UNIVERSIDADE
BEIRA INTERIOR

Surrogate Models

Otimização em Engenharia (15235)

2º Ciclo/Mestrado em Engenharia Aeronáutica

2024

Pedro V. Gamboa

Departamento de Ciências Aeroespaciais
Faculdade de Engenharia

0. Topics

- Surrogate-based optimization
- When to use a surrogate model
- Sampling methods
- Surrogate models

1. Introduction

- A surrogate model, also known as a response surface model or metamodel, is an approximate model of a functional output that represents a “curve fit” to some underlying data.
- The goal of a surrogate model is to build a model that is much faster to compute than the original function, but that still retains sufficient accuracy away from known data points.

2. Surrogate-Based Optimization

- Surrogate-based optimization (SBO) performs optimization using the surrogate model, as shown in Fig. 6.01.
- When used in optimization, the surrogate might define the full optimization model (i.e., the inputs are design variables, and the outputs are objective and constraint functions), or the surrogate could be just a component of the overall model.
- SBO is more targeted than the broader field of surrogate modelling.
- Instead of aiming for a globally accurate surrogate, SBO just needs the surrogate model to be accurate enough to lead the optimizer to the true optimum.

2. Surrogate-Based Optimization

- In SBO, the surrogate model is usually improved during optimization as needed but can sometimes be constructed beforehand and remain fixed during optimization.
- Some optimization algorithms interrogate both the surrogate model and the original model, an approach that is sometimes called surrogate-assisted optimization.

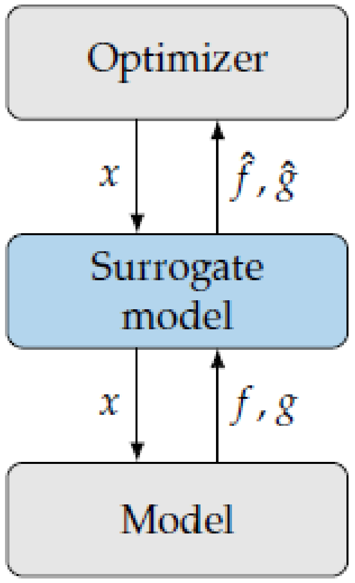


Figure 6.01 Surrogate-based optimization replaces the original model with a surrogate model in the optimization process.

Source: Martins et Ning, 2021

3. When to use a Surrogate Model

- There are various scenarios for which surrogate models are helpful.
- One scenario is when the original model is computationally expensive.
- Surrogate models can be queried with minimal computational cost, but constructing them requires multiple evaluations of the original model.
- Suppose the number of evaluations needed to build a sufficiently accurate surrogate model is less than that needed to optimize the original model directly.
- In that case, SBO may be a worthwhile option.
- Constructing a surrogate model becomes even more compelling when it is reused in multiple optimizations.

3. When to use a Surrogate Model

- Surrogate modelling can be effective in handling noisy models because they create a smooth representation of noisy data.
- This can be particularly advantageous when using gradient-based optimization.
- One scenario that leads to both expensive evaluation and noisy output is experimental data.
- When the model data are experimental and the optimizer cannot query the experiment in an automated way, we can construct a surrogate model based on the experimental data.
- Then, the optimizer can query the surrogate model in the optimization.
- Surrogate models are also helpful when we want to understand the design space, that is, how the objective and constraints (outputs) vary with respect to the design variables (inputs).

3. When to use a Surrogate Model

- By constructing a continuous model over discrete data, we obtain functional relationships that can be visualized more effectively.
- When multiple sources of data are available, surrogate models can fuse the data to build a single model.
- The data could come from numerical models with different levels of fidelity or experimental data.
- For example, surrogate models can calibrate numerical model data using experimental data.
- This is helpful because experimental data is usually much more scarce than numerical data.
- The same reasoning applies to low- versus high-fidelity numerical data.

3. When to use a Surrogate Model

- One potential issue with surrogate models is the curse of dimensionality, which refers to poor scalability with the number of inputs.
- The larger the number of inputs, the more model evaluations are needed to construct a surrogate model that is accurate enough.
- Therefore, the reasons for using surrogate models cited earlier might not be enough if the optimization problem has a large number of design variables.
- The SBO process is shown in Fig. 6.02.
- First, we use sampling methods to choose the initial points to evaluate the function or conduct experiments.
- These points are sometimes referred to as training data.

3. When to use a Surrogate Model

- Next, we build a surrogate model from the sampled points.
- We can then perform optimization by querying the surrogate model.
- Based on the results of the optimization, we include additional points in the sample and reconstruct the surrogate (infill).
- We repeat this process until some convergence criterion, or a maximum number of iterations is reached.

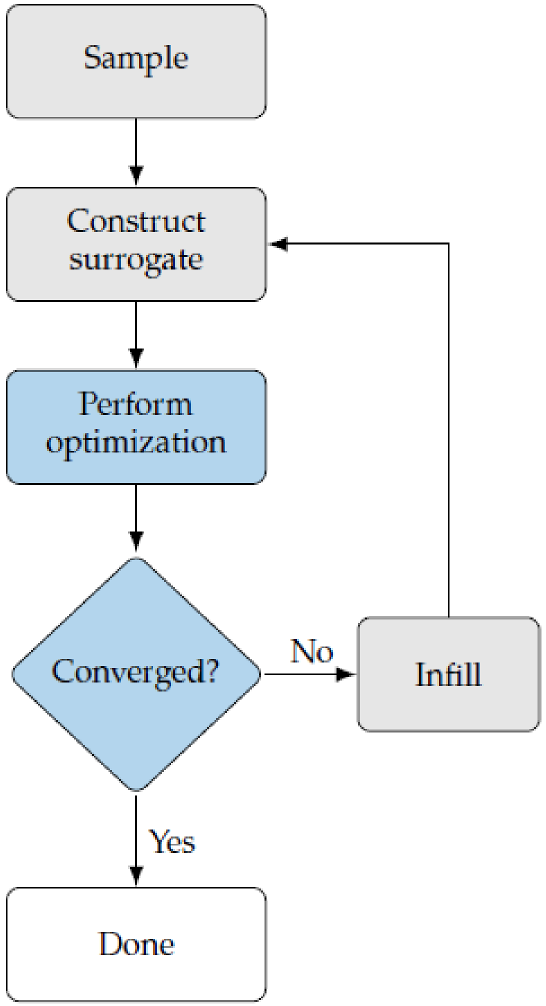


Figure 6.02 Overview of surrogate-based optimization procedure.

Source: Martins et Ning, 2021

3. When to use a Surrogate Model

- In some procedures, infill is omitted; the surrogate is entirely constructed upfront and not subsequently updated.
- The optimization step can be performed using any of the methods we covered previously.
- Because surrogate models are smooth and their gradients are easily computed, gradient-based optimization is preferred (see Chapter 4).
- However, some surrogate models can be highly multimodal, in which case a global search is preferred, either using gradient-based with multistart or a global gradient free method (see Chapter 5).
- This chapter discusses sampling, constructing a surrogate, and infill with some associated optimization strategies.

3. When to use a Surrogate Model

- We devote separate sections to two surrogate modelling methods that are more involved and widely used: kriging and deep neural networks.
- Many of the concepts discussed in this chapter have a wide range of applications beyond optimization.

4. Sampling

- Sampling methods, also known as sampling plans, select the evaluation points to construct the initial surrogate.
- These evaluation points must be chosen carefully.
- A straightforward approach is full factorial sampling, where we discretize each dimension and evaluate at all combinations of the resulting grid.
- This is not efficient because it scales exponentially with the number of input variables.
- One of the significant challenges of sampling methods is the dimensionality.
- For SBO, even with better sampling plans, using a large number of variables is costly.
- We need to identify the most important or most influential variables.

4. Sampling

- Knowledge of the particular domain is helpful, as is exploring the magnitude of the entries in a gradient vector across multiple points in the domain.
- We can use various strategies to help us decide which variables matter most, but for our purposes, we assume that the most influential variables have already been determined so that the dimensionality is reasonable.
- Having selected a set of variables, we are now interested in sampling methods that characterize the design space of interest more efficiently than full factorial sampling.

4. Sampling

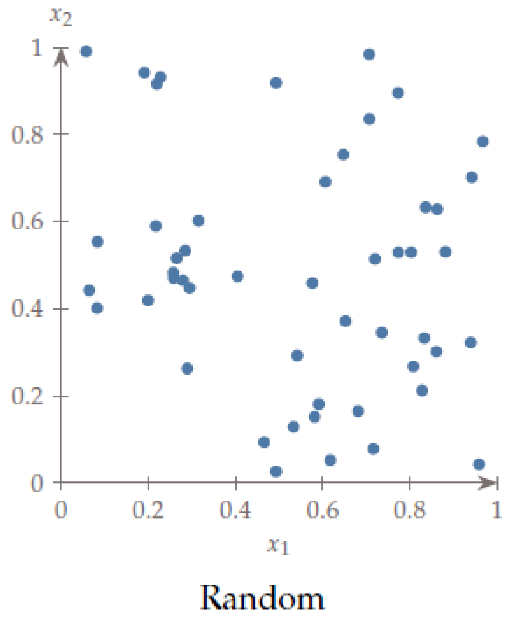
- In addition to their use in SBO, the sampling strategies discussed in this section are useful in many other applications: initializing a genetic algorithm (Section 5.7), a particle swarm optimization (Section 5.8) or a multistart gradient-based algorithm, or choosing the points to run in a Monte Carlo simulation.
- Because the function behaviour at each sample is independent, we can efficiently parallelize the evaluation of the functions.

4.1. Latin Hypercube sampling

- Latin hypercube sampling (LHS) is a popular sampling method that is built on a random process but is more effective and efficient than pure random sampling.
- Random sampling scales better than full factorial searches, but it tends to exhibit clustering and requires many points to reach the desired distribution (i.e., the law of large numbers).
- For example, Fig. 6.03 compares 50 randomly generated points across uniform distributions in two dimensions versus Latin hypercube sampling.
- In random sampling, each sample is independent of past samples, but in LHS, we choose all samples beforehand to ensure a well-spread distribution.

4.1. Latin Hypercube sampling

- To describe the methodology, consider two random variables with bounds, whose design space we can represent as a square.



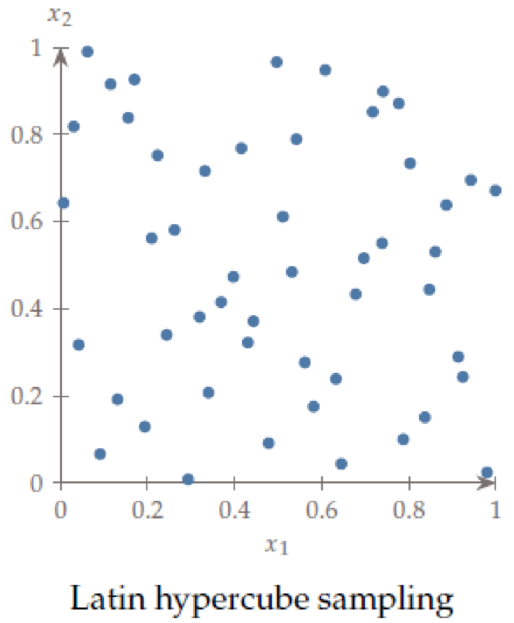


Figure 6.03 Contrast between random and Latin hypercube sampling with 50 points using uniform distributions.

Source: Martins et Ning, 2021

4.1. Latin Hypercube sampling

- Say we wanted only eight samples; we could divide the design space into eight intervals in each dimension, generating the grid of cells shown in Fig. 6.04.
- A full factorial search would identify a point in each cell, but that does not scale well.
- To be as efficient as possible and still cover the variation, we would want each row and each column to have one sample in it.

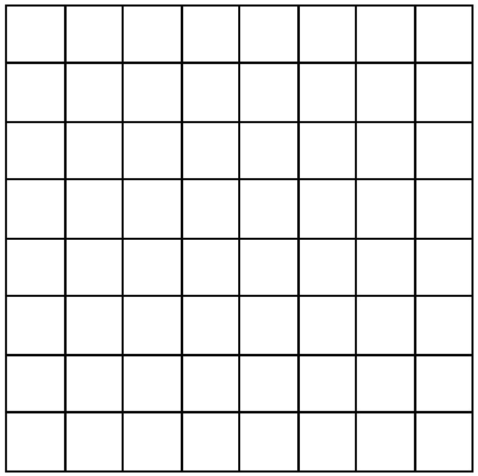


Figure 6.04 A two-dimensional design space divided into eight intervals in each dimension.

Source: Martins et Ning, 2021

4.1. Latin Hypercube sampling

- In other words, the projection of points onto each dimension should be uniform.
- For example, the left side of Fig. 6.05 shows the projection of a uniform LHS onto each dimension.
- We see that the points create a uniformly spread histogram.

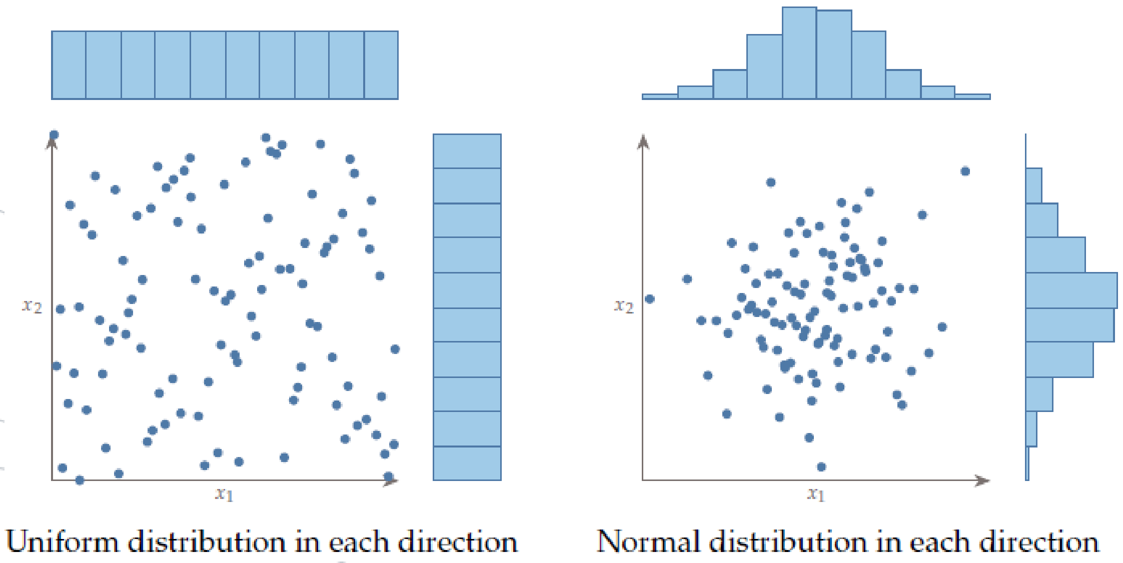


Figure 6.05 Example LHS with projections onto the axes.

Source: Martins et Ning, 2021

4.1. Latin Hypercube sampling

- The concept where one and only one point exists in any given row or column is called a Latin square, and the generalization to higher dimensions is called a Latin hypercube.
- There are many ways to achieve this, and some choices are better than others.
- Consider the sampling plan shown on the left of Fig. 6.06.
- This plan meets our criteria but clearly does not fill the space and likely will not capture the relationships between design parameters well.
- Alternatively, the right side of Fig. 6.06 has a sample in each row and column while also spanning the space much more effectively.

4.1. Latin Hypercube sampling

- LHS can be posed as an optimization problem where we seek to maximize the distance between the samples.
- The constraint is that the projection on each axis must follow a chosen probability distribution.

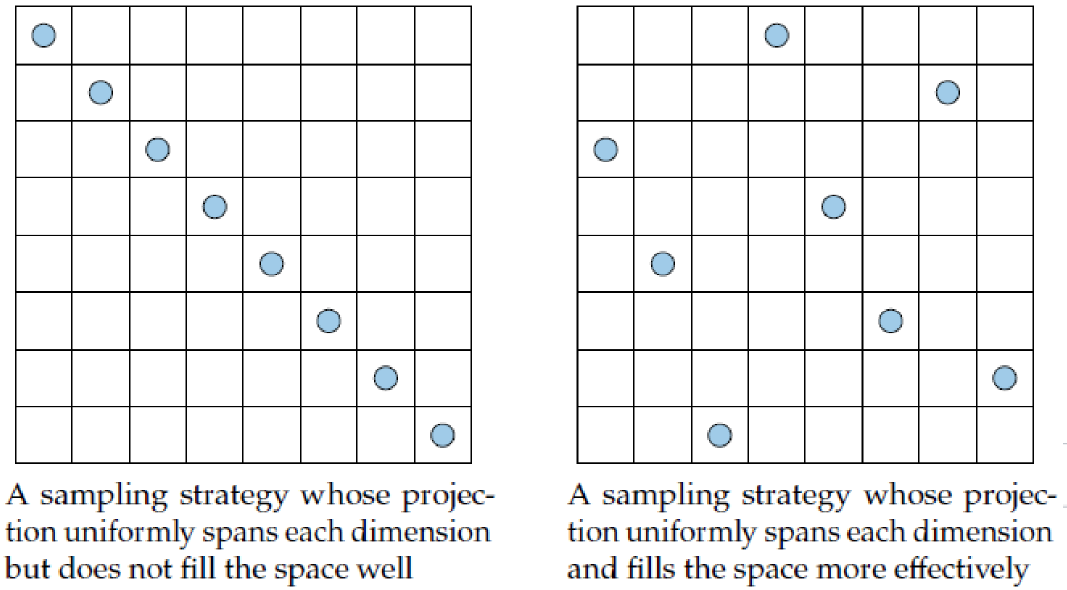

A sampling strategy whose projection uniformly spans each dimension but does not fill the space well
A sampling strategy whose projection uniformly spans each dimension and fills the space more effectively

Figure 6.06 Contrasting sampling strategies that both fulfill the uniform projection requirement.

Source: Martins et Ning, 2021

4.1. Latin Hypercube sampling

- The specified distribution is often uniform, as in the previous examples, but it could also be any distribution, such as a normal distribution, as shown on the right side of Fig. 6.05.
- This optimization problem does not have a unique solution, so random processes are used to determine the combination of points.
- Additionally, points are not usually placed in cell centers but at a random location within a given cell to allow for the possibility of reaching any point in the domain.
- The advantage of the LHS approach is that rather than relying on the law of large numbers to fill out our chosen probability distributions, we enforce it as a constraint.
- This method may still require many samples to characterize the design space accurately, but it usually requires far fewer than pure random sampling.

4.1. Latin Hypercube sampling

- Instead of defining LHS as an optimization problem, a much simpler approach is typically used in which we ensure one sample per interval, but we rely on randomness to choose point combinations.
- Although this does not necessarily yield a maximum spread, it works well in practice and is simple to implement.
- Before discussing the algorithm, we discuss how to generate other distributions besides just uniform distributions.
- We can convert from uniformly sampled points to an arbitrary distribution using a technique called inversion sampling.
- Assume that we want to generate samples x from an arbitrary probability density function (PDF) $p(x)$ or, equivalently, from the corresponding cumulative distribution function (CDF) $P(x)$.

4.1. Latin Hypercube sampling

- The probability integral transform states that for any continuous CDF, $y=P(x)$, the variable y is uniformly distributed (a simple proof, but it is not shown here to avoid introducing additional notation).
- The procedure is to randomly sample from a uniform distribution (e.g., generate y), then compute the corresponding x such that $P(x)=y$, which we denote as $x=P^{-1}y$.
- This latter step is known as an inverse CDF, a percent-point function, or a quantile function.
- This process is depicted in Fig. 6.07 for a normal distribution.
- This same procedure allows us to use LHS with any distribution, simply by generating the samples on a uniform distribution.

4.1. Latin Hypercube sampling

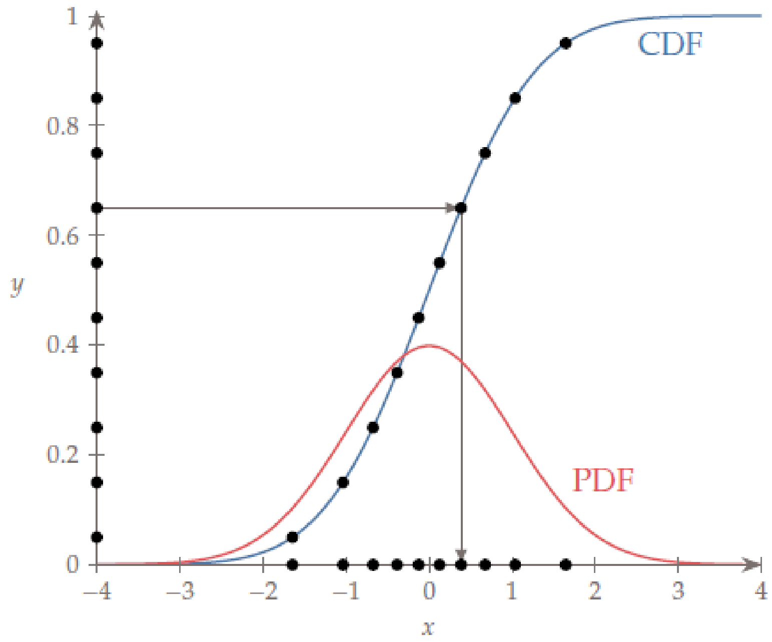


Figure 6.07 An example of inversion sampling with a normal distribution. A few uniform samples are shown on the *y*-axis. The points are evaluated by the inverse CDF, represented by the arrows passing through the CDF for a normal distribution. If enough samples are drawn, the resulting distribution will be the PDF of a normal distribution..

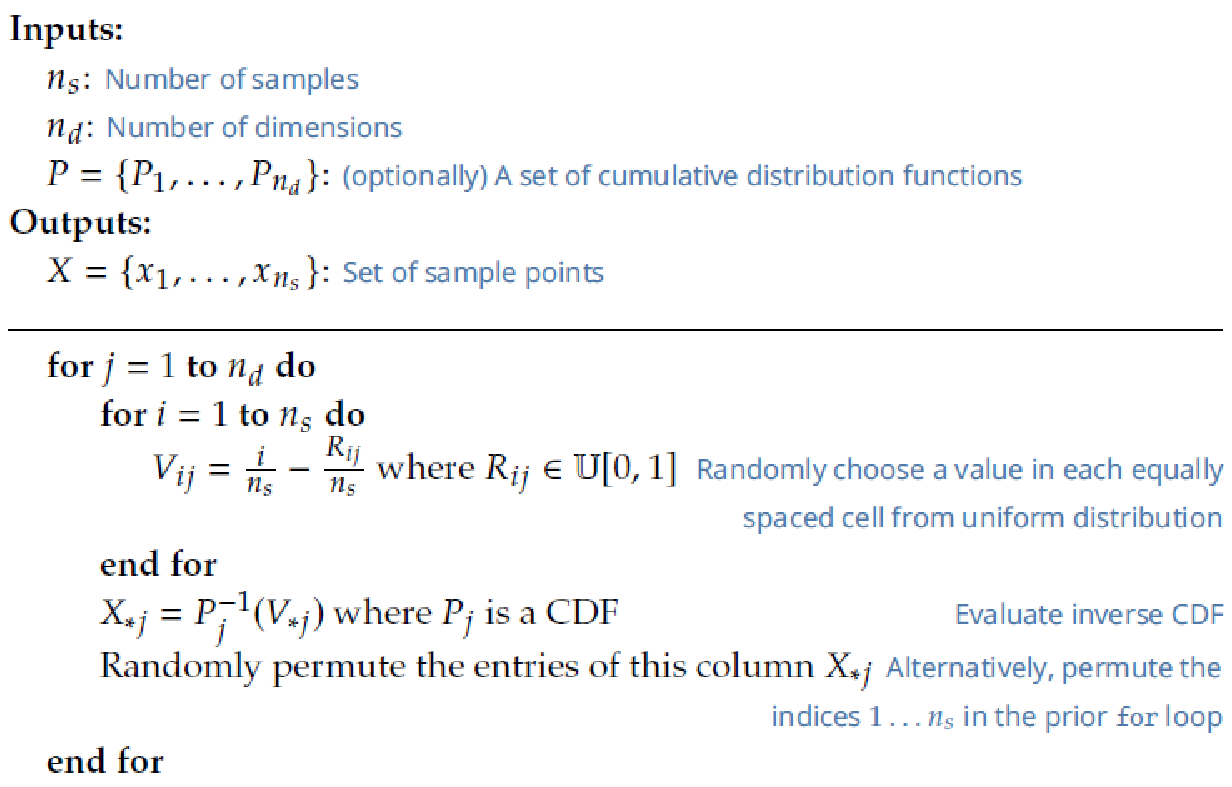
Source: Martins et Ning, 2021

4.1. Latin Hypercube sampling

- A typical algorithm is described in Alg. 6.1.
- For each axis, we partition the CDF in n_s evenly spaced regions (evenly spaced along the CDF, which means that each region is equiprobable).
- We generate a random number within each evenly spaced interval, where 0 corresponds to the bottom of the interval and 1 to the top.
- We then evaluate the inverse CDF as described previously so that the points match our specified distribution (the CDF for a uniform distribution is just a line $P(x)=x$, so the output is not changed).
- Next, the column of points for that axis is randomly permuted.
- This process is repeated for each axis according to its specified probability distribution.

4.1. Latin Hypercube sampling

Algorithm 6.1: Latin hypercube sampling



```
Inputs:
    $n_s$: Number of samples
    $n_d$: Number of dimensions
    $P = \{P_1, \ldots, P_{n_d}\}$: (optionally) A set of cumulative distribution functions
Outputs:
    $X = \{x_1, \ldots, x_{n_s}\}$: Set of sample points
________________________________________________________________
    for $j = 1$ to $n_d$ do
        for $i = 1$ to $n_s$ do
            $V_{ij} = \frac{i}{n_s} - \frac{R_{ij}}{n_s}$ where $R_{ij} \in \mathbb{U}[0,1]$    Randomly choose a value in each equally
                                                          spaced cell from uniform distribution
        end for
        $X_{*j} = P_j^{-1}(V_{*j})$ where $P_j$ is a CDF                Evaluate inverse CDF
        Randomly permute the entries of this column $X_{*j}$    Alternatively, permute the
                                                          indices $1 \ldots n_s$ in the prior for loop
    end for
```

4.1. Latin Hypercube sampling

- An example using this algorithm for eight points is shown in Fig. 6.08.
- In this example, we use a uniform distribution for x_1 and a normal distribution for x_2.
- There is one point in each equiprobable interval.
- As stated before, randomness does not necessarily ensure a good spread, but optimizing the spread is difficult because the function is highly multimodal.
- Instead, to encourage high spread, we could generate multiple Latin hypercube samples with this algorithm and select the one with the largest sum of the distance between points.

4.1. Latin Hypercube sampling

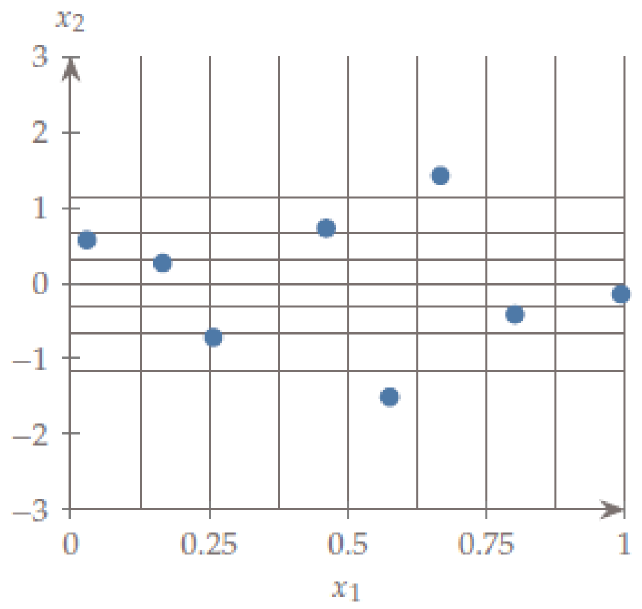


Figure 6.08 An example from the LHS algorithm showing uniform distribution in x_1 and a Gaussian distribution in x_2 with eight sample points. The equiprobable bins are shown as grid lines.

Source: Martins et Ning, 2021

4.2. Low-Discrepancy Sequences

- Low-discrepancy sequences generate deterministic sequences of points that are well spread.
- Each new point added in the sequence maintains low discrepancy - discrepancy refers to the variation in point density throughout the domain.
- Hence, a low-discrepancy set is close to even density (i.e., well spread).
- These sequences are called quasi-random because they often serve as suitable replacements for applications that use random sequences, but they are not random or even pseudorandom.
- An advantage of low-discrepancy sequences over LHS is that most of the approaches do not require selecting all the samples beforehand.

4.2. Low-Discrepancy Sequences

- These methods generate deterministic sequences; in other words, we generate the same sequence of points whether we choose them beforehand or add more later.
- This property is particularly advantageous in iterative procedures.
- We may choose an initial sampling plan and add more well-spread points to the sample later.
- This is not necessarily an advantage for the methods of this chapter because the optimization drives the selection of new points rather than continuing to seek spread out samples.
- However, this feature is useful for other applications, such as quadrature, Monte Carlo simulations, and other problems where an iterative sampling process is used to determine statistical convergence.

4.2. Low-Discrepancy Sequences

- Low-discrepancy sequences add more points that are well spread without having to throw out the existing samples.
- Even in non-iterative procedures, these sampling strategies can be a useful alternative.
- Several of these sequences are built on generalizations of the one-dimensional van der Corput sequence to more than one dimension.
- Such sequences are defined by representing an integer i in a given integer base b (the van der Corput sequence is always base 2):

$$i = a_0 + a_1 b + a_2 b^2 + \cdots + a_r b^r \quad where \quad a \in [0, b-1] \quad (6.01)$$

4.2. Low-Discrepancy Sequences

- If the base is 2, this is just a standard binary sequence.
- After determining the relevant coefficients (a_j), the ith element of the sequence is

$$\phi_b(i) = \frac{a_0}{b} + \frac{a_1}{b^2} + \frac{a_2}{b^3} + \cdots + \frac{a_r}{b^{r+1}} \qquad (6.02)$$

- An algorithm to generate an element in this sequence, also known as a radical inverse function for base b, is given in Alg. 6.2.
- For base 2, the sequence is as follows:

$$\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, \frac{1}{16}, \frac{9}{16}, \ldots \qquad (6.03)$$

4.2. Low-Discrepancy Sequences

Algorithm 6.2: Radical inverse function

Inputs:

- i: ith point in sequence
- b: Base (integer)

Outputs:

- ϕ: Generated point

```
$b_d = b$                                        Base used in denominator
$\phi = 0$
while $i > 0$ do
    $a = \text{mod}(i, b)$                       Coefficient
    $\phi = \phi + a/b_d$
    $b_d = b_d \cdot b$                          Increase exponent in denominator
    $i = \text{Int}(i/b)$                        Integer division
end while
```

4.2. Low-Discrepancy Sequences

- The interval is divided in half, and then each subinterval is also halved, with new points spreading out across the domain (see Fig. 6.09).

$$\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, \frac{1}{16}, \frac{9}{16}, \dots$$

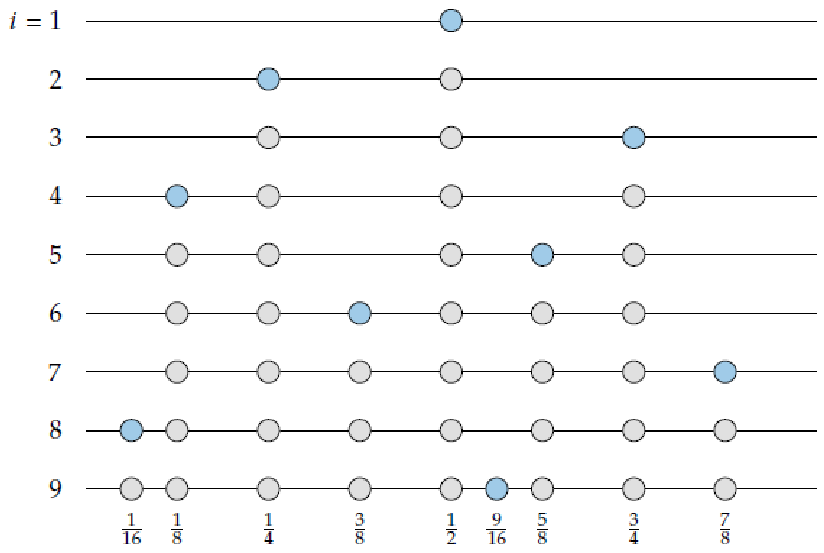


Figure 6.09 Van Der Corput sequence.

Source: Martins et Ning, 2021

4.2. Low-Discrepancy Sequences

- Similarly, for base 3, the interval is split into thirds, then each subinterval is split into thirds, and so on:

$$\frac{1}{3}, \frac{2}{3}, \frac{1}{9}, \frac{4}{9}, \frac{7}{9}, \frac{2}{9}, \frac{5}{9}, \frac{8}{9}, \frac{1}{27}, \dots \qquad (6.04)$$

4.2.1. Halton sequence

- A Halton sequence uses pairwise prime numbers (larger than 1) for the base of each dimension of the problem.
- The *i*th point in the Halton sequence is

$$\phi(i, b_1), \phi(i, b_2), \dots, \phi(i, b_{n_x}) \tag{6.05}$$

where the b_j set is pairwise prime.

- As an example in two dimensions, Fig. 6.10 shows 30 generated points of the Halton sequence where x_1 uses base 2, and x_2 uses base 3, and then a subsequent 20 generated points are added (in another colour), showing the reuse of existing points.
- If the dimensionality of the problem is high, then some of the base combinations lead to points that are highly correlated and thus undesirable for a sampling plan.

4.2.1. Halton sequence

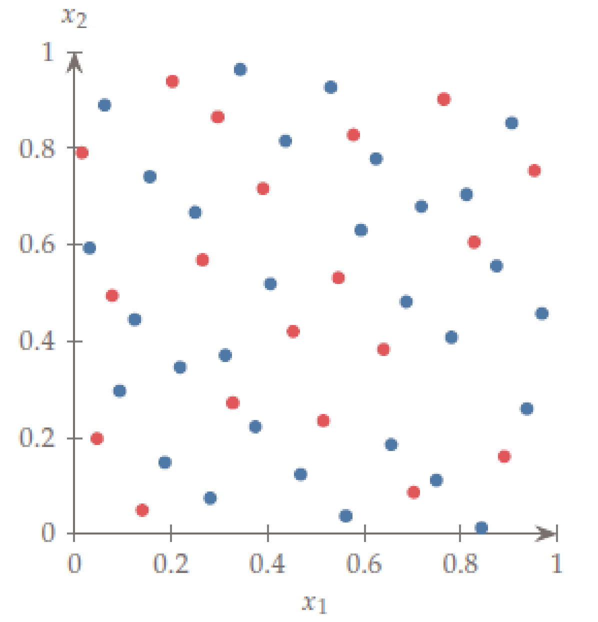


Figure 6.10 Halton sequence with base 2 for x_1 and base 3 for x_2. First, 30 points are selected (in blue), and then 20 points are added (in red). These points would be identical to 50 points chosen at once.

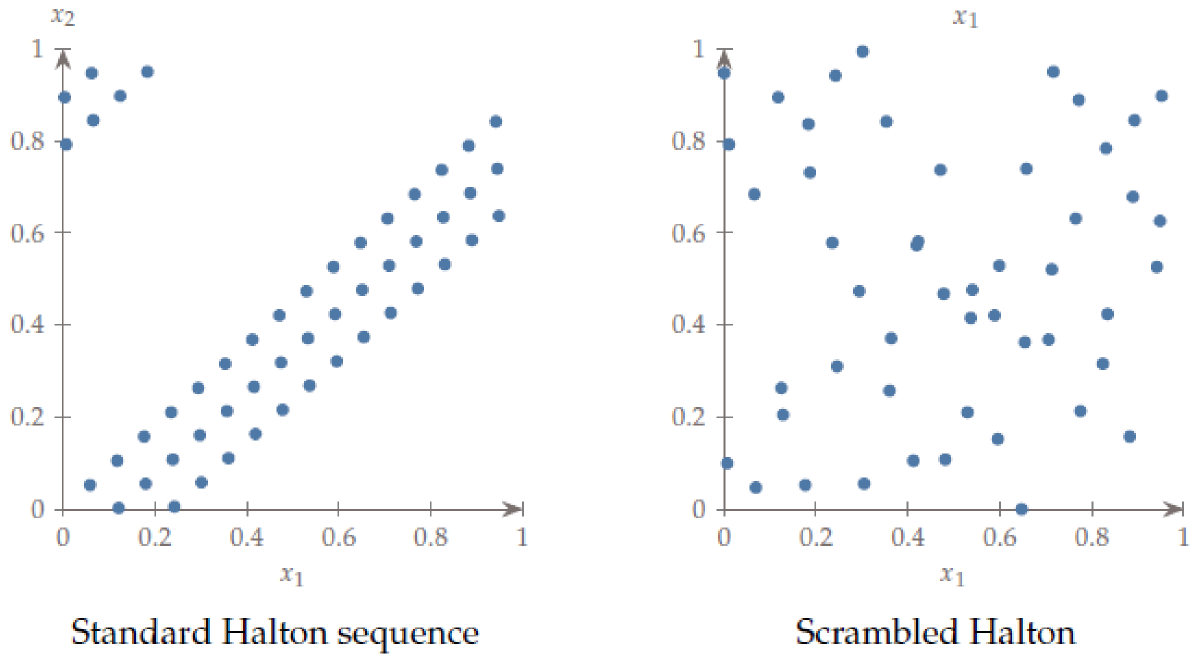


Figure 6.11 Halton sequence with base 17 for x_1 and base 19 for x_2.

Source: Martins et Ning, 2021

4.2.1. Halton sequence

- For example, the left of Fig. 6.11 shows 50 generated points where x_1 uses base 17, and x_2 uses base 19.
- To avoid this issue, we can use a scrambled Halton sequence.
- Scrambling can be accomplished by generating a permutation array containing a random permutation of the integers $p=[0,1,\ldots,b-1]$.
- Then, rather than using the integers a directly in Eq. 6.02, we use the entries of a as the indices of the permutation array.
- If p is the permutation array, we have:

$$\phi_b(i) = \frac{p_{a_0}}{b} + \frac{p_{a_1}}{b^2} + \frac{p_{a_2}}{b^3} + \cdots + \frac{p_{a_r}}{b^{r+1}} \qquad (6.06)$$

4.2.1. Halton sequence

- The permutation array is fixed for all digits 0 and for all n_p points in the domain.
- The right side of Fig. 6.11 shows the same example (with base 17 and base 19) but with scrambling to weaken the strong correlations.

4.2.2. Hammersley sequence

- The Hammersley sequence is closely related to the Halton sequence.
- However, it provides better spacing if we know beforehand the number of points (n_p) that we are going to use.
- This approach only needs n_x-1 bases (still pairwise prime) because the first dimension uses regular spacing:

$$\frac{i}{n_p}, \phi(i, b_1), \phi(i, b_2), \dots, \phi\left(i, b_{n_{x-1}}\right) \qquad (6.07)$$

- Because this sequence needs to know the number of points (n_p) beforehand, it is less useful for iterative procedures.
- However, the implementation is straightforward, and so it may still be a useful alternative to LHS.
- Figure 6.12 shows 50 points generated from a Hammersley sequence where the x_2-axis uses base 2.

4.2.2. Hammersley sequence

- Figure 6.12 shows 50 points generated from a Hammersley sequence where the x_2-axis uses base 2.

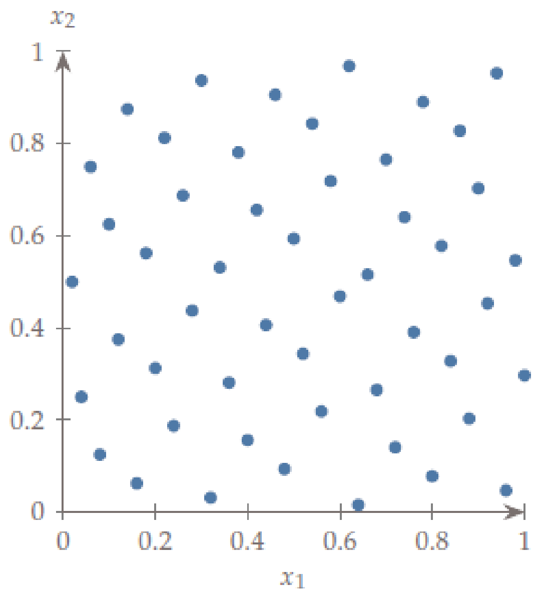


Figure 6.12 Hammersley sequence with base 2 for the x_2-axis.

Source: Martins et Ning, 2021

4.2.3. Other sequences

- A wide variety of other low-discrepancy sequences exist.
- The Faure sequence is similar to the Halton, but it uses the same base for all dimensions and uses permutation scrambling for each dimension instead.
- Sobol sequences use base 2 sequences but with a reordering based on "direction numbers".
- Niederreiter sequences are effectively a generalization of Sobol sequences to other bases.

5. Constructing a Surrogate

- Once sampling is completed, we have a list of data points, often called training data:

$$\left(x^{(i)}, f^{(i)}\right) \tag{6.08}$$

where $x^{(i)}$ is an input vector from the sampling plan, and $f^{(i)}$ contains the corresponding outputs from evaluating the model: $f^{(i)} = f\left(x^{(i)}\right)$.

- We seek to construct a surrogate model from this data set
- Surrogate models can be based on physics, mathematics, or a combination of the two.
- Incorporating known physics into a model is often desirable to improve model accuracy.

5. Constructing a Surrogate

- However, functional relationships are unknown for many complex problems, and a data-driven mathematical model can be more effective.
- Surrogate-based models can be based on interpolation or regression, as illustrated in Fig. 6.13.
- Interpolation builds a function that exactly matches the provided training data.

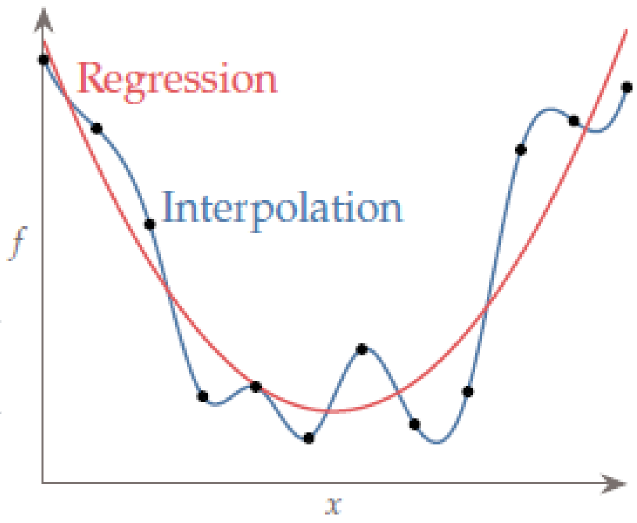


Figure 6.13 Interpolation models match the training data at the provided points, whereas regression models minimize the error between the training data and a function with an assumed trend.

Source: Martins et Ning, 2021

5. Constructing a Surrogate

- Regression models do not try to match the training data points exactly; instead, they minimize the error between a smooth trend function and the training data.
- The nature of the training data can help decide between these two types of surrogate models.
- Regression is particularly useful when the data are noisy.
- Interpolatory models may produce undesirable oscillations when fitting the noise.
- In contrast, regression models can find a smooth function that is less sensitive to the noise.
- Interpolation is useful when the data are highly multimodal (and not noisy).
- This is because a regression model may smooth over variations that are actually physical, whereas an interpolatory model can accurately capture those variations.

5. Constructing a Surrogate

- There are two main steps involved in either type of surrogate model.
  - First, we select a set of basis functions, which represent the form for the model.
  - Second, we determine the model parameters that provide the best fit to the provided data.
- Determining the model parameters is an optimization problem, which we discuss first.
- We discuss linear regression and nonlinear regression, which are techniques for choosing model parameters for a given set of basis functions.
- Next, we discuss cross validation, which is a critical technique for selecting an appropriate model form.
- Finally, we discuss common basis functions.

5.1. Linear Least Squares Regression

- A linear regression model does not mean that the surrogate is linear in the input variables but rather that the model is linear in its coefficients (i.e., linear in the parameters we are estimating).
- For example, the following equation is a two-dimensional linear regression model, where we use $\hat{f}$ to represent our estimated model of the function f:

$$\hat{f}(x) = w_1 x_1^2 + w_2 x_1 x_2 + w_3 \exp(x_2) + w_4 x_1 + w_5 \qquad (6.09)$$

- This function is highly nonlinear, but it is classified as a linear regression model because the regression seeks to choose the appropriate values for the coefficients w_i (and the function is linear in w).
- A general linear regression model can be expressed as

5.1. Linear Least Squares Regression

- A general linear regression model can be expressed as

$$\hat{f} = w^T \psi(x) = \sum_i w_i \psi_i(x) \qquad (6.10)$$

where w is a vector of weights, and ψ is a vector of basis functions.

- In this section, we assume that the basis functions are provided.
- In general, the basis functions can be any set of functions that we choose (and typically they are nonlinear).
- It is usually desirable for these functions to be orthogonal.
- The coefficients are chosen to minimize the error between our predicted function values $\hat{f}$ and the actual function values $f^{(i)}$.

5.1. Linear Least Squares Regression

- Because we want to minimize both positive and negative errors, we minimize the sum of the square of the errors (or a weighted sum of squared errors):

$$\underset{w}{\text{minimize}} \sum_i \left[\hat{f}\left(w; x^{(i)}\right) - f^{(i)}\right]^2 \tag{6.11}$$

- The solution to this optimization problem is called a least squares solution.
- If the regression model is linear, we can simplify this objective and solve the problem analytically.
- Recall that $\hat{f} = \psi^T w$, so the objective can be written as

$$\underset{w}{\text{minimize}} \sum_i \left[\psi(x^{(i)})^T w - f^{(i)}\right]^2 \tag{6.12}$$

5.1. Linear Least Squares Regression

- We can express this in matrix form by defining the following:

$$\Psi = \begin{bmatrix} - & \psi(x^{(1)})^T & - \\ - & \psi(x^{(2)})^T & - \\ - & \vdots & - \\ - & \psi(x^{(n_s)})^T & - \end{bmatrix} \tag{6.13}$$

- Matrix Ψ is of size $(n_s \times n_w)$, where n_s is the number of samples, n_w the number of parameters in w, and $n_s \geq n_w$.
- This means that there should be more equations than unknowns or that we have sampled more points than the number of coefficients we need to estimate.
- This should make sense because our surrogate function is only an assumed form and generally not an exact fit to the actual underlying function.
- Thus, we need more data to create a good fit.

5.1. Linear Least Squares Regression

- Then the optimization problem can be written in matrix form as:

$$\underset{w}{\text{minimize}} \quad \|\Psi w - f\|_2^2 \tag{6.14}$$

- Expanding the squared norm (i.e., $\|x\|_2^2 = x^T x$) gives

$$\underset{w}{\text{minimize}} \quad w^T \Psi^T \Psi w - 2 f^T \Psi w + f^T f \tag{6.15}$$

- We can omit the last term from the objective because our optimization variables are w, and the last term has no w dependence:

$$\underset{w}{\text{minimize}} \quad w^T \Psi^T \Psi w - 2 f^T \Psi w \tag{6.16}$$

5.1. Linear Least Squares Regression

- This fits the general form for an unconstrained quadratic programming (QP) problem, as shown in Chapter 3:

$$\underset{w}{\text{minimize}} \quad \frac{1}{2} x^T Q x - q^T x \tag{6.17}$$

onde

$$Q = 2\Psi^T \Psi \tag{6.18}$$

$$q = -2\Psi^T f \tag{6.19}$$

- Recall that an equality constrained QP (of which unconstrained is a subset) has an analytic solution as long as the QP is positive definite.
- In our case, we can show that & is positive definite as long as is full rank:

5.1. Linear Least Squares Regression

- In our case, we can show that Q is positive definite as long as Ψ is full rank:

$$x^T Q x = 2 w^T \Psi^T \Psi w = 2\|\Psi x\|_2^2 > 0 \tag{6.20}$$

- This is not surprising because the objective is a sum of squared values.
- Removing the portions associated with the constraints, the solution is

$$Qx = -q \tag{6.21}$$

- In our case, this becomes

$$2\Psi^T \Psi w = 2\Psi^T f \tag{6.22}$$

5.1. Linear Least Squares Regression

- After simplifying, we have an analytic solution for the weights:

$$w = (\Psi^T\Psi)^{-1}\Psi^T f \qquad (6.23)$$

- We sometimes express the linear relationship in Eq. 6.12 as $\Psi w = f$, although the case where there are more equations than unknowns does not typically have a solution (the problem is overdetermined).
- Instead, we seek the solution that minimizes the error $\|\Psi x - f\|^2$, that is, Eq. 6.23.
- The quantity $\Psi^+ = (\Psi^T\Psi)^{-1}\Psi^T$ is called the pseudoinverse of Ψ (or more specifically, the Moore–Penrose pseudoinverse), and thus we can write Eq. 6.23 in the more compact form

$$w = \Psi^+ f \qquad (6.24)$$

5.1. Linear Least Squares Regression

- This allows for a similar form to solving a linear system of equations where an inverse would be used instead.
- In solving both a linear system and the linear least-squares equation (Eq. 6.23), we do not explicitly invert a matrix.
- For linear least squares, a QR factorization is commonly used for improved numerical conditioning as compared to solving Eq. 6.23 directly.

5.1. Linear Least Squares Regression

Example 6.1: Linear regression.

- Consider the quadratic fit $\hat{f}(x) = ax^2 + bx + c$ where the coefficients we wish to estimate are $w = [a, b, c]$ and the basis functions are $\psi = [x^2, x, 1]$.
- We are provided n_s the data points, x and f, shown as circles in Fig. 6.14. From these data, we construct the matrix Ψ for our basis functions as follows

$$\Psi = \begin{bmatrix} {x^{(1)}}^2 & x^{(1)} & 1 \\ {x^{(2)}}^2 & x^{(2)} & 1 \\ & \vdots & \\ {x^{(n_s)}}^2 & x^{(n_s)} & 1 \end{bmatrix}$$

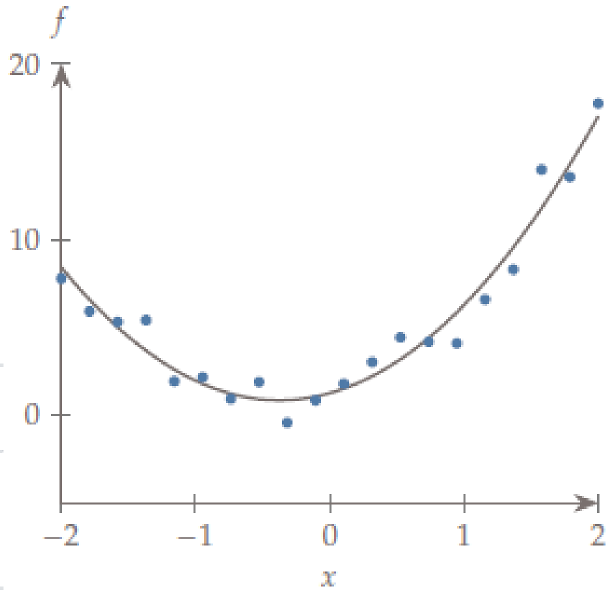


Figure 6.14 Linear least squares example with a quadratic fit on a one-dimensional function.

Source: Martins et Ning, 2021

5.1. Linear Least Squares Regression

Example 6.1: Linear regression (continued).

- We can then solve for the coefficients w using the linear least squares solution (Eq. 6.23).
- Substituting the coefficients and respective basis functions into Eq. 6.10, we obtain the surrogate model

$$\hat{f}(x) = w_1 x^2 + w_2 x + w_3$$

which is also plotted in Fig. 6.14 as a solid line.

5.2. Other regressions

- There are other possible regressions that one can use.
- For example:
  - Maximum Likelihood Interpretation
  - Nonlinear Least Squares Regression

5.3. Cross validation

- The other important consideration for developing a surrogate model is the choice of the basis functions in ψ.
- In some instances, we may know something about the model behaviour and thus what type of basis functions should be used, but generally, the best way to determine the basis functions is through cross validation.
- Cross validation is also helpful in characterizing error, even if we already have a chosen set of basis functions.
- One of the reasons we use cross validation is to prevent overfitting.
- Overfitting occurs when we have too many degrees of freedom and closely fit a given set of data, but the resulting model has a poor predictive ability.

5.3. Cross validation

- In other words, we are fitting noise.
- The following example illustrates this idea with a one-dimensional function.

5.3. Cross validation

Example 6.2: The dangers of overfitting.

- Consider the set of training data (Fig. 6.15, left), which we use to create a surrogate function.
- This is a one-dimensional problem so that it can be easily visualized.

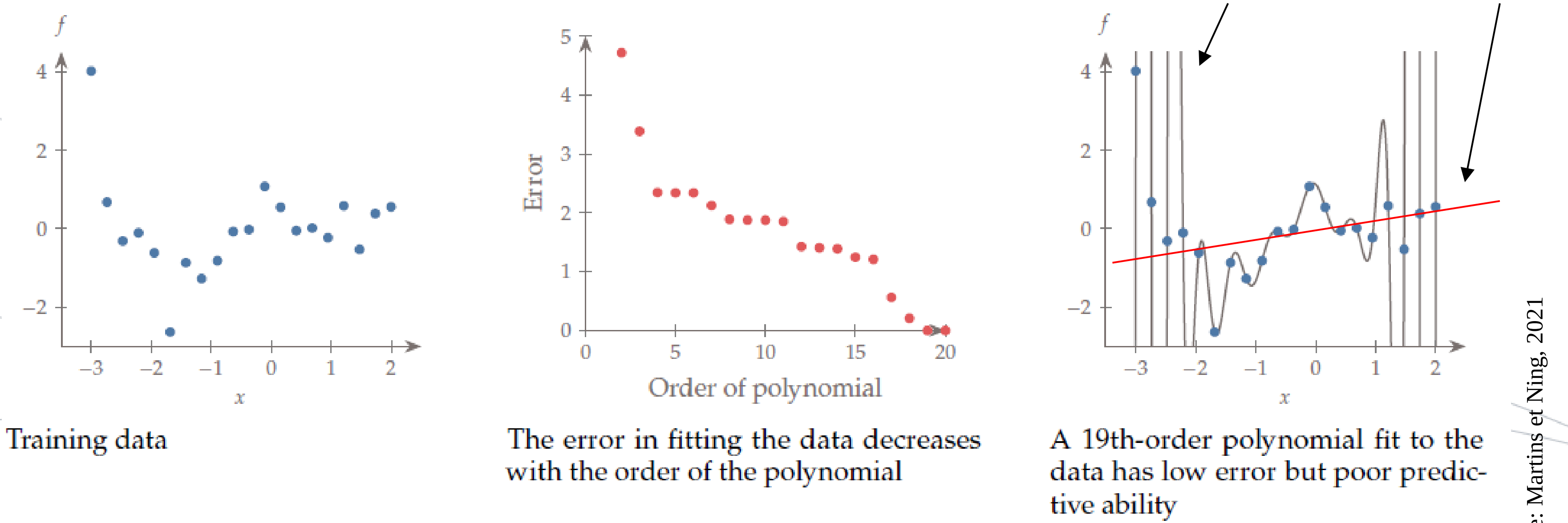


Figure 6.15 Fitting different order polynomials to data.

Source: Martins et Ning, 2021

5.3. Cross validation

Example 6.2: The dangers of overfitting (continued).

- In general, however, visualization is limited, and determining the right basis functions to use can be difficult.
- If we use a polynomial basis, we might attempt to determine the appropriate order by trying each case (e.g., quadratic, cubic, quartic) and measuring the error in our fit (Fig. 6.15, center).
- It seems as if the higher the order of the polynomial, the lower the error.
- For example, a 20th-order polynomial reduces the error to almost zero.
- The problem is that although the error is low on this set of data, the predictive capability of such a model for other data points is poor.

5.3. Cross validation

Example 6.2: The dangers of overfitting (continued).

- For example, the right side of Fig. 6.15 shows a 19th-order polynomial fit to the data.
- The model passes right through the points, but it does not work well for many of the points that are not part of the training set (which is the whole purpose of the surrogate).
- The opposite of overfitting is underfitting, which is also a potential issue.
- When underfitting, we do not have enough degrees of freedom to create a useful model (e.g., imagine using a linear fit for the previous example).

5.3. Cross validation

- The solution to the overfitting problem highlighted in Ex. 6.2 is cross validation.
- Cross validation means that we use one set of data for training (creating the model) and a different set of data for assessing its predictive error.
- There are many different ways to perform cross validation; we describe two.
- Simple cross validation is illustrated in Fig. 6.16.

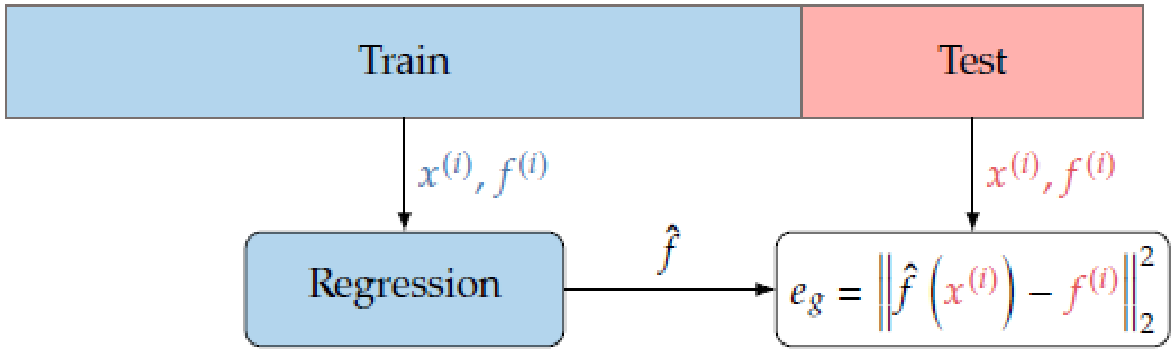

Train
Test
$x^{(i)}, f^{(i)}$
$x^{(i)}, f^{(i)}$
Regression
$\hat{f}$
$e_g = \left\| \hat{f}\left(x^{(i)}\right) - f^{(i)} \right\|_2^2$

Figure 6.16 Simple cross-validation process.

Source: Martins et Ning, 2021

5.3. Cross validation

- It consists of the following steps:
  1. Randomly split your data into a training set and a validation set (e.g., a 70–30 split).
  2. Train each candidate model (the different options for ψ) using only the training set but evaluate the error with the validation set. The error on previously unseen data is called the generalization error (e_g in Fig. 6.16).
  3. Choose the model with the lowest generalization error and optionally retrain that model using all of the data.
- An alternative option that is more involved but makes better use of the data is called k-fold cross validation.
- It is particularly advantageous when we have a small data set where we cannot afford to leave much out.

5.3. Cross validation

- This procedure is illustrated in Fig. 6.17 and consists of the following steps:
  1. Randomly split your data into *n* sets (e.g., *n*=10).
  2. Train each candidate model using the data from all sets except one (e.g., 9 of the 10 sets) and use the remaining set for validation. Repeat for all *n* possible validation sets and average the performance.
  3. Choose the model with the lowest average generalization error. Optionally, retrain with all the data.
- The extreme version of this process, when training data are very limited, is leave-one-out cross validation (i.e., each testing subset consists of one data point).

5.3. Cross validation

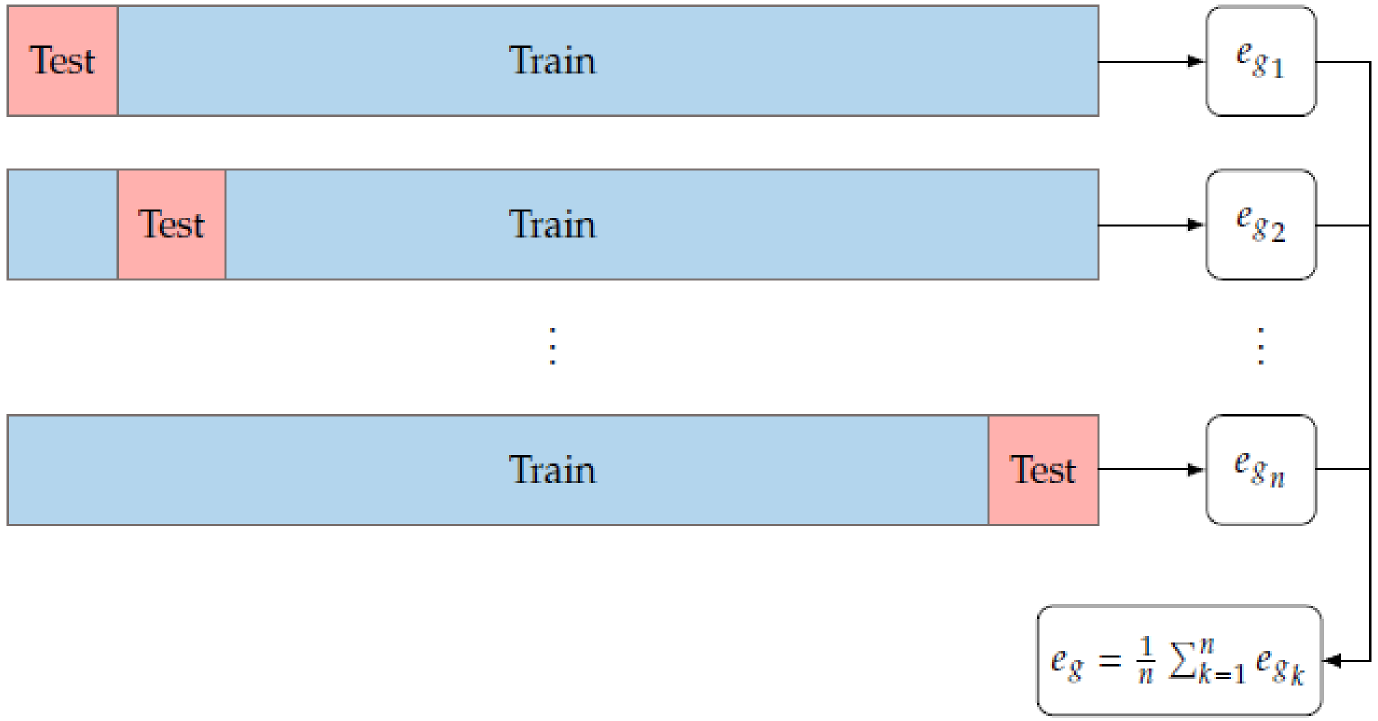


Figure 6.17 Diagram of k-fold cross validation process.

Source: Martins et Ning, 2021

5.3. Cross validation

Example 6.3: Cross validation helps to avoid overfitting.

- This example continues from Ex. 6.2.
- First, we perform k-fold cross validation using 10 divisions.
- The average error across the divisions using the training data is shown in Fig. 6.18 (with a smaller *y*-axis scale on the right).
- The error increases dramatically as the polynomial order increases.

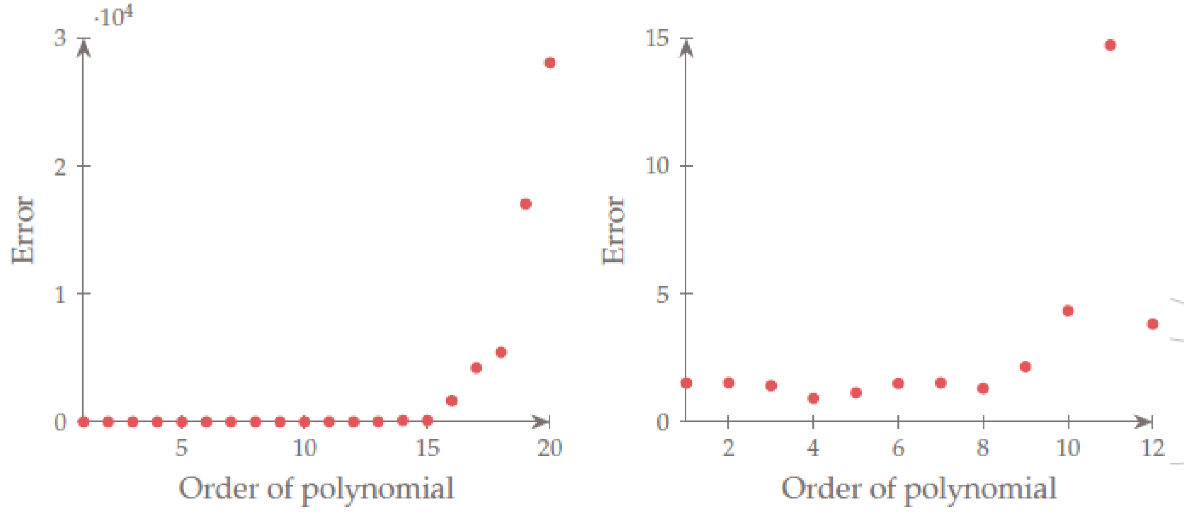


Figure 6.18 Error from k-fold cross validation.

Source: Martins et Ning, 2021

5.3. Cross validation

Example 6.3: Cross validation helps to avoid overfitting (continued).

- Zooming in on the flat region, we see a range of options with similar errors.
- Among the similar solutions, we generally prefer the simplest model.
- In this case, a fourth-order polynomial seems reasonable.
- A fourth-order polynomial is compared against the data in Fig. 6.19.
- This model has a much better predictive ability.

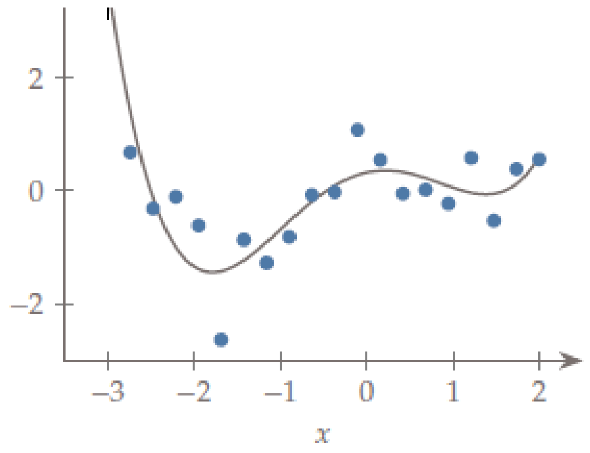


Figure 6.19 A fourth-order polynomial fit to the data.

Source: Martins et Ning, 2021

5.4. Common basis functions

- Although cross validation can help us find the lowest generalization error among a provided set of basis functions, we still need to determine what sets of options to consider.
- This selection is crucial because our model is only as good as the available options, but increasing the number of options increases computational time.
- The possibilities for basis functions are as numerous as the types of function.
- As stated before, it is generally desirable that they form an orthogonal set.
- We focus on a few commonly used functions.

5.4.1. Polynomials

- Polynomials, of which we have already seen a few examples, are useful in many applications.
- However, we typically use low-order polynomials for regression because high-order polynomials rarely generalize well.
- Polynomials can be particularly effective in cases where a knowledge of the physics suggests them to be an appropriate choice (e.g., drag varies quadratically with speed).
- Because a lot of structure is already built into the model form, fewer data points are needed to create a reasonable model (e.g., a quadratic function in n dimensions needs at least $n(n+1)/2+n+1$ points, so this amounts to 6 points in two dimensions, 10 points in three dimensions, and so on).

5.4.2. Radial basis functions

- Another common type of basis function is a radial basis function.
- Radial basis functions are functions that depend on the distance from some center point and can be written as follows:

$$\psi^{(i)} = \psi\left(\left\|x - c^{(i)}\right\|\right) = \psi\left(\left\|r^{(i)}\right\|\right) \qquad (6.25)$$

where *c* is the center point, and *r* is the radius about the center point.

- Although the center points can be placed anywhere, we usually choose the sampling data as centering points

$$\psi^{(i)} = \psi\left(\left\|x - x^{(i)}\right\|\right) \qquad (6.26)$$

5.4.2. Radial basis functions

- This is often a useful choice because it captures the idea that our ability to predict function behavior is related to how close we are to known function values (in other words, nearby points are more highly correlated).
- This form naturally lends itself to interpolation, although regularization can be added to allow for regression.
- Polynomials are often combined with radial basis functions because the polynomial can capture global function behavior, while the radial basis functions can introduce modifications to capture local behavior.
- One popular radial basis function is the Gaussian basis:

$$\psi^{(i)}(x) = exp\left(-\sum_j \theta_j \left|x - x_j^{(i)}\right|^2\right) \qquad (6.27)$$

5.4.2. Radial basis functions

where θ_j are the model parameters.

- One of the forms of kriging discussed in the following section can be viewed as a radial basis function model with a Gaussian basis.
- The surrogate modelling toolbox (SMT) (https://smt.readthedocs.io/) is a useful package for surrogate modelling, with a particular focus on providing derivatives for use in gradient-based optimization.
- SMT includes surrogate modelling techniques that utilize gradients as training data to enhance accuracy and scalability with the number of inputs.

5.4.2. Radial basis functions

Example 6.4: Aerodynamic coefficients of airfoils.

- Create a surrogate model for C_l, C_d and C_m as functions of Reynolds number, angle-of-attack, flap chord ratio, and flap deflection, using multiquadric representation of numerical computed data.
- See paper: “Use of Multiquadric Functions for Multivariable Representation of the Aerodynamic Coefficients of Airfoils” (https://doi.org/10.1155/2021/6615601)

6. Kriging

- Kriging is a popular surrogate modelling technique that can build approximations of highly nonlinear engineering simulations.
- We may not have a simple parametric form for such simulations that we can use with regression and expect a good fit.
- Instead of tuning the parameters of a functional form that describes what the function is, kriging tunes the parameters of a statistical model that describes how the function behaves.
- The kriging statistical model that approximates f consists of two terms: a function $\mu(x)$ that is meant to capture some of the function behaviour and a random variable $Z(x)$.
- Thus, we can write the kriging model as

6. Kriging

$$F(x) = \mu(x) + Z(x) \;\; where \;\; Z(x) \sim \aleph(0, \sigma^2) \qquad (6.28)$$

- When we evaluate the function, we want to approximate at point x, we get a scalar value $f(x)$.
- In contrast, when we evaluate the stochastic process (Eq. 6.28) at x, we get a random variable $F(x)$ that has a normal distribution with mean μ and variance σ^2.
- Although we wrote μ as a function of x, most kriging models consider this to be constant because the random variable term alone is effective in capturing the function behavior.
- For the rest of this section, we discuss the case with constant μ, which is called ordinary kriging.

6. Kriging

- Kriging is also referred to as Gaussian process interpolation, or more generally in the regression case discussed later in this section, as Gaussian process regression.
- The power of the statistical model lies in how it treats the correlation between the random variables.
- Although we do not know the exact form of the error term $Z(x)$, we can still make some reasonable assumptions about it.
- Consider two points in a sampling plan, $x^{(i)}$ and $x^{(j)}$, and the corresponding terms, $Z(x^{(i)})$ and $Z(x^{(j)})$.
- Intuitively, we expect $Z(x^{(i)})$ to be close to $Z(x^{(j)})$ whenever $x^{(i)}$ is close to $x^{(j)}$.
- Therefore, it seems reasonable to assume that the correlation between $Z(x^{(i)})$ and $Z(x^{(j)})$ is a function of the distance between the two points.

6. Kriging

- In kriging, we assume that this correlation is given by a kernel function $K(x^{(i)},x^{(j)})$:

$$K\left(x^{(i)},x^{(j)}\right)=corr\left(Z\left(x^{(i)}\right),Z\left(x^{(j)}\right)\right) \tag{6.29}$$

- As a matrix, the kernel is represented as $K_{ij}=K(x^{(i)},x^{(j)})$.
- Various kernel functions are used with kriging.
- The most commonly used kernel function is

$$K\left(x^{(i)},x^{(j)}\right)=exp\left(-\sum_{l=1}^{n_d}\theta_l\left|x_l^{(i)}-x_l^{(j)}\right|^{p_l}\right) \tag{6.30}$$

- where $\theta_l \geq 0$, $0 \leq p_l \leq 2$, and n_d is the number of dimensions (i.e., the length of the vector x).
- If every $p_l = 2$, this becomes a Gaussian kernel.

6. Kriging

- Let us examine how the statistical model F defined in Eq. 6.28 captures the typical behaviour of the function f.
- The parameter μ captures the typical value, and σ^2 captures the expected variance.
- The kernel (or correlation) function (Eq. 6.30) implicitly models continuous functions.
- If f is continuous, we know that, as $\left|x_l^{(i)} - x_l^{(j)}\right| \to 0$, then $\left|f\left(x_l^{(i)}\right) - f\left(x_l^{(j)}\right)\right| \to 0$.
- This is captured in the kernel function because as $\left|x_l^{(i)} - x_l^{(j)}\right| \to 0$, the correlation approaches 1.
- The parameter θ_l captures how active the function f is in the lth coordinate direction.

6. Kriging

- A unit difference in variable l ($\left|x_l^{(i)} - x_l^{(j)}\right| = 1$) has a more significant impact on the correlation when θ_l is large.
- The exponent p_l describes the smoothness of the function in the lth coordinate direction.
- Values of p_l close to 2 produce smooth functions, whereas values closer to zero produce functions with more variation.
- Kriging surrogate modelling involves two main steps.
  - The first step consists of using the data to estimate the statistical model parameters μ, σ^2, $\theta_1, \ldots, \theta_{n_d}$, and $p_1, \ldots, p_{n_d}$.
  - The second step consists of making predictions using the statistical model and these estimated parameter values.

6. Kriging

- The parameter estimation uses the maximum likelihood approach which is complicated.
- Let us denote the random variable as $F^{(i)} \equiv F\left(x^{(i)}\right)$ and the vector of random variables as $F = \left[F^{(1)}, \dots, F^{(n_s)}\right]$, where n_s is the number of samples.
- Similarly, $f^{(i)} \equiv f\left(x^{(i)}\right)$ and the vector of observed function values is $f = \left[f^{(1)}, \dots, f^{(n_s)}\right]$.
- Using this notation, we can say that the vector F is jointly normally distributed.
- This is also known as a multivariate Gaussian distribution.

6. Kriging

- The probability density function (PDF) (the likelihood that $F=f$) is

$$p(f) = \frac{1}{(2\pi)^{n_s/2}|\Sigma|^{1/2}} exp\left[-\frac{1}{2}(f - e\mu)^T\Sigma^{-1}(f - e\mu)\right] \quad (6.31)$$

where e is a vector of 1s with size n_s, and $|\Sigma|$ is the determinant of the covariance

$$\Sigma_{ij} = \sigma^2 K\left(x^{(i)}, x^{(j)}\right) \quad (6.32)$$

- The covariance between two elements $F^{(i)}$ and $F^{(j)}$ of F is related to correlation by the following definition:

$$\Sigma_{ij} = cov\left(F^{(i)}, F^{(j)}\right) = \sigma^2 corr\left(F^{(i)}, F^{(j)}\right) = \sigma^2 K\left(x^{(i)}, x^{(j)}\right) \quad (6.33)$$

- We assume stationarity of the second moment, that is, the variance σ^2 is constant in the domain.

6. Kriging

- The statistical model parameters $\theta_1, \ldots, \theta_{n_d}$, and $p_1, \ldots, p_{n_d}$ enter the likelihood (Eq. 6.31) via their effect on the kernel K (Eq. 6.30) and hence on the covariance matrix Σ (Eq. 6.32).
- We estimate the parameters using the maximum log likelihood approach; that is, we maximize the probability of observing our data f conditioned on the parameters μ and Σ.
- Using a PDF (Eq. 6.31) where μ is constant and the covariance is $\Sigma = \sigma^2 K$, yields the following likelihood function:

$$L(\mu, \sigma, \theta, p) = \frac{1}{(2\pi)^{n_s/2}\sigma^{n_s}|K|^{1/2}} exp\left[-\frac{(f - e\mu)^T K^{-1}(f - e\mu)}{2\sigma^2}\right] \tag{6.34}$$

- We now need to find the parameters μ, σ, θ_i, and p_i that maximize this likelihood function, that is, maximize the probability of our observations f.

6. Kriging

- We take the logarithm to form the log likelihood function:

$$\ell(\mu,\sigma,\theta,p) = -\frac{n_s}{2}ln(2\pi) - \frac{n_s}{2}ln(\sigma^2) - \frac{1}{2}ln|K| - \frac{(f-e\mu)^T K^{-1}(f-e\mu)}{2\sigma^2} \tag{6.35}$$

- We can maximize part of this term analytically by taking derivatives with respect to μ and σ, setting them equal to zero, and solving for their optimal values to obtain:

$$\mu^* = \frac{e^T K^{-1} f}{e^T K^{-1} e} \tag{6.36}$$

$$\sigma^{*2} = \frac{(f-e\mu^*)^T K^{-1}(f-e\mu^*)}{n_s} \tag{6.37}$$

6. Kriging

- We now substitute these values back into the log likelihood function (Eq. 6.35), which yields

$$\ell(\theta, p) = -\frac{n_s}{2} ln(\sigma^{*2}) - \frac{1}{2} ln|K| \tag{6.38}$$

- This function, also called the concentrated likelihood function, only depends on the kernel , which depends on θ and p.
- We cannot solve for optimal values of θ and p analytically.
- Instead, we rely on numerical optimization to maximize Eq. 6.38.
- Because θ can vary across a broad range, it is often better to search using logarithmic scaling.
- Once we solve that optimization problem, we compute the mean and variance in Eqs. 6.36 and 6.37.

6. Kriging

- Now that we have a fitted model, we can make predictions at new points where we have not sampled.
- We do this by substituting x_p into a formula called the kriging predictor.
- The formula is unique, but there are many ways to derive it.
- One way to derive it is to find the function value at x_p that is the most consistent with the behaviour of the function captured by the fitted kriging model.
- Let f_p be our guess for the value of the function at x_p.
- One way to assess the consistency of our guess is to add (x_p, f_p) as an artificial point to our training data (so that we now have n_s+1 points) and estimate the likelihood using the parameters from our fitted kriging model.

6. Kriging

- The likelihood of this augmented data can now be thought of as a function of f_p : high values correspond to guessed values of f_p that are consistent with function behavior captured by the fitted kriging model.
- Therefore, the value of f_p that maximizes the likelihood of this augmented data set is a natural way to predict the value of the function.
- This is an optimization problem with a closed-form solution, and the corresponding formula is the kriging predictor.
- Now we outline the derivation of the kriging predictor.
- With the augmented point, our function values are $\bar{f} = [f, f_p]$, where f is the n_s-vector of function values from the original training data.

6. Kriging

- Then, the correlation matrix with the additional data point is

$$\bar{K} = \begin{bmatrix} K & k \\ k^T & 1 \end{bmatrix} \tag{6.39}$$

where k is the correlation of the new point with the training data given by

$$k = \begin{bmatrix} corr\left(F\left(x^{(1)}\right), F\left(x_p\right)\right) = K\left(x^{(1)}, x_p\right) \\ \vdots \\ corr\left(F\left(x^{(n_s)}\right), F\left(x_p\right)\right) = K\left(x^{(n_s)}, x_p\right) \end{bmatrix} \tag{6.40}$$

- The 1 in the bottom right of the augmented correlation matrix (Eq. 6.39) is because the correlation of the new variable $F\left(x_p\right)$ with itself is 1.

6. Kriging

- The log likelihood function with these new augmented vectors and the previously determined parameters is as follows (see Eq. 6.35):

$$\ell(f_p) = -\frac{n_s}{2} ln(2\pi) - \frac{n_s}{2} ln(\sigma^{*2}) - \frac{1}{2} ln|\bar{K}| - \frac{\left(\bar{f} - e\mu^*\right)^T \bar{K}^{-1}(\bar{f} - e\mu^*)}{2\sigma^{*2}}$$

- We want to maximize this function with respect to f_p.
- Because only the last term depends on f_p (it is a part of $\bar{f}$) we can omit the other terms and formulate the following:

$$\underset{f_p}{\text{maximize}}\, \ell(f_p) = -\frac{\left(\bar{f} - e\mu^*\right)^T \bar{K}^{-1}(\bar{f} - e\mu^*)}{2\sigma^{*2}} \qquad (6.41)$$

6. Kriging

- This problem can be solved analytically, yielding the mean value of the kriging prediction

$$f_p = \mu^* + k^T K^{-1}(\bar{f} - e\mu^*) \tag{6.42}$$

- The mean square error of the kriging prediction (that is, the expected squared value of the error) is given by

$$\sigma_p^2 = \sigma^{*2}\left[1 - k^T K^{-1} k + \frac{(1 - k^T K^{-1} e)^2}{e^T K^{-1} e}\right] \tag{6.43}$$

- One attractive feature of kriging models is that they are interpolatory and thus match the training data exactly.
- To see how this is true, if x_p is the same as one of the training data points, $x^{(i)}$, then k is just ith column of K.

6. Kriging

- Hence, $K^{-1}k$ is a vector e_i, with all zeros except for 1 in the ith element.
- In the prediction (Eq. 6.42), $k^T K^{-1} = e_i^T$ and so the last term is $f^{(1)} - \mu^*$, which means that $f_p = f^{(1)}$.
- In the mean square error (Eq. 6.43), $k^T K^{-1} k$ is the same as $k^T e_i$.
- This is the ith element of k, which is 1.
- Therefore, the first two terms in the brackets in Eq. 6.43 cancel, and the last term is zero, yielding $\sigma_p^2 = 0$.
- This is expected: If we already sampled the point, the uncertainty about its function value should be zero.
- When describing a fitted kriging model, we often refer to the standard error as the square root of this quantity (i.e., $\sqrt{\sigma_p^2}$).

6. Kriging

Example 6.5: One-dimensional kriging model.

- Consider the decaying sinusoid:

$$f(x) = exp(-0.1x)\sin(x)$$

- We assume, however, that this function is unknown, and we sample at the following points:

$$x = [0.5,2,2.5,9,10]$$

- We can fit a kriging model to this data by following the procedure in this section.
- This includes solving the optimization problem of Eq. 6.38 using a gradient-based method with exact derivatives.
- We fix p=2 and search for θ in the range [10^{-3},10^{2}] with the exponent as the optimization variable.

6. Kriging

Example 6.5: One-dimensional kriging model (continued).

- The resulting interpolation is shown in Fig. 6.20, where we plot the mean line.
- The shaded area represents the uncertainty corresponding to ±1 standard error.
- The uncertainty goes to zero at the known data points and is largest when far from known data points.

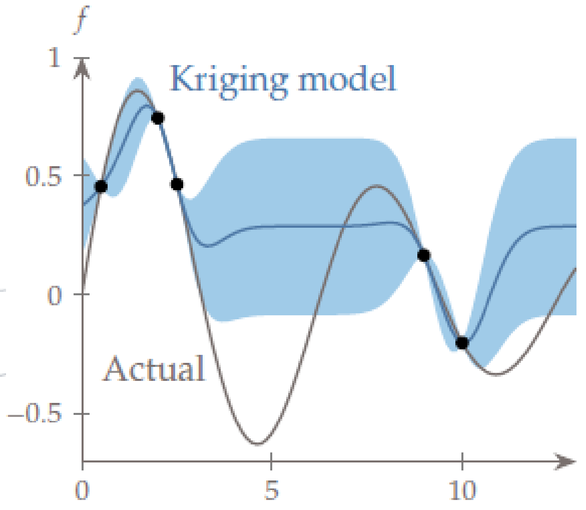


Figure 6.20 Kriging model showing the training data (dots), the kriging predictor (blue line) and the confidence interval corresponding to ±1 standard error (shaded areas), compared to the actual function (gray line).

Source: Martins et Ning, 2021

6. Kriging

- The standard error is directly related to the confidence interval (e.g., ±1 standard error corresponds to a 68% confidence interval).
- If we can provide the gradients of the function at the training data points (in addition to the function values), we can use that information to build a more accurate kriging model.
- This approach is called gradient-enhanced kriging (GEK).
- The methodology is the same as before, except we add more observed outputs (i.e., in addition to the function values at the sampled points, we add their gradients).
- In addition to considering the correlation between the function values at different sampled points, the kernel matrix needs to be expanded to consider correlations between function values and gradients, gradients and function values, and among gradient components.

6. Kriging

- We can use still use equation (Eq. 6.42) for the GEK predictor and equation (Eq. 6.43) for the mean square error if we plug in “expanded versions” of the outputs *f*, the vector *k*, the matrix *K*, and the vector of 1s, *e*.
- We expand the output vector to include not just the function values at the sampled points but also their gradients:

$$f_{GEK} = \begin{bmatrix} f_1 \\ \vdots \\ f_{n_s} \\ \nabla f_1 \\ \vdots \\ \nabla f_{n_s} \end{bmatrix} \qquad (6.44)$$

6. Kriging

- This vector is of length $n_s + n_s n_d$, where n_d is the dimension of x.
- The gradients are usually provided at the same x locations as the function samples, but that is not required.
- Recall that the term $e\mu^*$ in Eq. 6.42 for the kriging predictor represents the expected value of the random variables $F^{(1)}, \dots, F^{(n_s)}$.
- Now that we have expanded the outputs to include the gradients at the sampled points, the mean vector needs to be expanded to include the expected values of $\nabla F^{(1)}$, which are all zero.
- We can still use $e\mu^*$ in the formula for the predictor if we use the following definition:

$$e_{GEK} \equiv [1, \dots, 1, 0, \dots, 0] \tag{6.45}$$

6. Kriging

where 1 occurs for the first n_s entries, and 0 for the remaining $n_s n_d$ entries.

- The additional correlations (between function values and derivatives, and between derivatives and derivatives) are as follows:

$$corr\left(F\left(x^{(i)}\right), F\left(x^{(j)}\right)\right) = K_{ij} \tag{6.46}$$

$$corr\left(F\left(x^{(i)}\right), \frac{\partial F\left(x^{(j)}\right)}{\partial x_l}\right) = \frac{\partial K_{ij}}{\partial x_l^{(j)}} \tag{6.47}$$

$$corr\left(\frac{\partial F\left(x^{(i)}\right)}{\partial x_l}, F\left(x^{(j)}\right)\right) = \frac{\partial K_{ij}}{\partial x_l^{(i)}} \tag{6.48}$$

$$corr\left(\frac{\partial F\left(x^{(i)}\right)}{\partial x_l}, \frac{\partial F\left(x^{(j)}\right)}{\partial x_k}\right) = \frac{\partial^2 K_{ij}}{\partial x_l^{(i)} \partial x_k^{(j)}} \tag{6.49}$$

6. Kriging

- Here, we use *l* and *k* to represent a component of a vector, and we use $K_{ij} = K\left(x^{(i)}, x^{(j)}\right)$ as shorthand.
- For our particular kernel choice (Eq. 6.31), these correlations become the following

$$K_{ij} = exp\left(-\sum_{k=1}^{n_d} \theta_l \left(x_l^{(i)} - x_l^{(l)}\right)^2\right) \tag{6.50}$$

$$\frac{\partial K_{ij}}{\partial x_l^{(j)}} = 2\theta_l \left(x_l^{(i)} - x_l^{(l)}\right) K_{ij} \tag{6.51}$$

$$\frac{\partial K_{ij}}{\partial x_l^{(i)}} = -\frac{\partial K_{ij}}{\partial x_l^{(j)}} \tag{6.52}$$

6. Kriging

$$\frac{\partial^2 K_{ij}}{\partial x_l^{(i)} \partial x_k^{(j)}} = \begin{cases} -4\theta_l\theta_k \left(x_k^{(i)} - x_k^{(l)}\right)\left(x_l^{(i)} - x_l^{(l)}\right) K_{ij} & l \neq k \\ -4\theta_l^2 \left(x_l^{(i)} - x_l^{(l)}\right)^2 K_{ij} + 2\theta_l K_{ij} & l = k \end{cases} \qquad (6.53)$$

where we used p=2.

- Putting this all together yields the expanded correlation matrix:

$$K_{GEK} = \begin{bmatrix} K & J_K \\ J_K^T & H_K \end{bmatrix} \qquad (6.54)$$

- where the ($n_s \times n_s n_d$) block representing the first derivatives is

6. Kriging

$$J_K = \begin{bmatrix} \frac{\partial K_{11}}{\partial x^{(1)}}^T & \cdots & \frac{\partial K_{1n_s}}{\partial x^{(n_s)}}^T \\ \vdots & \ddots & \vdots \\ \frac{\partial K_{1n_s}}{\partial x^{(1)}}^T & \cdots & \frac{\partial K_{n_s n_s}}{\partial x^{(n_s)}}^T \end{bmatrix} \qquad (6.55)$$

and the $(n_s n_d \times n_s n_d)$ matrix of second derivatives is

$$H_K = \begin{bmatrix} \frac{\partial^2 K_{11}}{\partial x^{(1)} \partial x^{(1)}} & \cdots & \frac{\partial^2 K_{1n_s}}{\partial x^{(1)} \partial x^{(n_s)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 K_{n_s 1}}{\partial x^{(n_s)} \partial x^{(1)}} & \cdots & \frac{\partial^2 K_{n_s n_s}}{\partial x^{(n_s)} \partial x^{(n_s)}} \end{bmatrix} \qquad (6.56)$$

6. Kriging

- We can still get our estimates μ^* and σ^{*2} using Eqs. 6.36 and 6.37, but using the expanded versions of K, e, f and replacing n_s in Eq. 6.43 with $n_s(n_d+1)$, which is the new length of the outputs.
- The predictor equations (Eqs. 6.42 and 6.43) also apply with the expanded matrices and vectors.
- However, we also need to expand k in these computations to include the correlations between the gradients at the sampled points with the gradient at the point x where we make a prediction.

6. Kriging

- Thus, the expanded *k* is:

$$k_{GEK} = \begin{bmatrix} k \\ corr\left(\frac{\partial F\left(x^{(1)}\right)}{\partial x^{(1)}}, F\left(x_p\right)\right) = -\frac{\partial K\left(x^{(1)}, x_p\right)}{\partial x^{(1)}} \\ \vdots \\ corr\left(\frac{\partial F\left(x^{(n_s)}\right)}{\partial x^{(n_s)}}, F\left(x_p\right)\right) = -\frac{\partial K\left(x^{(n_s)}, x_p\right)}{\partial x^{(n_s)}} \end{bmatrix} \qquad (6.57)$$

- One difficulty with GEK is that the kernel matrix quickly grows in size as the dimension of the problem increases, the number of samples increases, or both.
- Various approaches have been proposed to improve the scaling with higher dimensions, such as a weighted sum of smaller correlation matrices or a partial least squares approach.

6. Kriging

Example 6.6: Gradient-enhanced kriging.

- By repeating Ex. 6.6 but this time including the gradients (Fig. 6.21) the standard error reduces dramatically between points.
- The additional information contained in the derivatives significantly helps in creating a more accurate fit.

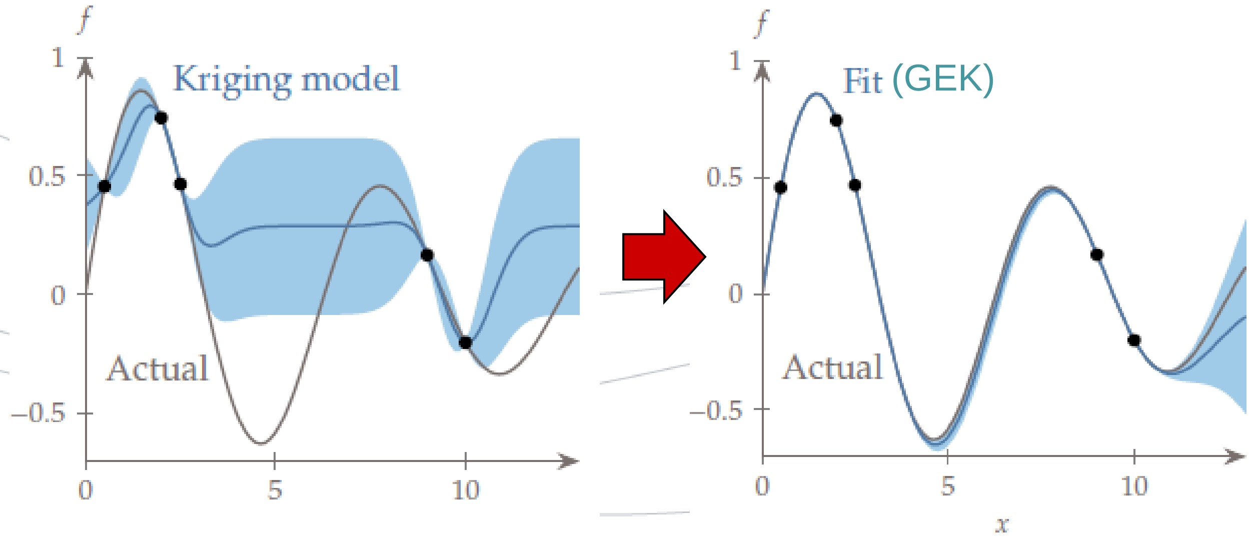


Figure 6.21 A GEK fit to the input data (circles) and a shaded confidence interval.

Source: Martins et Ning, 2021

6. Kriging

Example 6.7: Two-dimensional kriging.

- The Jones function is shown on the left in Fig. 6.22. Using GEK with only 10 training points from a Hammersley sequence (shown as the dots), created the surrogate model on the right. A reasonable representation of this multimodal space can be captured even with a small number of samples.

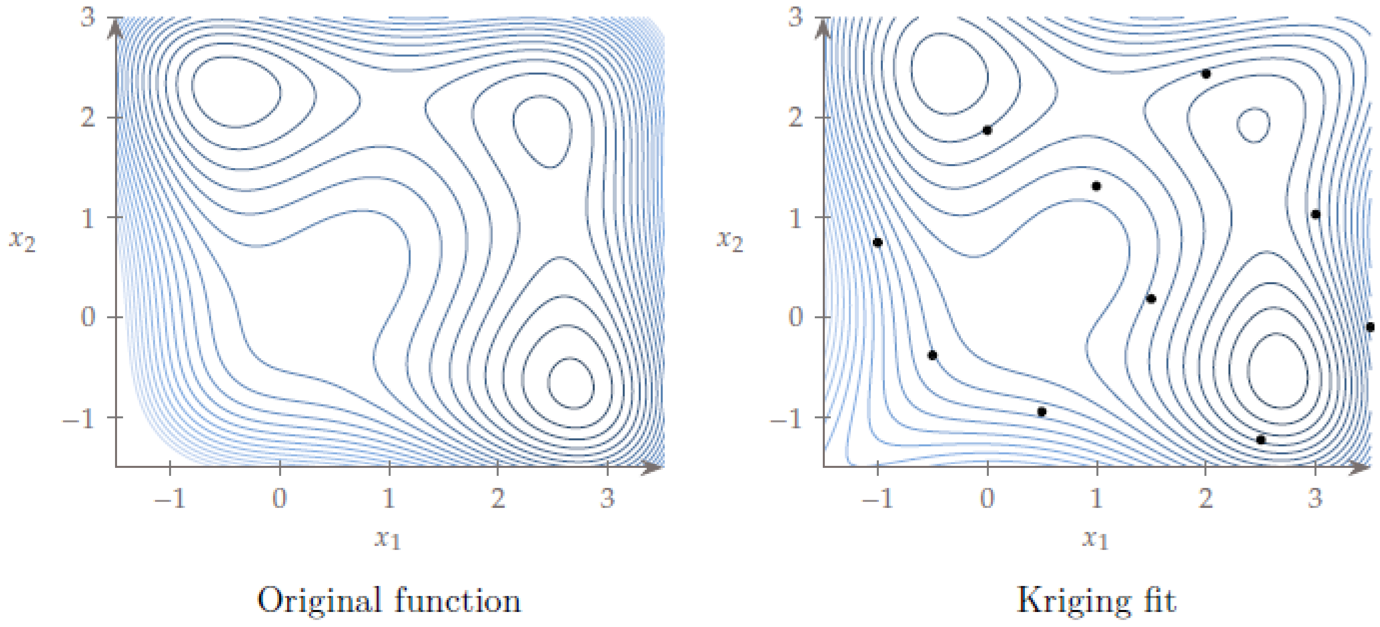


Figure 6.22 Kriging fit to the multimodal Jones function.

Source: Martins et Ning, 2021

6. Kriging

- The version of kriging in this section is interpolatory.
- For noisy data, a regression approach can be used by modifying the correlation matrix as follows:

$$K_{reg} = K + \tau I \tag{6.58}$$

with $\tau > 0$.

- This adds a positive constant along the diagonal, so the model no longer correlates perfectly with the provided points.
- The parameter τ is then an additional parameter to estimate in the maximum likelihood optimization.
- Even for interpolatory models, this term is often still added to the covariance matrix with a small constant value of τ (near machine precision) to ensure that the correlation matrix is invertible.

6. Kriging

- This section focused on some of the most common choices when using kriging, but many other versions exist.

7. Deep Neural Networks

- Like kriging, deep neural networks can be used to approximate highly nonlinear simulations where we do not need to provide a parametric form.
- Neural networks follow the same basic steps described for other surrogate models but with a unique model leading to specialized approaches for derivative computation and optimization strategy.
- Neural networks loosely mimic the brain, which consists of a vast network of neurons.
- In neural networks, each neuron is a node that represents a simple function.
- A network defines chains of these simple functions to obtain composite functions that are much more complex.

7. Deep Neural Networks

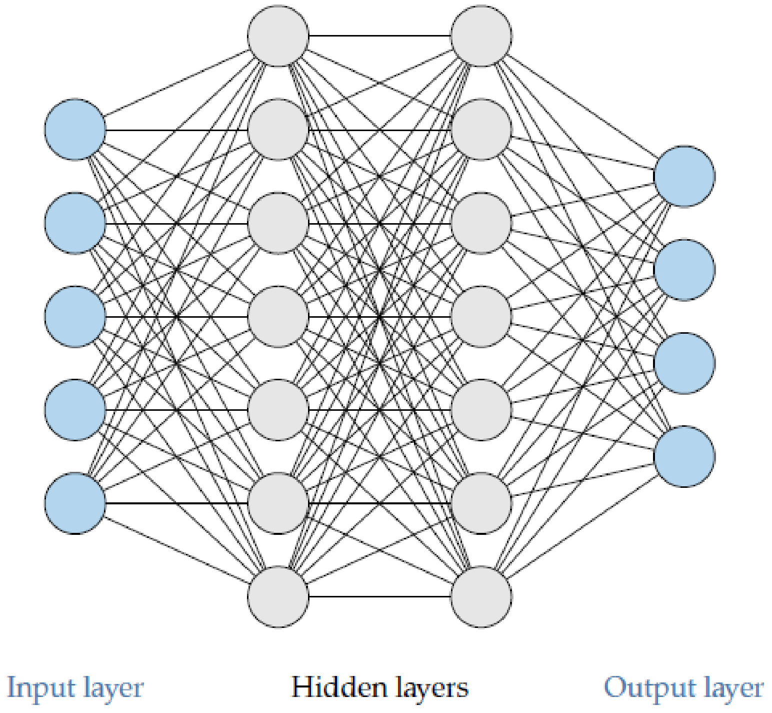
- For example, three simple functions, $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$, may be chained into the composite function (or network):

$$f(x) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(x)\right)\right) \tag{6.59}$$

- Even though each function may be simple, the composite function can express complex behaviour.
- Most neural networks are feed-forward networks, meaning that information flows from inputs x to outputs f.
- Recurrent neural networks include feedback connections.
- Figure 6.23 shows a diagram of a neural network.
- Each node represents a neuron.
- The neurons are connected between consecutive layers, forming a dense network.

7. Deep Neural Networks

- The first layer is the input layer, the last one is the output layer, and the middle ones are the hidden layers.
- The total number of layers is called the network's depth.
- Deep neural networks have many layers, enabling the modeling of complex behavior.



Source: Martins et Ning, 2021

Figure 6.23 Deep neural network with two hidden layers.

7. Deep Neural Networks

- The first and last layers can be viewed as the inputs and outputs of a surrogate model.
- Each neuron in the hidden layer represents a function.
- This means that the output from a neuron is a number, and thus the output from a whole layer can be represented as a vector x.
- We represent the vector of values for layer k by $x^{(k)}$, and the value for the ith neuron in layer k by $x_i^{(k)}$.
- Consider a neuron in layer k.
- This neuron is connected to many neurons from the previous layer k-1 (see the first part of Fig. 6.24).
- We need to choose a functional form for each neuron in the layer that takes in the values from the previous layer as inputs.

7. Deep Neural Networks

- Chaining together linear functions would yield another linear function.
- Therefore, some layers must use nonlinear functions.

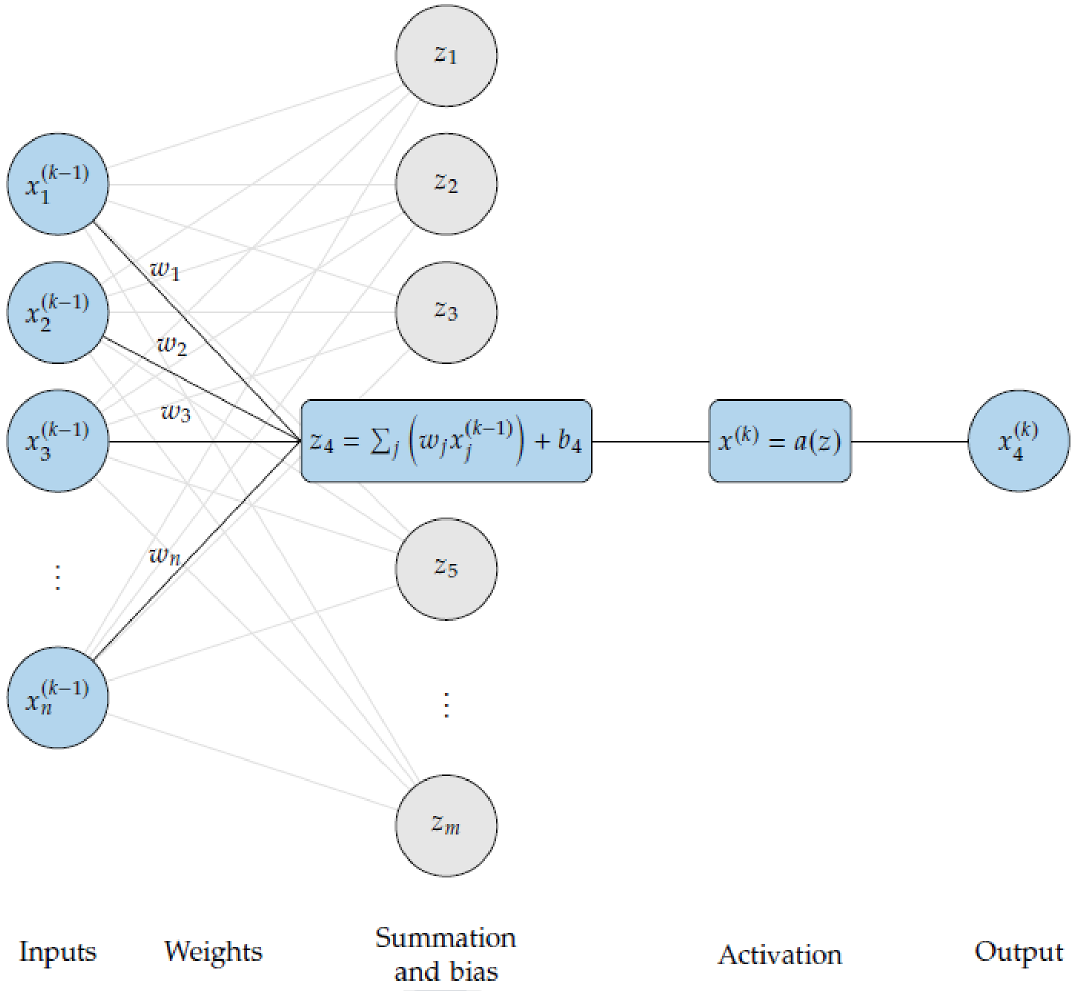


Figure 6.24 Typical functional form for a neuron in the neural net.

Source: Martins et Ning, 2021

7. Deep Neural Networks

- The most common choice for hidden layers is a layer of linear functions followed by a layer of functions that create nonlinearity.
- A neuron in the linear layer produces the following intermediate variable:

$$z = \sum_{j=1}^{n} w_j x_j^{(k-1)} + b \qquad (6.60)$$

- In vector form:

$$z = w^T x^{(k-1)} + b \qquad (6.61)$$

7. Deep Neural Networks

- The first term is a weighted sum of the values from the neurons in the previous layer.
- The w vector contains the weights.
- The term b is the bias, which is an offset that scales the significance of the overall output.
- These two terms are analogous to the weights used in the previous section but with the constant term separated for convenience.
- The second column of Fig. 6.24 illustrates the linear (summation and bias) layer.
- Next, we pass z through an activation function, which we call $a(z)$.

7. Deep Neural Networks

- Historically, one of the most common activation functions has been the sigmoid function:

$$a(z) = \frac{1}{1+e^{-z}} \tag{6.62}$$

- This function is shown in the top plot of Fig. 6.25.
- The sigmoid function produces values between 0 and 1, so large negative inputs result in insignificant outputs (close to 0), and large positive inputs produce outputs close to 1.
- Most modern neural networks use a rectified linear unit (ReLU) as the activation function:

$$a(z) = \max(0, z) \tag{6.63}$$

7. Deep Neural Networks

- This function is shown in the bottom plot of Fig. 6.25.
- The ReLU has been found to be far more effective than the sigmoid function in producing accurate neural networks.
- This activation function eliminates negative inputs.
- Thus, the bias term can be thought of as a threshold establishing what constitutes a significant value.
- The final two columns of Fig. 6.24 illustrate the activation step.

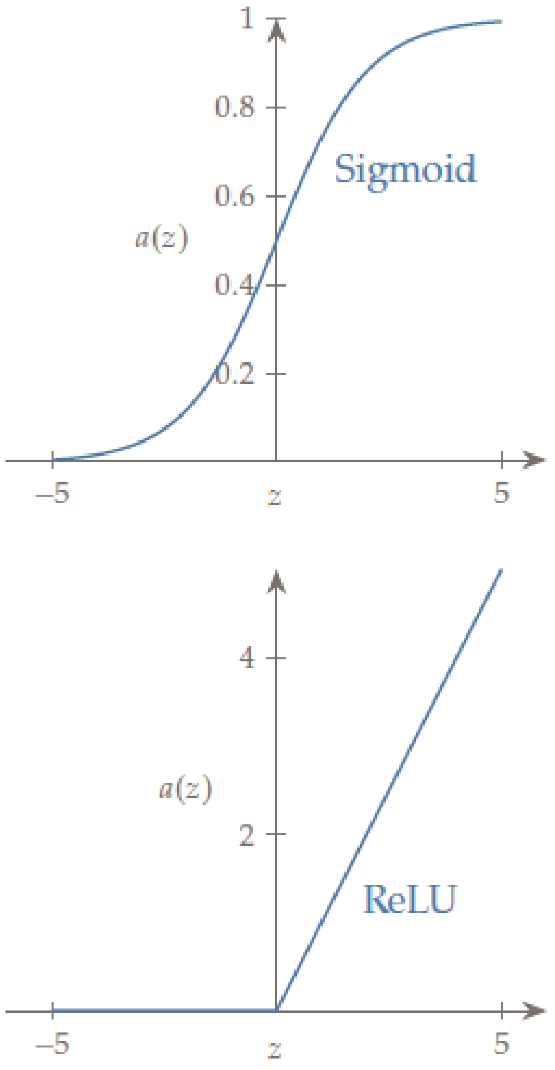


Figure 6.25 Activation functions.

Source: Martins et Ning, 2021

7. Deep Neural Networks

- Combining the linear function with the activation function produces the output for the *i*th neuron:

$$x_i^{(k)} = a\left(w^T x^{(k-1)} + b_i\right) \tag{6.64}$$

- To compute the outputs for all the neurons in this layer, the weights *w* for this one neuron form one row in a matrix of weights *W* and we can write :

$$\begin{bmatrix} x_1^{(k)} \\ \vdots \\ x_i^{(k)} \\ \vdots \\ x_{n_k}^{(k)} \end{bmatrix} = a\left(\begin{bmatrix} W_{1,1} & \cdots & W_{1,j} & \cdots & W_{1,n_k} \\ \vdots & & \vdots & & \vdots \\ W_{i,1} & \cdots & W_{i,j} & \cdots & W_{i,n_k} \\ \vdots & & \vdots & & \vdots \\ W_{n_k,1} & \cdots & W_{n_k,j} & \cdots & W_{n_k,n_k} \end{bmatrix} \begin{bmatrix} x_1^{(k-1)} \\ \vdots \\ x_j^{(k-1)} \\ \vdots \\ x_{n_{k-1}}^{(k-1)} \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_{n_k} \end{bmatrix}\right) \tag{6.65}$$

7. Deep Neural Networks

or

$$x^{(k)} = a\left(Wx^{(k-1)} + b\right) \tag{6.66}$$

- The activation function is applied separately for each row.
- The following equation is more explicit (where w_i is the ith row of W):

$$x_i^{(k)} = a\left(w_i^T W x_i^{(k-1)} + b_i\right) \tag{6.67}$$

- This neural network is now parameterized by a number of weights.
- Like other surrogate models, we need to determine the optimal value for these parameters (i.e., train the network) using training data.

7. Deep Neural Networks

- In the example of Fig. 6.23, there is a layer of 5 neurons, 7 neurons, 7 neurons, and then 4 neurons, and so there would be 5×7+7×7+7×4 weights and 7+7+4 bias terms, giving a total of 130 variables.
- This represents a small neural network because there are few inputs and few outputs.
- Large neural networks can have millions of variables.
- We need to optimize those variables to minimize a cost function.
- As before, we use a maximum likelihood estimate where we optimize the parameters θ (weights and biases in this case) to maximize the probability of observing the output data y conditioned on our inputs x.

7. Deep Neural Networks

- This results in a sum of squared errors function:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{n} \left[\hat{f}\left(\theta; x^{(i)}\right) - f^{(i)}\right]^2 \qquad (6.68)$$

- We now have the objective and variables in place to train the neural network.
- As with the other models discussed in this chapter, it is critical to set aside some data for cross validation.
- Because the optimization problem (Eq. 6.68) often has a large number of parameters θ, we generally use a gradient-based optimization algorithm.
- To solve Eq. 6.68 using gradient-based optimization, we require the derivatives of the objective function with respect to the weighs θ.

7. Deep Neural Networks

- Because the objective is a scalar and the number of weights is large, reverse-mode algorithmic differentiation (AD) is ideal to compute the required derivatives.
- Reverse-mode AD is known in the machine learning community as backpropagation.
- Whereas general-purpose reverse-mode AD operates at the code level, backpropagation usually operates on larger sets of operations and data structures defined in machine learning libraries.
- Although less general, this approach can increase efficiency and stability.
- The ReLU activation function (Fig. 6.25, bottom) is not differentiable at z=0, but in practice, this is generally not problematic - primarily because these methods typically rely on inexact gradients anyway, as discussed next.

7. Deep Neural Networks

- The objective function in Eq. 6.68 consists of a sum of subfunctions, each of which depends on a single data point $\left(x^{(i)}, f^{(i)}\right)$.
- Objective functions vary across machine learning applications, but most have this same form:

$$\underset{\theta}{\text{minimize}}\, f(\theta) \tag{6.69}$$

where

$$f(\theta) = \sum_{i=1}^{n} \ell\left(\theta; x^{(i)}, f^{(i)}\right) = \sum_{i=1}^{n} \ell_i(\theta) \tag{6.70}$$

- As previously mentioned, the challenge with these problems is that we often have large training sets where *n* may be in the billions.

7. Deep Neural Networks

- That means that computing the objective can be costly, and computing the gradient can be even more costly.
- If we divide the objective by n (which does not change the solution), the objective function becomes an approximation of the expected value:

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta) = \mathbb{E}\big(\ell(\theta)\big) \qquad (6.71)$$

- From probability theory, we know that we can estimate an expected value from a smaller set of random samples.
- For the application of estimating a gradient, we call this subset of random samples a minibatch $S = \{x^{(1)} \dots x^{(m)}\}$, where m is usually between 1 and a few hundred.

7. Deep Neural Networks

- The entries $x^{(1)}$, ..., $x^{(m)}$ do not correspond to the first n entries but are drawn randomly from a uniform probability distribution (Fig. 6.26).
- Using the minibatch, we can estimate the gradient as the sum of the subfunction gradients at different training points:

$$\nabla_\theta f(\theta) \approx \frac{1}{m} \sum_{i \in S}^{n} \nabla_\theta \ell\left(\theta; x^{(i)}, f^{(i)}\right) \qquad (6.72)$$

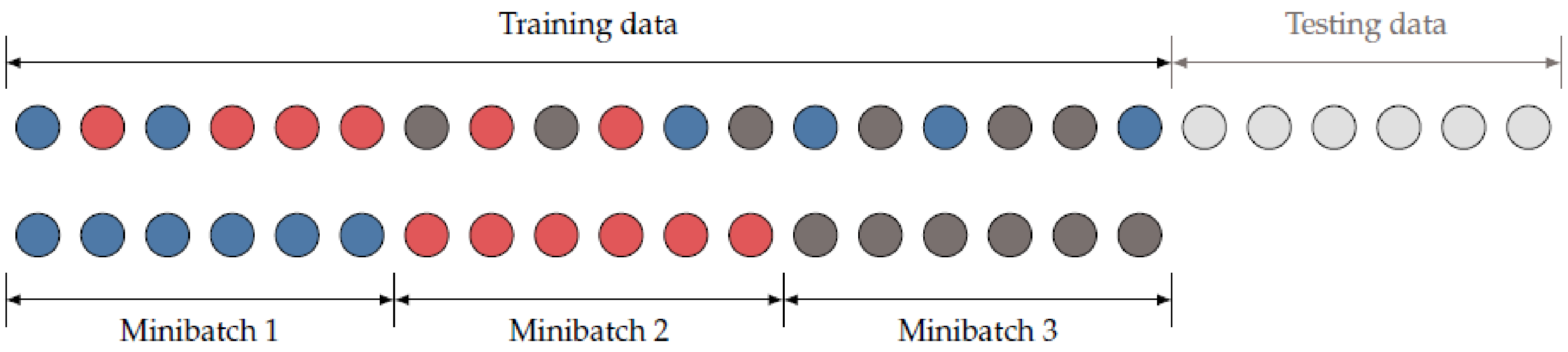


Figure 6.26 Minibatches are randomly drawn from the training data.

Source: Martins et Ning, 2021

7. Deep Neural Networks

- Thus, we divide the training data into these minibatches and use a new minibatch to estimate the gradients at each iteration in the optimization.
- This approach works well for these specific problems because of the unique form for the objective (Eq. 6.71).
- As an example, for one million training samples, a single gradient evaluation would require evaluating all one million training samples.
- Alternatively, for a similar cost, a minibatch approach can update the optimization variables a million times using the gradient estimated from one training sample at a time.
- This latter process usually converges much faster, mainly because we are only fitting parameters against limited data in these problems, so we generally do not need to find the exact minimum.

7. Deep Neural Networks

- Typically, this gradient is used with steepest descent methods (Chapter 4), more typically referred to as gradient descent in the machine learning communities.
- As discussed in Chapter 4, steepest descent is not the most effective optimization algorithm.
- However, steepest descent with the minibatch updates, called stochastic gradient descent, has been found to work well in machine learning applications.
- This suitability is primarily because:
  1. many machine learning optimizations are performed repeatedly,
  2. the true objective is difficult to formalize, and
  3. finding the absolute minimum is not as important as finding a good enough solution quickly.

7. Deep Neural Networks

- One key difference in stochastic gradient descent relative to the steepest descent method is that we do not perform a line search.
- Instead, the step size (called the learning rate in machine learning applications) is a preselected value that is usually decreased between major optimization iterations.
- Stochastic minibatching is easily applied to first-order methods and has thus driven the development of improvements on stochastic gradient descent, such as momentum, Adagrad, and Adam.
- Although some of these methods may seem somewhat ad hoc, there is mathematical rigor to many of them.
- Batching makes the gradients noisy, so second order methods are generally not pursued.

7. Deep Neural Networks

- However, ongoing research is exploring stochastic batch approaches that might effectively leverage the benefits of second-order methods.

8. Optimization and Infill

- Once a surrogate model has been built, optimization may be performed using the surrogate function values.
- That is, instead of minimizing the expensive function $f(x)$, we minimize the model $\hat{f}(x)$, as previously illustrated in Fig. 6.01.
- The surrogate model may be static, but more commonly, it is updated between optimization iterations by adding new training data and rebuilding the model.
- The process by which we select new data points is called infill.
- There are two main approaches to infill: prediction-based exploitation and error-based exploration.
- Typically, only one infill point is chosen at a time.
- The assumption is that evaluating the model is computationally expensive, but rebuilding and evaluating the surrogate is cheap.

8.1. Exploitation

- For models that do not provide uncertainty estimates, the only real option is exploitation.
- A prediction-based exploitation infill strategy adds an infill point wherever the surrogate predicts the optimum.
- The reasoning behind this approach is that in SBO, we do not necessarily care about having a globally accurate surrogate; instead, we only care about having an accurate surrogate near the optimum.
- The most logical point to sample is thus the optimum predicted by the surrogate.
- Likely, the location predicted by the surrogate will not be at the true optimum.
- However, evaluating this point adds valuable information in the region of interest.

8.1. Exploitation

- We rebuild the surrogate and re-optimize, repeating the process until convergence.
- This approach usually results in the quickest convergence to an optimum, which is desirable when the actual function is expensive to evaluate.
- The downside is that we may converge prematurely to an inferior local optimum for problems with multiple local optima.
- Even though the approach is called exploitation, the optimizer used on the surrogate can be a global search method (gradient-based or gradient-free), although it is usually a local search method.
- If uncertainty is present, using the mean value of the surrogate as the infill criteria results in essentially an exploitation strategy.

8.1. Exploitation

- The algorithm is outlined in Alg. 6.3.
- Convergence could be based on a maximum number of iterations or a tolerance for the objective function’s fractional change.

8.1. Exploitation

Algorithm 6.3: Exploitation-driven surrogate-based optimization

```
Inputs:
    $n_s$: Number of initial samples
    $\underline{x}, \overline{x}$: Variable lower and upper bounds
    $\tau$: Convergence tolerance
Outputs:
    $x^*$: Best point identified
    $f^*$: Corresponding function value

$x^{(i)} = \text{sample}(n_s, n_d)$                                  Sample
$f^{(i)} = f\left(x^{(i)}\right)$                                    Evaluate function
$k = 0$
while $k < k_{\max}$ and $\left(\hat{f}^* - f_{\text{new}}\right) / \hat{f}^* < \tau$ do
    $\hat{f} = \text{surrogate}\left(x^{(i)}, f^{(i)}\right)$        Construct surrogate model
    $x^*, \hat{f}^* = \min \hat{f}(x)$                               Perform optimization on the surrogate function
    $f_{\text{new}} = f(x^*)$                                        Evaluate true function at predicted optimum
    $x^{(i)} = x^{(i)} \cup x^*$                                     Append new point to training data
    $f^{(i)} = f^{(i)} \cup f_{\text{new}}$                          Append corresponding function value
    $k = k + 1$
end while
```

8.2. Efficient global optimization

- An alternative approach to infill uses error-based exploration.
- This approach requires using kriging (Section 6.6) or another surrogate approach that predicts not just function values but also error estimates.
- Although many infill metrics exist within this category, we focus on a popular one called expected improvement, and the associated algorithm, efficient global optimization (EGO).
- As stated previously, sampling where the mean is low is an exploitation strategy, but we do not necessarily want to sample where the uncertainty is high.
- That may lead to wasteful function calls in regions of the design space where the surrogate model is inaccurate but which are far from any optimum.

8.2. Efficient global optimization

- Let the best solution we have found so far be represented $f^* = f(x^*)$.
- The improvement for any new test point x is then given by

$$I(x) = \max(f^* - f(x), 0) \tag{6.73}$$

- If $f(x) \geq f^*$, there is no improvement, but if $f(x) < f^*$, the improvement is the objective decrease magnitude.
- However, $f(x)$ is not a deterministic value in this model but rather a probability distribution.
- Thus, the expected improvement is the expected value (or mean) of the improvement:

$$EI(x) = \mathbb{E}\left[\max(f^* - f(x), 0)\right] \tag{6.74}$$

8.2. Efficient global optimization

- The expected value for a kriging model can be found analytically as:

$$EI(x) = \left(f^* - \mu_f(x)\right)\Phi\left(\frac{f^* - \mu_f(x)}{\sigma_f(x)}\right) + \sigma_f(x)\phi\left(\frac{f^* - \mu_f(x)}{\sigma_f(x)}\right) \quad (6.75)$$

- where Φ and ϕ are the CDF and PDF, respectively, for the standard normal distribution, and μ_f and σ_f are the mean and standard error functions produced from kriging (Eqs. 6.42 and 6.43).
- The algorithm is similar to that of the previous section (Alg. 6.3), but instead of choosing the minimum of the surrogate, the selected infill point is the point with the greatest expected improvement.
- The corresponding algorithm is detailed in Alg. 6.4.

8. Optimization and Infill

8.2. Efficient global optimization

Algorithm 6.4: Efficient global optimization

Inputs:

n_s: Number of initial samples
$\underline{x}, \overline{x}$: Lower and upper bounds
τ: Minimum expected improvement

Outputs:

x^*: Best point identified
f^*: Corresponding function value

```
$x^{(i)} = \text{sample}(n_s, n_d)$                                  Sample
$f^{(i)} = f(x^{(i)})$                                               Evaluate function
$f^* = \min\{f^{(i)}\}$                                              Best point so far; also update corresponding $x^*$
$k = 0$
while $k < k_{\max}$ and $f_{ei} > \tau$ do
    $\mu(x), \sigma(x) = \text{GP}(x^{(i)}, f^{(i)})$                Construct Gaussian process surrogate model
    $x_k, f_{ei} = \max EI(x)$                                       Maximize expected improvement
    $f_k = f(x_k)$                                                   Evaluate true function at predicted optimum
    $f^* = \min\{f^*, f_k\}$                                         Update best point and $x^*$ if necessary
    $x^{(i)} \leftarrow [x^{(i)}, x_k]$                              Add new point to training data
    $f^{(i)} \leftarrow [f^{(i)}, f_k]$
    $k = k + 1$
end while
```

8.2. Efficient global optimization

Example 6.8: Expected improvement.

- Consider the same one-dimensional function of Ex. 6.5 using kriging (without gradients), where the data points and fit are shown again in Fig. 6.27.
- The best point we have found so far is denoted in the figure as (x^*, f^*).
- For a Gaussian process model, the fit also provides a 1-standard-error region, represented by the shaded region.

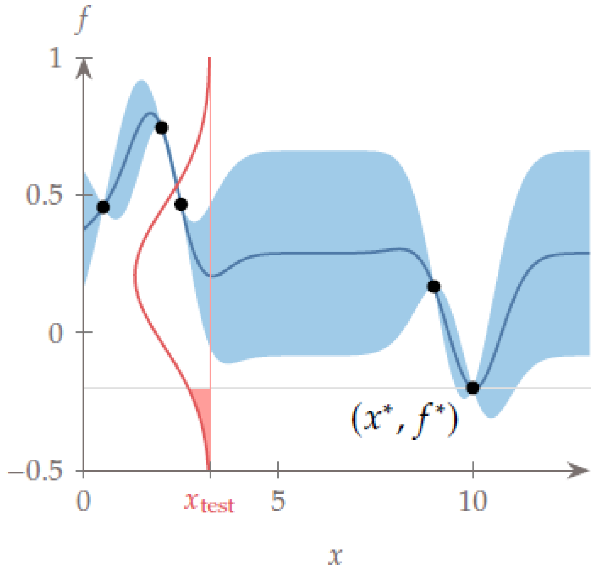


Figure 6.27 At a given test point (x_{test}=3.25), we highlight the probability distribution and the expected improvement in the shaded red region.

Source: Martins et Ning, 2021

8.2. Efficient global optimization

Example 6.8: Expected improvement (continued).

- Now imagine we want to evaluate this function at some new test point, x_{test}=3.25.
- In Fig. 6.27, the full probability distribution for the objective at x_{test} is shown in red.
- This probability distribution occurs at a fixed value of x, so we can visualize it in a dimension coming out of the page.
- Now, let us evaluate the expected improvement not just at x_{test}, but across the domain.
- The result is shown by the red function in the top left of Fig. 6.28.
- A few select iterations in the convergence process are shown in the remaining panes of Fig. 6.28.

8.2. Efficient global optimization

Example 6.8: Expected improvement (continued).

- On the top right, after the first promising valley is well explored, the middle region becomes the most likely location of potential improvements.
- Eventually, the potential improvements are minor and we terminate (bottom right).

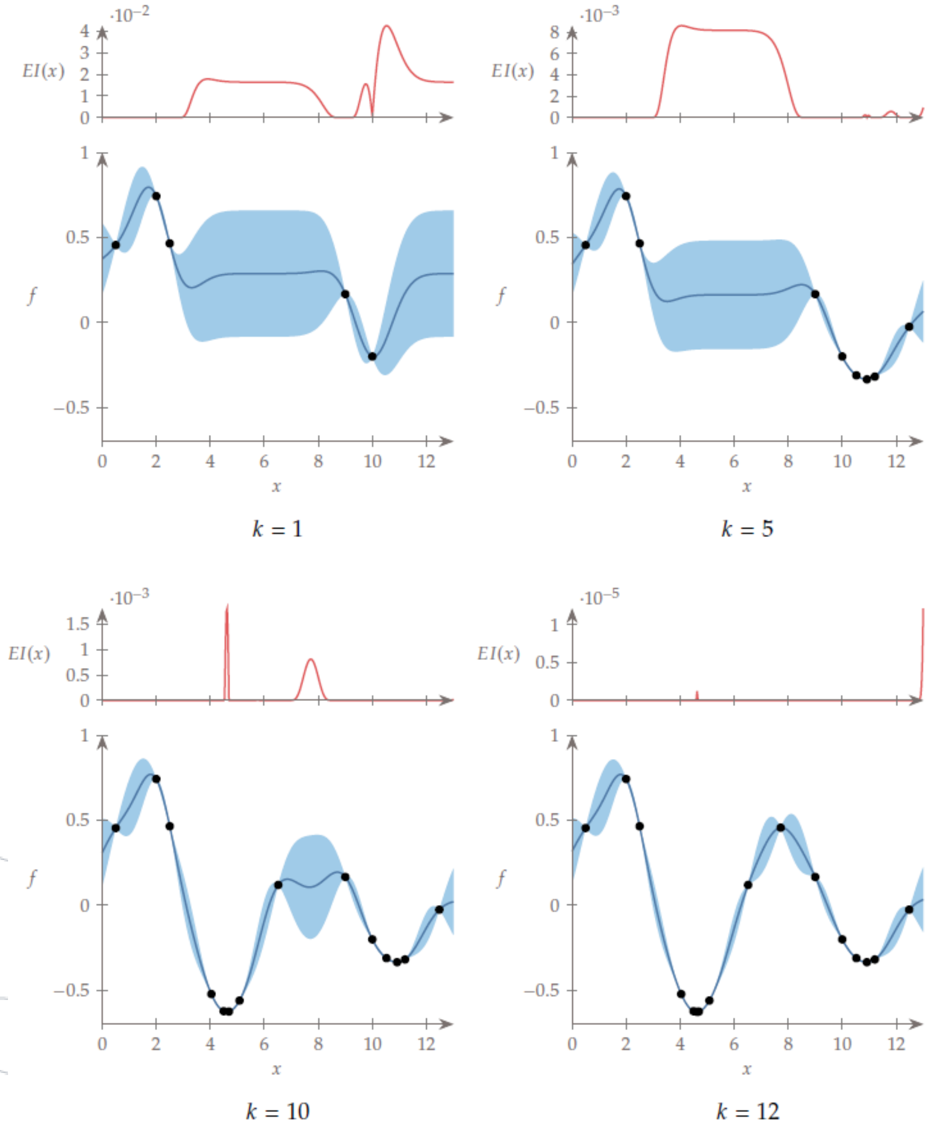


Figure 6.28 The process is repeated by evaluating expected improvement across the domain.

Source: Martins et Ning, 2021