Chapter 2
An Overview of Mathematical Methods for Numerical Optimization

Daniel E. Marthaler

Abstract This chapter serves as a basic overview of mathematical optimization problems and reviews how certain classes of these problems are solved. For the general category of nonlinear problems, both smooth and nonsmooth "*Derivative Free*" topics are discussed with and without constraints.

2.1 Introduction

This book is concerned with finding the "best" solution to particular metamaterial design problems. Best is put in quotations because the idea of what represents a good design is defined by the user, and very much depends on the application. The best design for some problems may be the one that reflects the most light transmitted at a given wavelength. Others might be those that absorb the most light throughout a range of wavelengths. Whatever the definition used to define what the "best" design implies, once it is established, we actually want to determine the structure that will yield this best solution. Mathematical optimization is the process we will use to select an optimal choice from a set of alternatives for this determination.

In this chapter, we give an overview of mathematical optimization and introduce the general (nonlinear) problem. The concepts introduced informally here will be covered in more detail in later chapters as specific applications and instantiations are discussed. We attempt to give a summary of the major work that has been done in this field, structuring it around different classes of the general problem. For a chapter of this type, brevity is a must, as the shear amount of material covered would (and does) fill entire textbooks.

Furthermore, when discussing mathematical optimization, we implicitly assume that we have a problem to optimize. For the scope of this book, we focus on metamaterial design problems. In general though, the problem we seek to optimize has an *objective function* and in most cases, actually determining the correct form of this function is one of the most difficult aspects to conduct.

D.E. Marthaler (✉)
GE Global Research: Industrial Internet Analytics, San Ramon, CA 94583, USA
e-mail: marthaler@ge.com

K. Diest (ed.), *Numerical Methods for Metamaterial Design*,
Topics in Applied Physics 127, DOI 10.1007/978-94-007-6664-8_2,

When modeling mathematical optimization problems, we separate them into different classes according to the type of problem they are attempting to solve. The problems may have models that are linear or nonlinear and may or may not be constrained. The objective and constraint functions might be differentiable or non-differentiable, convex or non-convex. In some cases, the problems may only be given via a *black box*, that is, we only know the outputs of the objective function given certain inputs, but not any actual analytical form. Nice references on fundamental theory, methods, algorithm analysis and advice on how to obtain and implement good algorithms for different classes of optimization are provided in [1, 2, 7, 8, 12, 29, 30, 37, 57] among others. We give only a cursory overview of various types of solution techniques. Interested readers are encouraged to refer to the references for more detail.

The rest of the chapter is organized as follows: Sect. 2.2 lays out the general optimization problem and includes a high level discussion on constructing viable objective functions. Section 2.3 discuses linear and convex models and solutions, in particular, the least squares method and different regularizers. Section 2.4 discusses optimization problems that utilize derivatives of the objective function, with subsections focusing on those with and without constraints. Finally, Sect. 2.5 looks at algorithms for optimization problems where derivative information is not available, either because the objective function is not differentiable, the derivative is not available, or the derivative is just too expensive to compute. We conclude with a short summary.

2.2 Mathematical Optimization

The present work considers general multi-objective optimization problems that may be written in the following form:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) = \left[f_1(\mathbf{x}),\, f_2(\mathbf{x}), \ldots,\, f_k(\mathbf{x})\right]^T \\ &\text{subject to} \\ &\quad g_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, m_{\text{ieq}}, \\ &\quad h_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, m_{\text{eq}}. \end{aligned} \tag{2.1}$$

Here $\mathbf{x} = (x_1, \ldots, x_n)$ is the variable to be minimized, $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^k$ is a multi-valued objective function, the functions $g_j : \mathbb{R}^n \to \mathbb{R}$, $j = 1, \ldots, m_{\text{ieq}}$, are the *inequality constraint functions*, and the functions $h_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m_{\text{eq}}$, are the *equality constraint functions*.

We define the space of *feasible solutions* or the *feasible set* as the set of all points that satisfy the constraints:

$$\Omega = \left\{\mathbf{y} \in \mathbb{R}^n : g_i(\mathbf{y}) \leq 0,\ i = 1, \ldots, m_{\text{ieq}} \text{ and } h_j(\mathbf{y}) = 0,\ j = 1 \ldots, m_{\text{eq}}\right\}.$$

The *attainable set* is the range of the feasible set under the objective function:

$$\mathbb{A} = \big\{\mathbf{F}(\mathbf{x}) : \mathbf{x} \in \Omega\big\}.$$

Typically in multi-objective optimization, there is no single global solution. It is often necessary to instead seek solutions satisfying Pareto optimality. A point $\mathbf{x}^* \in \Omega$ is *Pareto optimal* if and only if there is no other point $\mathbf{x} \in \Omega$ such that $\mathbf{F}(\mathbf{x}) \leq \mathbf{F}(\mathbf{x}^*)$ and $F_i(\mathbf{x}) < F_i(\mathbf{x}^*)$ for at least one i. That is, no element of $\mathbf{F}$ can be made better without (at least) one other element being made worse [32].

The concept of Pareto optimality invariably leads practitioners to decide which elements of $\mathbf{F}$ are "more important" than others. Having such a ranking of the elements of the objective function, the theory of preferences [38, 43, 44] allows for the construction of a *utility function*. This allows us to convert the general multi-objective function into a single scalar-valued objective function.

One of the most general utility functions is the weighted exponential sum:

$$U = \sum_{i=1}^{k} w_i \big[F_i(\mathbf{x})\big]^p \tag{2.2}$$

for some $p > 0$. Generally, p is proportional to the amount of emphasis placed on minimizing the function with the largest difference between $F_i(\mathbf{x})$ and the minimizer of $F_i(\mathbf{x})$ [28]. Without loss of generality, we can assume $F_i(\mathbf{x}) > 0$, for all i, otherwise we can rescale the objective function to make it so. Here, $\mathbf{w} = \{w_1, \ldots, w_k\}$ is a vector of weights, typically set by the practitioner, such that $\sum_{i=1}^{k} w_i = 1,\ w_i > 0$. Generally, the relative ordering of the weights reflects the relative importance of the objectives.

The most common implementation of Eq. (2.2) is to set $p = 1$, i.e.,

$$U = \sum_{i=1}^{k} w_i F_i(\mathbf{x}), \tag{2.3}$$

which is commonly referred to as the *weighted sum method*. If all of the weights are positive, then the minimum of Eq. (2.3) is Pareto optimal [56], that is, a minimizer of Eq. (2.3) is a Pareto solution of Eq. (2.1).

Selecting non-arbitrary weights is a difficult undertaking. Many approaches exist in selecting weights, surveys of which are provided by [16, 19, 23, 55]. Unfortunately, a satisfactory method to select appropriate weights does not guarantee that the final solution will be acceptable, that is, aligned with predefined preferences. In fact, it is known that weights must be functions of the original objectives in order for a weighted sum to mimic a list of preferences accurately [34]. They cannot be constants. Nevertheless, we proceed in assuming that our multi-objective function in Eq. (2.1) will be converted into a scalar objective, leading to our general problem

for the remainder of the chapter:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{subject to} \\ &\quad g_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, m_{\text{ieq}}, \\ &\quad h_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, m_{\text{eq}}, \end{aligned} \tag{2.4}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, and the other functions are as in Eq. (2.1).

2.3 Finding Solutions

In attempting to solve all but the most trivial of problems in the form of Eq. (2.4), a numerical algorithm is used to find a solution $\mathbf{x}^*$. Different objective functions f and constraint functions g, h are more efficiently solved with different types of algorithms. To deduce which algorithm would best assist in finding optimal solutions, we first determine the class of problem characterized by particular forms of the objective and constraint functions.

The simplest form of Eq. (2.4) is in fitting a regression line $y = mx + b$ through a pair of points (x_i, y_i), $i = 1, 2$. We choose the objective function $f(\mathbf{x}) = (\mathbf{y} - m\mathbf{x} - b)^2$ and there are no constraints. Here, $\mathbf{x} = (x_1, x_2)$, and $\mathbf{y} = (y_1, y_2)$. The optimal solution to this problem is given by

$$\begin{aligned} m &= \frac{y_2 - y_1}{x_2 - x_1}, \\ b &= y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1. \end{aligned}$$

When there are more than two points, it is usually impossible to fit a line through all of the points, so instead, we find the line that minimizes the total squared distance to the points:

$$\min \sum_{i=1}^{N} (a x_i + b - y_i)^2.$$

In higher dimensions, the analog to this line fitting problem is to find constants $(a_1, a_2, \ldots, a_n)$ that solve

$$\min \sum_{i=1}^{N} \left(a_i x_i^{(j)} - y_i^{(j)} \right)^2$$

for each $\mathbf{x}^{(j)}, \mathbf{y}^{(j)}$ pair (we omit b for clarity). In matrix notation, this is equivalent to finding the minimum of the function

$$f(\mathbf{a}) = |X\mathbf{a} - \mathbf{y}|_2^2$$

where X is the matrix whose ith row is $\mathbf{x}^{(i)}$ and $\mathbf{y} = (y_1, \ldots, y_N)^T$. A more common designation to this problem is writing X as A, $\mathbf{a}$ as $\mathbf{x}$ and $\mathbf{y}$ as $\mathbf{b}$. We then solve the problem $A\mathbf{x} = \mathbf{b}$. Problems of this type are referred to as *Least Squares* problems and formulating them as minimization problems

$$\min_{\mathbf{x}} |A\mathbf{x} - \mathbf{b}|_2^2 \tag{2.5}$$

leads to a *residual least squares* (RSS) problem. There are many algorithms that solve RSS problems. For a list and introduction, see, for example, [17].

It is well known that attempting to minimize an RSS problem via a numerical method can lead to instabilities. This occurs when the matrix A is not of full rank or when the matrix $A^T A$ is not invertible. In such situations, Eq. (2.5) is stabilized by including a *regularization* term:

$$|A\mathbf{x} - \mathbf{b}|_2^2 + |\Gamma\mathbf{x}|_2^2 \tag{2.6}$$

where Γ is a suitably chosen matrix called a Tikhonov matrix [50]. Usually, Γ is taken to be the identity $\Gamma = I$. An explicit solution to Eq. (2.6) is

$$\mathbf{x}^* = \left(A^T A + \Gamma^T \Gamma\right)^{-1} A^T \mathbf{b}, \tag{2.7}$$

and with $\Gamma = I$ the problem is usually formulated with a *regularization parameter* λ:

$$|A\mathbf{x} - \mathbf{b}|_2^2 + \lambda|\mathbf{x}|_2^2 \tag{2.8}$$

which is commonly known as *Ridge regression* since the parameter λ makes a "ridge" along the diagonal of $A^T A$.

Other regularizations are possible. In particular, we can take a different p-norm in the regularization term. A common choice is the 1-norm, producing the *Least Absolute Selection and Shrinkage Operator* (LASSO) formulation [49]:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_1. \tag{2.9}$$

The multitude of methods that can be used to solve problems of type Eq. (2.9) and its constrained formulation

$$\begin{aligned} &\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2 \\ &\text{s.t. } |\mathbf{x}|_1 \leq t \end{aligned} \tag{2.10}$$

are discussed within [8], but we mention here that there are many solvers that can be proved to solve the problem to a specified accuracy with a number of operations that does not exceed a polynomial of the problem dimensions.

Although the RSS and LASSO formulations described above were for linear formulations of the objective function, nonlinear formulations exist, one such can be seen in Chap. 6. In general, these problems belong to a class of problems known as *Convex* optimization. We classify a *convex* optimization problem as one in which the objective and constraint functions are convex, i.e., they satisfy the inequalities

$$\begin{aligned} f(\alpha\mathbf{x}+\beta\mathbf{y}) &\leq \alpha f(\mathbf{x})+\beta f(\mathbf{y}) \quad \text{and} \\ g_i(\alpha\mathbf{x}+\beta\mathbf{y}) &\leq \alpha g_i(\mathbf{x})+\beta g_i(\mathbf{y}), \quad i=1,\ldots,m_{\text{leq}}, \\ h_i(\alpha\mathbf{x}+\beta\mathbf{y}) &\leq \alpha h_i(\mathbf{x})+\beta h_i(\mathbf{y}), \quad i=1,\ldots,m_{\text{eq}} \end{aligned} \tag{2.11}$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and all $\alpha, \beta \in \mathbb{R}$ with $\alpha+\beta=1$, $\alpha \geq 0$, $\beta \geq 0$.

Most, if not all, metamaterial design problems will have nonlinear objective functions, and, when applicable, nonlinear constraints that unfortunately do not satisfy Eq. (2.11) everywhere in their domains. Fortunately though, many problems will have the property that Eq. (2.11) will be satisfied *locally* everywhere. That is, for any point $\mathbf{x}$ in the domain of f, there is a hypersphere about $\mathbf{x}$ where Eq. (2.11) is satisfied (although the α and β will be dependent upon the point $\mathbf{x}$). Such functions are called *locally convex*.

Unfortunately, the absence of global convexity limits the capability of most algorithms to guarantee finding the global minimum of Eq. (2.4). The best most algorithms can achieve is to find a local solution to the problem.

Techniques for solving Eq. (2.4) comprise two types: those that utilize gradient information and those that do not. Recall that a function has C^k smoothness if it is differentiable and its derivative is C^{k-1} smooth. This recursive definition starts with the class C^0, the continuous functions.

2.4 Algorithms Utilizing Gradient Information

We first discuss methods utilizing gradient information that are targeted for optimization problems with no constraints.

2.4.1 Unconstrained Nonlinear Optimization

To find the solution, $\mathbf{x}^*$, to Eq. (2.4) in the case where $\Omega = \mathbb{R}^n$ (i.e., an unconstrained problem), we must satisfy the *second order optimality conditions* [12]:

1. (necessity) If $\mathbf{x}^*$ is a local solution to Eq. (2.4), then $\nabla f(\mathbf{x}^*)=0$ and $\nabla^2 f(\mathbf{x}^*)$ is positive definite.
2. (sufficiency) If $\nabla f(\mathbf{x}^*)=0$ and $\nabla^2 f(\mathbf{x}^*)$ is positive definite, then there exists an $\alpha > 0$ such that $f(\mathbf{x}) \geq +\alpha\|\mathbf{x}-\mathbf{x}^*\|$ for all $\mathbf{x}$ near $\mathbf{x}^*$.

Satisfying these conditions only guarantees a *local* optimum for the general case. Most algorithms used to find solutions are iterative and take the form of Algorithm 2.1:

Algorithm 2.1: General iterative algorithm

input: Objective function f, initial point x_0

repeat

  Determine a descent direction $\mathbf{d}_k$

  Determine a step length α_k

  Update Candidate $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.

until $\nabla f(\mathbf{x}) \approx 0$

This is a consistent meme in solving mathematical optimization problems: from your current solution estimate, choose a better candidate and continue until the optimality conditions are satisfied. Algorithms for computing solutions to Eq. (2.4) differ in how they select descent directions $\mathbf{d}_k$ and step sizes α_k. We now discuss some possibilities for both.

2.4.1.1 Descent

Two methods for selection of a descent direction are:

1. Steepest Descent
2. Conjugate Gradient

The *steepest descent*, or gradient descent, algorithms choose descent directions $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$ based on the idea that f decreases fastest in the direction of its negative gradient. Unfortunately, due to the iterative nature of Algorithm 2.1, gradient descent's subsequent iterations may undo some minimization progress made on previous descents. To combat this, the *conjugate gradient* algorithm selects successive descent directions in a conjugate direction to previous descent directions. At iteration k, one evaluates the current negative gradient vector $-\nabla f(\mathbf{x}_k)$ and adds to it a linear combination of the previous descent iterates to obtain a new conjugate direction along which to descend. Initially, the descent is in the direction of the negative gradient, but each subsequent step moves in a direction that modifies the negative of the current gradient by a factor of the previous direction. The CG algorithm is shown in Algorithm 2.2.

Different Conjugate Gradient methods correspond to different choices for the scalar β_k. Three of the best known versions are:

- Fletcher–Reeves: $\beta_k^{\text{FR}} = \frac{\mathbf{s}_k^T \mathbf{s}_k}{\mathbf{s}_{k-1}^T \mathbf{s}_{k-1}}$
- Polak–Ribiére: $\beta_k^{\text{PR}} = \frac{\mathbf{s}_k^T (\mathbf{s}_k - \mathbf{s}_{k-1})}{\mathbf{s}_{k-1}^T \mathbf{s}_{k-1}}$
- Hestenes–Stiefel: $\beta_k^{\text{HS}} = \frac{\mathbf{s}_k^T (\mathbf{s}_k - \mathbf{s}_{k-1})}{\mathbf{d}_{k-1}^T (\mathbf{s}_k - \mathbf{s}_{k-1})}$

for a full list, consult [18].

Algorithm 2.2: Nonlinear conjugate gradient

input: Objective function f, initial point x_0

$\mathbf{d}_0 = -\nabla f(\mathbf{x}_0)$

(Line Search) $\alpha_0 = \arg\min_\alpha f(\mathbf{x}_0 + \alpha \mathbf{d}_0)$

$\mathbf{x}_1 = \mathbf{x}_0 + \alpha \mathbf{x}_0$ **repeat**

  Determine steepest direction $\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$

  Determine the scalar β_k (see below)

  Update the conjugate direction $\mathbf{d}_k = \mathbf{s}_k + \beta_k \mathbf{d}_{k-1}$

  Determine a step length (Line search) $\alpha_k = \arg\min_\alpha f(\mathbf{x}_k + \alpha \mathbf{d}_k)$

  Update Candidate $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.

until $\nabla f(\mathbf{x}) \approx 0$

2.4.1.2 Step Length

Having a descent direction, we must now determine how far along that direction to move for the next iterate. Ideally, we would move a length α along the line where α solves

$$\min_\alpha f(\mathbf{x}_k + \alpha \mathbf{d}_k), \tag{2.12}$$

i.e., the distance that minimizes the objective function in the direction $\mathbf{d}_k$. Notice that this is a one dimensional optimization problem in α. Finding an optimal solution to this problem would imply a method of solving the original nonlinear optimization problem! Therefore, instead of solving (2.12), we seek an efficient way of computing an acceptable α that guarantees that Algorithm 2.1 will converge to a $\mathbf{x}^*$.

To do this, we must find an α satisfying the following two conditions:

$$\begin{aligned} f(\mathbf{x}_k + \alpha \mathbf{d}_k) &\leq f(\mathbf{x}_k) + c_1 \alpha \mathbf{d}_k^T \nabla f(\mathbf{x}_k), \\ \mathbf{d}_k^T \nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k) &\geq c_2 \mathbf{d}_k^T \nabla f(\mathbf{x}_k) \end{aligned} \tag{2.13}$$

with $0 < c_1 < c_2 < 1$. The first condition is known as the *Armijo rule*. It ensures that the step length decreases f sufficiently for this iteration. The second condition is known as the *curvature condition*. It ensures that the slope of f has been reduced sufficiently for this iteration. Unfortunately, these two conditions may result in an α that is not close to an actual minimum of (2.12). Therefore, we modify the curvature condition to include

$$\left| \mathbf{d}_k^T \nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k) \right| \leq c_2 \left| \mathbf{d}_k^T \nabla f(\mathbf{x}_k) \right|, \tag{2.14}$$

and this ensures that α will lie close to a minimum critical point of Eq. (2.12). These three conditions taken together form the *Strong Wolfe conditions* [12] and are a prerequisite to any step length determination algorithm. Many methods exist for solving the general unconstrained problem, but they all utilize an algorithm similar to Algorithm 2.1 in their strategy.

2.4.1.3 Quasi-Newton Methods

In general, methods that utilize gradient information seek to find a stationary point of f by finding a zero of the gradient ∇f. A general class of methods, *quasi-Newton methods,* seek to do this by using Newton's method to find a root of ∇f. The underlying assumption in these methods is that the function f can locally be approximated by a quadratic.

Regular Newton's method updates candidate solutions at each iteration via

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[\nabla^2 f(\mathbf{x}_k)\right]^{-1} \nabla f(\mathbf{x}_k)$$

where $\nabla^2 f(\mathbf{x})$ denotes the Hessian, or the second derivative of f. Updates can be very expensive since we must find the inverse of an $n \times n$ matrix at every iteration. To ease computational cost, approximations to the Hessian and its inverse are used. There are multiple ways the Hessian can be approximated, one method that is extensively employed is from the Broyden family which uses a convex combination of Daviodon–Fletcher–Powell [14] and BFGS [45] updates. An extensive survey of Quasi-Newton methods may be found in [40].

2.4.2 Constrained Nonlinear Optimization

When dealing with the general form of Eq. (2.4), i.e., when the constraints exist, the first question to answer is how to ascertain if a candidate $\mathbf{x}^*$ is indeed a solution.

First, we define a constraint g_i to be *active* (resp., *inactive*) at a point $\mathbf{x}$ if $g_i(\mathbf{x}) = 0$ (resp., $g_i(\mathbf{x}) < 0$). (Note, equality constraints are always active.) We define the *active set* at $\mathbf{x}$, $\mathcal{A}(\mathbf{x})$, as the indices of those constraints $g_i(\mathbf{x})$ that are active at the given point. For a given candidate solution, $\mathbf{x}_k$, if no constraints are active, then the necessary and sufficient conditions are the same as for the unconstrained case. In the case where the candidate lies on the boundary of the feasible set (i.e., at least one constraint is active), the second order optimality conditions for the unconstrained case do not apply because the direction of the negative gradient (or even a descent direction in a conjugate direction) will push the next iterate into the infeasible set.

We will specify the optimality conditions for a solution $\mathbf{x}^*$ to solve Eq. (2.4) through the use of a Lagrangian function:

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \sum_{i=1}^{m_{\text{leq}}} \lambda_i g_i(\mathbf{x}) + \sum_{i=1}^{m_{\text{eq}}} \lambda_i h_i(\mathbf{x})$$

where $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{m_{\text{leq}}})$ and $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_{m_{\text{eq}}})$ are vectors called *KKT multipliers*. Now, if $\mathbf{x}^*$ is an optimal solution to Eq. (2.4), then there exist KKT multipliers

$\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that

$$\begin{aligned}
&\nabla f\left(\mathbf{x}^*\right) + \sum_{i=1}^{m_{\text{leq}}} \lambda_i^* \nabla g_i\left(\mathbf{x}^*\right) + \sum_{i=1}^{m_{\text{eq}}} \mu_i^* \nabla h_i\left(\mathbf{x}^*\right) = 0, \\
&g_i\left(\mathbf{x}^*\right) \leq 0 \quad \text{for } i = 1, \ldots, m_{\text{leq}}, \\
&h_i\left(\mathbf{x}^*\right) = 0 \quad \text{for } i = 1, \ldots, m_{\text{eq}}, \\
&\lambda_i^* \geq 0 \quad \text{for } i = 1, \ldots, m_{\text{leq}}, \\
&\mu_i^* \geq 0 \quad \text{for } i = 1, \ldots, m_{\text{eq}}, \\
&\lambda_i^* g_i\left(\mathbf{x}^*\right) = 0 \quad \text{for } i = 1, \ldots, m_{\text{leq}}.
\end{aligned} \tag{2.15}$$

The above conditions are known as the *Karush–Kuhn–Tucker* conditions (KKT conditions) [8]. Points that satisfy them are critical points of the original problem. To determine if these critical points are indeed solutions of Eq. (2.4), we impose second order conditions on the points (for they could be a maximizer or a saddle point).

Before stating the second order sufficient and necessary conditions, we first define the *tangent space* for feasible points $\bar{\mathbf{x}}$

$$T = \left\{\mathbf{v} : \nabla g_j(\bar{\mathbf{x}})\mathbf{v} = 0 \; \forall j \in \mathcal{A}(\bar{\mathbf{x}}), \; \nabla h(\bar{\mathbf{x}})\mathbf{v} = 0\right\}$$

where $\mathcal{A}(\bar{\mathbf{x}})$ denotes the active set.

For a KKT point, we also define the *relaxed tangent space*

$$T' = \left\{\mathbf{v} : \nabla g_j(\bar{\mathbf{x}})\mathbf{v} = 0 \; \forall j \in \{j : \lambda_j > 0\}, \; \nabla h(\bar{\mathbf{x}})\mathbf{v} = 0\right\}.$$

Having these definitions, we now state the second order necessary and sufficient conditions for a feasible candidate $\mathbf{x}^*$ with KKT multipliers $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ satisfying Eq. (2.15) to be a solution to Eq. (2.4):

$$\mathbf{w}^T \nabla_x L^2\left(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\right)\mathbf{w} > 0 \quad \forall \mathbf{w} \in T', \; \mathbf{w} \neq \mathbf{0}. \tag{2.16}$$

Methods for finding a suitable optimum satisfying Eqs. (2.15) and (2.16) for constrained optimization problems are ubiquitous. We focus on two categories:

1. Primal methods
2. Penalty and Barrier Methods

We will briefly describe each type below.

2.4.2.1 Primal Methods

Primal methods are those that solve Eq. (2.4) by starting with a candidate in the feasible set Ω and searching only the feasible set for an optimal solution. The main

characteristics of these algorithms is that they find new candidates that simultaneously decrease the objective function at each step, while remaining feasible. To update a given candidate $\mathbf{x}_k$, a vector $\mathbf{d}_k$ is chosen such that it is both descending and feasible. The following must hold for $\mathbf{d}_k$ to be a feasible direction:

$$\nabla f(\mathbf{x})^T \mathbf{d}_k < 0, \tag{2.17}$$

$$\nabla g_i(\mathbf{x})^T \mathbf{d}_k < 0, \tag{2.18}$$

$$\nabla h_i(\mathbf{x})^T \mathbf{d}_k = 0. \tag{2.19}$$

Equation (2.17) implies that we are descending, and Eqs. (2.18) and (2.19) imply that we are increasing feasibility (by moving in the direction tangential to the active set for the inequality constraints and parallel for the equality constraints).

Feasible direction methods suffer from requiring a feasible initial candidate, from situations where no feasible descent direction exists, and may be subject to jamming, or oscillations that prevent convergence of the algorithm [12].

Gradient projection methods are motivated from steepest descent algorithms in unconstrained optimization. Their basic idea is to take the negative of the gradient of the objective function and project it onto the *working surface* in order to determine a feasible descent direction. The working surface is the subset of the constraints that are currently active, i.e., the current active set.

Thus, at the current feasible point, one determines the active constraints and projects the negative gradient of the objective function onto the subspace tangent to the surface determined by these constraints. However, this may not be a feasible direction since the working surface may be curved. To deal with curvature, one searches for a feasible descent direction along an embedded curve within the constraint surface.

2.4.2.2 Penalty and Barrier Methods

Penalty and Barrier methods attempt to approximate constrained optimization problems with those that are unconstrained, and then apply standard unconstrained search techniques to obtain solutions. Penalty methods do this by adding a term to the objective function that penalizes violation of the constraints with a large factor. In the case of barrier methods, a term is added that favors points in the interior of the feasible region and penalizes those closer to the boundary.

The idea for penalty methods is to replace Eq. (2.4) with an unconstrained problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) + \beta\sigma(\mathbf{x}) \tag{2.20}$$

where $\beta > 0$ and $\sigma : \mathbb{R}^n \to \mathbb{R}$ is a function satisfying

1. $\sigma(\mathbf{x})$ is continuous;
2. $\sigma(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$;
3. $\sigma(\mathbf{x}) = 0 \Leftrightarrow g_i(\mathbf{x}) \leq 0,\ h_j(\mathbf{x}) = 0\ \forall i = 1, \ldots, m_{\text{leq}},\ j = 1, \ldots, m_{\text{eq}}$, i.e., $\mathbf{x}$ is feasible.

That is, we set up an unconstrained optimization problem where we generate a new objective function that greatly increases in value as $\mathbf{x}$ moves out of the feasible region. A standard choice for $\sigma(\mathbf{x})$ is the *quadratic loss function* [26]:

$$\sigma(\mathbf{x}) = -r \sum_{i=1}^{m_{\text{leq}}} \max\big(0, g_i(\mathbf{x})\big) + \frac{1}{r} \sum_{i=1}^{m_{\text{eq}}} \big(h_i(\mathbf{x})\big)^2. \tag{2.21}$$

For $\mathbf{x}$ values inside the feasible region, $g_i(\mathbf{x}) \leq 0$ and $h_i(\mathbf{x}) = 0$, giving a value of $\sigma = 0$. When $\mathbf{x}$ is outside of the feasible region, some of the $g_i > 0$ or $h_i \neq 0$, we begin to be penalized. To implement a penalty method, one needs to select a value for β. Standard techniques start with a relatively small value (and an infeasible point for $\mathbf{x}_0$) and monotonically increase β, solving subsequent unconstrained optimization problems (one for each β) and utilizing these intermediate solutions as the initial guess for the next problem. This *graduated optimization* method produces a sequence of solutions that converge to an optimal solution of the original constrained problem. Graduated optimization is a technique commonly used with hierarchical pyramid methods for matching objects within images [9].

Barrier methods are implemented when one does not wish to compute $f(\mathbf{x})$ outside of the feasible region. Thus, we would not be able to utilize a penalty function like Eq. (2.21). Instead, a selection would need to be made that was defined to converge for feasible points. A possible selection for problems with no equality constraints might be

$$\sigma(\mathbf{x}) = r \sum_{i=1}^{m} \frac{-1}{g_i(\mathbf{x})} \tag{2.22}$$

where $r > 0$ is the *barrier parameter*. As candidates get closer to the boundary of the feasible region, the value of the objective function becomes larger. The idea is to start with a feasible point and a relatively large value of the barrier parameter, preventing the candidates from nearing the boundary of the feasible set. Techniques then decrease the value of the barrier parameter monotonically until an optimum value for the original problem is achieved. Note that barrier methods require a feasible point from which to start. This can sometimes be difficult to find. Also, barrier methods do not work with equality constraints without cumbersome modifications to this basic approach, and by not allowing the method to ever leave the feasible region, much more computational effort is (usually) required.

Penalty methods are sometimes referred to as *external* methods since their augmented objective functions tend to utilize solutions in the exterior of the feasible region. Analogously, barrier methods are sometimes called *interior point* methods, for the opposite reason. There is a vast and vigorous field of research surrounding

these methods, and we suggest utilizing the references to find current implementations. A great start would be [26].

These two types of methods are among the most powerful for attacking the general scalar problem in Eq. (2.4). Of the two, exterior methods are preferable (when applicable) as they can deal with equality constraints, they do not require a feasible starting point, and their computational effort is substantially lower than for the interior methods.

2.5 Gradient-Free Algorithms

Looking at the form of Eq. (2.4), we denote f as a function, and this is typically seen as an analytical expression. Most industrial applications of the general problem may involve formulations that do *not* encode f analytically, but have some type of *black box* that computes values of $f(\mathbf{x})$. That is, given a value $\mathbf{x}$, there is some process (numerical simulation, physical experiment, etc.) that computes the output $f(\mathbf{x})$. Furthermore, the constraint functions may also be black-box functions. Typically, these black-box functions will not have any derivative information associated with them (although in rare occasions, there may be derivative information available via another black-box function). In these cases, f is expensive to calculate in terms of time, and methods that require many evaluations of f rapidly become infeasible to use in many applications. In particular, to produce viable step lengths satisfying the Strong Wolfe conditions in Eq. (2.14), hundreds of function evaluations may be required per iterate.

Moreover, when evaluating the objective function via a numerical simulation or physical experiment, inaccuracies may arise in the value that f takes at a given point. This generates many difficulties approximating derivatives via finite differences. This line of thinking dismisses the use of many of the techniques from Sect. 2.4. Even in cases where derivative information is available, function inaccuracies adversely effect most of these methods [15].

2.5.1 Direct Methods

Direct methods are those that attempt to solve the general problem *directly* by utilizing objective function values. Here, we introduce a number of methods starting with a variant of gradient descent for the derivative-free case.

2.5.1.1 Coordinate Descent

Perhaps the simplest method to solve an unconstrained version of Eq. (2.4) without using gradients is to do successive line searches in each coordinate direction for each

iteration. That is, one does a line search in a coordinate direction for each iteration, changing coordinates for each, and looping cyclically as the number of dimensions are reached. This process is called *Coordinate Descent* (CD). Iterations of a cycle of line search in all coordinate directions is equivalent to one gradient descent direction, but the number of function evaluations may prove to be prohibitive.

More efficient algorithms have been constructed in an attempt to limit the number of function evaluations made to reach convergence. In particular, choosing a random direction to do line search for each iteration, the so-called *Random Coordinate Descent,* was shown to converge, on average, in fewer iterations than CD [36, 42]. In general, one seeks an appropriate coordinate system where CD would operate optimally. The *Adaptive coordinate descent* algorithm [24] gradually builds a transformation of the coordinate system such that the new coordinates are as decorrelated as possible with respect to the objective function.

Instead of finding a pointwise trajectory to the minimum, other techniques attempt to locate a set wherein the optimal solution resides. The oldest and most famous of these is the simplex algorithm.

2.5.1.2 Nelder–Mead Simplex Algorithm

The Nelder–Mead (NM) algorithm [35] solves the general problem by containing the solution within a *simplex.* A simplex is the generalization of a polygon to n dimensions. The NM algorithm starts with a set of points in $\mathbb{R}^n$ forming a simplex and at each iteration, the objective function is evaluated at the vertices of the simplex.

The algorithm replaces the worst point on the simplex with a point reflected through the centroid of the remaining n points. If this point is better than the best current point, then the simplex is stretched exponentially out along this line. If not, then the simplex stretches across a valley, so the simplex is shrunk towards a (hopefully) better point. A few of the other means of replacing the chosen point include: reflection, expansion, inside and outside contractions.

The Nelder–Mead algorithm remains popular, mostly through its simplicity, but McKinnon [33] established analytically that convergence can occur to points with $\nabla f(\mathbf{x}) \neq 0$, even when the function is convex and twice continuously differentiable. Tseng [54] proposed a globally convergent simplex-based search method that considers an expanded set of candidate replacement points (besides those listed above). Other modifications are presented in [13].

2.5.1.3 Mesh Adaptive Direct Search (MADS)

The Mesh Adaptive Direct Search (MADS) [3] is a generalization of several existing direct search methods [25, 51–53]. MADS was introduced to extend direct search methods to deal with the constrained problem in Eq. (2.4), while improving both the practical and theoretical convergence results seen in previous methods.

MADS handles constraints $x \in \Omega$ by the so-called *extreme barrier method*, which simply consists in rejecting any trial point which does not belong to Ω. The term *extreme barrier method* comes from the fact that this approach can be implemented by solving the unconstrained minimization of

$$f_\Omega(x) = \begin{cases} f(x) & \text{if } x \in \Omega, \\ \infty & \text{otherwise} \end{cases}$$

in place of Eq. (2.4). Note that this may impose severe discontinuities on the problem. A more subtle way of handling quantifiable constraints is presented in [4], and is summarized in Chap. 4.

Each MADS iteration proceeds as follows: Given a candidate solution $\mathbf{x}_k$, the SEARCH step produces a list of tentative trial points. Any mechanism can be used to create the list, as long as it contains a finite number of points located on a *conceptual mesh*. The conceptual mesh is defined by a *mesh parameter* $\Delta_k^M > 0$. This parameter, along with a finite set of positive spanning directions D, forms the mesh at iteration k:

$$M_k = \left\{ \mathbf{x} + \Delta_k^M \mathbf{d} : \mathbf{x} \in V_k, \mathbf{d} \in D \right\} \tag{2.23}$$

where V_k is a set containing all previous points where the objective function has been evaluated. A *positive spanning set* of $\mathbb{R}^n$ is a set $D = \{\mathbf{d}_1, \ldots, \mathbf{d}_m\}$ of vectors in $\mathbb{R}^n$ such that every vector in $\mathbb{R}^n$ is a linear combination of the $\mathbf{d}_i$ with nonnegative coefficients. Many methods exist for computing a set of points on the conceptual mesh: speculative search [21], Latin hypercube sampling [47], variable neighborhood searches [11], surrogates, and many others [48].

Having an initial set of points, the objective function is evaluated at each of the points until either a better candidate than $\mathbf{x}_k$ is found, or all of the points are evaluated. In the latter case, a POLL step is implemented that conducts a local exploration near the candidate point. Following an unsuccessful SEARCH step, the POLL step generates a list of mesh points near the incumbent x_k. The term *near* is tied to the so-called *poll size parameter* $\Delta_k^p > 0$. Similar to the SEARCH step, the POLL step may be interrupted as soon as an improvement point over the candidate is found.

Parameters are updated at the end of each iteration. There are two possibilities: If either the SEARCH or the POLL step generated a mesh point $\mathbf{p} \in M_k$ which is better than $\mathbf{x}_k$, then the candidate point $\mathbf{x}_{k+1}$ is set to $\mathbf{p}$ and both the mesh size and poll size parameters are increased or kept to the same value. For example, $\Delta_{k+1}^M \leftarrow \min\{1, 4\Delta_k^M\}$ and $\Delta_{k+1}^p \leftarrow 2\Delta_k^p$. Otherwise, $\mathbf{x}_{k+1}$ is set to $\mathbf{x}_k$ and the poll size is decreased and the mesh size parameter decreased or kept the same. For example, $\Delta_{k+1}^m \leftarrow \min\{1, \frac{1}{4}\Delta_k^m\}$ and $\Delta_{k+1}^p \leftarrow \frac{1}{2}\Delta_k^p$. At any iteration of the MADS algorithm, the poll size parameter Δ_k^p must be greater than or equal to the mesh size parameter Δ_k^M. Termination conditions arise when either the poll parameter matches the mesh size parameter or a predefined number of iterations have been reached.

2.5.2 Surrogate Methods

As mentioned above, surrogates may be used to determine a set of points for use in the SEARCH step for direct search. These methods build a model interpolating between the known points stored in V_k. This section looks at methods that do not restrict themselves to interpolation with a local search; rather, they utilize a *global* surrogate function to assist in the optimization.

There are many ways of employing surrogates. In particular, there is a standard engineering process [5] for using them:

1. Choose a surrogate s for the objective function f that is either
   (a) A simplified model of f (as is used in Chap. 6) or
   (b) A *response surface* of f generated from a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_q$ where f takes a finite value;
2. Minimize over the surrogate s, obtaining a candidate point $\mathbf{x}^s$;
3. Evaluate the objective function at $\mathbf{x}^s$ and repeat the process.

In cases where we do not have a simplified model for f and wish to generate a response surface (or metamodel) $\hat{f}$, the question arises as to which method to use. Barton [6] enumerates a list, including splines, radial basis functions, kernel smoothing, spatial correlation models, and frequency domain approaches. Regardless of the method employed, its quality depends crucially upon choosing an appropriate sampling technique [39]. The remainder of this subsection describes a state of the art response surface methodology known as *Gaussian Process Regression*. We will see its implementation in Chap. 3.

2.5.2.1 Gaussian Process Regression

Gaussian Process Regression (GPR) [41] is also known as Kriging prediction, Kolmogorov–Wiener prediction, or best linear unbiased prediction. It is a technique for estimating the objective function value at a new point $\mathbf{x}_*$ utilizing noisy observations $f(\mathbf{x})$ at points $\mathbf{x}_1, \ldots, \mathbf{x}_m$. The surrogate is a process that generates data such that any finite subset follows a multivariate Gaussian distribution.

A typical assumption for the surrogate is that the mean of the data is zero everywhere (if not, we can subtract the mean and work with the transformed dataset). Then, pairs of points in GPR are related to each other by the *covariance function*. A popular choice is the *squared exponential*:

$$k(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 \exp\left[\frac{-\|\mathbf{x}_p - \mathbf{x}_q\|_2^2}{2L^2}\right] \tag{2.24}$$

where the maximum allowable covariance is σ_f^2.

Note, that the covariance between the outputs is written as a function of the inputs. For this particular covariance function, we see that the covariance is almost

maximal between variables whose corresponding inputs are very close, and decreases as their distance in the input space increases. The covariance function has a characteristic length scale L, which informally can be thought of as roughly the distance you have to move in input space before the function value can change significantly. Alternatively, this relates how much influence distant points will have on each other.

We create the *covariance matrix* between all pairs of points

$$K(\mathbf{X},\mathbf{X}) = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2) & \dots & k(\mathbf{x}_1,\mathbf{x}_m) \\ k(\mathbf{x}_2,\mathbf{x}_1) & k(\mathbf{x}_2,\mathbf{x}_2) & \dots & k(\mathbf{x}_2,\mathbf{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_m,\mathbf{x}_1) & k(\mathbf{x}_m,\mathbf{x}_2) & \dots & k(\mathbf{x}_m,\mathbf{x}_m) \end{bmatrix}. \tag{2.25}$$

Observations from the data are often noisy, for a various number of reasons. As is typical in most regression schemes, we model the observations as

$$y = f(\mathbf{x}) + \mathcal{N}\big(0,\sigma_\nu^2\big),$$

and the covariance between two points becomes

$$\operatorname{cov}(y_p,y_q) = k(\mathbf{x}_p,\mathbf{x}_q) + \sigma_\nu^2\delta_{pq} \quad \text{or} \quad \operatorname{cov}(\mathbf{y}) = K(\mathbf{X},\mathbf{X}) + \sigma_\nu^2 I, \tag{2.26}$$

where δ_{pq} is the Kronecker delta function which is 1 when $p = q$ and 0 otherwise. Here, I is the $m \times m$ identity matrix.

The purpose of generating the surrogate is to predict values of the observables at previously unseen points. The assumptions underpinning GPR state that the joint distribution of the observed data and unknown data point $\mathbf{x}_*$ is given by:

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X},\mathbf{X}) + \sigma_\nu^2 I & K(\mathbf{X},\mathbf{x}_*) \\ K(\mathbf{x}_*,\mathbf{X}) & K(\mathbf{x}_*,\mathbf{x}_*) \end{bmatrix}\right). \tag{2.27}$$

where y_* denotes the value of the surrogate at the unseen point $\mathbf{x}_*$. We seek the conditional probability $p(y_*|\mathbf{y})$, or "how likely is a certain prediction for y_* given the data?" As derived in [41], this probability follows the distribution

$$p(y_*|\mathbf{y}) \sim \mathcal{N}\big(K_* K^{-1}\mathbf{y},\, K_{**} - K_* K^{-1} K_*^T\big) \tag{2.28}$$

where T denotes transposition and we use the short hand notation of K being the covariance matrix, $K_* = [k(\mathbf{x}_*,\mathbf{x}_1)\ k(\mathbf{x}_*,\mathbf{x}_2)\ \dots\ k(\mathbf{x}_*,\mathbf{x}_m)]$ and $K_{**} = k(\mathbf{x}_*,\mathbf{x}_*)$.

Thus, the best estimate for y_* is the mean of this distribution

$$y_* = K_*\big(K + \sigma_\nu^2\big)^{-1}\mathbf{y}, \tag{2.29}$$

and the uncertainty is captured in the variance

$$\operatorname{var}(y_*) = K_{**} - K_*\big(K + \sigma_\nu^2\big)^{-1} K_*^T. \tag{2.30}$$

We note here for completeness that if our original data set did not have zero mean, but instead had mean $\mathbf{m}(\mathbf{X})$, then Eq. (2.29) would become

$$y_* = m(\mathbf{x}_*) + K_*\left(K + \sigma_\nu^2\right)^{-1}\left(\mathbf{y} - \mathbf{m}(\mathbf{X})\right) \tag{2.31}$$

where $m(\mathbf{x}_*)$ denotes the mean of the new data. The variance remains unchanged from Eq. (2.30).

For actual implementations of the above equations, we need to determine values for the parameters σ_f, L, σ_ν. This collection of parameters are referred to as *hyperparameters*. Most methods for determining the hyperparameters from data attempt to optimize the marginal likelihood of $p(y_*|\mathbf{y})$ with respect to the hyperparameters, given the data. This is itself a rich and interesting optimization problem having a long history in spatial statistics [31].

2.5.3 Stochastic Search Algorithms

In the above formulations, some assumption about the smoothness of the function, or continuity of the function is made. This is manifested either in the direct usage of gradients or in methods like the polling step of direct search, where shrinking the polling step parameter is assumed to lead to a better solution.

Sometimes, functions are not continuous for large swaths of the space over which we seek to optimize. This section presents approaches that rely on non-deterministic algorithmic steps. This is a more delicate way of saying that the algorithms "guess" which direction to search for a better candidate solution. Most algorithms of this type have a heuristic for choosing how to "guess." Some approaches occasionally allow new candidates that are "worse" (in terms of the objective function) than the current solution; the idea being that accepting a worse candidate at this iteration will lead to a better overall solution as the algorithm iterates. This idea allows the algorithm to theoretically find global solutions. The literature on stochastic algorithms is very extensive, especially on the applications side, since their implementation is rather straightforward compared to deterministic algorithms. See, for example, [22, 46, 58] for a general overview.

2.5.3.1 Random Search

The simplest algorithm of this type is random search. Random algorithms compare the current iterate $\mathbf{x}$ with a randomly generated candidate (no heuristic). The current iterate is updated only if the candidate is a better point (in terms of the objective function). The determination of new candidates is based on two random components: A direction $\mathbf{d}$ is generated using a uniform distribution over the unit sphere in $\mathbb{R}^n$, and a step α is generated from a uniform distribution over the set of steps S in a way that $\mathbf{x} + \alpha\mathbf{d}$ is feasible. Bélisle et al. [10] generalized these types of algorithms

by allowing arbitrary distributions to generate both the direction **d** and step α, and proved convergence to a global optimum under mild conditions for continuous optimization problems. Unfortunately, the number of function evaluations for this type of method become prohibitive.

2.5.3.2 Genetic Algorithms

In an effort to chose points with less randomness than simply guessing, *Genetic Algorithms* (GA) were originally introduced by Holland [20] wherein a method was designed that mimics the process of natural evolution.

The GA operates on a population of individuals that are each represented by a chromosome **x**. Initially, a random population is chosen and the objective function is evaluated on each member. The better performing members are chosen to mate and form a new generation, mimicking the process of natural selection. A mating pool is first formed by either sorting the population according to objective function value and then keeping the top performing members, or by using a threshold such as the mean or the median cost to eliminate any population members with a worse performance than the threshold value. Members in the mating pool are eligible for breeding. For each new solution to be produced, a pair of "parent" solutions is selected from the mating pool. These parents produce a "child" solution using *crossover*, creating a new solution which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated.

After selection and crossover have been performed to fill out the population for the next generation, a small percentage of elements in the new population are mutated in order to continue exploring new parts of the parameter space. If an individual is randomly selected for mutation, then its value is given a new random value within its allowed range. Typical mutation probabilities are on the order of a few percent, and different distributions are employed for new variates.

The final step in populating the new generation is to optionally enforce *elitism*. Elitism ensures that the best global fitness is maintained between generations by copying the chromosome with the best fitness from the previous generation into the new population. At this point, the new population is ready to be evaluated by the fitness function.

Different crossover methods and Nature-Inspired Optimization routines, including Genetic Algorithms, will be discussed in detail in Chap. 5.

2.5.3.3 Non-dominated Sorting Genetic Algorithm

We now introduce a method that attempts to produce the Pareto front for a general multi-objective optimization problem. We will see that Chap. 3 generates such a problem, and here we discuss the method used to solve it. This method, Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II) [27] assumes our multi-objective function has k dimensions.

Again, we adopt the general idea of a genetic algorithm, but with some changes. The algorithm starts with a random parent population P of size N. Binary tournament selection, recombination, and mutation operators are used to create a child population of P of size N. We combine the parent and children populations then sort them via the principle of *non-domination*. An element $\mathbf{p} \in P$ dominates another element $\mathbf{q} \in P$ if there is an i with $p_i < q_i$ and $p_j \leq q_j$ for all other j. Here, the ith element of $\mathbf{p}$, denoted p_i represents the ith objective value for this population element.

Each solution is assigned a fitness equal to its non-domination level. Those elements with no dominating elements are given fitness 1. Those elements only dominated by elements with fitness $= 1$ are given fitness 2, etc. For each fitness level, we sort the elements in that level via *crowding comparison.* To do so, we first find the *local crowding distance* for each element. This distance is calculated by finding the average distance of the two nearest neighbors to this point along each of the objective axes.

We sort within each fitness level, giving preference to those solutions that are "more spread out," i.e., have a larger crowding distance. The new population is then generated by taking the first N elements of the sorted fitness levels. The process repeats itself (children are generated, combined with parents, sorted via non-domination, etc.) until either all elements of the population have fitness level 1 or a predetermined number of iterations are reached.

2.6 Summary

We have described a range of mathematical optimization problems and their respective solution techniques. Methods that utilize derivative information, both for constrained and unconstrained problems, were briefly introduced. These methods, combined with parametrized models of metamaterial structures to be simulated, are too often trapped in numerous local minima. As a result, their usefulness for metamaterial design is minimal, and they will not be covered further in the text.

Many methods for solving problems that do not take advantage of derivative information, either because it does not exist or is not available, were also discussed. These techniques, which will be covered over the next three chapters, are well-established methods of optimization. They are all robust against non-smooth optimization surfaces, and coincidentally are all direct search methods. Additionally, both Mesh Adaptive Direct Search in Chap. 4, and Nature Inspired Optimization in Chap. 5 work efficiently in high dimensions.

The last two chapters of the book do not focus solely on the optimization method itself. These chapters integrate both optimization routines with novel methods for calculating and representing the shapes of the individual resonant structures within a metamaterial. These approaches are both gradient-based, but they are able to circumvent the normal pitfalls of gradient-based optimization by transforming the space over which the optimization occurs. Both techniques are new to the field of

metamaterial design; however, their applicability extends far beyond the focus of this book. This is clearly illustrated by the range of design examples that are covered throughout the last two chapters.

References

1. N. Andr'easson, A. Evgrafov, M. Patriksson, *An Introduction to Continuous Optimization: Foundations and Fundamental Algorithms*. Studentlitteratur (2005)
2. K.A. Atkinson, *An Introduction to Numerical Analysis*, 2nd edn. (Wiley, New York, 1988)
3. C. Audet, J.E. Dennis Jr., Mesh adaptive direct search algorithms for constrained optimization. SIAM J. Optim. **17**(2), 188–217 (2006)
4. C. Audet, J.E. Dennis Jr., A progressive barrier for derivative-free nonlinear programming. SIAM J. Optim. **20**(1), 445–472 (2009)
5. J.-F.M. Barthelemy, R.T. Haftka, Approximation concepts for optimum structural design—a review. Struct. Optim. **5**, 129–144 (1993)
6. R.R. Barton, Metamodeling: a state of the art review, in *Proceedings of the 1994 Winter Simulation Conference* (1994), pp. 237–244
7. D.P. Bertsekas, *Nonlinear Programming*, 2nd edn. (Athena Scientific, Belmont, 1999)
8. S.P. Boyd, L. Vandenberghe, *Convex Optimization* (Cambridge University Press, Cambridge, 2003)
9. P.J. Burt, Fast filter transformations for image processing. Comput. Graph. Image Process. **16**, 20–51 (1981)
10. H.E. Romeijn, C.J. Bélisle, R.L. Smith, Hit-and-run algorithms for generating multivariate distribution. Math. Oper. Res. **18**, 255–266 (1993)
11. V.B.C. Audet, S. Le Digabel, Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. J. Glob. Optim. **41**(2), 299–318 (2008)
12. E.K.P. Chong, S.H. Zak, *An Introduction to Optimization* (Wiley, New York, 1996)
13. A.R. Conn, K. Scheinberg, L.N. Vicentee, *Introduction to Derivative-Free Optimization* (SIAM, Philadelphia, 2009)
14. W.C. Davidon, Variable metric method for minimization. SIAM J. Optim. **1**, 1–17 (1991)
15. J.E. Dennis, H.F. Walker, Inaccuracy in quasi-Newton methods: local improvement theorems. Math. Program. Stud. **22**, 70–85 (1984)
16. R.T. Eckenrode, Weighting multiple criteria. Manag. Sci. **12**, 180–192 (1965)
17. G.F. Golub, C.F. Van Loan, *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences (1996)
18. W.W. Hager, H. Zhang, A survey of nonlinear conjugate gradient methods. Pac. J. Optim. **2**, 35–58 (2006)
19. B.F. Hobbs, A comparison of weighting methods in power plant siting. Decis. Sci. **11**, 725–737 (1980)
20. J.H. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, 1975)
21. R. Hooke, T.A. Jeeves, Direct search solution of numerical and statistical problems. J. ACM **8**, 212–229 (1961)
22. H.H. Hoos, T. Stützlew, *Stochastic Local Search: Foundations and Applications* (Morgan Kaufmann/Elsevier, San Mateo, 2004)
23. C.-L. Hwang, K. Yoon, Multiple attribute decision making methods and applications: a state-of-the-art survey, in *Lecture Notes in Economics and Mathematical Systems*, ed. by M. Beckmann, H.P. Kunzi (Springer, Berlin, 1981)
24. M. Schoenauer, I. Loshchilov, M. Sebag, Adaptive coordinate descent, in *Genetic and Evolutionary Computation Conference (GECCO)* (ACM Press, New York, 2011), pp. 885–892

25. J.E. Dennis Jr., V. Torczon, Direct search methods on parallel machines. SIAM J. Optim. **1**(4), 448–474 (1991)
26. P.A. Jensen, J.F. Bard, *Operations Research Models and Methods* (Wiley, New York, 2003)
27. S. Agarwal, K. Deb, A. Pratap, T. Meyarivan, Physical programming: effective optimization for computational design. IEEE Trans. Evol. Comput. **6**(2), 182–191 (2002)
28. J. Koski, R. Silvennoinen, Norm methods and partial weighting in multicriterion optimization of structures. Int. J. Numer. Methods Eng. **24**, 1101–1121 (1987)
29. D.G. Luenberger, *Linear and Nonlinear Programming* (Addison-Wesley, Reading, 1984)
30. H.D. Sherali, M.S. Bazaraa, C.M. Shetty, *Nonlinear Programming: Theory and Algorithms*, 2nd edn. (Wiley, New York, 1993)
31. K.V. Mardia, R.J. Marshall, Maximum likelihood estimation for models of residual covariance in spatial regression. Biometrika **71**(1), 135–146 (1984)
32. R.T. Marler, J.S. Arora, Survey of multi-objective optimization methods for engineering. Struct. Multidiscip. Optim. **26**, 369–395 (2004)
33. K.I.M. McKinnon, Convergence of the Nelder–Mead simplex method to a nonstationary point. SIAM J. Optim. **9**, 148–158 (1998)
34. A. Messac, Physical programming: effective optimization for computational design. AIAA J. **34**, 149–158 (1996)
35. J.A. Nelder, R. Mead, A simplex method for function minimization. Comput. J. **7**, 308–313 (1965)
36. Y. Nesterov, Efficiency of coordinate descent methods on huge-scale optimization problems. CORE Discussion Paper (2010)
37. J. Nocedal, S.J. Wright, *Numerical Optimization*. Springer Series in Operations Research (Springer, New York, 1999)
38. N. Otto, K.N. Otto, A formal representational theory for engineering design. Technical report, Ph.D. Thesis, California Institute of Technology (1992)
39. M.R. Kirby, P.A. Barros Jr., D.N. Mavris, Impact of sampling techniques selection on the creation of response surface models. SAE Trans., J. Aerosp. **113**, 1682–1693 (2004)
40. M. Papadrakakis, G. Pantazopoulos, A survey of quasi-Newton methods with reduced storage. Int. J. Numer. Methods Eng. **36**, 1573–1596 (1993)
41. C. Rassmussen, C. Williams, *Gaussian Processes for Machine Learning* (MIT Press, Boston, 2006)
42. M. Richtárik, P. Takáč, Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. arXiv:1107.2848 (2011)
43. M.J. Scott, Formalisms for negotiation in engineering design, in *The 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference*, 1996
44. M.J. Scott, E.K. Antonsson, Aggregation functions for engineering design tradeoffs, in *Fuzzy Sets and Systems* (1998), pp. 253–264
45. D. Shanno, Conditioning of quasi-Newton methods for function minimization. Math. Comput. **24**(111), 647–656 (1970)
46. J.C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control* (Wiley, New York, 2003)
47. M. Stein, Large sample properties of simulations using Latin hypercube sampling. Technometrics **29**, 143–151 (1987)
48. R.M. Lewis, T.G. Kolda, V. Torczon, Optimization by direct search: new perspectives on some classical and modern methods. SIAM Rev. **45**(3), 385–482 (2003)
49. R. Tibshirani, Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B **58**(1), 267–288 (1996)
50. A.N. Tikhonov, On the stability of inverse problems. Dokl. Akad. Nauk SSSR **39**(5), 195–198 (1943) (in Russian)
51. V. Torczon, Pattern search methods for nonlinear optimization. SIAG/OPT Views News **6**, 7–11 (1995)
52. V. Torczon, M.W. Trosset, From evolutionary optimization to parallel direct search: pattern search algorithms for numerical optimization. Comput. Sci. Stat. **29**, 396–401 (1998)

53. M.W. Trosset, I know it when I see it: toward a definition of direct search methods. SIAG/OPT Views News **9**, 7–10 (1997)
54. P. Tseng, Fortified-descent simplicial search method: a general approach. SIAM J. Optim. **10**, 269–288 (1999)
55. H. Voogd, *Multicriteria Evaluation for Urban and Regional Planning* (Pion, London, 1983)
56. L.A. Zadeh, Optimality and non-scalar-valued performance criteria. IEEE Trans. Autom. Control **AC-8**, 59–60 (1987)
57. W.I. Zangwill, *Nonlinear Programming: A Unified Approach* (Prentice-Hall, Englewood Cliffs, 1969)
58. A.A. Zhigljavsky, *Theory of Global Random Search* (Kluwer Academic, Dordrecht, 1991)

http://www.springer.com/978-94-007-6663-1

Numerical Methods for Metamaterial Design
Diest, K. (Ed.)
2013, XVI, 213 p., Hardcover
ISBN: 978-94-007-6663-1