Fitting Very Flexible Models: Linear Regression With Large Numbers of Parameters

David W. Hogg[1,2,3,4] and Soledad Villar[5,6]

[1] Flatiron Institute, a division of the Simons Foundation, USA
[2] Center for Cosmology and Particle Physics, Department of Physics, New York University, USA
[3] Center for Data Science, New York University, USA
[4] Max-Planck-Institut für Astronomie, Heidelberg, Germany
[5] Department of Applied Mathematics & Statistics, Johns Hopkins University, USA
[6] Mathematical Institute for Data Science, Johns Hopkins University, USA

Abstract

There are many uses for linear fitting; we consider here the interpolation and denoising of data, as when the goal is to fit a smooth, flexible function to a set of noisy data points. Investigators often choose a polynomial basis, or a Fourier basis, or wavelets, or something equally general. They also choose an order, or number of basis functions to fit, and (often) some kind of regularization. We discuss how this basis-function fitting is done, with ordinary least squares and extensions thereof. We emphasize that it can be valuable to choose *far more parameters than data points*, despite folk rules to the contrary: Suitably regularized models with enormous numbers of parameters generalize well and make good predictions for held-out data; over-fitting is not (mainly) a problem of having too many parameters. It is even possible to take the limit of infinite parameters, at which, if the basis and regularization are chosen correctly, the least-squares fit becomes the mean of a Gaussian process, or a kernel regression. We recommend cross-validation as a good empirical method for model selection (for example, setting the number of parameters and the form of the regularization), and jackknife resampling as a good empirical method for estimating the uncertainties of the predictions made by the model. We also give advice for building stable computational implementations.

Key words: Regression – Linear regression – Gaussian Processes regression

1. Introduction

In contexts in which we want to fit a flexible function to data, for interpolation or denoising, we often perform linear fitting in a generic basis, such as polynomials, Fourier modes, wavelets, or spherical harmonics. This kind of linear fitting arises in astronomy when, for example, we want to calibrate the relationship between wavelength and position on the detector in a spectrograph: We have noisy measurements of calibration data and we want to fit a smooth function of position that denoises and interpolates the calibration data. It also arises when we want to make a data-driven interpolation, extrapolation, or local averaging of data, as with light-curve de-trending, continuum estimation, and interpolation or extrapolation of instrument housekeeping (or other) data.

When faced with problems of this kind, investigators have three general kinds of choices that they have to make: They have to choose the basis in which they are working (Fourier, polynomial, wavelet, etc.). They have to choose to what order they extend the basis—that is, how many components to use in the fit. And they have to decide how (or whether) to regularize the fit, or discourage fit coefficients from getting out of line when the data are noisy or the basis functions are close to (or strictly) degenerate.

If you have encountered and solved problems like these, you have made these three kinds of choices (sometimes implicitly). The second choice—about the number of coefficients to fit—is usually made heuristically, and often subject to the strongly believed opinion that we *must have fewer parameters than data points*. Here we show that this folk rule is not valid; we can go to extremely large numbers of coefficients without trouble. But like most folk rules, it has a strong basis in reality: There are extremely bad choices possible for the number of coefficients, and especially when the number of parameters is close or comparable to the number of data points. As we will discuss below, these choices ought to be made with care.

In many cases the third kind of choice—about regularization—is made implicitly, not explicitly. Here we are going to emphasize this choice and its importance, and its value in improving results.

Alternatively, if you are unhappy with the three choices of basis, order, and regularization, you might decide to avoid such decisions and go fully *non-parametric*: Instead of fitting a basis expansion, you can use a strict interpolator (like a cubic spline), or you can fit a Gaussian process to your data. Here we will show that the choice of any Gaussian process kernel function is equivalent to choosing a basis and a regularization and letting the number of fit components go to infinity. That is, going non-parametric does not really get you out of making these choices. It just makes these choices more implicit. And it is a pleasure to note that any time you have gone non-parametric you have implicitly chosen to use a basis with way more fit parameters than data points! The fact that non-parametrics work so well is strong evidence against the folk rule about the number of parameters needing to be less than the number of data points.

An important assumption or setting for the problems we are addressing in this *expository note* will be that we care about predicting new data or interpolating the data, but we explicitly *do not care* about the parameters of the fit or the weights of the basis functions *per se*. In this setting there are no important meanings to the components of the model. That is—for us— only the data exist. The details of the model are just choices that permit high-quality interpolations and predictions in the space of the data.

In what follows, we will look at applications that look like interpolation; there are many other contexts in which regressions have gone to very large numbers of parameters. For example, there are contexts in which there are enormous numbers of possible natural features, like for instance when the data are images or videos: Every pixel of every frame of the video—or any linear (or nonlinear) combination of pixels—can become a feature for the regression. Also, there are contexts in which features are generated randomly from, say, a space of functions that can act on the natural features (Rahimi & Recht 2007). These settings are not what we are addressing here, but they are relevant and related (many good books exist, for example, Bishop 2006; Hastie et al. 2009; Agresti 2015; Gelman et al. 2020). In some ways, the most flexible of models currently in use are deep networks, where it is both the case that the input data often have enormous numbers of natural features, *and* the deep network is capable of generating (effectively) far more, internally.

In some contexts we have strong beliefs about the noise affecting our measurements. In other cases we do not. In some cases we have strong reasons to use a particular basis. In other cases we do not. The differences in these beliefs and the differences in our objectives will change what methods we choose and how we use and analyze them.

This *expository note* delivers a theoretical and practical discussion of linear regression and function interpolation; it does not deliver new research results. Our only novel contribution is to make an explicit connection between regularized linear regression and Gaussian processes.

Ordinary least squares is reviewed in Section 2. The extensions of weighted least squares and ridge regression are shown in Section 3. The over-parameterized case (more parameters than data) is discussed in Section 4. The concept of feature weighting for controlling regularization in over-parameterized fits is introduced in Section 5. Cross-validation is explained and used to choose the number of parameters in Section 6, and the double descent phenomenon is shown. The Gaussian Process appears as the limit of infinite parameters in Section 7. Jackknife resampling is explained and used to estimate uncertainties in Section 8. Numerical implementation considerations are discussed in Section 9 and some final remarks are made in Section 10.

2. Standard Linear Fitting: Ordinary Least Squares with a Feature Embedding

Our setup will be that there are n scalar data points y_i. Each of these data points has an associated coordinate or location t_i. In the machine-learning lexicon, these will be our "training data." The location t_i could be thought of as a time at which the data point was taken, or a position, or it can be a higher dimensional vector or blob of housekeeping data associated with the data point. Critically, we are going to imagine that the t_i are known very well (not very uncertain or noisy), while the y_i are possibly very uncertain or noisy measurements. We will return to these assumptions at the end.

We are going to fit these data y_i with a linear sum of p basis functions $g_j(t)$. These basis functions are functions of the coordinates t. That is, our implicit generative model is

$$y_i = \sum_{j=1}^{p} \beta_j \, g_j(t_i) + \text{noise}, \tag{1}$$

where the p values β_j are parameters or coefficients of the linear fit. We can assemble the evaluations of the p functions $g_j(t)$ at the n data coordinates t_i into a $n \times p$ design matrix or feature matrix X such that

$$[X]_{ij} = g_j(t_i). \tag{2}$$

The transformation of the n locations t_i into a $n \times p$ matrix X is the opposite of a dimensionality reduction. It is called variously a *feature map* or a *feature embedding*. The embedding raises the dimensionality of the locations t into the p-dimensional space of the rows of X.

For a concrete example, one common choice is to make the feature embedding functions $g_j(t)$ terms in a Fourier series

$$g_j(t) = \begin{cases} \cos \omega_j t & \text{for } j \text{ odd} \\ \sin \omega_j t & \text{for } j \text{ even} \end{cases} \tag{3}$$

$$\omega_j = \frac{\pi}{T} \left\lfloor \frac{j}{2} \right\rfloor, \tag{4}$$

where T is a (large) length-scale in the coordinate space (t space) and $\lfloor j/2 \rfloor$ indicates the floor of $j/2$ (integer division).
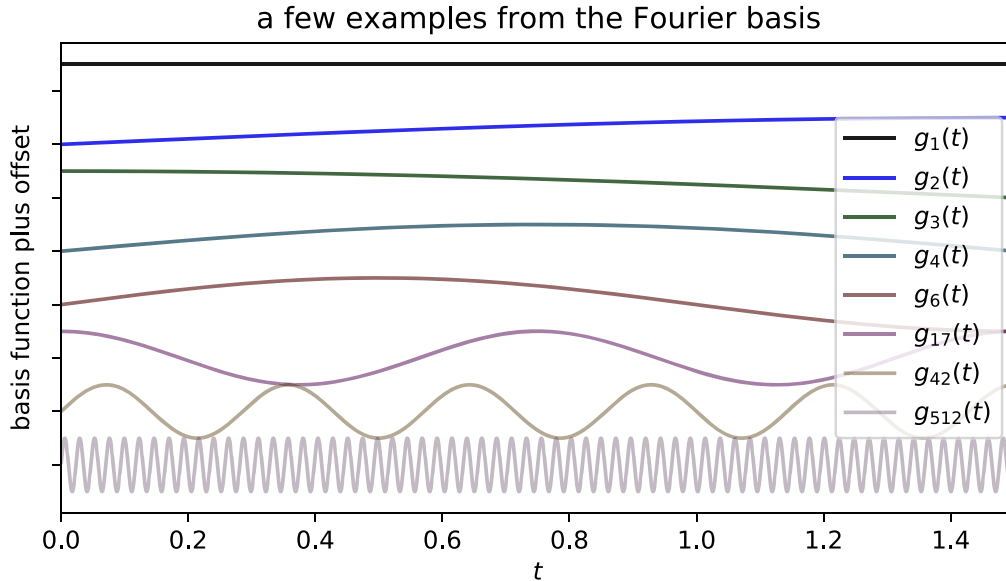
Figure 1. Examples of basis functions $g_j(t)$ from the basis given in Equation (3). This basis was constructed with length-scale parameter $T = 3$. The wavelength in the location space decreases, and the frequency increases, with index j.

Example functions $g_j(t)$ from this basis are shown in Figure 1. Alternative common choices would be to make the embedding functions $g_j(t)$ polynomials or other kinds of ordered basis functions, such as wavelets or spherical harmonics (the latter if, say, the t_i are positions on the sphere). Another choice that is not common in the natural sciences, but studied in machine learning (for example, Rahimi & Recht 2007), is to choose the features randomly from a distribution (rather than on a regular grid in frequency, as we do here). That is beyond our scope, but we come back to it in Section 10.

The idea of least-squares fitting is that the "best" values of the parameters β_j are the values that minimize the sum of squares of the differences between the data and the linear combination of features:

$$\hat{\beta} = \arg\min_{\beta} \|Y - X\,\beta\|_2^2, \qquad (5)$$

where $\hat{\beta}$ is the p-vector (column vector) of the p best-fit values $\hat{\beta}_j$ of the parameters β_j, and $\|q\|_2^2$ denotes the squared L2-norm, or the *sum of squares* of the components of a vector q

$$\|q\|_2^2 \equiv q^\top q. \qquad (6)$$

This optimization objective (5) is convex and the optimization problem has a solution in closed form[7] as long as the number of parameters (and features) p is less than the number of training points n (and the matrix $X^\top X$ is invertible):

$$\hat{\beta} = (X^\top X)^{-1} X^\top Y. \qquad (7)$$

We will treat the case in which the matrix is not invertible below. When the investigator knows uncertainties on the training data y_i, this expression will change a bit; we begin to discuss that in the next section.

But recall our setting: We are using the linear fit to interpolate the data, or de-noise the data, or predict new data. In these contexts, we typically do not care about the parameter vector $\hat{\beta}$ itself. We care only about the predictions at a set of new "test" locations t_*, which will usually be different from the training locations t_j. From the new test times t_* we create the test feature matrix X_* (by the same feature embedding functions $g_j(t)$). The prediction $\hat{Y}_*$ for the y values at the test locations t_* becomes

$$\hat{Y}_* = X_*(X^\top X)^{-1} X^\top Y. \qquad (8)$$

Examples of OLS applied to some toy data[8] are shown in Figure 2. This form (8) of the prediction of new test data is called ordinary least squares (OLS). It has many good properties, some of which are encoded in the Gauss–Markov theorem. In particular, if the noise contributions in the model given in Equation (1) are uncorrelated, have zero mean, and equal variances for $i = 1, ..., n$ then the Gauss–Markov theorem states that the OLS estimator has the lowest variance within the class of unbiased estimators that are linear in Y (see, for example, Hastie et al. 2009, Ch. 3).

---

[7] Note that $\nabla_\beta \|Y - X\beta\|^2 = 2X^\top(Y - X\beta)$ so critical points are such that $X^\top Y = X^\top X\beta$. Since the objective is convex all these critical points are global minima. Further analysis is given in Appendix.

[8] The algorithm by which the toy data were made—and indeed all of the code used to make the figures for this *expository note*—is available online at https://github.com/davidwhogg/FlexibleLinearModels.
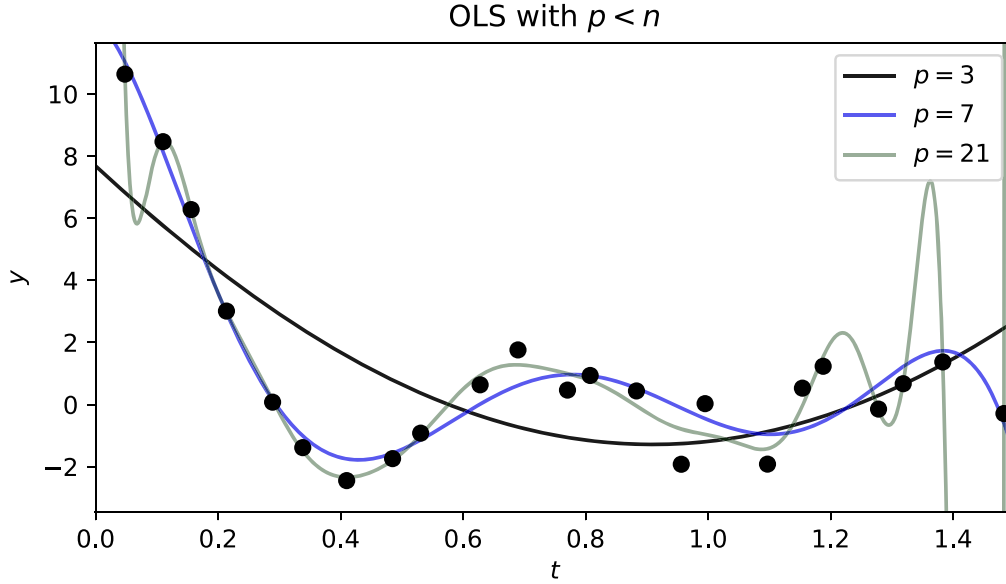
Figure 2. Ordinary least-squares (OLS) fits (continuous lines) to a set of example data points (black dots). Fits are shown for different values of the number of basis functions p; there are $n = 23$ data points. Here (and in all the Figures to follow) we are using the Fourier basis functions in (3) and shown in Figure 1. The data y_i were generated using a function that does not reside in the function space spanned by the basis. Here the X_* matrices used in the predictions $\hat{Y}_*$ were generated from a fine grid of locations in the location coordinate t; the plots of the fine grid of predictions are the continuous lines. The fits with larger p have more flexibility to fit the data than the fit with $p = 3$, but the fit at the highest p shows evidence of over-fitting.

3. Discussion and Extensions of OLS

The prediction (8) when $p < n$ (the under-parameterized or traditional regime) is *affine invariant* in that p-dimensional rotations or rescalings of the rectangular feature matrix X do not affect predictions. That is, if R is an invertible $p \times p$ matrix, the prediction using $X' \leftarrow X R$ will be identical to the prediction using the original X. This affine invariance will be modified in the over-parameterized regime, below.

Although the OLS prediction (8) is affine invariant with respect to p-dimensional transformations, it is *not* affine invariant with respect to n-dimensional transformations, such as a re-weighting of the input data. Indeed, if we know weights or inverse variances for our data points, conceptually we can put them into a weight matrix C^{-1} (written this way to emphasize that reweighting is usually inverse-variance weighting) and write

$$\hat{Y}_* = X_*(X^\top C^{-1} X)^{-1} X^\top C^{-1} Y, \tag{9}$$

where the weight matrix C^{-1} is $n \times n$ (and often diagonal in standard applications). The weight matrix C^{-1} is sometimes called the information tensor (or information matrix) and its inverse C is often called the covariance matrix or the noise variance tensor.

This form (9) of least squares is called *weighted least squares* (WLS) because of the data weighting (not to be confused with *feature* weighting, to appear below). It is also called *chi-squared fitting* because it optimizes the scalar objective commonly called

chi-squared:

$$\chi^2 = (Y - X \beta)^\top C^{-1} (Y - X \beta). \tag{10}$$

In Figure 3 a comparison of OLS and WLS is shown, for a case of non-trivial data weights, where the data weights are set to be the inverse squares of individual data point uncertainties.

We will say more about noisy data and the propagation of uncertainties below in Section 8. It is true that in realistic scenarios there are uncertainties not just in the data points y_i, but also often in the *locations* t_i of the data as well. It turns out that taking the latter into account is a much harder problem; we will discuss this briefly in Section 8.

It is common to include a regularization that discourages the fit from making use of large amplitudes β_j. There are many options, but the simplest is ridge regression (or Tikhonov regularization or L2 regularization), which (in the form that does not have data weights) looks like

$$\hat{\beta} = \arg\min_{\beta} \|Y - X \beta\|_2^2 + \lambda \|\beta\|_2^2, \tag{11}$$

where $\lambda > 0$ is a regularization parameter that penalizes large values for elements β_j of the parameter vector. This optimization is also convex. The ridge-regularized prediction for new data looks like

$$\hat{Y}_* = X_*(X^\top X + \lambda I)^{-1} X^\top Y, \tag{12}$$

where I is the $p \times p$ identity (see Appendix). In the language of Bayesian inference, λ can be seen as the inverse of a prior
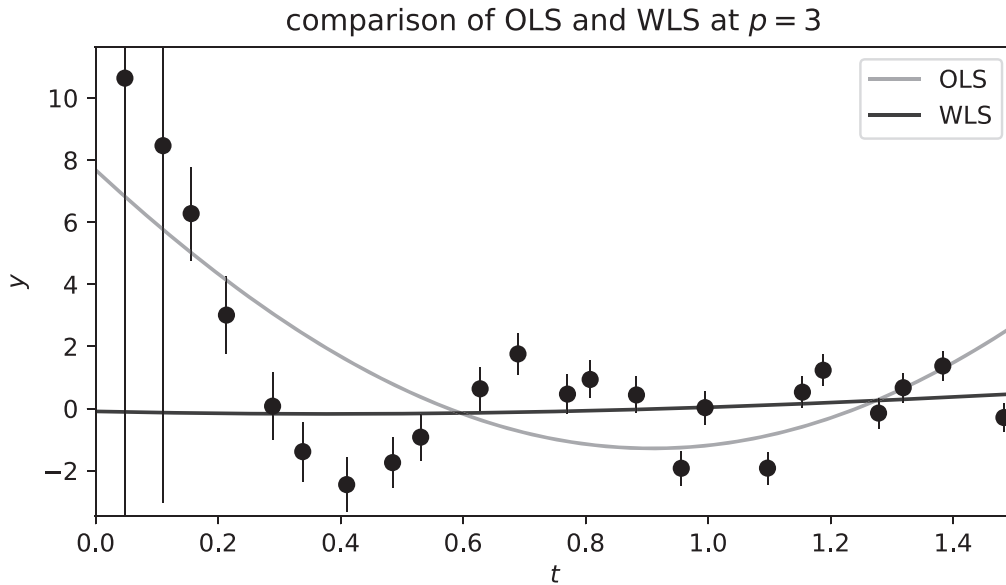
comparison of OLS and WLS at $p = 3$

Figure 3. Comparison of ordinary least-squares (OLS) and weighted least-squares (WLS) fits (continuous lines) to the example data (black dots). In order to illustrate the differences, we assigned non-trivial error bars to the data. The error bars are ignored in the OLS fit, but in the WLS fit, the weight matrix C^{-1} is diagonal with the diagonal entries set to the inverses of the squares of those error bars. The WLS fit "pays less attention to" the points on the left with the largest error bars.

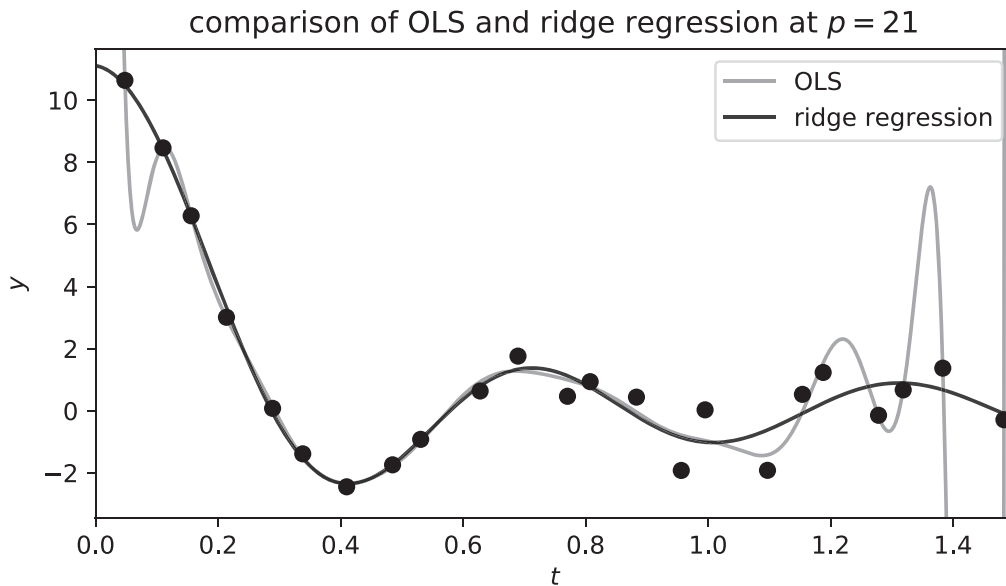## comparison of OLS and ridge regression at $p = 21$

Figure 4. Comparison of ordinary least-squares (OLS) and ridge-regression fits (continuous lines) to the example data (black dots). In order to illustrate the differences, we chose the $p = 21$-parameter fit, which shows evidence of over-fitting at the edges of the fit range. The regularized fit looks more sensible, though it fits the individual data points less precisely. The choice of regularization parameter λ matters; here we used $\lambda = 0.1$ (chosen heuristically by hand).

variance for the parameters β_j. The salutary effect of the ridge regularization is shown in Figure 4.

The ridge brings with it a choice: How to set the hyper-parameter λ? We generally recommend cross-validation, to be discussed below in Section 6; cross-validation gives good results but it produces a slightly biased estimate (Liu & Dobriban 2019).

Ridge regression is not affine invariant in the sense that the standard OLS prediction (8) is; that is, the effect of the regularization depends on the amplitudes and linear combinations of features placed in the feature matrix. This will become important later when we consider feature weights below. It is also the case that the regularization need not be proportional to

5

the identity matrix: In principle any positive definite matrix Λ could be used in place of λI; this makes sense to consider when we have detailed prior beliefs about all the parameters β_j or if those parameters are measured with different units (say).

Combining both the point weighting from WLS and the generalized ridge regression into a weighted ridge yields to estimates

$$\hat{Y}_* = X_*(X^\top C^{-1} X + \Lambda)^{-1} X^\top C^{-1} Y. \qquad (13)$$

This form has good properties for many real-world physics applications, where data point error bars are often known, and functions are often expected to be smooth. We will discuss this form (13) more below in Section 5, but briefly we can say here that it appears in Bayesian inference contexts where the data points y_i are treated as having Gaussian noise associated with them (with known variance tensor or covariance matrix C) and there is a Gaussian prior on the parameter vector β (with known variance Λ^{-1}). We have discussed that model elsewhere (Hogg et al. 2020).[9] It is also useful sometimes to think of the "units" or dimensions of the quantities in (13): In the Bayesian setting, the units of C would be the square of the units of Y and the units of Λ^{-1} would be the square of the units of the ratio Y/X (the square of the units of β).

4. Over-parameterization

We are taught folklore, that we can never fit for more parameters than we have data. That is, we can never work at $p > n$. This is not true! Not only is it the case that we *can* work at $p > n$, in many cases we *should* work at $p > n$, and many real-world regressions do. In the over-parameterized case, there are typically many choices for the parameters β_j that will literally zero out the differences between the data Y and the linear prediction $X \beta$. In the Introduction, we said that we are considering the case of linear regression in which *we do not care about the values of the parameters*. That is, the parameters of our linear fits are just nuisance parameters. This means that we do not have very detailed opinions about how these redundancies in the model (redundancies that appear when $p > n$) are resolved. The OLS solution, in this case, is defined (somewhat arbitrarily) to be the minimum-norm parameter vector β that interpolates the data:

$$\hat{\beta} = \arg\min_\beta \|\beta\|_2^2 \ \text{subject to} \ Y = X \beta. \qquad (14)$$

Technically this formulation depends on an additional assumption that the feature matrix X is full rank or that the data Y lie in the subspace spanned by X. This is true almost always when $p > n$. This optimization is again convex and has a unique solution, although that solution will depend on feature weights that we discuss in a moment. The under-parameterized

---

[9]   In our previous discussion of this product of Gaussians, the notation differs. What's called Λ^{-1} here is called Λ in Hogg et al. (2020).

and over-parameterized optimization statements (5) and (14) can be unified into one form by considering the limit of light L2 regularization:

$$\hat{\beta} = \lim_{\lambda \to 0^+} \left[\arg\min_\beta \|Y - X \beta\|_2^2 + \lambda \|\beta\|_2^2 \right]; \qquad (15)$$

in the limit, this delivers the OLS solution in either case and does not require the constraint in (14) to be satisfied.

In the over-parameterized case ($p > n$), the prediction looks like

$$\hat{Y}_* = X_* X^\top (X X^\top)^{-1} Y, \qquad (16)$$

provided that $X X^\top$ is invertible (which will usually be the case). Like (8), this prediction is *also* called OLS, or sometimes "min-norm least-squares" to emphasize the point that it is making the minimum-norm choice of β among many degenerate solutions. Examples of OLS fits in the over-parameterized regime are shown in Figure 5. Note that the fits with different numbers of parameters p lead to very different predictions, but they all go through the data exactly.

It is slightly off-topic to notice that in Figure 5, as p gets very large, the OLS solution approaches $y = 0$ everywhere that it can. This behavior is a direct consequence of using the Fourier basis and Plancharel's theorem, which states that the Fourier transform is unitary (see for instance Folland 1994). This means that the integral over frequency of the square of the Fourier transform is equal to the integral over location of the square of the original function. Thus the min-norm solution delivered by OLS, which chooses the interpolating function that minimizes the squares of the component amplitudes β in the Fourier basis used to make Figure 5, will choose the interpolating function $y(t)$ that minimizes the mean square of the value of the function in the location space. It will try to stay as close to $y = 0$ as possible. That's probably not desirable in most applications! We will fix that problem in the next section.

The two equations for OLS—(8) and (16)—can be unified into one equation (and also generalized to handle non-invertible matrices $X^\top X$ and $X X^\top$) if we define the pseudo-inverse $X^\dagger$:

$$\hat{Y}_* = X_* X^\dagger Y. \qquad (17)$$

The pseudo-inverse of a diagonal matrix is defined as the diagonal matrix made by inverting the non-zero diagonal entries. And for any non-diagonal (or any non-square) matrix X the pseudo-inverse is defined by taking the singular-value decomposition of X, $X = U S V$, with S diagonal and U, V orthogonal, then $X^\dagger \equiv V^\top S^\dagger U^\top$.

This pseudo-inverse form (17) of OLS is extremely general: It works for both the $p \leqslant n$ and $p > n$ cases, and it works when the $X X^\top$ or $X^\top X$ matrices are not invertible. There can be significant numerical issues with implementing the pseudo-inverse; we comment on those in Section 9.
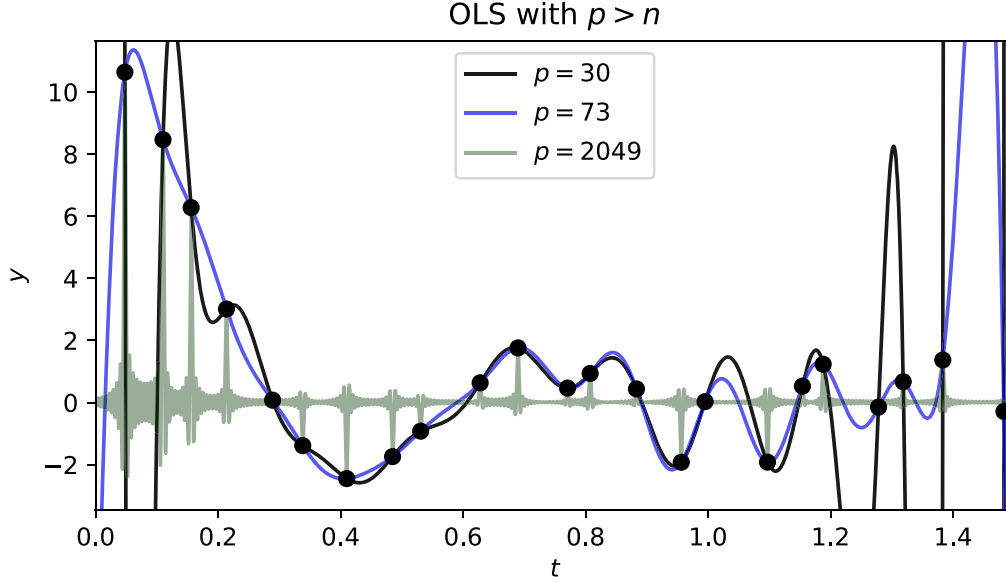
Figure 5. Ordinary least-squares (OLS) or min-norm least-squares fits (continuous lines) to a set of example data points (black dots), but now for a few over-parameterized cases. There are $n = 23$ data points. As in Figure 2, the data points y_i were generated using a function that does not precisely reside in the function space spanned by the basis. The three fits are very different, but they all go through all the data points exactly. As p gets large, the fit function approaches $y = 0$ almost everywhere; this is a consequence of Plancharel's Theorem and the use of the Fourier basis (see text).

5. Feature Weighting

The OLS prediction for $p > n$ is *not* affine invariant with respect to p-dimensional rotations or rescalings. That is, rotations and scalings in the feature space *will* affect predictions. It behooves us to re-scale the features (the p n-vectors of the feature matrix X) in a sensible way, like for instance, to encourage the fit to use low-frequency features more than high-frequency features (Bah & Ward 2016; Rauhut & Ward 2016; Xie et al. 2020). We can encode these feature weights in a $p \times p$ diagonal weight matrix Λ^{-1} and the prediction becomes

$$\hat{Y}_* = X_* \, \Lambda^{-1} X^\top (X \, \Lambda^{-1} X^\top)^{-1} \, Y. \tag{18}$$

This equation is a special case of the fully general form we will give below in (24), and equivalent to a special case of (13) above. This (18) can be seen as the prediction for new data resulting from the following optimization (again, assuming the data can be interpolated):

$$\hat{\beta} = \arg\min_\beta \|\Lambda^{1/2} \, \beta\|_2^2 \ \text{subject to} \ Y = X \, \beta, \tag{19}$$

which—in analogy to the optimizations (14) and (15)—can also be written as

$$\hat{\beta} = \lim_{\lambda \to 0^+} \left[\arg\min_\beta \|Y - X \, \beta\|_2^2 + \lambda \, \|\Lambda^{1/2} \, \beta\|_2^2 \right]. \tag{20}$$

This does not require the constraints in (19) to be satisfied. This optimization penalizes more strongly the parameters β_j corresponding to features $g_j(t)$ with larger values of $[\Lambda^{1/2}]_{jj}$.

In the Fourier case this re-weighting can be very straightforward: Each of the p embedding functions $g_j(t)$ has an associated frequency ω_j; we can control the fit by weighting the j features by a function $f(\omega)$. For demonstration purposes, we can choose

$$f(\omega) = \frac{1}{s^2 \, \omega^2 + 1}, \tag{21}$$

where s is a hyper-parameter controlling the (inverse) width of the weighting function in frequency space, and the frequency input will be ω_j for each feature j. That is,

$$[\Lambda^{-1}]_{jj} = [f(\omega_j)]^2. \tag{22}$$

We have chosen this form (21) for $f(\omega)$ for specific reasons that will become obvious below. The introduction of the feature weights dramatically changes the predictions; this is shown in Figure 6. High frequencies are suppressed and the prediction becomes smooth.

This all illustrates that, while OLS is affine invariant in the under-parameterized setting, the affine non-invariance in the over-parameterized setting is a property that can be exploited. When the different features are weighted or normalized differently, the amplitudes of the components β_j of the parameter vector β have to change in response, which in turn changes its norm $\|\beta\|_2^2$. That is, the details of the min-norm data-fitting parameter vector depends on the details of how the features are normalized or weighted. In feature-weighted OLS, this property can be exploited to make the fits smooth, or meet other desiderata.
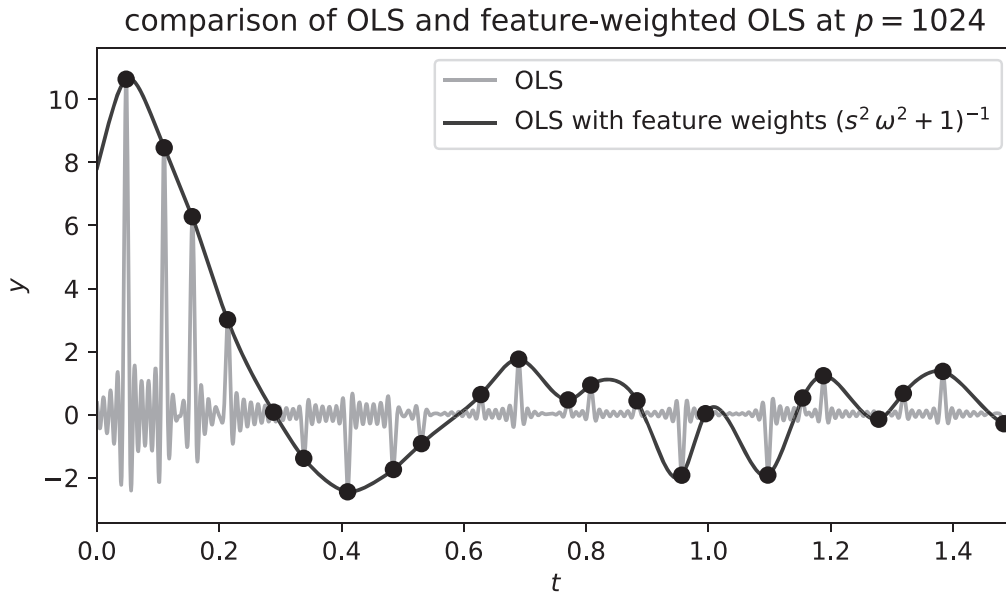
Figure 6. Comparison of ordinary least squares (OLS) to OLS with a specific feature weighting. The feature-weighting function is given in the text in Equations (21) and (22); in this and the following Figures, we (somewhat arbitrarily) set the width parameter to be $s = 0.05$. Because this weighting function penalizes more strongly the higher-frequency features, the min-norm solution gives them smaller amplitudes, making the fit smoother.

If we think of the OLS choice $\hat{\beta}$—the min-norm vector among all vectors β that thread the data (satisfy $X\,\beta = Y$)—as the result of a kind of light regularization, then the feature weighting is an adjustment of the form of that regularization: It asks the optimization to pull some components of β toward zero harder than others. In our view, feature weighting should be considered part of the investigator's choice of regularization. The feature weighting can be thought of as altering the details of that choice, or it can be thought of as making the standard min-norm choice but in a carefully chosen, rescaled basis.

In the most general case, you have not just a set of feature weights Λ^{-1}, you also have a set of data point weights C^{-1} (which, in standard settings, would be the inverses of the variances of the noise affecting the data points y_i). When you put these all together, the feature-weighted, data-weighted least squares predictions are given by either of these two equivalent expressions:

$$\hat{Y}_* = X_*(X^\top C^{-1} X + \Lambda)^{-1} X^\top C^{-1} Y \qquad (23)$$

$$\hat{Y}_* = X_* \, \Lambda^{-1} X^\top (X \, \Lambda^{-1} X^\top + C)^{-1} Y. \qquad (24)$$

The equivalence of these is due to the Woodbury matrix identity (Henderson & Searle 1981). These expressions appear in Bayesian-inference contexts (see, for example, Hogg et al. 2020) when Λ^{-1} is the variance of the prior on the parameter vector β and C is the variance of the noise on the data vector Y. They are solutions to the optimization

$$\hat{\beta} = \arg\min_{\beta} \|C^{-1/2}(Y - X\,\beta)\|_2^2 + \|\Lambda^{1/2}\,\beta\|_2^2. \qquad (25)$$

The units of C must be the units of Y squared, and the units of Λ^{-1} must be the square of the ratio of the units of Y to the units of X (the square of the units of β). These weighted-feature, weighted-data least-square forms (23) and (24) are good because (if the weightings are chosen appropriately) they lead to smooth solutions that are not required to pass precisely through every data point. This is appropriate in common, real situations in which data are noisy and the world is smooth. Which form to choose, between (23) and (24), depends on a few things, but primarily p/n. If $p < n$ it is both faster and more stable to use (23); if $p > n$ it is faster and more stable to use (24). We show a toy example of a fit with both feature weights Λ^{-1} and data weights C^{-1} in Figure 7.

6. How to Set the Number of Parameters and the Double Descent Phenomenon

There is a lot of literature analyzing the performance of linear regressions as a function of the sizes n and p, regularization strengths and forms, and so on (for example, Dobriban & Wager 2018; Bartlett et al. 2020; Hastie et al. 2019). They often refer to the "risk," which is a statistics term for the expected squared error (mistake) made when predicting new data not in the training set. In order to deliver values or bounds on the risk, this literature depends on knowing how the data were generated, or the family of distributions from which the data (X and Y in our nomenclature) were drawn. When working with real data, we do not get this luxury (real data comes without documentation). Indeed, understanding the generating process of the world is the goal of investigations
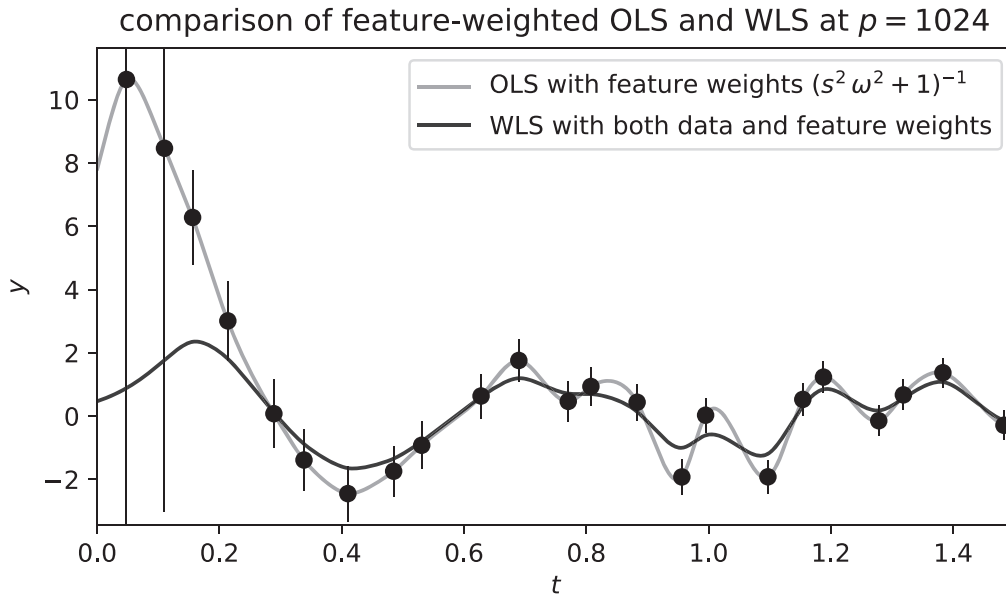
Figure 7. Comparison of the feature-weighted OLS shown in Figure 6 to the same but also including data weights C^{-1}, as in Equations (23) and (24). Similarly to Figure 3, the data weights on the diagonal of C^{-1} are the inverses of the squares of the uncertainties shown as vertical error bars. The feature weights encourage the prediction to be smooth; the data weights permit it to be even smoother because they permit the prediction to miss the training data. Prediction results now depend on the specific amplitude or prefactor multiplying the feature weights; a particular value of $0.07\, f^2(\omega)$ was chosen for the diagonal of the feature weight matrix Λ^{-1} for this demonstration.

in the natural sciences (such as astronomy); the investigator does not know the generating process at the outset.

Given a data set (location–data pairs t_i, y_i), what is the best way to empirically estimate, from those data, the out-of-sample prediction error? That is, how do we estimate how well we are likely to predict new data? A reasonable answer to this is cross-validation: In cross-validation, we leave out a part of the data (in the most extreme form, leave out one single data point at a time), train the model using all but the left-out part, and predict the left-out part. Then this process is iterated over all choices of what part (or point) to leave out. This process is illustrated in Figure 8, where we show 23 fits, each of which is trained to the 22 points remaining when we leave one of the $n = 23$ points out. Also shown is the prediction, in each leave-one-out fit, for the left-out point.

The prediction for the mean-squared error (MSE) for new data is the mean of the square of the differences between the leave-one-out predictions and the left-out data. This leave-one-out cross-validation MSE (CVMSE) will be different for different choices for the basis, size (p), and regularization (including feature-weight functions) of the fits you do. It is a fairly reliable and well-studied method for assessing predictive accuracy (see, for example, Stone 1974), although it does depend on some assumptions (for example, that data points are not duplicated or strongly corrrelated, and that the predictive information in the data is distributed among multiple individual data points). An example of CVMSE for two models (the feature-weighted OLS and the ridge regression) are shown for our toy data, as a

function of the number of features p, in Figure 9. The predictive accuracy is indeed a very strong function of p.

In particular, the CVMSE for the feature-weighted OLS fits is bad when $p \approx n$, and much better at $p \ll n$ and $p \gg n$. This phenomenology is not particular to this problem. It is extremely general. There is an effect known in many kinds of regression called "double descent" or "peaking phenomenon" or "jamming" in which predictive accuracy becomes very poor when the number of free parameters p comes close to the number of data points n (Jain & Chandrasekaran 1982; Spigler et al. 2019; Belkin et al. 2019; Geiger et al. 2019). For linear models, the "risk"—the out-of-sample prediction error—blows up when the model capacity just becomes excessive at $p = n$ (Hastie et al. 2019). The fundamental reason for this phenomenon is that the ordinary least-squares estimates (8) and (16) require computation of the inverses $(X^\top X)^{-1}$ and $(X\, X^\top)^{-1}$ respectively, which are very badly conditioned around $p \approx n$. This translates to a large variance for the estimator $\hat{\beta}$ which implies a large risk. One way to think about this is that when the condition number of the matrix $X^\top X$ or $X\, X^\top$ is large, some directions in the data space —some linear combinations of the elements of the data vector Y —are very strongly amplified.[10] This makes the regression

---
[10] The condition number comes back up below in Section 9. It turns out that when the condition number is large, not only do the output predictions become very sensitive to the input training data, but also the numerical (computational) stability of the linear algebra can also be badly affected. But we emphasize here that the risk goes bad when the condition number is large *even if* it is possible to perform the linear algebra correctly at high precision.
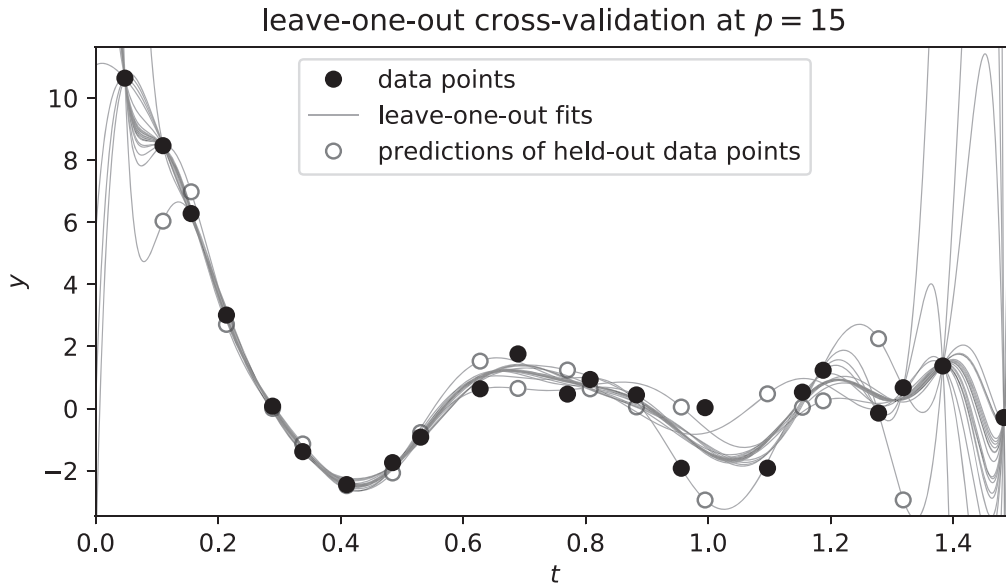
Figure 8. A demonstration of leave-one-out cross-validation for one particular model. The 23 lines are the 23 fits, for each of which one data point was held out. Also plotted on each of the 23 lines is the prediction made for that fit's held-out data point. Some of the predictions do not appear within the plot window, because the predictions obtain large amplitudes at the edges.
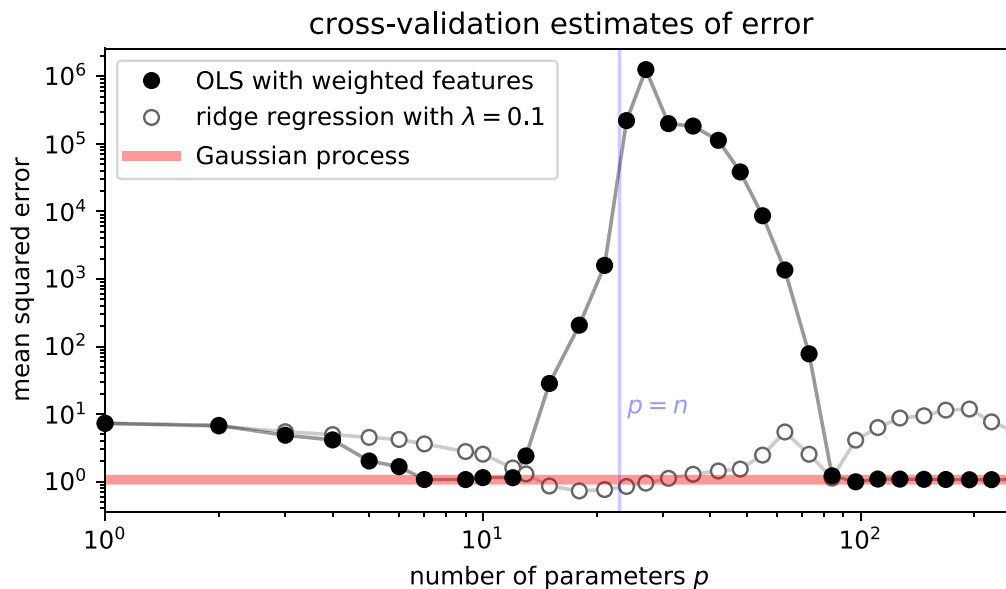

Figure 9. Leave-one-out cross-validation estimates of mean squared prediction error for the feature-weighted OLS fits, as a function of the number of features p. Also shown are the same for ridge regression with $\lambda = 0.1$ and also the Gaussian process that we will introduce in Section 7 and show in Figure 10; the GP result is shown as a flat line because it does not have an associated number p. In detail the first (lowest-t) and last (highest-t) data points were not used in computing the mean squared error; that choice is debatable, but we are imagining an assessment of the quality of the *interpolations*, not extrapolations, of these models. Note that the OLS predictions are much, much worse at $p \approx n$ (check out all the orders of magnitude on the vertical axis) than they are at very low or very high p, but that ridge regression does not show this behavior.

unreasonably sensitive to noise in the data. The high-risk behavior at $p \approx n$ disappears with certain kinds of regularization. These include the ridge regularization shown in Figure 9 (also known as Tikhonov regression), but also early stopping

(Hastie et al. 2019), and dimensionality reduction (Huang et al. 2020).

Sometimes the word "over-fitting" is used to describe models that are too flexible. In our view, a model is properly

called "over-fit" when the prediction of the regression on held-out data—what the statisticians call "risk"—is bad. So over-fitting happens not universally when the number of parameters is large, but instead when the number of parameters is comparable to the number of data points, and (also) the regularization is inappropriate. Figure 9 shows that models with both small and large numbers p of parameters can make good predictions for held-out data, and that regularization can also protect a model from over-fitting, even when $p \approx n$. We do not mean to imply that over-fitting is not a problem in regression; it can be! Our recommendation is just that the problem of over-fitting be analyzed empirically through cross-validation, not through intuitive ideas about the number of parameters.

The CVMSE gets better and worse at different values of p. That's not surprising; there are places where the fit basis and size and feature weighting are all more appropriate for your specific problem. These good-CVMSE places are the best places to work, if we can afford to search for them and find them. In general, if we care about predictive accuracy, it is worth doing a search for the values of p (and other hyper-parameters, like regularization strengths and feature-weighting function parameters) that minimize the CVMSE. The minimum-CVMSE choices for p and the hyper-parameters will generally be very close to the choices that lead to the best predictions for new data.

That said, there is one more point to make about over-fitting, which is that, when our data set is small, it is possible to over-fit the cross-validation. This problem is beyond the scope of this *expository note*; all we will say here is that it is not possible to make precise settings of many hyper-parameters through cross validation. Because in cross validation we are making use of our data to set the properties of our regression, there are dangers of over-adapting our method to our data. It is safer to have a fully independent validation data set, but this is rarely practical (and it does not completely protect us either; see Recht et al. 2018).

7. The Gaussian Process: The Limit of Infinite Features

The Gaussian process (GP; see Rasmussen & Williams 2005 for a complete introduction) is a non-parametric regression that takes training data y_i at coordinates t_i, plus a kernel function, and makes predictions $\hat{y}_*$ for new data at new positions t_*. The Gaussian process mean prediction looks like

$$\hat{Y}_* = K_* K^{-1} Y, \qquad (26)$$

where K is a square $n \times n$ kernel matrix between training locations and themselves

$$[K]_{i,i'} = k(t_i, t_{i'}), \qquad (27)$$

and K_* is the same except it is the rectangular kernel matrix between test locations t_* and training locations t_i; the function $k(\cdot,\cdot)$ is a symmetric, positive semi-definite kernel function.

Stated this simply, the GP looks like magic; our goal here is to connect this to the feature-weighted OLS from Section 5.

First, what does it mean for a kernel function $k(\cdot,\cdot)$ to be positive semi-definite? It means that for all sets of n locations $\{t_i\}_{i=1}^n$ the $n \times n$ kernel matrix K made from the function according to (27) has only non-negative, real eigenvalues. The kernel function must be positive semi-definite because, among other things, it is describing the variance of a process, and variances are always non-negative. It turns out that we can guarantee that a kernel function is positive semi-definite if we can show that it is, itself, the Fourier transform of a non-negative function (Bochner's Theorem, see for example Folland 1994).[11]

Although the kernel matrix K is, by construction, always positive semi-definite, it can have extremely bad or even infinite condition number. That is, it can lead your code or implementation into very unstable linear-algebra operations. We discuss how to handle these issues below in Section 9.

Above, we called the GP prediction the "mean." This is because the Gaussian process is a model for a mean *and variance* in function space. In addition to the mean $\hat{Y}_*$ predicted[12] in (26), the GP also predicts a *variance* in the Y_* space around that mean. In this *expository note*, we are going to treat the GP as producing only a mean prediction, where the shape of the kernel function matters, but the amplitude of the kernel function does not (the prediction is a ratio of kernel matrices, so the kernel amplitudes cancel). However, in Bayesian-inference contexts the amplitude of the GP kernel is important, and the predicted variances are important. When we consider only the mean prediction of the GP, as we do here, then the GP is a kind of linear filter that operates on data Y and predicts or interpolates to new data Y_*. Classically, this linear filter is sometimes called kernel regression, a Wiener filter, or *kriging* (possibly because of Krige 1951).

Importantly for us, there is a strong connection between the feature-weighted OLS and the GP. In particular, when we take the limit of infinite features ($p \to \infty$; provided the limit exists) we get kernel matrices K in place of the $X \Lambda^{-1} X^\top$ matrix products:

$$\lim_{p \to \infty} X \Lambda^{-1} X^\top = K \qquad (28)$$

$$\lim_{p \to \infty} X_* \Lambda^{-1} X^\top = K_*, \qquad (29)$$

where Λ^{-1} is the diagonal matrix of weights, and element i, i' of K is obtained by evaluating a kernel function $k(t_i, t_{i'})$.

---

[11] It is *not* required that the function $k(\cdot,\cdot)$ itself be non-negative; non-negative definiteness is a different condition entirely from non-negativity.

[12] There is an additional point worthy of mention here, which is that the expression (26) is implicitly for a GP with a zero prior mean. There is a more general expression that subtracts a prior mean function from the Y values and adds the prior mean back in to the $\hat{Y}_*$ values. See Rasmussen & Williams (2005) for all the math.

Equivalently, the limit is

$$\lim_{p \to \infty} \sum_{j=1}^{p} [\Lambda^{-1}]_{jj}\, g_j(t) g_j(t') = k(t, t'), \qquad (30)$$

where we have used the diagonality of Λ to make a single sum over j. The specific form of the kernel function $k(\cdot,\cdot)$ depends on the basis (the features) we choose, and the weighting of the basis functions in the OLS.

The connection between the infinite basis chosen (the form and weighting of the features) and the kernel function is governed by Mercer's theorem (see Minh et al. 2006). However, if the basis is Fourier, as we chose above in (3), and the spacing between modes ($\Delta\omega = \omega_{j+2} - \omega_j$) is small enough, the kernel approximates the Fourier transform of the square of the weighting function $f(\omega)$ we use to weight the features. The convergence of the minimum weighted norm interpolation solution to the kernel-induced solution is formally discussed in recent work (Li 2020).

In the case of the specific example of weight function $f(\omega)$ given in Equation (21), we can connect our feature-weighted OLS to an equivalent GP if we know the Fourier transform of the square of $f(\omega)$. We chose that specific form for $f(\omega)$ because it has a square that is a member of a Fourier-transform pair:

$$\mathscr{F}[F(t)] = [f(\omega)]^2 \qquad (31)$$

$$F(t) = \sqrt{\frac{\pi}{8}} \left(1 + \frac{|t|}{s}\right) \exp{-\frac{|t|}{s}}. \qquad (32)$$

This latter function is also known as the Matérn 3/2 kernel function; it will become the kernel function for the Gaussian process when we take the limit. In the limit $\Delta\omega \to 0$ the specific example of the $p \to \infty$ feature-weighted OLS given here becomes a GP with kernel function $F(\cdot)$ (under the mild assumptions that guarantee that the discrete Fourier transform converges to the continuous Fourier transform, see Epstein 2005):

$$k(t_i, t_{i'}) \to F(t_i - t_{i'}) \qquad (33)$$

$$= \sqrt{\frac{\pi}{8}} \left(1 + \frac{|t_i - t_{i'}|}{s}\right) \exp{-\frac{|t_i - t_{i'}|}{s}}. \qquad (34)$$

Technically, the kernel function $k(\cdot,\cdot)$ converges to the Fourier transform of the square of the weighting function $f^2(\cdot)$ in the limit $\Delta\omega \to 0$ (the definition of $k(\cdot,\cdot)$ already assumes $p \to \infty$). However, provided that the spacing of the Fourier modes in frequency space is $\Delta\omega \ll 1/\Delta t_{\max}$ and the maximum frequency is $\lfloor p/2 \rfloor \Delta\omega \gg 1/\min(s, \Delta t_{\min})$, where

$$\Delta t_{\min} \equiv \min_{i \neq j} |t_i - t_j| \qquad (35)$$

$$\Delta t_{\max} \equiv \max_{i,j} |t_i - t_j|, \qquad (36)$$

it will be true that the OLS with feature weights $f(\omega)$ will closely approximate the mean of a GP with kernel $k(\Delta t) = \mathscr{F}[f^2]$ (Epstein 2005). The comparison of the OLS and GP is shown in Figure 10.

In our toy examples, we use the feature weighting that generates the Matérn 3/2 kernel. It is important to emphasize that there are literally infinite alternative choices that can be made here, and hundreds even if you restrict to kernels with known closed forms. Indeed, any function $F(t - t')$ that has a finite, all-positive Fourier transform $f^2(\omega)$ can be substituted for the Matérn kernel and associated frequency weighting function we use here.

The particular kernel we obtain in Equation (34) is a stationary kernel, meaning it depends only on absolute values of time differences $|t - t'|$. Not all $p \to \infty$ kernels will be stationary. The limit $p \to \infty$ of $X\, \Lambda^{-1} X^\top$ in (29) leads to a stationary kernel $k(\cdot,\cdot)$ in this case because the feature embedding in X is a set of sines and cosines. Sines and cosines form a basis for translation-invariant function spaces. If we had made a different choice for our basis, such as polynomials or wavelets, we would have obtained a non-stationary kernel in the $p \to \infty$ limit.

At the end of Section 5, we discussed including not just feature weights in a matrix Λ^{-1} but also data weights in a matrix C^{-1}. The GP also permits this, and it is often a very good idea. The generalization of Equations (23) and (24) to the GP case is

$$\hat{Y}_* = K_*(K + C)^{-1}\, Y. \qquad (37)$$

As before, in contexts where we have independent uncertainties on each element y_i of our training data Y, C^{-1} would naturally be set to the diagonal matrix containing the inverses of the variances of those uncertainties (so C would contain the variances). This form (37) is used a lot in astronomy and cosmology (for example, Zaroubi et al. 1995; Aigrain et al. 2016; Foreman-Mackey et al. 2017) and it is also the standard form given for the Wiener filter, where the kernel function generating K is the Fourier transform of the "signal power" and the kernel function generating C is the Fourier transform of the "noise power." In this form, the amplitude of the kernel function matters, because it is competing, in some sense, the variance K of the GP against the variance C of the noise. It is not sufficient to get the shape of the kernel function right to make a stable prediction of the mean, when using form (37).

We remark that a naive implementation of the regularized regression using (37) has an $O(n^3)$ time complexity. However, Gaussian processes can be implemented in something close to $O(n)$ for certain classes of one-dimensional models (Ambikasaran et al. 2015; Foreman-Mackey et al. 2017). Given this, you might ask: Why use $O(n^3)$ feature-weighted over-parameterized regressions when you could just use a fast GP? Unfortunately there are not algorithms that are so fast for higher dimensional GPs, and there are no good algorithms at all for many situations (for example GPs on the surface of a sphere). Furthermore, the standard GP implementations focus on stationary GPs, which do not encompass all the options that one might construct with non-stationary basis functions. Given all this, it is often the case that
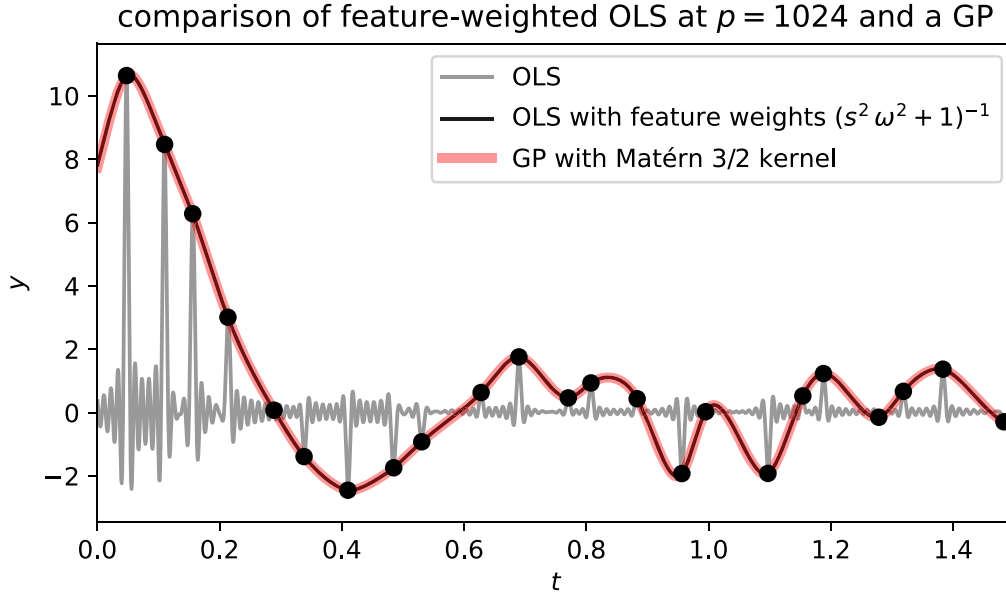
Figure 10. Comparison of feature-weighted ordinary least squares with a Gaussian process. The figure shows the OLS fit, the feature-weighted version with the particular feature weighting given in (21), and the GP fit using the kernel (32) that is the Fourier transform of the square of the feature weighting. The feature-weighted fit and the GP are essentially identical, as expected.

one should or must use linear-regression frameworks, when appropriate GPs are not available.

8. Uncertainties on the Predictions

Often we need to compute not just a prediction, but also an uncertainty on that prediction. How faithfully we must compute that uncertainty will depend strongly on the context in which we are doing the fitting or interpolation. However, it is often the case in the physical sciences that predictions are required to come with good or conservative estimates of uncertainty. There are four-ish sources of uncertainty in the predictions we are making in problems like these: (A) The data points y_i we have are individually noisy. (B) There are finitely many of those data points (there are n of them) and there are gaps between them in the location space t. (C) The data points might have uncertain locations or location measurements t_i. (D) And the predictions we make depend on hyper-parameter choices, such as the form of the basis, the number of parameters p, and any regularization or feature weighting. These four different sources of uncertainty propagate differently and are differently "simple" to deal with. In particular it turns out that the easiest sources of uncertainty to understand and propagate are those coming from (A) the noise in the y_i and (B) the number n and locations t_i of the data. The uncertainties coming from (C) uncertainties in the locations t_i, and (D) model choices, are both much harder to model and propagate.

In the best-case scenario, we might have accurate estimates of the variances $[C]_{ii} = \sigma_i^2$ of the noise contributions affecting our training data values y_i, the noise might be Gaussian with zero mean, we might have locations t_i that are extremely

accurately known, and we might have justifiable prior variances $[\Lambda^{-1}]_{jj}$ on our parameters β_j. In this case, we can assume that the linear model with p parameters is a good model for our data, and write down a likelihood function and a prior. In this case, we can turn the Bayesian crank and deliver a posterior density for the predicted values Y_* at the test locations t_*. The second derivative of the natural logarithm of the posterior density can then be processed into a standard error on the predictions $\hat{Y}_*$ (or we can deliver full posterior densities somehow). The details of this are way beyond the scope of this *expository note*, but this (or a closely related) problem is discussed in some detail by us elsewhere (Hogg et al. 2020).

That scenario is best-case, but it does involve a lot of assumptions. That is, if we do not trust our estimates of the noise variances, or if we think the noise might be non-Gaussian, then the likelihood approaches will give poor uncertainty estimates. In this sense, it is more conservative to make use of more empirical methods for uncertainty estimation. The leading candidates are jackknife or bootstrap resamplings (for example, Efron 1979). Here we will focus on jackknife.

The idea of jackknife resampling is that we make k subsamples of the data, in each of which we have dropped (or held out) a unique fraction $1/k$ of the data. The variance of the results (in this case the prediction $\hat{Y}_*$) across the jackknife samples can be transformed into an estimate of the variance of the estimator acting on the full data set. In the most extreme form—leave-one-out jackknife—we set $k = n$. In this case, the jackknife estimate $\hat{\Sigma}_*$ of the variance of the best-fit predictions

$\hat{Y}_*$ is given by

$$\hat{\Sigma}_* = \frac{n-1}{n} \sum_{i=1}^{n} (\hat{Y}_{*i} - \hat{Y}_*)(\hat{Y}_{*i} - \hat{Y}_*)^\top, \qquad (38)$$

where $\hat{Y}_{*i}$ is the estimate of Y_* made after *leaving out* the one training point t_i, y_i, and $\hat{Y}_*$ is the estimate made using *all* the data. The pre-factor $(n-1)/n$ in (38) comes from the fact that the jackknife subsamples are extremely correlated; the pre-factor is computed to amplify the jackknife variance into an unbiased estimate of the variance of the estimator $\hat{Y}_*$. The square-root of the trace of this uncertainty variance $\hat{\Sigma}_*$ (that is, what we might call the jackknife estimate of the standard error) is shown in Figure 11. Note the similarity between jackknife and cross-validation; it is relevant and important: These empirical estimates of uncertainty and prediction error are closely related.

One issue with jackknife that is worth noting is that it (effectively) assumes that the data being held out are uncorrelated in a measurement sense. And yet, many data we take might have spatial or time correlations in the measurement uncertainties. If you suspect that your data include local correlations in measurement uncertainties (from, say, calibration, or instrument drift), it is more responsible to jackknife by holding out groups of correlated points, rather than individual points. A full discussion of responsible jackknife design is beyond our scope, but many good references exist (for example, Efron 1982).

These methods—full likelihood analysis and jackknife—take into account (A) the noise in the data y_i and (B) the number and spacing of the training data points. They do not, by themselves, take into account (C) the uncertainties on the locations t_i. There are no simple methods for uncertainty propagation from the locations t_i into the predictions $\hat{Y}_*$. One option is to resample the t_i according to our uncertainty estimates, and re-do the fits. That's potentially expensive, and overly conservative (because it ignores the information about the t_i coming from the y_i). Another is to linearize the first derivative of the best-fit curve $y(t)$ and propagate uncertainty using those derivatives. This is not conservative, because it only works if the location uncertainties are small relative to substantial changes in the slope of the predictions with t. The most extreme option would be to *simultaneously fit* for the prediction $\hat{Y}_*$ and *all* of the noisily measured t_i. That's a great idea, but that fit would be extremely computationally expensive, and no reasonable fit objective would be convex. All of these ideas are out of scope here.

Finally, in order to take account of (D) the uncertainty coming from our choices of basis, number p, and regularization, we might have to know (or learn) the distribution over *these* things. Since these choices are hyper-parameters, any inference that propagates uncertainties coming from these

choices would have to be hierarchical in structure. That is also out of scope.

9. Implementation Notes

All of the code used to make the figures for this *expository note* is available publicly.[13] Although the examples are toys, the implementation of everything can be generalized for real-data situations. In our code, there are some aspects of the linear-algebra implementation that might seem odd. Here are some comments.

Mostly, linear algebra stability comes down to the *condition number* of the matrix in question. The condition number, for our purposes, is the ratio of the largest eigenvalue (for non-negative definite matrices) to the smallest *non-zero* eigenvalue. For rectangular feature matrices it is the ratio of the largest singular value to the smallest nonzero singular value.

When the condition number is large, different linear-algebra approaches will differ. For example, the function call `solve(A, b)` should give us the best estimate it can for the product $A^{-1} b$, whereas the function call `dot(inv(A), b)`, which is the same on paper, will give us the dot product between `b` and the best estimate that `inv()` can find for the inverse of `A`. When the condition number of A is large, these two values can be very different, and the `solve()` will be better.[14] We recommend to use `solve(A, b)` and never `dot(inv(A), b)` unless there are compelling code-structure reasons to use the latter, such as repeated calls (but even then, a Cholesky decomposition followed by repeated Cholesky solves is probably a more stable solution).

If our condition number gets very large, even `solve()` can give bad results, because very small eigenvalues of the matrix are being poorly estimated and then inverted. That's unstable. It is better to zero out those small eigenvalues—it is better to destroy bad information than to use it—and perform a pseudo-inverse in those cases. So if we really want to be safe (and we do, here) we should really use `lstsq(A, b, rcond=tiny)` instead of `solve(A, b)`. The `lstsq()` function requires a `rcond` input, which says at what (dimensionless) precision to zero-out low eigenvalues. It usually makes sense to set this dimensionless `rcond` input to something like machine precision, which is about `1e-16` for our current hardware–software setups.

One might think that all of this is academic, but it really is not when the number of features p is close to the number of data points n. For example, in our toy-data OLS experiments in this paper with $n = 23$, the matrix $X^\top X$ has a condition number (ratio of largest eigenvalue to smallest nonzero eigenvalue) that

---

[13] https://github.com/davidwhogg/FlexibleLinearModels

[14] This claim contradicts what is stated in the abstract of Druinsky & Toledo (2012), but this claim is based on our real numerical experiments on real matrices, so we stand by it. What is uncontroversial is that `solve(A, b)` will always perform non-worse than `dot(inv(A), b)`; sometimes it will perform far better.
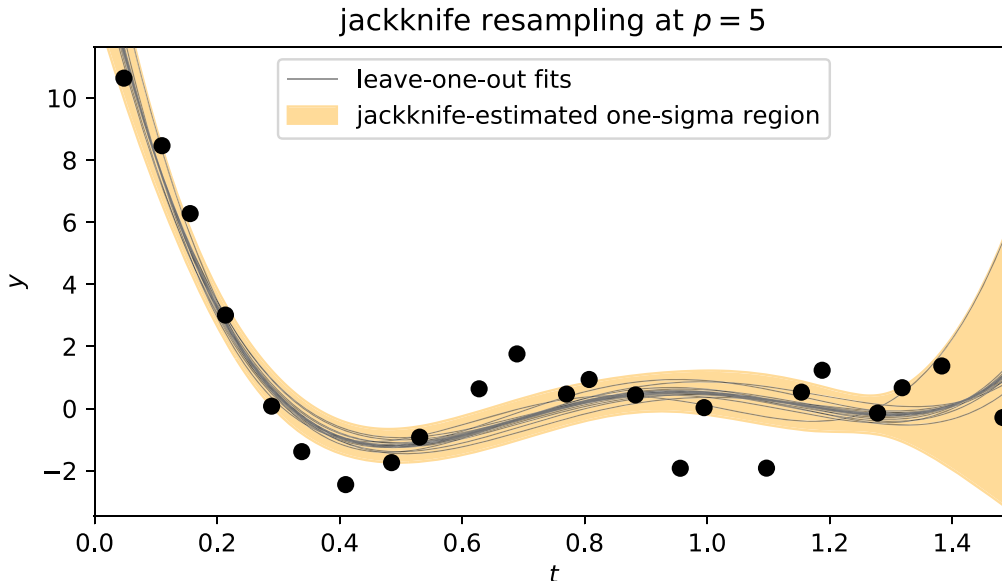
Figure 11. The jackknife-estimated uncertainty on an ordinary least squares fit. What is shown is the diagonal of the empirical jackknife estimate $\hat{\Sigma}_*$ of the variance tensor on the prediction $\hat{Y}_*$ of an OLS fit. In other words, the off-diagonal entries of $\hat{\Sigma}_*$ are being ignored here; there are important covariances that are hard to visualize (and beyond our scope). The 1σ region is larger than the span of the jackknife trials; this is because the jackknife trials are strongly correlated; the jackknife formula (38) compensates for those correlations.

saturates machine precision for the entire range $12 < p < 50$. And that's for $n = 23$; things generally get worse as n gets larger.

Related to this, if we have a choice between formulations that involve an inverse of a square, like $(X^\top X)^{-1} X^\top Y$, and formulations that involve a pseudo-inverse, like $X^\dagger Y$, you should do the latter, because $X^\top X$ has the *square* of the condition number of X. When we want to execute $X^\dagger Y$, as we do in (17), we should again use `lstsq()`, which was literally designed for these applications; we use `lstsq(X, Y, rcond=tiny)`. This function returns the best estimate it can for $X^\dagger Y$, so it should be better than computing a pseudo-inverse `pinv(X)` and then matrix-multiplying `pinv(X)` with `Y`. Again here it usually makes sense to set this `rcond` input to something like machine precision.

Once p is large enough, if we are using a feature weighting with Λ^{-1} with diagonal entries $[\Lambda^{-1}]_{jj}$ that decrease to zero with j, at some point, at machine precision, the additional columns we are adding to X are effectively all zeros. They will literally underflow the linear-algebra representation. That's not a problem if we implement our linear algebra well (that is, use `lstsq()` appropriately), but it does mean that our predictions and cross-validations will saturate at some p (as we see them do in Figure 9).

In multiple places, but especially (23) and (24), we are performing operations on matrices that are diagonal (C and Λ and their inverses are all diagonal). We avoid ever constructing diagonal matrices. We multiply X or $X^\top$ by a diagonal matrix by just multiplying the rows (or columns) by the diagonal entries. And we invert by just inverting the diagonal entries. We recommend to avoid constructing and operating on operators that are almost entirely zeros, unless we are using a very efficient sparse linear algebra implementation.

And finally, it is most numerically stable to operate on the smallest matrices we can. For example, when you have the choice between the formulation in Equation (23) and that in Equation (24), we should choose the former when $p < n$ and the latter when $p > n$. That way we are always doing the heavy linear algebra (`solve()` and `lstsq()` function calls) at a size $\min(p, n)$, which is both faster and more stable than linear algebra at $\max(p, n)$. And it is much faster when we make these choices correctly: Linear algebra scales naively as the dimension *cubed*. (In practice—with excellent packages—it actually scales with a power more like 2.6 than 3, but still!)

10. Discussion

This *expository note* is about linear fitting with very flexible models, for interpolation, prediction, and de-noising of data. We make a case for considering using very big models, but being intentional about regularization. These settings (over-parameterized, but carefully regularized) are adaptive and useful, and they connect, as we showed, to Gaussian processes in the limit, which are well-studied workhorses of machine learning and data science.

At the present day, most important regression techniques involve *nonlinear fitting*, with the key example being *deep*

learning (LeCun et al. 2015, for example). We did not discuss any of that. However, many of the high-level lessons in this *expository note* carry over: All of these tools work well when the function space is flexible but also carefully regularized (Belkin et al. 2019). In these nonlinear settings, regularization takes many additional new forms, like early stopping (Yao et al. 2007), dropout (Srivastava et al. 2014), and restricted network structure (such as convolutional; see for example Bishop 2006).

We only considered the setting in which we care about predicting new data Y_* and never the setting in which we care about the internal parameters β of the linear fit. However, if we are, say, measuring the power spectrum of a process (as we are, often, in cosmology), then we care about these amplitudes themselves. In this case, any regularization we apply takes the role of a prior or prior information. That prior must be chosen (in those cases) with great care, because the answers we get will be influenced by that choice; when it does not properly conform to our true beliefs, our answers will be distorted in bad ways. And, technically, when we take this view (that the parameters β matter), we also have to get serious about the noise model on the data. That is, the beliefs encoded in the data noise variance tensor C also must represent our true beliefs if we do not want our answers distorted in bad ways. All of that is out of scope here, but we say things about it all elsewhere (for example, Hogg et al. 2010).

In many cases, when the goal is just to interpolate or de-noise data, investigators use running means (or medians), low-pass filters, or explicit interpolators (like cubic spline interpolation). These methods all have close relationships with what is written here. Indeed, a running mean, a low-pass filter, and a cubic-spline interpolator can all be written as a linear operator (the details of which depends on the locations t_i of the data) acting on the input data vector Y, just like our linear fits produce linear operators. This means that in many cases, these methods can be translated into versions of the methods we have presented in this *expository note*. A full translation is beyond our scope, though. And our view is that the value of making explicit investigator choices about basis and the regularization makes the linear fitting approach better in general.

Regularization was perhaps the biggest theme of this *expository note*. But we only really considered variants of L2 regularization (or ridge or Tikhonov). There are other regularizations, even other convex regularizations. A valuable and useful option is L1 or the lasso (Tibshirani 1996), which encourages sparsity—it encourages parameters β_j to take the value $\beta_j = 0$ exactly, where possible. This kind of regularization makes sense when we have prior beliefs that the functional forms we seek will in fact be sparse in our chosen basis. Usually this kind of consideration is less important in interpolation, prediction, and de-noising settings, but it is not unheard of. A nice recent result is that the feature weighting that we employ here can be used with L1 regularization to obtain simultaneously smoothness *and* sparsity (Rauhut & Ward 2016).

We discussed deterministic, ordered expansions like Fourier series and polynomials, but there is another class of *random features* methods (see, for example, Rahimi & Recht 2007) that we did not discuss. Briefly, the idea with random features is that instead of weighting features that are regularly spaced in frequency space with a weighting function $f(\omega)$, we could have generated unweighted features but randomly from a probability distribution $\propto f^2(\omega)$. The same Gaussian process limit appears as $p \to \infty$ (provided we choose those features with appropriately random phases too). These random-feature approaches have rarely been used in the natural sciences, but they are potentially of interest in many applications.

In the toy examples used throughout this *expository note*, we considered data with locations t_i in a one-dimensional location space or ambient space. This was a choice for simplicity of visualization; in principle the locations t_i could be higher dimensional, or live in a different kind of space. The most important consideration about the dimensionality and range of the locations is that the functions $g_j(t)$ have to sensibly take the locations t_i as input. In astronomy and cosmology contexts, it is common for the locations t_i to be positions on the sphere, and it is common for the natural basis functions to be spherical harmonics, for example.

The methods in this *expository note* are all *discriminative*, in the sense that we took the locations t_i to be prior or primary; the goal was to find a function of locations t_i that predicts data y_i. This asymmetry between the locations and the data led to concerning statements in Section 8 when we considered the possibility that the locations t_i themselves might be noisy or uncertain. An alternative formulation to these discriminative models are *generative* models, in which the model attempts not just to predict the y_i from the t_i but instead predicts both the t_i and the y_i. A model that generates both can be used to make predictions y_* at new locations t_* by executing an inference or inverse problem on the generative model. Generative models are generally non-convex—and harder to execute—if we want the relationship between the t_i and the y_i to be nonlinear (for an example of this in astronomy, see, for example, Ness et al. 2015). In these cases we do not get closed-form solutions, and there are not known guarantees (like the Gauss–Markov theorem) about performance. But they are more general, and often more appropriate, especially when both the locations t_i and the data y_i are comparably noisy measurements.

In our toy examples, we used a Fourier series. That is just one choice among many. However, *it is often a great choice*. When we choose the Fourier basis (and—importantly—for every cosine term you include the corresponding sine term), the matrix $X \Lambda^{-1} X^\top$ (and the limiting kernel matrix K at $p \to \infty$) has the property that every matrix element $[X \Lambda^{-1} X^\top]_{ii'}$ depends on (or can be calculated from) just the absolute

difference $|t_i - t_{i'}|$. That is, in this basis, all fitting methods become technically *stationary*. This all relates to the translation-independence properties of the Fourier basis. So although the Fourier basis is just one choice among many, it is the *right choice* when we think our problem has (or might have) certain kinds of translation invariances.

It is a pleasure to thank Viviana Acquaviva (CUNY), Jed Brown (CU Boulder), Edgar Dobriban (UPenn), Nestor Espinoza (STScI), Dan Foreman-Mackey (Flatiron), Alessandro Gentilini, Teresa Huang (JHU), Weilin Li (NYU), Sam Roweis (deceased), Adrian Price-Whelan (Flatiron), Bernhard Schölkopf (MPI-IS), Kate Storey-Fisher (NYU), Rachel Ward (UT Austin), and Lily Zhao (Yale) for valuable conversations and input. S.V.'s work on this project was supported by NSF DMS 2044349, EOARD FA9550-18-1-7007, and the NSF–Simons Research Collaboration on the Mathematical and Scientific Foundations of Deep Learning (MoDL) (NSF DMS-2031985). All code and data used in this project are available at https://github.com/davidwhogg/FlexibleLinearModels.

Appendix
Optimization Arguments

As mentioned in the main text, the optimal solutions of the unconstrained objectives (5) and (11) can be obtained by computing the first-order critical points. The first-order critical points are global minima because all the objectives considered are convex in the regression coefficients β.

To derive the optimal solutions of the constrained optimization (14) and similar, we use a classical result in quadratic optimization (see for instance Nocedal & Wright 2006, Ch. 16): Consider the quadratic optimization problem

$$\arg\min_x \frac{1}{2}x^T G\,x + x^\top c \quad \text{subject to } A\,x = b. \tag{A1}$$

The first-order necessary conditions for x^* to be a solution of (A1) is that there is a vector λ^* (known as Lagrange multipliers) such that the following system of equations is satisfied:

$$\begin{bmatrix} G & A^\top \\ A & 0 \end{bmatrix}\begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix}. \tag{A2}$$

Equation (A2) is typically known as the Karush–Kuhn–Tucker conditions (KKT). If the objective function in (A1) is convex, the KKT conditions are also sufficient for optimality.[15]

Using this formulation it is easy to check that the solution of (14) leads to predictions in (12), and the solution of (19) produce (18). For instance, in order to show the latter we

---

[15] The KKT conditions can also characterize optimality in more general cases. For instance, if A has full row rank and Z is the nullspace of A, if $Z^\top G\,Z$ is positive semidefinite the KKT conditions are sufficient for optimality. If $Z^\top G\,Z$ is positive definite we also have that the solution of (A1) is unique (which is typically the case in the under-parameterized linear regression but not in the over-parameterized).

consider

$$\hat\beta = \arg\min_\beta \|\Lambda^{1/2}\,\beta\|_2^2 \quad \text{subject to } Y = X\,\beta, \tag{A3}$$

or equivalently

$$\hat\beta = \arg\min_\beta \frac{1}{2}\,\beta^\top \Lambda\,\beta \ \text{subject to } X\,\beta = Y, \tag{A4}$$

obtaining the formulation in (A1) for $G = \Lambda$, $c = 0$, $A = X$, $b = Y$ and $x = \beta$. Using the KKT conditions we obtain

$$\begin{bmatrix} \hat\beta \\ \lambda^* \end{bmatrix} = \begin{bmatrix} \Lambda & X^\top \\ X & 0 \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ Y \end{bmatrix}, \tag{A5}$$

(assuming the KKT matrix is invertible). Luckily there exists a complete characterization for the inverse of 2×2 block matrices (see, for example, Lu & Shiou 2002). In particular if A and $D - CA^{-1}B$ are invertible matrices we have

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1}$$
$$= \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix}. \tag{A6}$$

In the particular case of (A5) it suffices to compute the top right block the inverse. If both Λ and $X\,\Lambda^{-1}X^\top$ are invertible we obtain

$$\hat\beta = \Lambda^{-1}X^\top(X\,\Lambda^{-1}X^\top)^{-1}\,Y. \tag{A7}$$

References

Agresti, A. 2015, Foundations of Linear and Generalized Linear Models (New York: Wiley)
Aigrain, S., Parviainen, H., & Pope, B. J. S. 2016, MNRAS, 459, 2408
Ambikasaran, S., Foreman-Mackey, D., Greengard, L., Hogg, D. W., & O'Neil, M. 2015, ITPAM, 38, 252
Bah, B., & Ward, R. 2016, ITSP, 64, 3145
Bartlett, P. L., Long, P. M., Lugosi, G., & Tsigler, A. 2020, PNAS, 117, 30063
Belkin, M., Hsu, D., Ma, S., & Mandal, S. 2019, PNAS, 116, 15849
Bishop, C. M. 2006, Pattern Recognition and Machine Learning (Berlin: Springer)
Dobriban, E., & Wager, S. 2018, AnSta, 46, 247
Druinsky, A., & Toledo, S. 2012, arXiv:1201.6035
Efron, B. 1979, AnSta, 7, 1
Efron, B. 1982, The Jackknife, the Bootstrap and Other Resampling Plans (Philadelphia, PA: SIAM)
Epstein, C. L. 2005, CPAM, 58, 1421
Folland, G. B. 1994, A Course in Abstract Harmonic Analysis (Boca Raton, FL: CRC Press)
Foreman-Mackey, D., Agol, E., Ambikasaran, S., & Angus, R. 2017, AJ, 154, 220
Geiger, M., Spigler, S., d'Ascoli, S., et al. 2019, PhRvE, 100, 012115
Gelman, A., Hill, J., & Vehtari, A. 2020, Regression and Other Stories (Cambridge: Cambridge Univ. Press)
Hastie, T., Montanari, A., Rosset, S., & Tibshirani, R. J. 2019, arXiv:1903.08560
Hastie, T., Tibshirani, R., & Friedman, J. 2009, The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.; Berlin: Springer)
Henderson, H. V., & Searle, S. R. 1981, SIAMR, 23, 53

Hogg, D. W., Bovy, J., & Lang, D. 2010, arXiv:1008.4686
Hogg, D. W., Price-Whelan, A. M., & Leistedt, B. 2020, arXiv:2005.14199
Huang, N., Hogg, D. W., & Villar, S. 2020, arXiv:2011.11477
Jain, A. K., & Chandrasekaran, B. 1982, Handbook of Statistics, Vol. 2 (Amsterdam: Elsevier), 835
Krige, D. G. 1951, Journal of the Southern African Institute of Mining and Metallurgy, 52, 119
LeCun, Y., Bengio, Y., & Hinton, G. 2015, Natur, 521, 436
Li, W. 2020, SIMODS, 3, 414
Liu, S., & Dobriban, E. 2019, arXiv:1910.02373
Lu, T.-T., & Shiou, S.-H. 2002, CMwA, 43, 119
Minh, H. Q., Niyogi, P., & Yao, Y. 2006, Int. Conf. on Computational Learning Theory (Berlin: Springer), 154
Ness, M., Hogg, D. W., Rix, H. W., Ho, A. Y. Q., & Zasowski, G. 2015, ApJ, 808, 16
Nocedal, J., & Wright, S. 2006, Numerical Optimization (Berlin: Springer)

Rahimi, A., & Recht, B. 2007, in NIPS'07: Proc. 20th Int. Conf. on Neural Information Processing Systems (Red Hook, NY: Curran), 1177
Rasmussen, C. E., & Williams, C. K. I. 2005, Gaussian Processes for Machine Learning (Cambridge, MA: MIT Press)
Rauhut, H., & Ward, R. 2016, Appl. Comput. Harmon. Anal., 40, 321
Recht, B., Roelofs, R., Schmidt, L., & Shankar, V. 2018, arXiv:1806.00451
Spigler, S., Geiger, M., d'Ascoli, S., et al. 2019, JPhA, 52, 474001
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014, The Journal of Machine Learning Research, 15, 1929
Stone, M. 1974, Journal of the Royal Statistical Society: Series B (Methodological), 36, 111
Tibshirani, R. 1996, Journal of the Royal Statistical Society: Series B (Methodological), 58, 267
Xie, Y., Ward, R., Rauhut, H., & Chou, H.-H. 2020, arXiv:2006.08495
Yao, Y., Rosasco, L., & Caponnetto, A. 2007, Constr. Approx., 26, 289
Zaroubi, S., Hoffman, Y., Fisher, K. B., & Lahav, O. 1995, ApJ, 449, 446