Approaches to Curve Fitting

Table of contents

References **1**

1 Introduction **2**

2 The Deterministic Approach **2**
2.1 The Model 2
2.2 Least-Squares Estimation 3
2.3 Overfitting 4

3 A Probabilistic Approach **4**
3.1 The Model 4
3.2 Maximum Likelihood Estimation 5
3.3 Maximum a posteriori Estimation 7
3.4 Bayesian Estimation 9

Appendix **11**
1 Solving for the parameters 11
2 Solving for the parameters with a penalty term 14

References

Pattern Recognition and Machine Learning, **Christopher M. Bishop**, 2006

Bayesian Inference: An Introduction to Principles and Practice in Machine Learning **Michael E. Tipping**, 2004

1 Introduction

In a canonical scenario, we observe a real-valued variable t and we'd like to *model* it. In other words, we'd like to create a process that could plausibly generate t so that we can gather insights and make predictions. In this post, we consider the supervised setting where we model a target variable t as a function of some observed input variable x. To this end, we have gathered a set of N realizations t_n of t together with a corresponding set of realizations x_n of x,

$$\mathbf{x} = \{x_1, x_2, ..., x_N\}^\top, \; \mathbf{t} = \{t_1, t_2, ..., t_N\}^\top.$$

In this post, we will consider synthetic data where the underlying data generating process is

$$y = \sin(2\pi x) + \mathcal{N}(0, \beta^{-1}), \tag{1}$$

where we've denoted the variance as β^{-1} so that $\beta = 1/\sigma^2$ which we call the *precision*. The goal is to exploit our observed data $\{\mathbf{x}, \mathbf{t}\}$ to model the *underlying generative function* (1) to then do something useful; for example, predict unseen values of t for new values of x.

2 The Deterministic Approach

2.1 The Model

In the deterministic approach, we simply consider our target variable t to be a parameterized function of the input variable x and some unknown parameters $\mathbf{w}$. In other words, we assume t is the result of a non-random data-generating process which can be described by a function within the family of functions

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j, \tag{2}$$

where M is the order of the polynomial, and the polynomial coefficients $w_0, ..., w_M$ are collectively denoted by the vector $\mathbf{w}$. Although $y(x, \mathbf{w})$ is a nonlinear function of x, it is linear in the unknown parameters $\mathbf{w}$. Functions that are linear in the unknown parameters have special properties and are called *linear models*.

2.2 Least-Squares Estimation

Although we're trying to model the variable t as a function of the input variable x, we only have realizations of these variables which we've stored in $\{\mathbf{x}, \mathbf{t}\}$. In the deterministic approach we try and find a setting of $\mathbf{w}$ that best agrees with our dataset $\{\mathbf{x}, \mathbf{t}\}$ by minimizing some notion of "error". This error measures the misfit between a realized model – the function $y(x, \mathbf{w})$ for a given value of $\mathbf{w}$ – and the training data points. A widely used (and simple) error function is given by the sum-of-squares of the Euclidean distance between predictions $y(x_n, \mathbf{w})$ and the corresponding target values t_n,

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2. \tag{3}$$

The value of $E(\mathbf{w})$ is always non-negative and zero only when $y(x, \mathbf{w})$ passes exactly through every training point. Using the error function (3) in order to find the parameter values for $\mathbf{w}$ is called the method of *least-squares*. It is visualized in Figure 1.

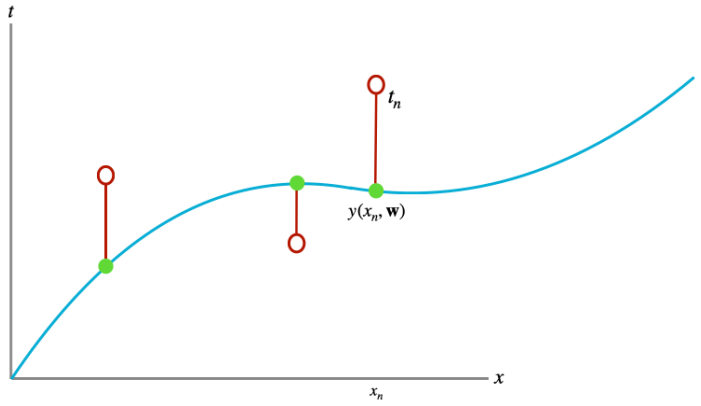

Figure 1: Figure 1: The sum-of-squares error function in (3) is computed by taking one half the sum of the squared distances of each data point from the function $y(x, \mathbf{w})$. These displacements are shown in red.

Solving for $\mathbf{w}$ in this setting is fairly straightforward. Because the error function in (3) is a quadratic function of the parameters $\mathbf{w}$, its derivative with respect to $\mathbf{w}$ will be linear in the elements of $\mathbf{w}$. Therefore, the minimization of the error function has a unique solution, which we denote $\mathbf{w}^*$. It can be found in closed form (Appendix **1**). We can then use $y(x, \mathbf{w}^*)$ to predict new values of the target t for new values of the input x.

2.3 Overfitting

There remains the need to choose the order M of the polynomial, which falls within the realm of *model selection*. Choosing a large M yields a flexible set of models for us to fit, but they are susceptible to *overfitting*. We will see later that the least-squares method represents a special case of *maximum likelihood*, and that overfitting can be viewed as a general symptom of maximum likelihood.

For now, we can continue with a particular method to avoid overfitting – regularization. This involves adding a new term to the error function (3) that penalizes the parameters $\mathbf{w}$ for being too large. The simplest way to do so is to add the sum-of-squares of the weights. This leads to a new error function,

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w})\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2, \tag{4}$$

where $\|\mathbf{w}\|^2 = \mathbf{w}^\top\mathbf{w} = w_0^2 + w_1^2 + \cdots + w_M^2$ and the parameter λ controls the strength of regularization. Like (3), the error function (4) can be minimized in close form (Appendix **2**). Instituting such a penalty term as we did takes on different names depending on the literature. In the statistics literature they are known as shrinkage methods, and in the context of neural networks it is known as weight decay. Lastly, the specific case of (4) is known as ridge regression.

3 A Probabilistic Approach

3.1 The Model

In the deterministic approach we assumed t to be the result of a deterministic function of x and unknown parameters $\mathbf{w}$. We now consider a *probabilistic* model so that we can express uncertainty in our predictions.

In the deterministic approach, we considered t as being the result of a process described by a parameterized function $y(x, \mathbf{w})$. However, we may not want to make such a strong statement as saying t *is exactly equal to* $y(x, \mathbf{w})$. This could be because we think there is noise in the observations $\mathbf{t}$, for example due to measurement error. To introduce such uncertainty, we need to place a distribution over the target variable t. A sensible distributional assumption is to place a Gaussian distribution over t, with its mean given by the parameterized function $y(x, \mathbf{w})$, and its variance being fixed and unknown. This is visualized in Figure 2.

To understand what we're effectively saying when we create such a model, it is useful to re-emphasize and apply the *data-generating process* perspective. We can think of our model as describing a process that produces t from a given x and parameter $\mathbf{w}$. Last section, this

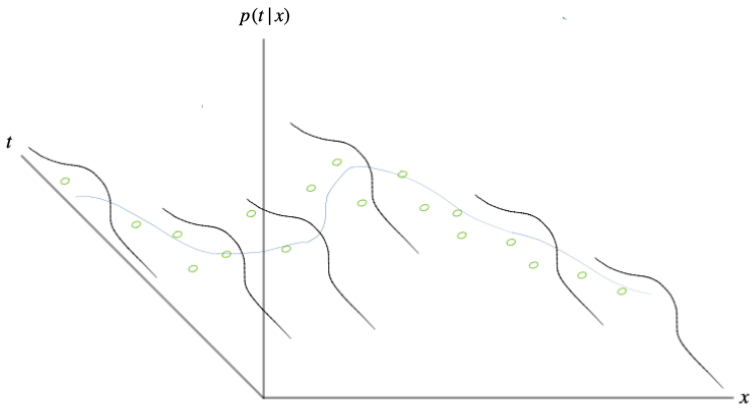

Figure 2: Figure 2: Illustration of a Gaussian conditional distribution over t conditioned on x where the mean of the distribution is given by some function of x, and the variance is fixed

process was a deterministic function $y(x, \mathbf{w})$. In this section, we extend the process by making the assumption that each t is the result of $y(x, \mathbf{w})$ *and some additive uncertainty*, where that additive uncertainty takes the form of a zero-mean Gaussian distribution with unknown variance. This is to say that we are assuming, to have gotten a particular instance of t:

- we are given an instance of x,
- this instance of x is then used to obtain the output of the parameterized function $y(x, \mathbf{w})$,
- to which we add a sample from a zero-mean Gaussian with fixed and unknown variance.

This leads to the following model,

$$\begin{aligned} p(t|x, \mathbf{w}, \beta) &= y(x, \mathbf{w}) + \mathcal{N}(0, \beta^{-1}) \\ &= \mathcal{N}(y(x, \mathbf{w}), \beta^{-1}), \end{aligned} \tag{5}$$

where we've used the scaling property of the Gaussian distribution's mean. (5) is an *observation model* and is more specifically referred to as the *Gaussian noise* model or a *conditional Gaussian* model.

3.2 Maximum Likelihood Estimation

In order to use the training dataset $\{\mathbf{x}, \mathbf{t}\}$ to determine the values of the unknown parameters $\mathbf{w}$ and β, we will use a more general approach than error minimization – *maximum likelihood estimation*. As the name suggests, we will search for a setting of $\mathbf{w}$ and β so that the likelihood of our observed data $\{\mathbf{t}\}$ is maximized. In other words, we've defined a data-generating process,

and we want to find the setting of the parameters so that the likelihood of our process having *created* our observed $\{\mathbf{t}\}$ is maximized.

In order to use maximum likelihood estimation, we need to have a *likelihood function*. A likelihood function is *derived* from an observation model. It can be thought of as an observation model being *applied* to a particular dataset. Assuming the data $\{\mathbf{t}\}$ were independently sampled from (5), the likelihood function is the product of evaluating how consistent the model is with each datapoint (t_n, x_n), and is evaluated for a particular setting of $\mathbf{w}$ and β,

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(x_n, \mathbf{w}), \beta^{-1}). \tag{6}$$

Each time we choose a setting for $\mathbf{w}$ and β and plug them into our model, we are defining a conditional distribution given by (5). This conditional distribution may agree with the dataset we have, or it may not. Examples of agreement and disagreement are shown in Figure 3.

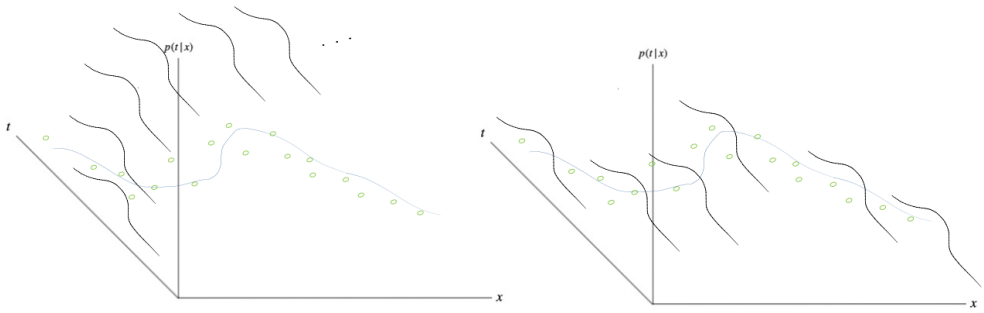

Figure 3: Figure 3: A Gaussian noise model shown for a handful of x_i, with two different settings for $\mathbf{w}$ and β. On the left is a setting of $(\mathbf{w}, \beta)$ that induces a model that disagrees with our observed data. On the right is a setting of $(\mathbf{w}, \beta)$ that induces a model that agrees much better with our observed data. Maximum likelihood looks for a setting of $(\mathbf{w}, beta)$ that best agrees with our observed data.

We now demonstrate how, in practice, we compute the maximum likelihood estimates for $\mathbf{w}$ and β. In this example, it can be done in closed form and amounts to taking the derivative of the likelihood (6), setting it equal to zero, and then solving for $\mathbf{w}$ or β. We begin with $\mathbf{w}$. It is common to instead maximize the log likelihood instead of the likelihood (6) for numerical stability and convenience. We can write the log likelihood as

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = -\frac{\beta}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi). \tag{7}$$

3.2.1 Maximum Likelihood's connection to Least-Squares

In taking the derivative of (7) with respect to $\mathbf{w}$, we can omit the last two terms as they do not depend on $\mathbf{w}$. We can also replace the coefficient $\frac{\beta}{2}$ with $\frac{1}{2}$ since scaling (7) by a constant won't change the location of the maximum of (7) with respect to $\mathbf{w}$. Lastly, we can equivalently minimize the *negative* log likelihood. This leaves us with minimizing

$$\frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2. \tag{8}$$

And so we see that the sum-of-squares error function has arisen as a consequence of maximizing the likelihood under the assumption of a Gaussian noise distribution. In fact, for a *Gaussian noise model*, maximum likelihood estimation and least-squares estimation find the same $\mathbf{w}$; in particular, the one that minimizes (8). Once we've found the maximum likelihood estimate for $\mathbf{w}$, which we will denote $\mathbf{w}_{\mathrm{ML}}$, we can use it to find the setting for the precision parameter β of the Gaussian conditional distribution. Maximizing (7) with respect to β yields

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}_{\mathrm{ML}}) - t_n\}^2. \tag{9}$$

And so we see that the maximum likelihood procedure yields a variance σ^2 being the average squared deviation between the observed data points and the fitted $y(x, \mathbf{w}_{\mathrm{ML}})$.

3.2.2 Maximum Likelihood's predictive distribution

The predictive distribution as a result of the maximum likelihood approach amounts to plugging in the maximum likelihood estimates $\mathbf{w}_{\mathrm{ML}}$ and β_{ML} into the observation model (5):

$$p(t|x, \mathbf{w}_{\mathrm{ML}}, \beta_{\mathrm{ML}}) = \mathcal{N}(y(x, \mathbf{w}_{\mathrm{ML}}), \beta_{\mathrm{ML}}^{-1}). \tag{10}$$

3.3 Maximum a posteriori Estimation

Introducing a prior distribution over the parameters $\mathbf{w}$ is a way of introducing our *prior* beliefs (perhaps through domain expertise) about the parameters before observing our dataset. Additionally, as we will see, it serves as a *regularizer* for our estimate of $\mathbf{w}$. Importantly, it is also one of the components in Bayes' theorem, and takes us a step towards a full Bayesian treatment. For simplicity, we introduce a simple Gaussian prior

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left[-\frac{\alpha}{2}\mathbf{w}^\top\mathbf{w}\right], \tag{11}$$

where α is the precision of the distribution and $M+1$ is the number of elements in $\mathbf{w}$ for an M order polynomial function. Variables such as α are called *hyperparameters* since we have to choose them. Now that we have a prior, we can use Bayes' theorem to yield a quantity proportional to the posterior,

$$\overbrace{p(\mathbf{w}|\mathbf{x},\mathbf{t},\alpha,\beta)}^{\text{Posterior } p(\mathbf{w}|D)} \propto \underbrace{p(\mathbf{t}|\mathbf{x},\mathbf{w},\beta)}_{\text{Likelihood } p(D|\mathbf{w})} \overbrace{p(\mathbf{w}|\alpha)}^{\text{Prior } p(\mathbf{w})}, \tag{12}$$

where the proportion relation $\propto$ comes from the fact that the denominator of Bayes' theorem, $p(D) = p(\mathbf{t}|\mathbf{x})$, does not depend on $\mathbf{w}$. Maximizing the right hand size of (12) with respect to $\mathbf{w}$ is equivalent to maximizing the posterior with respect to $\mathbf{w}$ due to the proportionality. By optimizing $\mathbf{w}$ to maximize the posterior, we are finding the most probable parameter values $\mathbf{w}$ considering our observed data and also our prior knowledge about $\mathbf{w}$ encapsulated in the (data independent) prior $p(\mathbf{w}|\alpha)$. This yields a tradeoff between what we believed about $\mathbf{w}$ before seeing our data, and the $\mathbf{w}$ that best fits our data. This technique is referred to *maximum a posteriori estimation*, MAPE, or MAP.

3.3.1 Maximum a posteriori's connection to Least-Squares

Taking the negative logarithm of (12) and combining with the log likelihood in (7) and the prior in (11), it can be shown that the maximum of the posterior is equivalenty the minimum of the following expression:

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\alpha}{2}\mathbf{w}^\top\mathbf{w}. \tag{13}$$

If we define $\lambda = \alpha/\beta$, we can rewrite (13) as

$$\frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}. \tag{14}$$

And so we see that minimizing the regularized sum-of-squares function introduced in (4) arises naturally from maximizing the posterior of a Gaussian noise model with Gaussian prior.

3.3.2 Maximum a posteriori's predictive distribution

Similar to Maximum Likelihood, the result of the *maximum a posteriori estimation* method are point-estimates for the parameters $\mathbf{w}$ and β. We can similarly plug in those estimates, denoted $\mathbf{w}_{\text{MAP}}$ and β_{MAP}, into the observation model (5):

$$p(t|x, \mathbf{w}_{\text{MAP}}, \beta_{\text{MAP}}) = \mathcal{N}(y(x, \mathbf{w}_{\text{MAP}}), \beta_{\text{MAP}}^{-1}). \tag{15}$$

3.4 Bayesian Estimation

So far we have been making a point estimate of $\mathbf{w}$, which does not yet amount to a Bayesin treatment. In a Bayesian treatment, we take into account all possible $\mathbf{w}$ that could have explained our data. To predict a new value of t for a new x, we marginalize over all possible settings of $\mathbf{w}$, yielding the *posterior predictive distribution* or *Bayesian model average*. The goal of the Bayesian treatment is to compute the posterior over the weights $p(\mathbf{w}|D)$ which represents all possible settings of $\mathbf{w}$ given our observed data D. To this end, we must use Bayes' theorem:

$$p(\mathbf{w}|D) = \frac{p(D|\mathbf{w})(p(\mathbf{w}))}{p(D)}. \tag{16}$$

And for our regression problem in particular, (16) can be further specified as

$$p(\mathbf{w}|\mathbf{t}, \mathbf{x}, \alpha, \beta) = \frac{\overbrace{p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)}^{\text{Likelihood}} \cdot \overbrace{p(\mathbf{w}|\alpha)}^{\text{Prior}}}{\underbrace{p(\mathbf{t}|\mathbf{x})}_{\text{Evidence}}}. \tag{17}$$

To simplify this, we will assume the parameters α and β are fixed and known in advance, and we will solely focus on the unknown $\mathbf{w}$. (16) then turns into

$$p(\mathbf{w}|\mathbf{x}, \mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{x}, \mathbf{w}) \cdot p(\mathbf{w})}{p(\mathbf{t}|\mathbf{x})}. \tag{18}$$

So instead of computing a single *point estimate* of the weights (as we've done thus far); now, given the likelihood and the prior, we can compute the *posterior distribution* over $\mathbf{w}$ via Bayes' rule. Computing the posterior can be facilitated by our choice of prior (a modeling choice) so that we can calculate the posterior in closed form. Those priors are referred to as *conjugate priors*. There are a number of other techniques used to instead *approximate* the posterior $p(\mathbf{w}|\mathbf{t}, \mathbf{x})$ when conjugacy is impossible. These include *variational inference* and *markov chain monte carlo sampling*. We save all these methods for another post and assume we've found the exact posterior.

3.4.1 Bayesian's predictive distribution

Once it is found, the posterior represents all possible settings of $\mathbf{w}$ that could explain our data. To incorporate this information into a *predictive distribution* so that we can predict new values of an unobserved t given an unobserved x, we marginalize over all possible settings of $\mathbf{w}$ like so:

$$p(t|x, \mathbf{x}, \mathbf{t}) = \int p(t|x, \mathbf{w}) p(\mathbf{w}|\mathbf{x}, \mathbf{t}) \mathrm{d}\mathbf{w}, \tag{19}$$

where $p(t|x, \mathbf{w})$ is our model given by (5) omitting the dependence on β, and $p(\mathbf{w}|\mathbf{x}, \mathbf{t})$ is the posterior over the weights $\mathbf{w}$. In (19),

- we look at a possible setting of $\mathbf{w}$ which we will denote $\mathbf{w}_i$ according to the posterior $p(\mathbf{w}|\mathbf{x}, \mathbf{t})$. Placing $\mathbf{w}_i$ into our observation model $p(t|x, \mathbf{w})$ defines a "fitted model" $p(t|x, \mathbf{w}_i)$.
- this "fitted" model is *multiplied* by the probability of that setting of $\mathbf{w}_i$ given by the posterior. In other words, the probability that $p(t|x, \mathbf{w}_i)$ having been used to generate our observed data $\mathbf{t}$ from the inputs $\mathbf{x}$.
- we do this for every possible setting $\mathbf{w}_i$ according to the posterior and integrate.

Thus, we are taking a weighted average of all possible "fitted" models, where the weights of each component are determined by how likely the setting of $\mathbf{w}$ isi according to its posterior.

3.4.2 Analyzing our specific predictive distribution

We will now alnalyze the specific form of (19) for our example problem. A consequence of us selecting a Gaussian likelihood and a Gaussian prior is that we can analytically compute the posterior; and it is Gaussian as well – an example of *conjugacy*. More so, we can analytically solve the integration in (19) to get a predictive distribution that itself is Gaussian. In fact, it takes on the more specific form

$$p(t|x, \mathbf{x}, \mathbf{t}) = \mathcal{N}(t|m(x), s^2(x)), \tag{20}$$

where the mean and variance are given by

$$m(x) = \beta \boldsymbol{\phi}(x)^\top \mathbf{S} \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) t_n \tag{19}$$

$$s^2(x) = \overbrace{\beta^{-1}}^{\text{noise}} + \underbrace{\boldsymbol{\phi}(x)^\top \mathbf{S} \boldsymbol{\phi}(x)}_{\text{Parameter Uncertainty}} \tag{20}$$

The $\mathbf{S}$ is a matrix, and is given by

$$\mathbf{S}^{-1} = \alpha\mathbf{I} + \beta \sum_{n=1}^{N} \boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x),^{\top} \tag{21}$$

where $\mathbf{I}$ is the unit matrix, and we have defined the vector $\boldsymbol{\phi}(x)$ with elements $\phi_i(x) = x^i$ for $i = 0, ..., M$. In the predictive distribution (20), we now see that the variance depends on x. It's expansion is described by (21) and it contains two additive components. The first component, as was already expressed in the maximum likelihood predictive distribution β_{ML}, is the noise on the target variables. The second component, which *has not* been expressed until now, arises from the *uncertainty in the parameters* $\mathbf{w}$ and is a consequence of treating $\mathbf{w}$ as a random variable – a mainstay of the Bayesian treatment.

Appendix

1 Solving for the parameters

1.1 Analytical Setup

We have the error function $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$, where $y(x, \mathbf{w}) = \sum_{j=0}^{M} w_j x^j$. We'd like to find the optimal setting for $\mathbf{w}$ in the sense that it minimizes $E(\mathbf{w})$. We will see that we can cast this minimization problem as solving a system of linear equations. We will then solve that system of linear equations with code to attain the optimal $\mathbf{w}^*$.

Claim: The w_i in $\mathbf{w} = (w_0, w_1, w_2..., w_M)$ that minimize the error function $E(\mathbf{w})$ are given by the solution to the following set of linear equations,

$$\sum_{j=0}^{M} A_{ij} w_j = T_i \quad \text{where} \quad A_{ij} = \sum_{n=1}^{N} (x_n)^{(i+j)},\ T_i = \sum_{n=1}^{N} (x_n)^i t_n$$

Proof: We will take the derivative of $E(\mathbf{w})$ with respect to $\mathbf{w}$, set it to zero, and then rearrange terms to prove the claim above.

By the chain rule,

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \frac{\partial E(\mathbf{w})}{\partial y(x_n, \mathbf{w})} \frac{\partial y(x_n, \mathbf{w})}{\partial w_i} \tag{1}$$

Solving the two terms on the right hand side yields

$$\frac{\partial E(\mathbf{w})}{\partial y(x_n, \mathbf{w})} = \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}$$
$$\frac{\partial y(x_n, \mathbf{w})}{\partial w_i} = \frac{\partial}{\partial w_i}(w_0 + w_1 x_1 + ... w_i x_n^i \cdots + w_m x_n^M) = x_n^i$$

Substituting back into (1) yields

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\} x_n^i \tag{i}$$
$$= \sum_{n=1}^{N} (\sum_{j=0}^{M} w_j x_n^j - t_n) x_n^i \tag{ii}$$
$$= \sum_{n=1}^{N} (\sum_{j=0}^{M} w_j x_n^i x_n^j - t_n x_n^i) \tag{iii}$$
$$= \sum_{n=1}^{N} (\sum_{j=0}^{M} w_j x_n^{(i+j)} - t_n x_n^i) \tag{iv}$$

- (i) − (ii): definition of $y(x_n, \mathbf{w})$.
- (ii) − (iii): distribute x_n^i into the parentheses.
- (iii) − (iv): exponent rule.

Setting the derivative to 0 and rearranging,

$$\sum_{n=1}^{N} (\sum_{j=0}^{M} w_j x_n^{(i+j)} - t_n x_n^i) = 0$$
$$\sum_{n=1}^{N} \sum_{j=0}^{M} w_j x_n^{(i+j)} = \sum_{n=1}^{N} t_n x_n^i$$
$$\sum_{j=0}^{M} A_{ij} w_j = T_i$$

■

1.2 Code

```
import numpy as np
import matplotlib.pyplot as plt

def create_A(xs: np.array, M: int) -> np.array:
    """
    Create the matrix A where A_ij = sum_n=1^N [x_n^(i+j)].
    """
    A = np.zeros((M, M))
    # For each row
    for i in range(M):
        # For each column
        for j in range(M):
            # Fill in A_ij = sum_n=1^N (x_n^(i+j))
            A[i][j] = np.sum(xs**(i+j))
    return A

def create_T(xs: np.array, ts: np.array, M: int) -> np.array:
    """
    Create the vector T where T_i = sum_n=1^N (x_n^i) t_n.
    """
    T = np.zeros((M,))
    # For each row
    for i in range(M):
        # Fill in T_i = sum_n=1^N (x_n^i)t_n
        T[i] = np.sum((xs ** i) * ts)
    return T

def get_w(xs: np.array, ts: np.array, M: int) -> np.array:
    """
    Creates A and T using `create_A` and `create_T` and then solves
    the linear system of equations to get w.
    """
    A = create_A(xs=xs, M=M)
    T = create_T(xs=xs, ts=ts, M=M)
    return np.linalg.solve(a=A, b=T)

def predict(w: np.array, x: np.array, M) -> np.array:
    """
    Feature expand the input x and then run the linear
    transformation involving w,x to get predictions for target t.
    """
```

```
    # Get [x**1, x**2, x**3 ... x**M] for each x
    N = x.shape[0]
    powers = np.arange(0, M )
    powers_expanded = np.tile(powers, (N,)).reshape(N, M)
    xs_expanded = x.repeat(M).reshape(N, M)
    xs_powered = np.power(xs_expanded, powers_expanded)

    # Apply w
    return w @ xs_powered.T
```

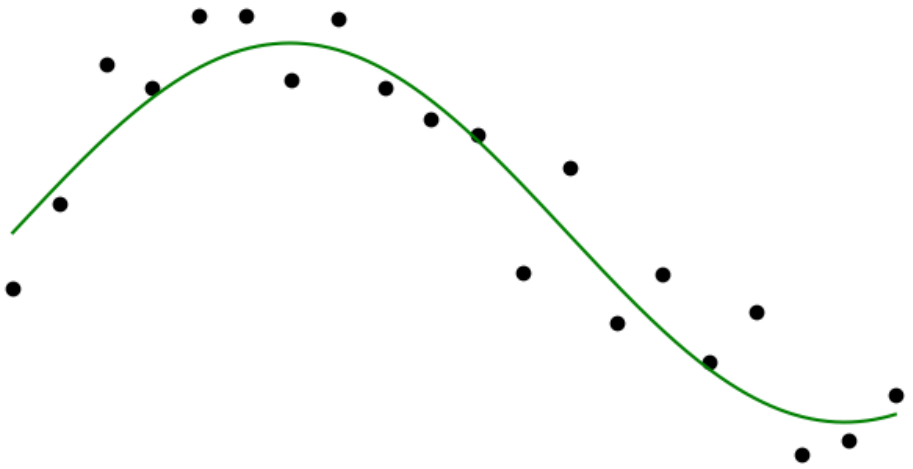

Figure 4: Signal (green) and training data (black).

2 Solving for the parameters with a penalty term

2.1 Analytical Setup

We have the error function $\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w})\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$, where $\|\mathbf{w}\|^2 = \mathbf{w}^\top\mathbf{w} = w_0^2 + w_1^2 + \cdots + w_M^2$ and the parameter λ controls the strength of regularization. We'd like to find the optimal setting for $\mathbf{w}$ in the sense that it minimizes $\widetilde{E}(\mathbf{w})$. We will see that we can cast this minimization problem as solving a system of linear equations. We will then solve that system of linear equations with code to attain the optimal $\mathbf{w}^*$.

Claim: The w_i in $\mathbf{w} = (w_1, w_2, ..., w_M)$ that minimize the error function $\widetilde{E}(\mathbf{w})$ are given by the solution to the following set of linear equations,

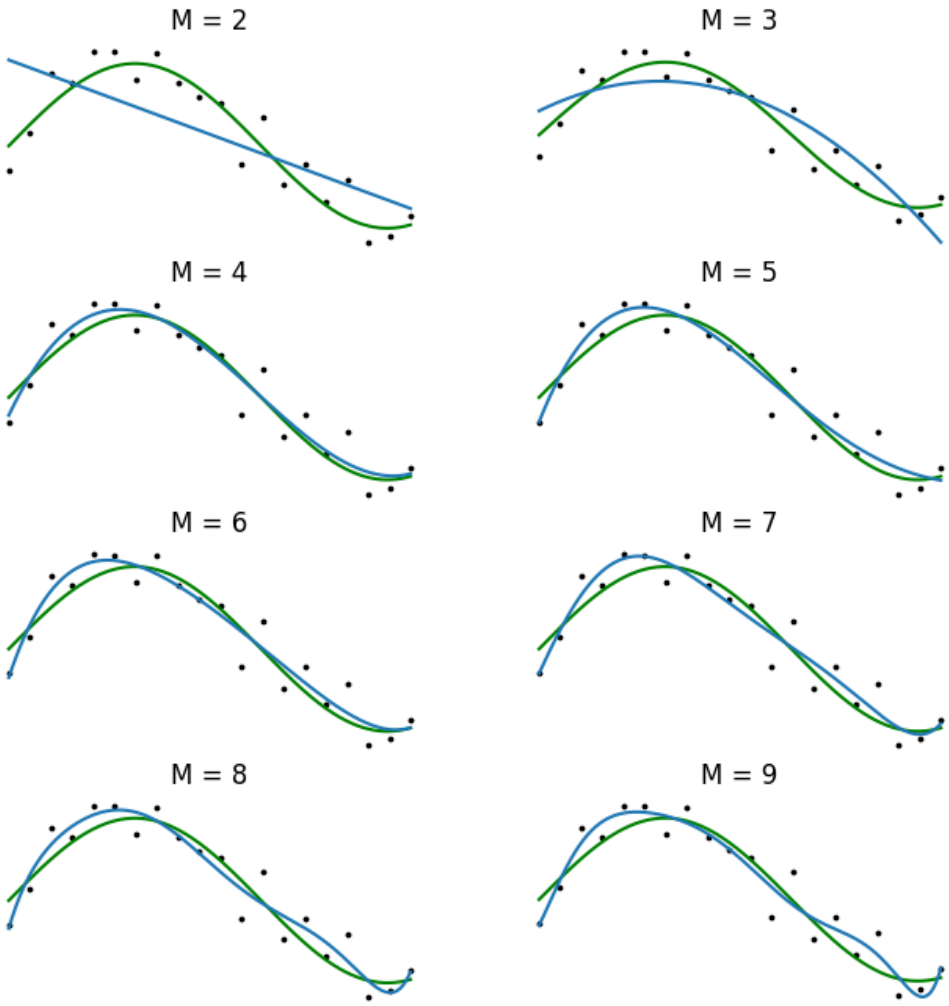

Figure 5: Signal (green), training data (black), and fitted curves (blue) for various settings of M – the order of the polynomial.

$$\sum_{j=0}^{M} A_{ij} w_j + \lambda w_i = T_i \quad \text{where} \quad A_{ij} = \sum_{n=1}^{N} (x_n)^{(i+j)},\ T_i = \sum_{n=1}^{N} (x_n)^i t_n$$

Proof: We will take the derivative of $\widetilde{E}(\mathbf{w})$ with respect to $\mathbf{w}$, set it to zero, and then rearrange terms to prove the claim above.

By the chain rule,

$$\frac{\partial \widetilde{E}(\mathbf{w})}{\partial w_i} = \frac{\partial E(\mathbf{w})}{\partial y(x_n, \mathbf{w})} \frac{\partial y(x_n, \mathbf{w})}{\partial w_i} + \frac{\lambda}{2} \frac{\partial \mathbf{w}^\top \mathbf{w}}{\partial w_i} \tag{2}$$

Solving the two terms on the right hand side yields

$$\begin{aligned}
\frac{\partial \widetilde{E}(\mathbf{w})}{\partial y(x_n, \mathbf{w})} &= \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\} \\
\frac{\partial y(x_n, \mathbf{w})}{\partial w_i} &= \frac{\partial}{\partial w_i}(w_0 + w_1 x_1 + \dots w_i x_n^i \cdots + w_m x_n^M) = x_n^i \\
\frac{\partial \mathbf{w}^\top \mathbf{w}}{\partial w_i} &= \frac{\partial}{\partial w_i}(w_0^2 + w_1^2 + \dots w_i^2 + \dots w_M^2) = 2w_i
\end{aligned}$$

Substituting back into (2) yields

$$\begin{aligned}
\frac{\partial \widetilde{E}(\mathbf{w})}{\partial w_i} &= \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\} x_n^i + \frac{\lambda}{2} 2 w_i && \text{(i)} \\
&= \sum_{n=1}^{N} (\sum_{j=0}^{M} w_j x_n^j - t_n) x_n^i + \lambda w_i && \text{(ii)} \\
&= \sum_{n=1}^{N} (\sum_{j=0}^{M} w_j x_n^i x_n^j - t_n x_n^i) + \lambda w_i && \text{(iii)} \\
&= \sum_{n=1}^{N} (\sum_{j=0}^{M} w_j x_n^{(i+j)} - t_n x_n^i) + \lambda w_i && \text{(iv)}
\end{aligned}$$

- (i) $-$ (ii): definition of $y(x_n, \mathbf{w})$ and $\frac{\lambda}{2} \cdot 2 = \lambda$.
- (ii) $-$ (iii): distribute x_n^i into the parentheses.
- (iii) $-$ (iv): exponent rule.

Setting the derivative to 0 and rearranging,

$$\sum_{n=1}^{N}(\sum_{j=0}^{M} w_j x_n^{(i+j)} - t_n x_n^i) + \lambda w_i = 0$$

$$\sum_{n=1}^{N}\sum_{j=0}^{M} w_j x_n^{(i+j)} + \lambda w_i = \sum_{n=1}^{N} t_n x_n^i$$

$$\sum_{j=0}^{M} A_{ij} w_j + \lambda w_i = T_i$$

■

2.2 Code

```
# ===== Reproducibility ====== #
# Create generator for reproducible resutls
seed = 123
generator = np.random.default_rng(seed)

# ===== Data Generation ======= #
# N = num datapoints; M = order of polynomial
N = 20
# Create signal and noisy signal
signal_fn = lambda x: np.sin(x)
signal_noisy_fn = lambda xs: signal_fn(xs) + generator.normal(loc=0, scale=0.5, size=(N,),
# Create training data
xs = np.linspace(start=0, stop=5, num=N)
ts = signal_noisy_fn(xs)
# Create "signal"
x_axis = np.linspace(start=0, stop=5, num=1000)
signal = signal_fn(x_axis)

# ===== Plot Data ====== #
# Create figure
figure1 = plt.figure(figsize=(8, 4))
plt.axis("off")
# Plot Signal
line = np.linspace(start=0, stop=5, num=1000)
plt.plot(x_axis, signal, c="green")
# Plot training data
```

```
plt.scatter(xs, ts, c="black")
plt.show()

# ======= Functions to get W and Predict ====== #
def create_A(xs: np.array, M: int) -> np.array:
    """
    Create the matrix A where A_ij = sum_n=1^N [x_n^(i+j)].
    """
    A = np.zeros((M, M))
    # For each row
    for i in range(M):
        # For each column
        for j in range(M):
            # Fill in A_ij = sum_n=1^N (x_n^(i+j))
            A[i][j] = np.sum(xs**(i+j))
    return A

def create_T(xs: np.array, ts: np.array, M: int) -> np.array:
    """
    Create the vector T where T_i = sum_n=1^N (x_n^i) t_n.
    """
    T = np.zeros((M,))
    # For each row
    for i in range(M):
        # Fill in T_i = sum_n=1^N (x_n^i)t_n
        T[i] = np.sum((xs ** i) * ts)
    return T

def create_lambdaI(size: int, ln_lambda: float) -> np.array:
    """
    Create an identity matrix with lambda as the diagonal elements.
    """
    lambda_ = np.exp(ln_lambda)
    identity = np.eye(N=size)
    return lambda_ * identity

def get_w(xs: np.array, ts: np.array, M: int, ln_lambda: float) -> np.array:
    """
    Creates A and T using `create_A` and `create_T` and then solves
    the linear system of equations to get w.
    """
```

```
    A = create_A(xs=xs, M=M)
    lambdaI = create_lambdaI(size=M, ln_lambda=ln_lambda)
    A_plus_lambdaI = A + lambdaI
    T = create_T(xs=xs, ts=ts, M=M)
    return np.linalg.solve(a=A_plus_lambdaI, b=T)

def predict(w: np.array, x: np.array, M) -> np.array:
    """
    Feature expand the input x and then run the linear
    transformation involving w,x to get predictions for target t.
    """
    # Get [x**1, x**2, x**3 ... x**M] for each x
    N = x.shape[0]
    powers = np.arange(0, M )
    powers_expanded = np.tile(powers, (N,)).reshape(N, M)
    xs_expanded = x.repeat(M).reshape(N, M)
    xs_powered = np.power(xs_expanded, powers_expanded)
    # Apply w
    return w @ xs_powered.T


# ======== Find W, Predict, and Plot for various M ======= #
figure = plt.figure(figsize=(8, 8))
cols, rows = 2, 4
for i in range(1, cols * rows + 1):
    M = i + 1
    w = get_w(
        xs=xs,
        ts=ts,
        M=M,
        ln_lambda=0,
    )
    t_hats = predict(w, line, M)
    figure.add_subplot(rows, cols, i)
    plt.axis("off")
    plt.title(f"M = {M}")
    # Plot signal
    plt.plot(line, signal_fn(line), label="Signal", c="green")
    # Plot data
    plt.scatter(xs, ts, s = 3, label="Data", c="black")
    # Plot predictions
```

```
    plt.plot(line, t_hats, label="Predicted Signal")
plt.show()
```

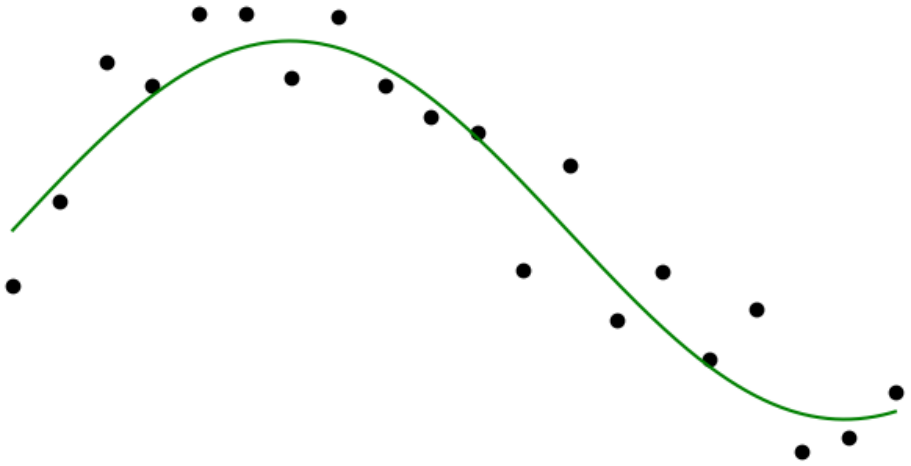

Figure 6: Signal (green) and training data (black).

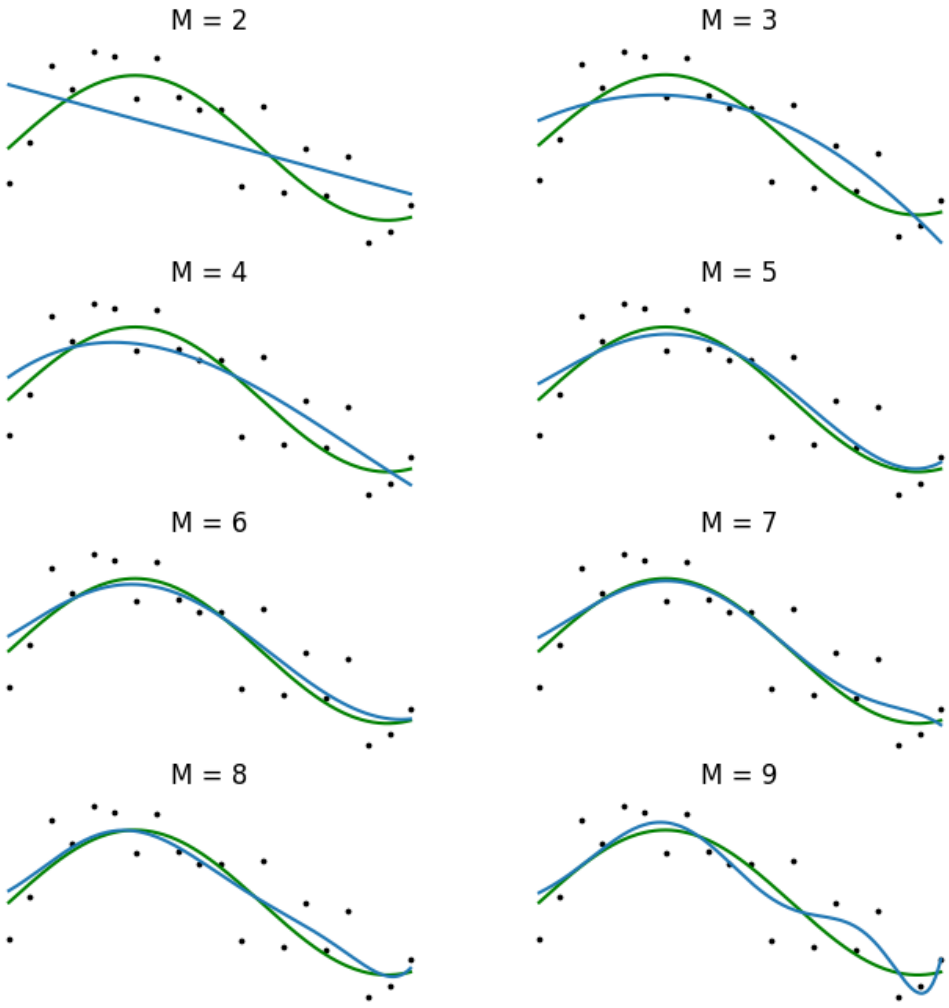

Figure 7: Signal (green), training data (black), and fitted curves (blue) for various settings of M – the order of the polynomial, but with with regularization governed by $\ln \lambda = 0$.