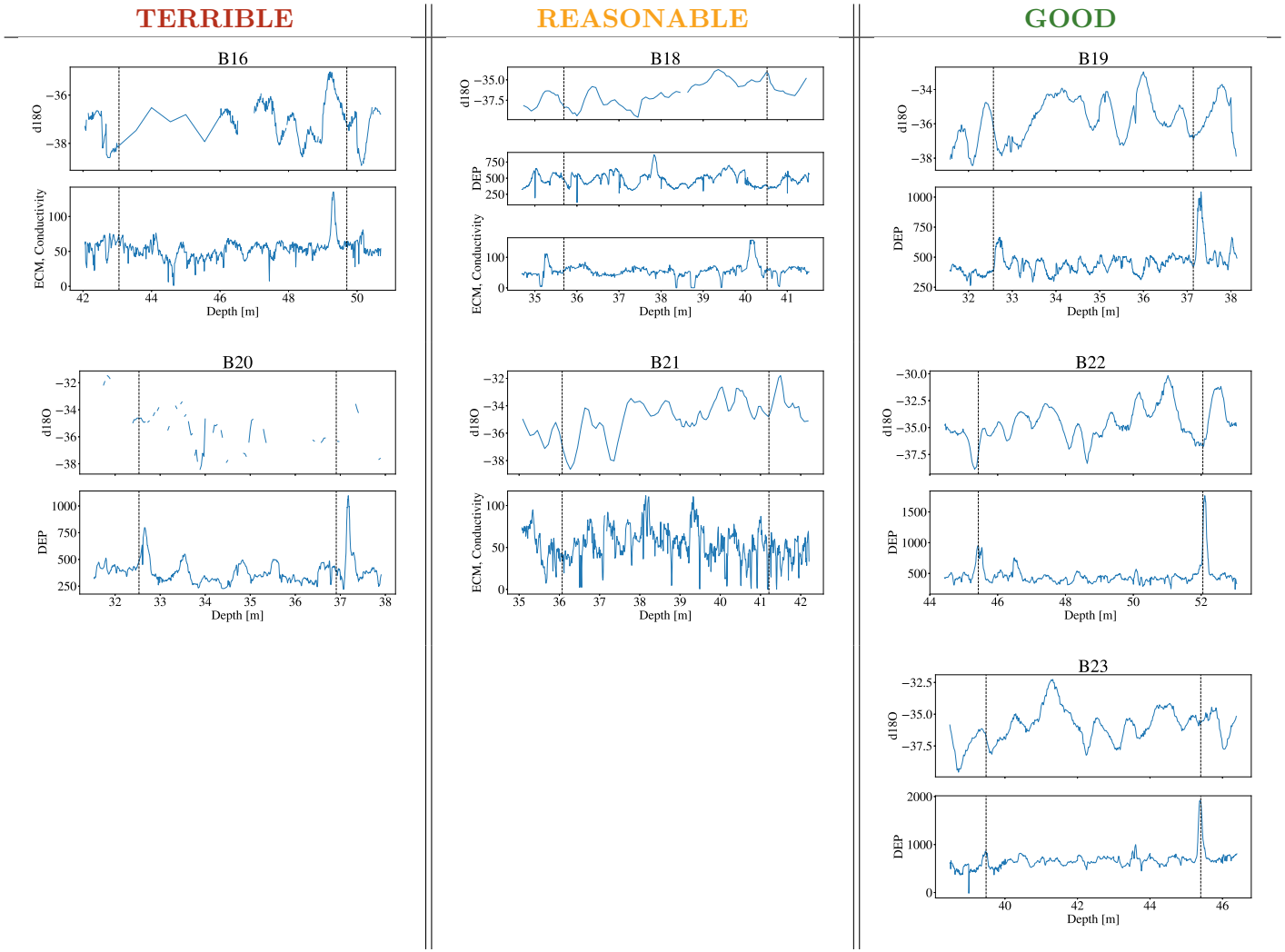
0.1 APPENDIX I: Firn Diffusivity

0.2 APPENDIX II: Data - AWI B-cores


TERRIBLE
REASONABLE
GOOD
B16
B18
B19
B20
B21
B22
B23
d18O
DEP
ECM, Conductivity
Depth [m]

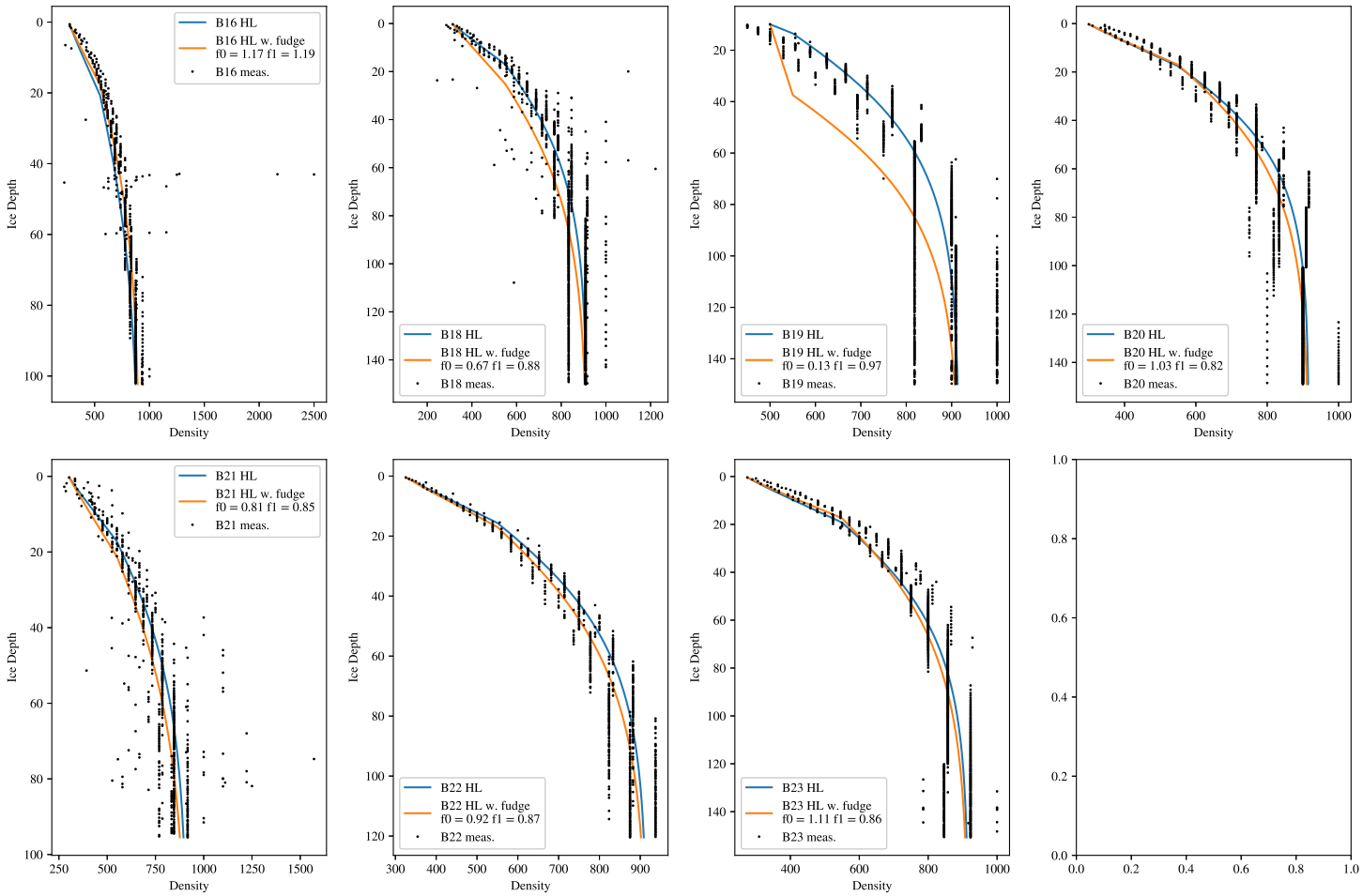


Figure 0.1: AWI B-cores density measurements and HL-modeled densities.

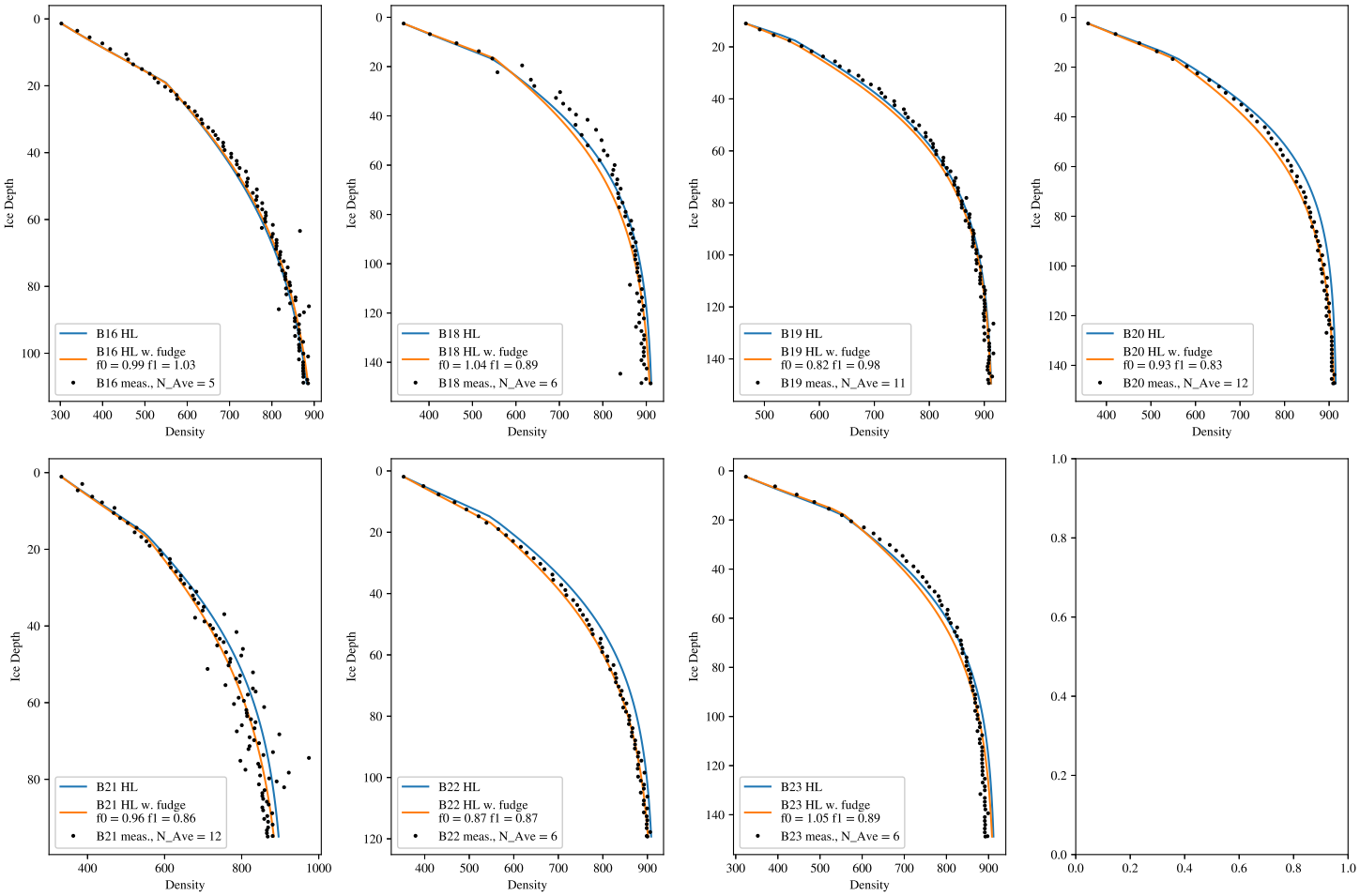


Figure 0.2: AWI B-cores averaged density measurements and related HL-modeled densities.

0.3 APPENDIX III: Data - Alphabet Cores

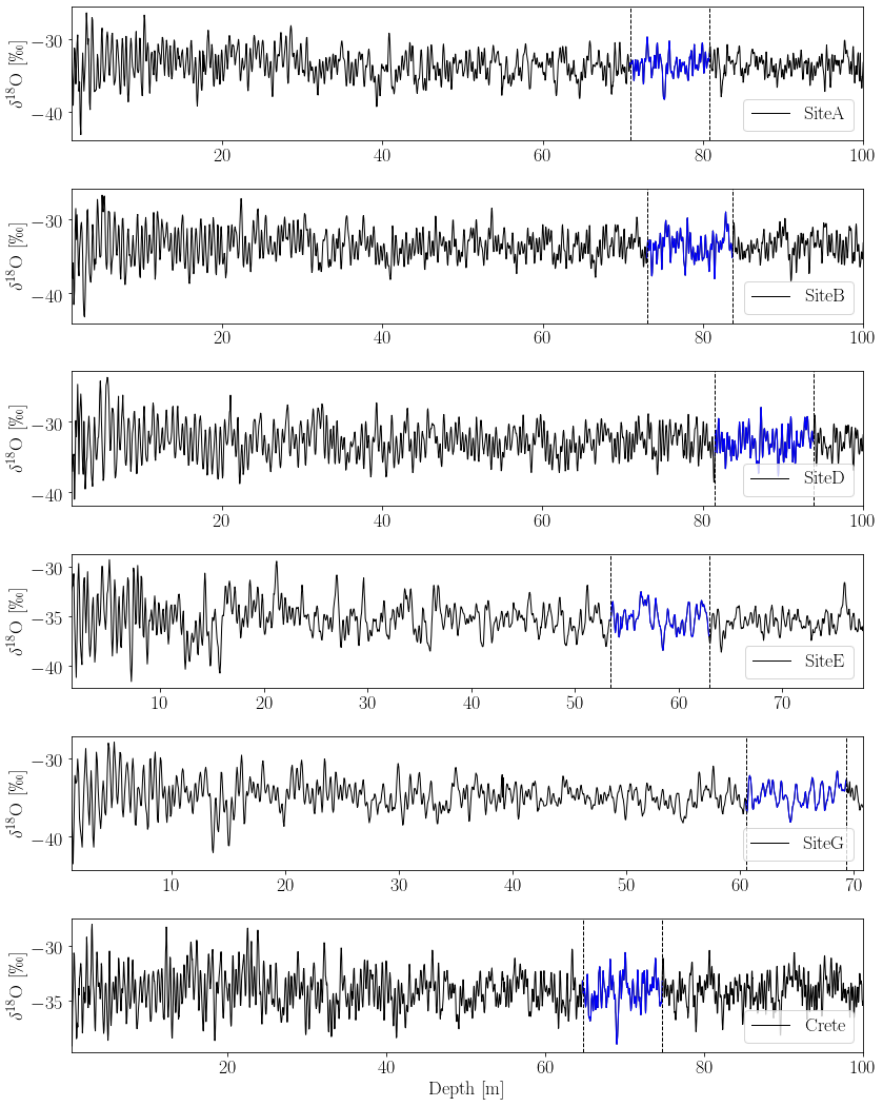


Figure 0.3: All six Alphabet cores under examination, with Laki to Tambora depth section highlighted.

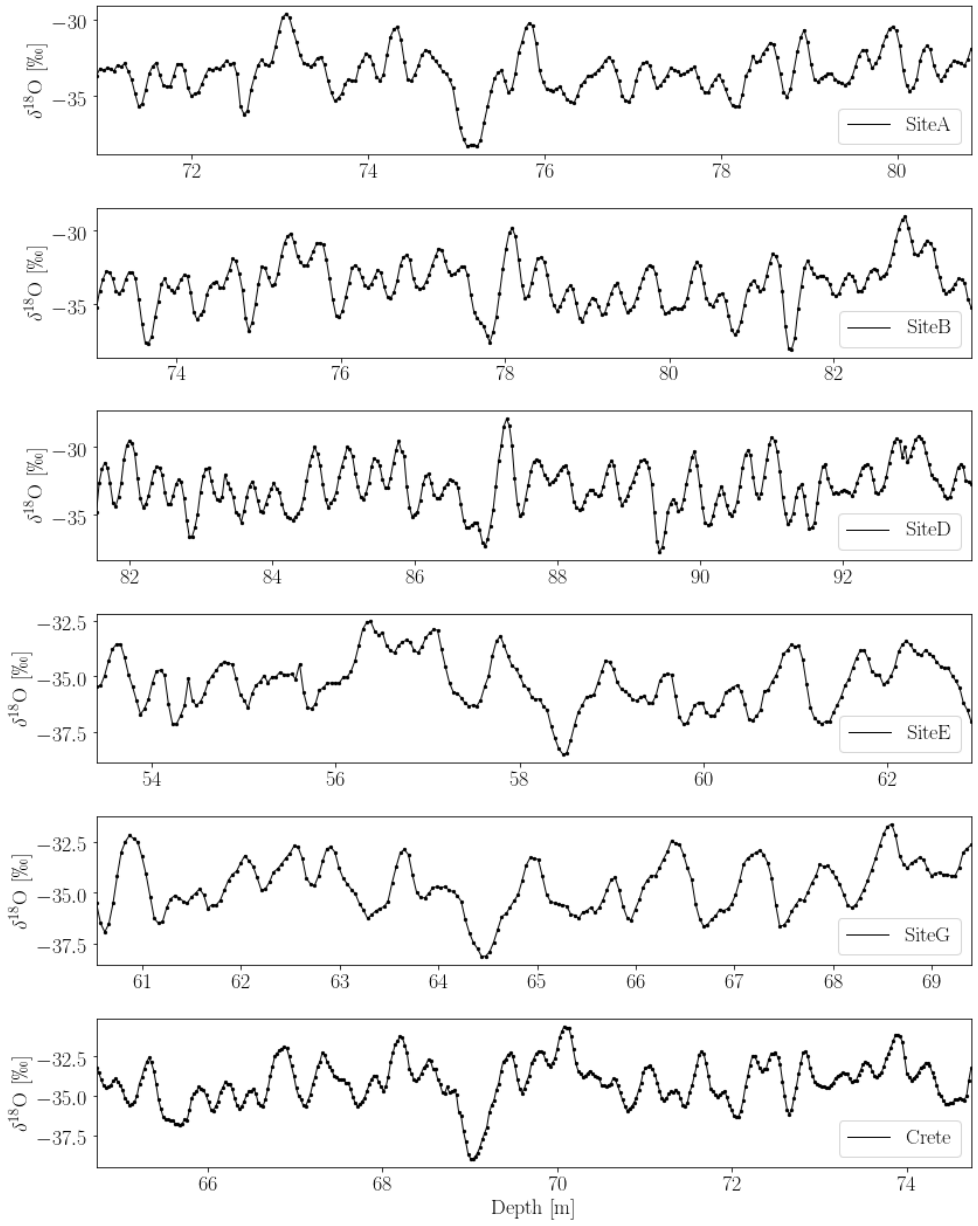


Figure 0.4: All six Alphabet cores, showing only the depth series concerning the Laki to Tambora section.

0.4 APPENDIX IV: Spectral Transforms

0.4.0.1 Discrete and Fast Fourier Transform

The definition of the continuous Fourier transform and its inverse was presented in the above. The Fourier transform is as seen a way of representing the function under consideration as an infinite sum of periodic components. When the function is discrete, so will the Fourier transform be, and the integral is replaced with a sum. This gives us the Discrete Fourier Transform (DFT) which transforms the signal into a sum of separate components contributing at different frequencies. The DFT is dependent on the sampling interval, Δ, and we can describe our discrete signal X as a function of N discrete time steps $t_k = k \cdot \Delta$, where $k = 0, 1, ..., N - 1$:

$$X_k \equiv X(t_k) \tag{1}$$

This sample size is supposed to be representative for the entire discrete function, if the function continues beyond the N sampled points. When sampling discretely at interval Δ, there will be a special frequency, the Nyquist critical frequency, defined through the sampling size as:

$$f_{NQ} \equiv \frac{1}{2\Delta}. \tag{2}$$

This frequency is of great importance in transformation of discrete signals. If the continuous signal is sampled at an interval Δ is bandwidth limited to frequencies smaller in magnitude than f_{NQ}, $\tilde{X}(f) = 0$ for $|f| \geq f_{NQ}$ - i.e. the transformed function has only non-zero values inside the Nyquist interval, $\tilde{X}(-f_{NQ}), ..., \tilde{X}(f), ..., \tilde{X}(f_{NQ})$. This means that the function is completely determined since we have all information about the signal contained in our available frequency space.
On the other hand, which is much more likely, if the continuous signal consists of frequencies both inside and outside the Nyquist interval, then all spectral information outside of this range will be falsely interpreted as being inside this range. Thus a wave inside the interval with a frequency of f_n will have a number of wave siblings outside of the interval, with frequencies of $k \cdot \frac{1}{\Delta} f_n$, k being integers, which will be aliased into the Nyquist interval and give rise to an increased power at the frequency f_n.
When analyzing an already measured discrete signal, this might give rise to some headache. What can be done is to assume that the signal has been sampled competently and then assume that the Fourier transform is zero outside of the Nyquist interval. After the analysis it will then be possible to determine if the signal was indeed competently sampled, as the Fourier series will

go to zero at f_{NQ} given a correct assumption, and go to a fixed value, if the sampling was not done competently.
Now with the basics of understanding the limits of frequency transform of a discretely sampled signal, it is possible to estimate the DFT of the signal $X_k \equiv X(t_k)$. Since the Fourier transform is a symmetric transformation it is easiest to assume that N is even.

Since the input information is of size N we should expect only to sample the frequency transform $\tilde{X}(f)$ at only discrete values of f in the range between the upper and lower critical Nyquist frequencies, $-f_{NQ}$ to f_{NQ}:

$$f_n \equiv \frac{n}{N\Delta}, \qquad n = -\frac{N}{2}, ..., \frac{N}{2} \tag{3}$$

This will indeed actually give rise to $N + 1$ values, since 0 will be in the interval as well, but the limit frequencies are actually not independent, but all frequencies between are, which reduces it to N samples.
Now the integral from Equation **??** needs to be estimated as a sum:

$$\tilde{X}(f_n) = \int_{-\infty}^{\infty} X(\tau) e^{2\pi i f_n \tau} dt \approx \sum_{k=0}^{N-1} X_k e^{2\pi i f_n t_k} \Delta = \Delta \sum_{k=0}^{N-1} X_k e^{2\pi i k \frac{n}{N}} \tag{4}$$

The Discrete Fourier Transform is thus defined as:

$$\tilde{X}_n \equiv \sum_{k=0}^{N-1} X_k e^{2\pi i k \frac{n}{N}} \tag{5}$$

This gives the approximate relation between the DFT estimate and the continuous Fourier transform $\tilde{X}(f)$ when sampling at size Δ as:

$$\tilde{X}(f_n) \approx \Delta \tilde{X}_n \tag{6}$$

The inverse DFT is given as:

$$X_n \equiv \frac{1}{N} \sum_{n=0}^{N-1} X \tilde{X}_n e^{-2\pi i k \frac{n}{N}} \tag{7}$$

Computation of the DFT can be very slow and tiresome, since it involves complex multiplication between a number of vectors and matrices. If we write Equation 5 as $\tilde{X}_n = \sum_{k=0} N - 1 W^{nk} X_k$, where W is a complex number $W \equiv e^{2\pi i/N}$. This shows that the vector X_k must be multiplied with a complex matrix which (n,k)th component consists of the constant W to the power

of nk. This matrix multiplication evidently leads to a process of $O(N^2)$. Fortunately, a number of different algorithms implementing a wide range of different theories from complex number arithmetic and prime-factoring to group and number theory ([**?**],[**?**], [**?**] and others) have been developed for fast and efficient computation of the discrete Fourier transform. One of these is called the Fast Fourier Transform (FFT), which can reduce the computations to just $\mathcal{O}(N \log_2 N)$. In this thesis the FFT used is the one implemented in the `scipy.fft` Python package [**?**], which is based on the works of [**?**]. See said article for implementation details. One important thing about this specific algorithm is that for the algorithm to function most efficiently, the number of points computed in the frequency space must be of a power of 2, following the use of base $\log_2$

0.4.0.2 Nonuniform Discrete Fourier Transform

All FFT algorithms evaluates the DFT definitions from Eqs. 5 to 7 in fast and efficient ways. But one key assumption for these methods is that the data under examination are equispaced, i.e. uniformly distributed, based on the summation in Eq. 5. The computations thus expect uniform data as input and returns uniform data as output. Unfortunately this is not always the case for data collected in physical experiments. In this case the basic assumptions for the calculations of both DFT and FFT are flawed.
The most general form of a nonuniform transform would be the one that takes non-equispaced data as input and also returns non-equispaced transforms as output. Firstly we wish to create a nonuniform discrete Fourier transform (NUDFT) that transforms a sequence of N complex numbers $X_0, ..., X_{N-1}$ to a different sequence of M complex numbers, $\tilde{X}_0, ..., \tilde{X}_{M-1}$. The one-dimensional NUDFT the computes the transformed vector $\tilde{\boldsymbol{X}} = (\tilde{X}_0, ..., \tilde{X}_{M-1})^{\mathrm{T}}$, with entries computed as the sum

$$\tilde{X}_k = \sum_{n=0}^{N-1} X_n e^{-2\pi i p_n f_k}, \qquad 0 \leq k \leq M-1. \tag{8}$$

The values $X_0, ..., X_{N-1}$ are sample values, $p_0, ..., p_{N-1}$ are sample positions and $f_0, ..., f_{M-1}$ are frequencies. The NUDFT vector $\tilde{\boldsymbol{X}}$ is found by computing M sums with each N terms. This meaning that the computational cost will be of order $\mathcal{O}(M \cdot N)$, and if $M = N$ then of $\mathcal{O}(N^2)$. The NUDFT reduces to the DFT if the points are equispaced, $p_n = \frac{n}{N}$, and the frequencies are integers, $f_k = k$, and can be computed at the cost of the FFT, $\mathcal{O}(N \log_2 N)$. In the literature there are many who have presented different ways to develop a

fast NUDFT ([?], [?], [?], [?] among others), generally referred to as NUFFT or NFFT. In this work though, the main focus is on the discrete cosine transform, and the NFFT methods will not be described in depth

0.4.0.3 Discrete Cosine Transform

The Fourier transform in any of its many forms is designed to process complex-valued signals, always producing a complex-valued spectrum, even for signals that were strictly real-valued. The real-valued or complex-valued part of the Fourier spectrum is on their own not enough to represent the full information of the signal, since neither the cosine nor he sine functions (corresponding to the real and the complex parts of the spectrum respectively), constitute a complete set of basis functions. Nonetheless, a purely real-valued signal has a symmetric Fourier spectrum, meaning that it is only necessary to compute half the number of spectral coefficients, without losing any signal information. Since the signals analyzed in this thesis are strictly real, one way to use this knowledge to improve on the works of this project is to consider a different, less expensive, purely real spectral transform. The cosine transform [?] seems to do the trick: it uses only cosine functions as basis functions and operates with only real-valued signal and spectral coefficients, and have properties similar to the Fourier transform.
For the discrete version of the cosine transform, DCT, and its inverse, IDCT, a number of different definitions have been proposed, but for this work, the originally formulations by [?] are used. These are often referred to as "The DCT" and "The IDCT", and other times as DCT-II and DCT-III. The entries of the computed discrete cosine transform vector,$\tilde{X}_0, ..., \tilde{X}_{M-1}$, of a real-valued signal of N data points, $X_0, ...X_{N-1}$, is computed as

$$\tilde{X}_k = 2 \sum_{n=0}^{N-1} X_n \cos\left(\frac{\pi(2n+1)k}{2N}\right), \qquad 0 \leq k < M. \tag{9}$$

To orthonormalize the base functions,$\phi_k(n)$, the coefficients are multiplied by a scaling factor f:

$$f = \begin{cases} \frac{1}{\sqrt{2N}}, & \text{if} k = 0 \\ \frac{1}{\sqrt{4N}}, & \text{otherwise} \end{cases} \tag{10}$$

so that the base functions, $\phi_k[n] = 2f \cos\left(\frac{\pi(2n+1)k}{2N}\right)$, meet the condition:

$$\sum_{n=0}^{N-1} \phi_k[n]\phi_l[n] = \delta_{lk}. \tag{11}$$

The inverse of the DCT, the so called DCT-III, is defined, unnormalized, as:

$$X_k = \tilde{X}_0 + 2\sum_{n=1}^{N-1} \tilde{X}_n \cos\left(\frac{\pi n(2k+1)}{2N}\right), \qquad 0 \leq k < N \tag{12}$$

and orthonormalized:

$$X_k = \frac{\tilde{X}_0}{\sqrt{N}} + \sqrt{\frac{2}{N}}\sum_{n=1}^{N-1} \tilde{X}_n \cos\left(\frac{\pi n(2k+1)}{2N}\right), \qquad 0 \leq k < N \tag{13}$$

Only when the DCT-III is orthonormalized is it exactly the inverse of the orthonormalized DCT-II. If they are both unnormalized, the DCT-III is the inverse of the DCT-II up to a factor $2N$. As with the DFT, the DCT can directly be computed at a cost of $\mathcal{O}(N \cdot M)$, and can also reduced to $\mathcal{O}(N \log N)$. The fast DCT algorithm(FCT) used here is based on [**?**] as it is implemented in the `scipy.fft.dct` package[**?**].

0.4.0.4 Nonuniform Discrete Cosine Transform

Again, as with the FFT, the FCT works under the key assumption that data is equispaced. Though when data is nonuniform, the DCT is described as:

$$\tilde{X}_k = 2\sum_{n=0}^{N-1} X_n \cos\left(2\pi f_k\left(p_n + \frac{1}{2N}\right)\right), \qquad 0 \leq k < M-1 \tag{14}$$

with, in the most general case, nonuniformly spaced signal, $p_o, ..., p_{N-1}$, data and frequency data, $f_0, ..., f_{M-1}$. The inverse of NDCT, the INDCT, is computed as:

$$X_k = \frac{\tilde{X}_0}{\sqrt{N}} + \sqrt{\frac{2}{N}}\sum_{n=1}^{N-1} \tilde{X}_n \cos\left(\left(p_n + \frac{1}{2N}\right)2\pi f_k\right), \qquad 0 \leq k < N-1 \tag{15}$$

It is possible to develop algorithms with the computational cost of $\mathcal{O}(N \log N)$ for NDCT and INDCT as it is for the NDFT ([**?**], [**?**]) but it has showed to be out of the scope of this project and has not been implemented. This of course slows down the final optimization algorithm, as it requires a number of spectral transformations. In Section **??** it is described how the final algorithm has been designed to minimize the use of NDCT, and thus speeding up the final computations.

0.4.0.5 Maximum Entropy Method (Burg's Method)

SIGNAL-MEM: Write this entire section - maybe not necessary? Maybe use in reconstruction of missing data...

0.5 APPENDIX V: Splines and Interpolations

0.5.0.1 Existence, Uniqueness and Conditioning

Considering any attempt to create an interpolant to fit a number of data points, the questions of uniqueness and existence is a matter of matching the data points with the number of parameters in the interpolant. If there are too few parameters, the interpolant does not exist, as it will not pass through all data points. If there are too many, the interpolant will not be unique. Formally this can be described through a system of linear equations.
For any data set consisting of $(t_i, y_i),\ i = 1, ..., m$ points, an interpolant can be chosen from a function space spanned by some suitable set of basis functions, $\phi_1(t), ..., \phi_n(t)$. The interpolant can then be described as a linear combination of these basis functions:

$$f(t) = \sum_{j=1}^{n} x_j \phi_j(t) \tag{16}$$

The interpolant can then be found by determining the parameters x_j by requiring that the interpolant f must pass through the M data points (t_i, y_i):

$$f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i) = y_i, \quad i = 1, ..., m \tag{17}$$

This can of course also be written compactly in matrix form as a system of linear equations:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \tag{18}$$

In this equation $\boldsymbol{A}$ is the $m \times n$ basis matrix, which entries consists of the value of the n basis functions evaluated at the m data points, $a_{ij} = \phi_j(t_i)$, the m vector $\boldsymbol{y}$ consists of the known data values y_i, and the n vector $\boldsymbol{x}$ consists of the unknown, to be determined, parameters x_j.
From linear algebra we know, that if we choose the number of basis function ot be equal to the number of data points, $n = m$, the basis matrix will be square, and thus - given the matrix is nonsingular - the system will be determined, and the data points can be fit exactly. Though in some problems it is beneficial to choose the system to be either overdetermined(less parameters than data points, the data cannot be fit exactly) or underdetermined(more parameters than data points, giving freedom to allow satisfaction of additional properties or conditions).
So the existence and uniqueness of an interpolant is given by the non-singularity of the basis matrix, be it square or not and the conditioning of the matrix

points to the parameters' sensitivity to perturbations. An ill-conditioned basis matrix will lead to high sensitivity in the parameters, but this problem can still be approximately solvable through Gaussian elimination with partial pivoting, but this solution will mean that the coefficients may be poorly determined.

0.5.0.2 Polynomial Interpolation

The most common way to determine an interpolant is through polynomials. Denoting a set of all polynomials of degree at most k, $\quad k \geq 0$ as $\mathbb{P}_k$, it can be seen that this set forms a vector space of dimension $k+1$. The basis functions that span this vector space can be chosen to be composed of a number of different functions and this choice has a great influence on both the cost of computation and manipulation of the interpolant, and the sensitivity of the parameters, i.e. the conditioning of the basis matrix.
Considering n data points it is obvious to choose $k = n-1$ so that the dimension of the vector space matches the number of data points. The maybe most natural choice of basis for $\mathbb{P}_{n-1}$ is one that consists of the first n monomials[1],

$$\phi_j(t) = t^{j-1}, \quad j = 1, ..., n. \tag{19}$$

Thus any given polynoial $p_{n-1} \in \mathbb{P}_{n-1}$ will be of the form

$$p_{n-1}(t) = x_1 + x_2 t + \cdots + x_n t^{n-1}. \tag{20}$$

In this basis the system of $n \times n$ linear equations will be of the form

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 1 & t_1 & \cdots & t_1^{n-1} \\ 1 & t_1 & \cdots & t_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_1 & \cdots & t_1^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \boldsymbol{y}. \tag{21}$$

COMP-INTERP: REFERENCE!!!

This type of matrix with geometric progression, i.e. the columns are successive powers of some independent variable t is called a Vandermonde matrix. When using the monomial basis and using a standard linear equation solver to determining the interpolants coefficients requires $\mathcal{O}(n^3)$ work and often results in ill-conditioned Vandermonde matrices $\boldsymbol{A}$, especially for high-degree polynomials. This ill-conditioning is due to the monomials of higher and higher degree being more and more indistinguishable from each other. This

[1] Roughly speaking, a polynomial with only one term.

makes the columns of $\boldsymbol{A}$ nearly linearly dependent, resulting in almost singular matrices, and thus highly sensitive coefficients. For high enough n, the Vandermonde matrix becomes efficiently singular, to computational precision at least, though, as mentioned, this can be worked around, but requires some additional computational work.

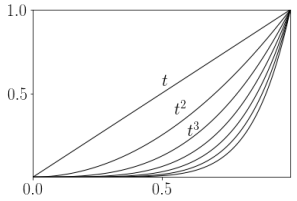


Figure 0.5: Illustration of the first eight monomials.

0.5.0.3 Piecewise Polynomial Interpolation and Splines

The amount of work needed to solve the system as well as the conditioning of the system can be improved by using a different basis all together. Some different bases superior to the monomial that are worth mentioning are the Lagrange basis functions, the Newton basis functions and the orthogonal polynomials. But for this thesis we take a step further into the interpolation theory, as the choice of basis functions might not be enough to work around some of the problems connected with fitting a single polynomial to a large number of data points(i.e. oscillatory behaviour in the interpolant, nonconvergence or issues around the boundaries).

COMP-INTERP: REFERENCES!!

These practical and theoretical issues can be avoided through the use of piecewise polynomial interpolation, with the advantage that a large number of data points can be fitted with low-degree polynomials.

When turning to piecewise polynomial interpolation of the data points (t_i, y_i), $i = 1, ..., n$, $t_1 < t_2 < \cdots < t_n$, a different polynomial is chosen for each subinterval $[t_i, ti + 1]$. Each point t_i, where the interpolant changes is called knots or control points. The simplest piecewise interpolation is piecewise linear interpolation, where each knot is connected with a straight line. If we consider this simple example it appears that by eliminating the problems of nonconvergence and unwanted oscillatory behaviour, the smoothness of the interpolant is sacrificed. This might be true for this simplistic example but since there are a number of degrees of freedom in choosing each piecewise polynomial interpolant, the smoothness can be reintroduced by explotiting a number of these measures. One way of doing this is by demanding knowledge of both the values and the derivatives of the interpolant at each data pint. This just adds more equations to the system, and thus to have a well-defined solution, the number of equations must match the number of parameters. This type of interpolation is known as Hermite interpolation. The most common choice for this interpolation, to still maintain simplicity and computational efficiency, is cubic Hermite interpolation. This introduces a piecewise cubic polynomial with n knots, and thus $n-1$ interpolants each with 4 parameters to fit, leading to $4(n-1)$ parameters to be determined. Since each of the $n-1$ cubics must match the data points at each end of the subinterval, it results in $2(n-1)$

equations, and requiring the derivative to be continuous, i.e. match at the end points, an additional of $n-2$ equations are taken in. This leads to a system consisting of $2(n-1)+(n-2) = 3n-4$ equations to fit to the $4n-4$ parameters. This leaves n free parameters, meaning that a cubic Hermite interpolant is not unique and the remaining free parameters can be used to accommodate further or additional constraints that might be around the problem at hand.

Cubic Spline Interpolation

A spline is a piecewise polynomial of degree k that is continuously differentiable $k-1$ times.

One way of using the remaining free parameters is by introducing *splines*. A cubic spline is, given the spline definition, a piecewise cubic polynomial, a polynomial of degree $k = 3$, and must then be $k-1 = 2$ times differentiable. Thinking back on the Hermite cubic, we were left with n free parameters. By demanding continuity of also the second derivative, we introduce $n-2$ new parameters, leaving only 2 final parameters to be free. These 2 remaining parameters can be fixed through a number of different requirements, e.g. by forcing the second derivative at the endpoints to be zero, which leads to the *natural* spline.
The Hermite and spline interpolations are useful for different cases. The Hermite cubic might be more appropriate for preserving monotonicity if it is known that the data are monotonic. On the contrary, the cubic spline may enforce a higher degree of smoothness as it takes the second derivative into account as well.
For the case of this study, cubic spline interpolation is used to either evenly redistribute slightly unevenly sampled data or to enhance resolution for more precise peak detection. The general method for cubic spline interpolation used here is described in the following.

Assuming the original depth array $\mathbf{d}$ is distributed as $d_{i-1} < d_i < d_{i+1}$ with $i = 0, ..., n-1$ has a minimum sampling distance as $\Delta_{\min}$ we define the new sampling distance for the new depth array $\hat{\mathbf{d}}$ as $\Delta = \Delta_{\min}$ - again assuming that $\hat{\mathbf{d}}_{j-1} < \hat{\mathbf{d}}_j < \hat{\mathbf{d}}_{j+1}$ with $j = 0, ..., \hat{n}-1$. This makes it possible to define the first and last value of the new array as

$$\hat{\mathrm{d}}_0 = \Delta \lceil \frac{\mathrm{d}_0}{\Delta}, \rceil \tag{22}$$

$$\hat{\mathrm{d}}_{\hat{n}-1} = \Delta \lfloor \frac{\mathrm{d}_{n-1}}{\Delta} \rfloor. \tag{23}$$

From this the number of values in the new array, $\hat{n}$, can be determined as

$$\hat{n} = 1 + \frac{\hat{\mathrm{d}}_{\hat{n}-1} - \hat{\mathrm{d}}_0}{\Delta}, \quad \hat{n} \in \mathbb{Z}. \tag{24}$$

Thus our new depth array will be given as

$$\hat{\mathbf{d}} = \hat{\mathrm{d}}_0 + j \cdot \Delta, \qquad j = 0, ..., \hat{n} - 1. \tag{25}$$

The original data are then used to define a cubic spline interpolation function to which the redistributed depth data points can be matched. For this part of the data analysis the `SciPy.interpolate` Python (REFERENCE) package with `SciPy.interpolate.CubicSpline` for the cubic spline interpolation.

0.6 APPENDIX VI: Parallelization

COMP-PARAL: Do some actual parallelization! And write this entire section

0.7 APPENDIX V: Splines and Interpolations