1. **Stochastic complexity:** In this question, you will see numerically that with $N$ data points sampled from a parametric source, the optimal number of representation levels scales as $\sqrt{N}$. To see this, consider a length-$N$ binary sequence $X$; note that $X_i \in \{0, 1\}$. Moreover, the probability that a symbol $X_i$ is 1 will be governed by an (unknown) parameter $\Theta$. Because $\Theta$ is unknown, our approach to encoding the sequence $X$ proceeds as follows. First, we count the number of ones and zeros in $X$, which are denoted by $N_1$ and $N_0$, where $N=N_0+N_1$. Second, we quantize $N_1/N$ to the nearest representation level (the set of all possible levels will be described later). Third, with $K$ levels we encode the index $k$ corresponding to representation level $r_k$ using $\log_2(K)$ bits; what we have described so far is known as the first part (encoding the estimated parameter) of a two-part code. In the second part of a two-part code, we encode the actual sequence $X$ using $\text{len}_\Theta(X) = -N_0 \log_2(1-r_k) - N_1 \log_2(r_k)$ bits. The rationale for this expression for $\text{len}_\Theta(X)$ is that the symbol 1 appears $N_1 \approx N r_k$ times, each time it appears its probability is $r_k$, requiring a coding length of $-N_1 \log_2(r_k)$; in an analogous fashion, 0 appears $N_0$ times and is encoded with length $-N_0 \log_2(1-r_k)$ bits. (At this stage you may be scratching your heads and wondering how we can encode ones and zeros with a non-integer number of bits. It turns out that for sequences of characters $X$, it is indeed possible to encode the entire sequence $X$ in a way that the coding length is roughly $-\log_2(\text{Pr}(X))$, where $\text{Pr}(X)$ is the probability of the entire sequence. This technique is called arithmetic coding, and you may want to read about it.)

To keep things simple, let’s say that the coding length is $\log_2(K) - N_0 \log_2(1-r_k) - N_1 \log_2(r_k)$. Note, however, that if $\Theta$ was known, then the coding length would be $N$ times the entropy of $\Theta$, $H(\Theta) = -\log_2(\Theta) - (1-\Theta) \log_2(1-\Theta)$. The *redundancy* of the two-part code, or excess coding length above the entropy, is defined as $R(X, \Theta) = \log_2(K) + \text{len}_\Theta(X) - N H(\Theta)$, which can be simplified as follows,

$$R(X, \Theta) = \log_2(K) - N_0 \log_2(1-r_k) - N_1 \log_2(r_k) + N (1-\Theta) \log_2(1-\Theta) + N \Theta \log_2(\Theta)$$

$$= \log_2(K) - N (1-\Theta) \log_2(1-r_k) - N \Theta \log_2(r_k) + N (1-\Theta) \log_2(1-\Theta) + N \Theta \log_2(\Theta)$$

$$= \log_2(K) + N (1-\Theta) \log_2 \left( \frac{1-\Theta}{1-r_k} \right) + N \Theta \log_2 \left( \frac{\Theta}{r_k} \right).$$

Your goal in this question will be to show numerically what range of values for $K$ makes sense. To do so, plot the redundancy as a function of two variables. On the horizontal axis, vary the parameter $\Theta$...
from 0 to 1. (You will see that the redundancy may become somewhat larger at the extremes where \( \Theta \) approaches 0 or 1, but should focus on the interior \((0,1)\).) On the vertical axis, vary the number of bins \( K \) from 1 to \( N \), where you will plot things as a function of \( \log(K) \). For each \( K \), define the set of possible representation levels as \( 0.5/K, 1.5/K, \ldots, (K-0.5)/K \); it can be seen that there are \( K \) possible representation levels. For each \((\Theta, K)\) pair, compute the redundancy \( R \). Once you’ve populated the entire \( R \) matrix, your two dimensional plot can be something like `imshow(R)`, where the horizontal axis will correspond to \( \Theta \), the vertical to \( \log(K) \), and the color or gray shading in the image will reflect the redundancy. For each bin \( K \), there will be \( K \) representation levels which will affect \( r_k \). So the redundancy calculations will vary in each bin \( K \) both because of values of \( K \) as well as the value of \( r_k \) chosen, in that particular bin.

**Note:** There is no need to consider all \( N \) possible values of \( K \). Because you’re plotting \( K \) on a logarithmic plot, it will be fine to space out the values of \( K \) being considered geometrically. What you should see is that when \( K \) is close to \( \sqrt{N} \), the redundancy \( R \) will be close to \( 0.5 \log_2(N) \). In contrast, for large values of \( K \), the redundancy will approach \( \log_2(N) \), because the first part of the two-part code costs roughly \( \log_2(N) \) bits, while the redundancy induced by the second part is small. For small values of \( K \), for example, 2 or 3, the redundancy of the second part of the two-part code could grow as \( O(N) \), which exceeds \( \Theta(\log_2(N)) \). Therefore, what you should see is that typical values of \( R \) are the smallest for \( K = O(\sqrt{N}) \).

Please provide various plots, relevant Matlab/Python code, and discuss your findings.

2. **Error, probability, and coding length:** In the previous question, you saw numerically that the number of parameters, \( K \), should be proportional to \( \log(N) \). While this correspondence may seem reasonable when compressing sequences of bits, it might be more complicated when dealing with real-valued data. To work out the details of a situation where MDL can be applied to more complicated examples, we will delve into details of the example from the supplement on least squares and error distributions. We have two hypotheses, \( H_3 \) and \( H_4 \), which represent possibilities that our data \( T \) (recall that in this example \( X \) are input variables and \( T \) are noisy target variables) was generated by polynomials of order 3 or 4. Let us compare the coding length under both hypotheses.

   a. A single “flag bit” encodes whether we rely on \( H_3 \) or \( H_4 \) when processing the data. For example, we could decide that a flag bit being 0 corresponds to \( H_3 \), and 1 corresponds to \( H_4 \).

   b. Having indicated which of the two hypotheses seems to correspond to the data better, we now encode the parameters of the curve fitting. You can assume that each parameter is encoded using \( 0.5 \log_2(N) \) bits, and having 3 (respectively, 4) parameters will require \( 1.5 \log_2(N) \) (respectively, \( 2 \log_2(N) \)) bits. Parts a and b of our encoding process could be merged together and viewed as the first part of a two part code, which encodes the model \( c \). We could say that there are two model classes \( C_3 \) and \( C_4 \), corresponding to hypotheses \( H_3 \) and \( H_4 \), respectively. The overall model class \( C \) is a union of \( C_3 \) and \( C_4 \), i.e., \( C = C_3 \cup C_4 \). While part a differentiates between \( C_3 \) and \( C_4 \), part b specifies our model \( c \in C \).

   c. We now use the model \( c \) to encode the actual data \( T \). The framework used in this curve fitting problem may seem counter intuitive at first; let’s discuss it carefully. Our goal in part c is to encode the noisy target variables \( T \) given the input variables \( X \). That is, \( X \) is side information, which is available to us when we encode \( T \). Recall that \( t_n = W^T x_n + \text{noise}_n \) where \( x_n \) is input \( n \), \( t_n \) is noisy observation \( n \), \( W \) is a column vector of polynomial weights, \( W^T \) is its transposed version, \( \text{noise}_n \) is the noise in \( t_n \), and \( n \) is the index, i.e., \( n \in \{1, \ldots, N\} \). We want to encode \( t_n \) using \( x_n \), where we have access to \( W \), which has been encoded.

To encode \( T \), we need a probabilistic model for the error, \( \text{noise}_n \). Our probabilistic model will assume that \( \text{noise}_n \) follows a Gaussian distribution with probability density function (pdf),
\[ f(\text{noise}_n) = \frac{1}{\sqrt{2\pi \sigma^2}} e^{-\frac{(\text{noise}_n)^2}{2\sigma^2}}, \]

where \( f(\text{noise}_n) \) denotes the density, and \( \sigma^2 \) is the variance of the Gaussian random variable. (You can estimate \( \sigma^2 \) using the empirical average of squared errors, \( \frac{1}{N} \sum_{n=1}^{N} (\text{noise}_n)^2 \).

To show how this pdf corresponds to probability, suppose that \( \text{noise}_n \) is quantized into bins of width \( \Delta \). Denoting the quantization operator by \( Q(\cdot) \), \( b_n = Q(\text{noise}_n) = \Delta \cdot \text{round}(\text{noise}_n / \Delta) \), where \( b_n \) denotes the quantization bin corresponding to \( \text{noise}_n \). (For example, if the bin width is \( \Delta = 0.5 \) and \( \text{noise}_n = 1.7 \), then \( 1.7 / 0.5 = 3.4 \) is rounded down to 3, and multiplying by the bin width yields 1.5. That is, we round to the nearest multiple of \( \Delta \).) In principle, we could assign each bin a probability corresponding to the probability mass within the bin. Because bin \( b \) (\( b \) is an integer) spans the range \( [\Delta(b-0.5), \Delta(b+0.5)) \), the probability \( P_b \) of bin \( b \) integrates over the pdf within the bin,

\[ P_b = \int_{\Delta(b-0.5)}^{\Delta(b+0.5)} f(\text{noise}_n) \, d\text{noise}_n. \]

In practice, you can approximate \( P_b \) based on the pdf computed at \( \Delta \cdot b \), the middle of the bin. That is, \( P_b \approx \Delta \cdot f(\Delta \cdot b) \). We encode bin \( b \) using \(-\log_2(P_b)\) bits. To summarize part c, encoding \( T \) requires \(-\sum_b \log_2(P_b)\) bits.

Please revisit the curve fitting question of HW1 using these relations between the error, probability, and coding length. To keep things simple, you can assume that the true function is a sine (as in the slides), that model orders 3 and 4 (corresponding to \( H_3 \) and \( H_4 \)) are viable, and the bin size \( \Delta \) can be proportional to the noise amplitude used to generate your data (in the code online it is 0.3, and choosing \( \Delta = 0.1 \) seems reasonable).

Your role is to develop software (Matlab or Python) that selects the optimal model order and outputs the overall coding length for both possible model orders. The existing example code online computes optimal weights \( W \) for various model orders, applies \( W \) to \( X \) to derive predicted target variables \( \text{predict}_n \), and computes the noise corresponding to the predictions, i.e., \( \text{noise}_n = t_n - \text{predict}_n \). Your new code will quantize each \( \text{noise}_n \) to some corresponding bin \( b_n \), estimate the variance \( \sigma^2 \) using the empirical average of squared errors, compute the probability of each bin, and apply the bin probabilities to compute the coding length required to encode \( T \) using \( W \). Finally, adding the overheads required from parts a and b above to specify the model \( c \), which implicitly corresponds to \( W \), your implementation will select between \( H_3 \) and \( H_4 \).

If you are more ambitious, you can evaluate your implementation on other non-linear functions as in the curve fitting question of HW1. You can also evaluate a wider range of model orders, for example \( H_0 \) through \( H_{10} \) (the number of flag bits will need to increase).