some algorithms tend to have a bias. While by looking at the table and keeping in mind that some algorithms have super-linear run time, it might be tempting to cut long datasets into several short data sets, as these will run faster and a certain level of precision is reached within a shorter time. However, the increased precision is paid by lower accuracy, as it is clear from the previous subsections, that runs with small data sets will produce less accurate results. Depending on the application, it might of course be worth paying.

The next step is to take into account the run time. This is done by simply extrapolating the measured run times to the number of data points calculated in table 1. This way, one knows not only how many input data points one needs, but also how much time it will cost to process them using a certain algorithm. Depending on how long it takes to measure the data compared to process the data (or how much it costs or some other metric), different trade-offs might be chosen. This information is displayed in table 2.

Generally, one can see, that for input sequences up to about 2^{12}, the standard algorithm needs the fewest input to achieve the desired precision. For longer inputs, wavelet based algorithms start to be competitive. This gets even clearer when taking run time into account. As the wavelet based algorithms are in $O(n)$ compared to the standard algorithm which is in $O(n^2)$, they need much less time to achieve the desired precision. The optimal algorithm completely loses this test at least for input sizes up to 2^{20}. However, its strengths are its resilience against noise.

8 Conclusions

The following conclusions can be made after the measurements. There is no free lunch. Except for the rescaled range algorithm developed by Harold Hurst himself which is dominated by other algorithms in all tested areas, all algorithms have their strengths and weaknesses.

- The **standard algorithm** can be used for short inputs (such as 2^6). Its the only algorithm which delivers reasonable results for small input sets. However, it has quadratic run time and is therefore significantly slower

Algo / Run length	2^6	2^8	2^{12}	2^{16}	2^{20}
RS	-	14,710	23,850	39,560	694,430
Std	1,459	2,095	3,994	40,605	91,119
Wavelet (D4)	-	6,730	8,520	66,200	276,510
Wavelet (D16)	-	31,920	7,910	27,450	128,490
Wavelet (D64)	-	-	9,593	12,542	50,170
Opt (D16)	-	17,600	347,400	897,600	3,054,600

Table 1: Total number of measured data points needed to achieve a confidence interval of ± 0.05 given a certain length of each individual run shown for different algorithms using synthetic traces with $H = 0.5$.

Algo / Run length	2^6	2^8	2^{12}	2^{16}	2^{20}
RS	-	0.0103	0.0298	0.0698	1.204
Std	0.0486	0.1039	0.3047	4.7592	10.8804
Wavelet (D4)	-	0.0066	0.0065	0.0284	0.1434
Wavelet (D16)	-	0.0249	0.0060	0.0484	0.2416
Wavelet (D64)	-	-	0.0293	0.0885	0.3677
Opt (D16)	-	0.0192	1.0602	5.8637	62.1483

Table 2: Time needed to achieve a confidence interval of ± 0.05 using synthetic traces with $H = 0.5$. All values are in seconds.

than the wavelet-based estimators on long inputs. But as a positive aspect, it needs least input data for small input sets up to 2^{12} to achieve a certain precision. On the other hand, its also very prone to noise in the data and has some bias in contrast to the wavelet-based algorithms and the optimal algorithm which are bias-free.

- The **wavelet-based algorithms** have a broad range where they perform well. Their good convergence means that they need less input data as other algorithms specially for large time series and their linear run time makes them fast. For very long input sets such as 2^{20} or larger, wavelets with more coefficients start to get interesting, as they need less input than smaller wavelets. Also, the wavelet-based algorithms have some resilience against noise. The drawback is that wavelet algorithms can only run on data sets with a certain minimum length, e.g. 128 or even 256 as they need some input data to fill their filter banks. They are not suitable for short input sets.

- The **optimal algorithm** shows weaker performance than the wavelet-based algorithms in most tests up to the maximum input size tested. On the other hand, its resilience against noise is superior to all other tested algorithms. It's the only algorithm which can be used in the tested noise model with multiplicative noise with a distribution of $\mathcal{N}(1, 0.1)$ or more. However, also the optimal algorithm is not suitable for small input sets, as it shares the same problems as the standard wavelet-based algorithms.

In general, different aspects must be taken in to account before choosing the algorithm. For short inputs, only the standard algorithm is suitable. As it is prone to noise, this algorithm must be used with care. For large input

sets, wavelet-based algorithms will show better overall performance. For noisy data sets, depending on the level of noise, the optimal algorithm or the other wavelet-algorithms can be used. In this case, one needs to use larger input data sets.

9 Higher dimensional cases

9.1 Standard algorithm

The standard method can be easily extended to two and more-dimensional applications. For the two-dimensional case, the equation (8) can be adjusted like this:

$$\mu_\nu(\Delta n) := \frac{1}{N-\Delta n} \sum_{n=0}^{N-\Delta n-1} \|\boldsymbol{x}_{n+\Delta n} - \boldsymbol{x}_n\|_2^\nu \tag{19}$$

Note that the absolute value has now changed to the L_2 norm.

9.2 Rescaled range algorithm

Also the rescaled range method can be extended to a more-dimensional case. However, equation (2) needs to be reformulated like this:

$$R(n) = max_{i,j}(dist(\boldsymbol{x}_i, \boldsymbol{x}_j)) \tag{20}$$

As this leads to a significant computational effort, I have chosen to do the following instead for every time window starting at index l and of size n:

$$\bar{\boldsymbol{x}} = \frac{1}{n} \sum_{i=l}^{l+n} \boldsymbol{x_i} \tag{21}$$

$$R(n) = 2max_{\boldsymbol{x}}(dist(\boldsymbol{x}, \bar{\boldsymbol{x}})) \tag{22}$$

The standard deviation $S(n)$ is computed as

$$S(n) := \frac{1}{N-1} \sum_{i=1}^{N} \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_2$$

9.3 Other algorithms

The use of wavelet algorithms for multidimensional cases is not that easy, as multidimensional wavelets would be needed for this. I have not followed this path.

9.4 Measurement results

9.4.1 Synthetic data

First, these algorithms have been run against synthetic data sets to verify their correctness. The synthetic data sets have $H \in \{0.0, 0.5, 1.0\}$ and $l \in \{2^6, 2^8, 2^{12}, 2^{16}, 2^{20}\}$. One graph for each value of H is shown. As one can

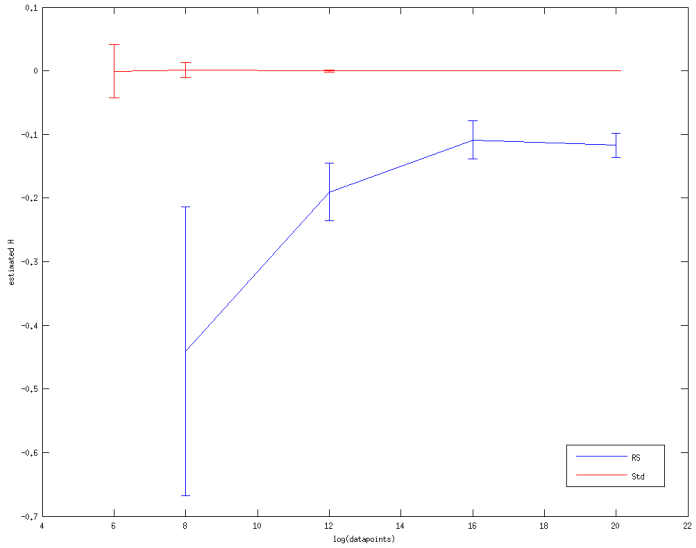

Figure 14: Results for synthetic data with $H = 0.0$. The error bars for Std are so small that they are not visible.

Test case	estimated H	Ci lower bound	Ci upper bound
All3T6_2	0.112	0.0685	0.155
All3T6_4	0.0723	0.0519	0.0928
All3T6	0.1347	0.115	0.155
AllCtrl	−0.0205	−0.0293	−0.0117
AllLatA	0.373	0.352	0.395
AllMCD	0.0146	0.00679	0.0223
AllMCDLatA	0.0718	0.0401	0.103

Table 3: Measurements for real data.

see, the rescaled range algorithm shows a much poorer convergence than in the one dimensional case. I suppose that this is because of some design decisions made such as estimating he maximum distance between two points using $R(n) = 2max_{\boldsymbol{x}}(dist(\boldsymbol{x}, \bar{\boldsymbol{x}}))$ which of course is not exactly the true value. However, I did not further follow this path, as the rescaled range algorithm is known to have poor performance from the one dimensional case and should generally not be used.

9.4.2 Real data

The tests with real two-dimensional data have only been made with the standard algorithm in 2D. The data consists of a number of different test cases with several individual runs each containing several hundred two dimensional data points. Table 3 shows test case, estimated Hurst exponent and the 95% confidence interval bounds.

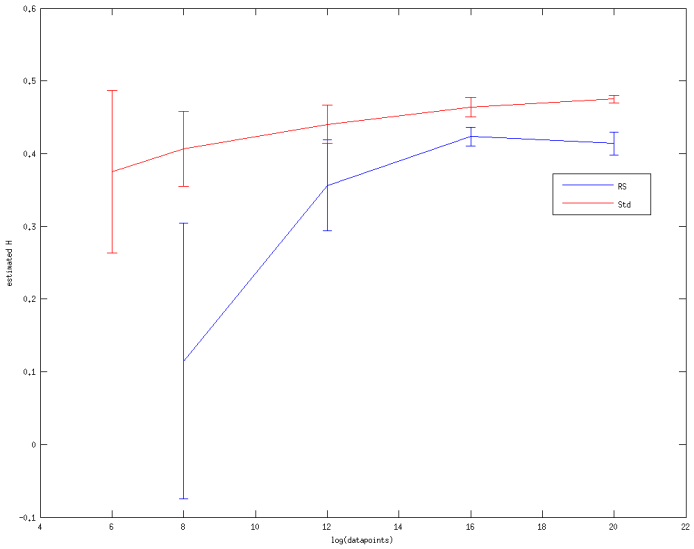

Figure 15: Results for synthetic data with $H = 0.5$

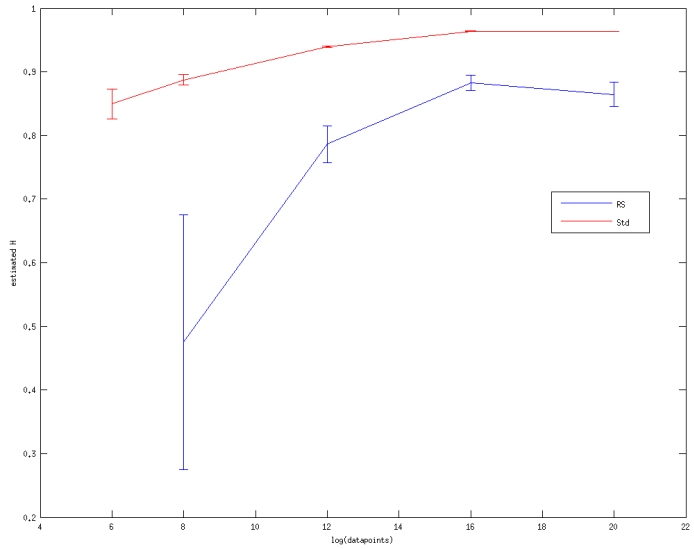

Figure 16: Results for synthetic data with $H = 1.0$

10 Future work

10.1 Multidimensional wavelet estimator

Multidimensional Hurst estimators only been treated very limited. More work could be done to adapt wavelet-based estimators for 2D or even higher dimensional applications. The implementation which has been made in this thesis is based on the "standard algorithm" and therefore is very slow for large input sets.

10.2 Wavelet packet algorithm

As described in [6], the Hurst exponent can also be estimated using the wavelet packet algorithm. This algorithm is widely used in image compression and gives more degrees of freedom than the classical wavelet decomposition. Instead of only decomposing the low-pass filtered output of one decomposition step, the wavelet function is applied to the high-pass filtered output as well. Afterwards, a best basis is built from all of the decomposed data.

A Other applications

A.1 Stock market

I have applied the Hurst estimators to historical closing prices of the blue chips of the Swiss stock market. Some measurement results (using a static decomposition wavelet estimator) are shown in the table below. The data used is from June 2005 to May 2010.

ABB	0.44
Actelion	0.47
Baer	0.43
Credit Suisse	0.31
Holcim	0.40
Lonza	0.35
Nestlé	0.48
Novartis	0.38
Richemond	0.51
Roche	0.42
Swiss Re	0.34
Swisscom	0.42
Swiss Life	0.49
Syngenta	0.49
Synthes	0.42
UBS	0.48
Swatch	0.52
Zurich	0.45

This does not look very promising, if the values were clearly over 0.5 in all cases, making money with stocks would be a lot easier. Probably, more information could be taken out of the data, if not only closing prices (i.e. prices at the end of the day) would be taken into account or if a time window had been chosen which does not contain huge turbulences on the financial markets. It would

also be interesting to investigate whether the Hurst exponent has dropped after the introduction of electronic trading as well as after making real-time trading available to the broad public through on-line trading. However, appropriate historical data is not freely available.

A.2 Picking on seismic signals

As I am working at the Swiss seismological service and we are currently evaluating new software to locate earthquakes, I have tested a wavelet-based algorithm described below against other algorithms we are evaluating. Given a measurement signal (basically a time series) as the input, the problem is to find as exactly as possible the onset time of an arriving P wave (primary wave, the fastest seismic wave and therefore the first to hit a measurement station). The complicated part about this is to cope with background noise and to get the onset time as exactly as possible. A usual problem for all of the algorithms is that they might miss the P wave and only declare the onset time when the larger but slower S wave (secondary wave) arrives, a problem which all algorithms using threshold values are prone to. Having some picks on the P and some on the S wave will distort the results. Various algorithms have been developed to pick earthquakes and to filter mispicks.

I have implemented a wavelet-based picker which basically does the following

1. Make a stationary wavelet transform. This gives several time series, each containing one octave of the original signal.

2. For each octave, estimate the average and the standard deviation of the energy (squared decomposed values) on the first n seconds. This is an estimate for the noise in the respective frequency band.

3. Run a sliding window over the rest of the data, declare a pick if the energy reaches at least m times the average plus l times the standard deviation. This might result in no pick as well if there is too much noise or too little signal.

4. Doing this on all octaves generally results in a number of picks. If there are more than k picks, a linear least squares fit is run. As every filter dilutes the signal (because the wavelet is a windowed function), using the least squares fit allows an estimate on what the pick would be on the undecomposed signal. For every decomposition step, an equation like this is produced:
$$t_i m + c = i \tag{23}$$
where i is the number of decompositions already done (e.g. for $i = 0$ this would be the original signal), t_i the picking time on the corresponding octave, m and c unknowns. Solving the linear least squares problem and setting i = 0 leads to the desired value.

Good values for the parameters k, l, m and n need to be found by prior knowledge and testing.

This procedure can be refined by using a wavelet with a high resolution in the frequency domain (but a poor in the time domain) to do a first round of picking. This way, noise is more probably contained in just a few frequency

bands, but picks are poor as the resolution in the time domain is poor. After this initial round, a second round is done with a wavelet with a high resolution in the time domain (but a poor in the frequency domain). Instead of picking all over again, only fine adjustments of the original picks are made within a tiny window. This way, noise is not that much of a problem any more. The wavelets D64 and D16 have been used for this purpose.

The assessment of the wavelet picker was made against the Baer-Kradolfer picker from ETH (see [3]) and AR-AIC (see [10]) using a predefined set of various events which have been reviewed manually before. All of the algorithms have been run as a plugin to seiscomp (s. [7]). Testing is still ongoing. Some results show that the wavelet picker generally makes less picks than other available pickers and has far less false positives. However, it also misses more picks than other algorithms. Parameters can be tuned for all available algorithms, and depending on the testing metric (how much to punish false positives and how much to punish missed picks) results can be quite different.

A.3 Images

To illustrate the principle and the usefulness of a stationary wavelet transform, two typical seismic signals as well as their static wavelet decomposition are shown. The first one (BRANT) shows a typical signal of a remotely located station which is distant to the earthquake. Most of the noise is low-periodic. Also note that little energy is in the high frequency part of the earthquake signal. This shows that the earthquake is at quite some distance, as the earth acts as a low-pass filter. The second shows a typical signal of a station located close to houses and close to the earthquake (WILA). A lot of high periodic noise shows the presence of civilization (cars, machines etc.). Also note that the energy is spread over the spectrum, this shows that the earthquake has happened close. If the distance between earthquake and station were bigger, less energy would be in the high frequency part of the earthquake signal. The colors have been adjusted to use the full possible range and do not correspond with specific exact values.

The method on how to graphically display the information is taken from [2].

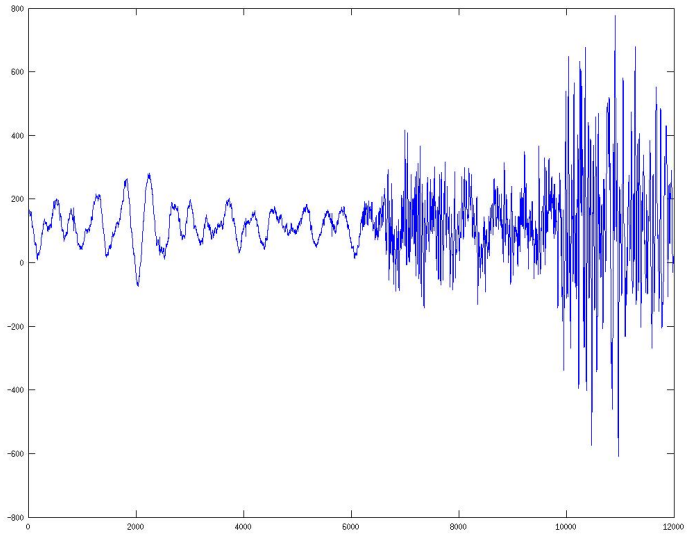

Figure 17: Seismic signal of station BRANT, vertical component, x axis in 1/120s, y axis velocity in raw counts

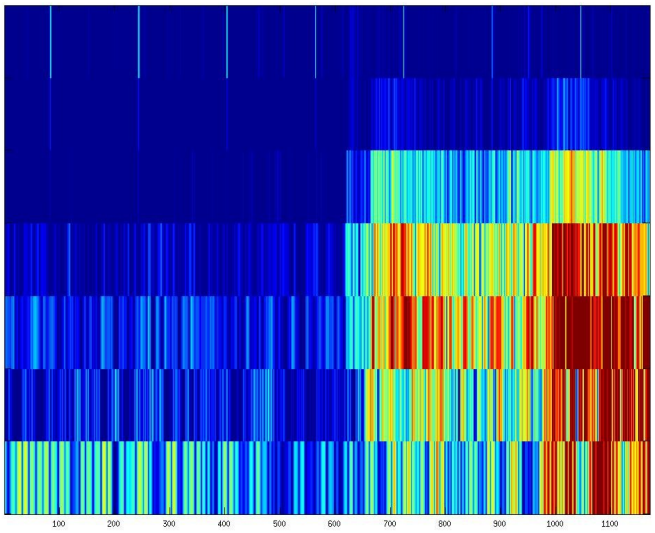

Figure 18: Scalogram display of static wavelet decomposed data for station BRANT, x axis: 1/12s, y axis: top highest octave, bottom lowest displayed octave, red: lot of energy, blue little energy

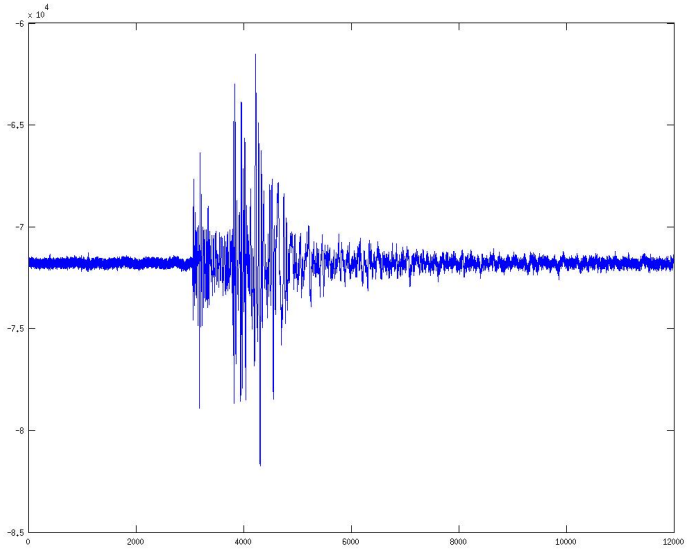

Figure 19: Seismic signal of station WILA, vertical component, x axis in 1/120s, y axis velocity in raw counts

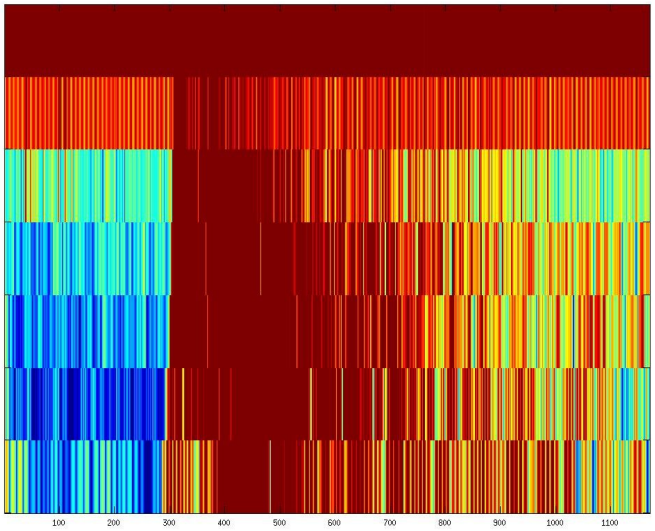

Figure 20: Scalogram display of static wavelet decomposed data for station WILA, x axis: 1/12s, y axis: top highest octave, bottom lowest displayed octave, red: lot of energy, blue little energy

B Software manual

B.1 Overview

B.2 Compiling and installing

The core part of the software consists of C++ template classes. No other libraries than the standard C++ library is needed to compile them, so this should work on every common platform. The web front end is written in Perl and only uses the libraries which are distributed together with the Perl compiler, no additional software installations are required. It makes use of the CGI interface. The synthetic traces have been created by a Matlab routine delivered by Ivo Sbalzarini. The testing routines to drive the test benches partially make use of the Perl library Math::Random which is not part of the core distribution of Perl but is available as a separate package on common Linux platforms and on http://www.cpan.org. However, this part is not necessary for using the software.

Compiling the software should be as simple as changing to the source directory and typing "make" and "make install" which will copy the files to /usr/local/bin.

To install the web interface, copy web interface.pl to a web server directory and make sure that CGI is enabled for the particular web directory. You might need to adjust the parameters in the first section of the script according to your needs (maximum file size, location of the hurst estimator binary). As executing the web interface script with large volumes of data can impose a heavy load on the system, you will probably want to limit access to the script to particular users, e.g. by password protection or any other mechanism supported by the web server.

B.3 Use

B.3.1 Binaries

After compiling, the following binaries should have been produced:

- **hurst**: This is the main binary. A help message is displayed when run without parameters.
- **test_1d**: This is a binary for testing the library against 1D input data. This binary is designed to be driven through testbench_1d.pl.
- **test_2d**: This is a binary for testing the library against 2D input data. This binary is designed to be driven through testbench_2d.pl.
- **static_wavelet_transform**: This binary takes a time series as input and creates output which can then in turn be used as input for the Matlab "image" command to produce visual output of a static wavelet transform. The chosen parameters are optimized to display seismic waveform signals in a format which is typical at my working place. In case of other inputs, the routines will need to be fine-adjusted in order to produce an optimal visual result.

B.3.2 Web service

The web service should be pretty straightforward to use. Files can be uploaded in plain text format. The following input format is expected: In case of an 1D algorithm, the input format might be two values on one line or one single value on a line. In case of a single value on a line, the whole input file is treated as a single time series, in this case, the output will be the estimated hurst exponent for all chosen algorithms.

In case of two values per line, the first value is assumed to be an index and the second one the corresponding value. In case an index is lower than its predecessor, the program assumes that a new data set has begun. In case of multiple data sets, the output value will be the average value, the standard deviation, the lower and upper 95% confidence interval.

In case a 2D algorithm has been chosen, in case of two values per line, the algorithm assumes a single input set, in case of three values per line, the first value is treated as an index. In all other aspects, the same information is displayed as in the 1D case. 1D and 2D algorithms cannot be mixed on the same data set (this would make absolutely no sense anyway).

Blanks, tabulators, commas and semicolons or any combination of them are accepted as separators. In case the input is invalid or too short for a certain hurst estimator, the result will be -1.

B.4 Delivered files

As a general remark, the standard method is called "MSS" in all of the delivery files as well as in the source code.

- **measurement_output**: The measurement output resides here. The files result_1d*.txt contain the one dimensional measurements, result_2d*.txt the two dimensional measurements. For the one dimensional case, noise measurements have been made as described in section 7.5. In this case, the value at the end of the file name describes the level of noise introduced into the data, e.g. result_1d_0.2.txt is the measurement output for multiplicative noise distributed as $\mathcal{N}(1, 0.2)$. The output files are formatted as follows: For every measured combination of simulated H and input length, a section starts with H l. After this on every subsequent line of the section, a line is formatted like this: Name of the estimator, average, standard deviation, $\frac{stddev}{avg}$, lower bound of the 95% confidence interval, upper bound of the 95% conficence interval, running time.

  Example: RS: 0.12461 0.01705 0.13682 0.11871 0.13052 2.18125

  This describes a measurement with the rescaled range algorithm with average 0.12461, standard error 0.01705, $fracstddevavg = 0.13682$, lower bound of the conficence interval 0.11871, upper bound 0.13052 and run time 2.18125s.

- **src**: The source files as well as the perl scripts as described earlier in this section reside here. The directories data_1d and data_2d contain the synthetic traces used to test the algorithms as gzipped files. The content of the files is coded like this: 0.2_10_27.data.gz means the file contains a synthetic trace with $H = 0.2$, containing 2^{10} data points. 27 is the run

number. data_real contains real world data from biological applications in 2D. data_seismic contains two sample files for static_wavelet_transform. The directories html and latex are generated by doxygen.

- **report**: Contains this thesis as well as the source files.

References

[1] Wikipedia article on rescaled range.

[2] W. Bäni. *Wavelets, eine Einführung für Ingenieure (2. Auflage)*. Oldenbourg, München, 2005.

[3] M. Bär and U. Kradolfer. An automatic phase picker for local and teleseismic events. *Bulletin of the Seismological Society of America*, 77(4):1437–1445, 1987.

[4] Arnaud Gloter and Marc Hoffmann. Estimation of the hurst parameter from discrete noisy data. *Annals of Statistics*, 35(5):1947–1974, 2007.

[5] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[6] C.L. Jones, G. T. Lonergan, and D. E. Mainwaring. Wavelet packet computation of the hurst exponent. *Journal of Physics A: Mathematical and General*, 29, 1996.

[7] Geoforschungszentrum Potsdam. Seiscomp3, http://www.seiscomp3.org.

[8] John A. Rice. *Mathematical Statistics and Data Analysis, Third Edition*. Thomsoon Brook/Cole, 2006.

[9] Ivo F. Sbalzarini. Moments of displacement and their spectrum.

[10] R. Sleeman and T. van Eck. Robust automatic p-phase picking: An online implementation in the analysis of broadband seismogram recordings. *Physics of the earth and planetary interiors*, 113:265–275, 1999.