NUMERICAL OPTIMIZATION IN THE CONTEXT OF CLASSIFICATION PROBLEMS

MARIA CAMERON

Contents

1. Motivation: classification problems

Classification problem[1] is one of major problems considered in data science and machine learning. Instances of this problem include text categorization, image recognition, separating results of screening medical tests to positive and negative, etc. We consider the classification problem with only two classes. This is not a big loss of generality, as in practice, if classification into $K > 2$ categories is required, for each $1 \le k \le K$, one solves the problem of attributing or not attributing a given object to category k. Mathematically, the input data for a classification problem are of the form of a collection of pairs vector-label:

$$(\mathbf{z}_i, y_i) \quad \mathbf{z}_i \in \mathbb{R}^D, \quad y_i \in \{1, -1\}, \quad i = 1, \dots, n, \tag{1}$$

The mainstream approach to solving classification problems is to map the input vectors $\mathbf{z}_i$ to a feature space $X \subset \mathbb{R}^d$, i.e., $\mathbf{z}_i \mapsto \phi(\mathbf{z}_i) =: \mathbf{x}_i \in X$, in which the images of points with labels 1 and -1 are expected to be separable using a hyperplane. The space feature space X is often high-dimensional. The map ϕ is often nonlinear. I will give two examples borrowed from the review paper by L. Bottou, F. Curtis, and J. Nocedal "Optimization Methods for Large-Scale Machine Learning" [2] (see Section 2).

1.1. Text categorization.

Reuters Corpus Volume I (RCV1) [3] is a manually categorized archive of over 800,000 news stories. Most stories consist of less than 1000 words (less than two A4 pages). Each story is mapped to the 47,152-dimensional feature space corresponding to the vocabulary of 47,152 words. In this space, points corresponding to each category are separable by a hyperplane from the others. Hence, the task is to find a "good" hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$ such that

$$\begin{cases} \mathbf{w}^\top \mathbf{x}_i + b > 0, & y_i = 1, \\ \mathbf{w}^\top \mathbf{x}_i + b < 0, & y_i = -1. \end{cases} \tag{2}$$

Once such a hyperplane is found, the *prediction function* h can be chosen to be

$$h(\mathbf{x}, \mathbf{w}, b) := \mathbf{w}^\top \mathbf{x} + b, \tag{3}$$

Then one evaluates

$$\mathsf{sign}\,(h(\mathbf{x}_i, \mathbf{w}, b)) \tag{4}$$

in order to attribute $\mathbf{x}_i$ to one class or the other.

In 1995, Cortes and Vapnik [1] proposed a successful approach based on duality called Support-Vector Machine or Support-Vector Network for finding the optimal hyperplane for which the right-hand sides of the inequalities in (2) are replaced with 1 and -1 respectively.

[1]The first classification algorithm was proposed by R. A. Fisher in 1936 where he derived an optimal decision function to separate two Gaussian distributions (see [1]).

In the same paper, extended this approach for nonlinear dividing hypersurfaces and applied it to a number of benchmark datasets including NIST handwritten digit dataset. We will discuss this approach as it is quite interesting.

However, as we will see, it involves some numerical difficulties that make it less appealing than an approach sharpened for robustness rather than for optimality. This approach is based on the use of a smooth *loss function* and regularization, *Tikhonov* or *lasso*. For example, a *log-loss* function is of the form

$$l(h, y) := \log(1 + \exp[-yh(\mathbf{x}_i, \mathbf{w}, b)]). \tag{5}$$

Moreover, the optimization problem is often regularized by adding an extra Tikhonov's regularizing term $\frac{\lambda}{2}\|\mathbf{w}\|^2$ with a positive parameter λ:

$$\min_{(\mathbf{w},b)\in\mathbb{R}^d\times\mathbb{R}} \frac{1}{n}\sum_{i=1}^{n} l\left[h(\mathbf{x}_i, \mathbf{w}, b), y_i)\right] + \frac{\lambda}{2}\|\mathbf{w}\|^2. \tag{6}$$

Adding such a term renders the optimization problem *convex*.

1.2. **Image recognition and deep neural networks.** The problem of image recognition is much harder than the problem of text categorization. Even the task of digit recognition is difficult as there is no rule describing properties of pixel combinations that define each digit (see, e.g. Fig. 2.1 in [2] or Wiki: MNIST database). As for today, the most successful approaches for image recognition are based on *deep neural networks* (DNNs). DNNs are often described with jargon borrowed from neuroscience as they were originally inspired by models of biological neurons.

The input vectors $\mathbf{z}_i \equiv \mathbf{x}_i^{(0)}$ are mapped to the feature space by a composition of functions of the form

$$\mathbf{x}_i^{(j)} = s\left(W_j\mathbf{x}_i^{(j-1)} + \mathbf{b}_j\right) \in \mathbb{R}^{d_j}, \quad j = 1, \ldots, J, \tag{7}$$

where J is the number of *layers*, i.e., the number of functions in the composition, the $d_j \times d_{j-1}$-matrix W_j is vector $\mathbf{b}_j \in \mathbb{R}^{d_j}$ determine parameters of jth layer, and s is a chosen nonlinear *activation* function acting component-wise. Popular choices are ReLU (rectified linear unit) $s(x) = \max\{0, x\}$ and sigmoid $s(x) = (1 + \exp(-x))^{-1}$ shown in Fig. 1. To simplify notation, we introduce a parameter vector w that collects all parameters:

$$\mathsf{w} := \{(W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2), \ldots, (W_J, \mathbf{b}_J)\}. \tag{8}$$

The optimization problem becomes

$$\min_{\mathsf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} l\left[h(\mathbf{x}_i, \mathsf{w}), y_i)\right], \tag{9}$$

where l is the chosen loss function. In contrast to (9) this optimization problem is highly nonlinear and nonconvex, i.e., intractable to solve to global optimality [2].

Here are some examples of DNNs for image recognition[2]:

- AlexNet (2012), 8 layers. The original paper by A. Krizhevsky, I. Sutsever, and G. E. Hinton [4] is available here.

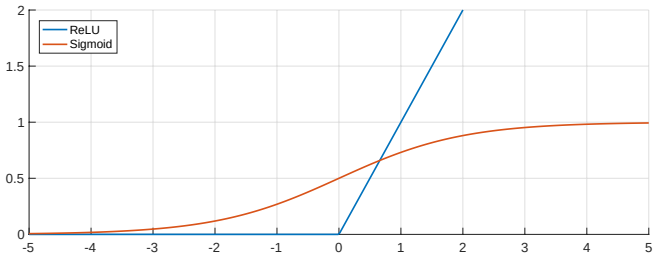


FIGURE 1. Popular choices of the activation function in DNNs: rectified linear unit (ReLU) $s(x) = \max\{0, x\}$ and sigmoid $s(x) = (1 + \exp(-x))^{-1}$.

- ResNet (2016), a resudial neural network, proposed in [5] may have up to 152 layers. Apparently, ResNet-18 is implemented in Matlab.

2. A WARM-UP CLASSIFICATION PROBLEM

Let us start with a simple classification problem. We are given a set of data $\{\mathbf{x}_i, y_i\}$, $i = 1, \dots, n$, where $\mathbf{x}_i \in \mathbb{R}^d$ are vectors and $y_i \in \{1, -1\}$ are labels. We assume that the sets of points with labels 1 and -1 are separable by a hyperplane. The problem posed in [1] is the following (see Fig. 2):

find a hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$ *such that*

$$f(\mathbf{w}, b) := \frac{1}{2}\|\mathbf{w}\|^2 \to \min \quad \text{subject to} \tag{10}$$

$$y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1, \quad i = 1, \dots, n. \tag{11}$$

Let us explain (10). The distance between the two parallel hyperplanes in (11), $\mathbf{w}^\top \mathbf{x}_i + b - 1 = 0$ and $\mathbf{w}^\top \mathbf{x}_i + b + 1 = 0$, is equal to the distance between the points of intersection of these hyperplanes with the normal line $\{\alpha \mathbf{w} \mid \alpha \in \mathbb{R}\}$. These two points are

$$\frac{1-b}{\|\mathbf{w}\|}\frac{\mathbf{w}}{\|\mathbf{w}\|} \quad \text{and} \quad \frac{-1-b}{\|\mathbf{w}\|}\frac{\mathbf{w}}{\|\mathbf{w}\|}. \tag{12}$$

Hence, the distance between these hyperplanes is

$$\rho := \left| \frac{1-b}{\|\mathbf{w}\|} - \frac{-1-b}{\|\mathbf{w}\|} \right| = \frac{2}{\|\mathbf{w}\|}. \tag{13}$$

Therefore, (10) maximizes the distance between the these two planes. The constraint (11) is equivalent to

$$\begin{cases} \mathbf{w}^\top \mathbf{x}_i + b \geq 1, & y_i = 1, \\ \mathbf{w}^\top \mathbf{x}_i + b \leq -1, & y_i = -1. \end{cases} \quad i = 1, \dots, n. \tag{14}$$

[2]Thanks to Avi Schwarzschild, AMSC PhD student.

Compare it with (2).

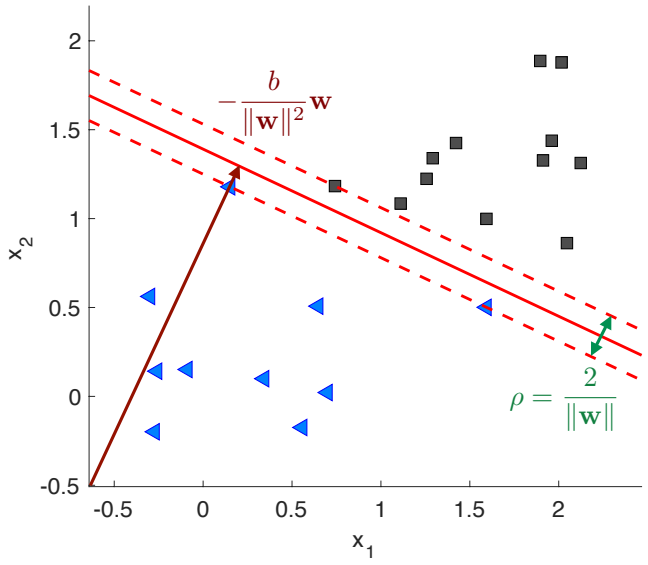


FIGURE 2. An illustration to optimization problem (10)–(11).

The optimization problem (10)–(11) is a *quadratic program*, i.e., the objective function is quadratic, while the constraints defining the feasible set are linear. Note that the objective function is convex but not strictly convex. In order to solve it, we will need to learn some theory and methodology of constrained optimization.

3. Basics of constrained optimization

As the methods that we will use for solving the optimization problem (10)–(11) are suitable for a much larger class of problems than quadratic programs, we will generalize the problem in-hand to the following:

$$
\begin{aligned}
& f(\mathbf{x}) \to \min \quad \text{subject to} \\
& c_i(\mathbf{x}) = 0, \quad i \in \mathcal{E}, \quad \text{(equality constraints)} \\
& c_i(\mathbf{x}) \geq 0, \quad i \in \mathcal{I}, \quad \text{(inequality constraints)}.
\end{aligned} \tag{15}
$$

We assume that f and c_i, $i \in \mathcal{E} \cup \mathcal{I}$, are continuously differentiable.

In order to understand how to solve (15), first recall the method of Lagrange multipliers from your calculus course. A helpful quick reminder is given in the Wiki article "Lagrange Multiplier"–look at Figure 1 there.

Imagine that we have a single equality constraint $c(\mathbf{x}) = 0$. Then the minimizer of $f(\mathbf{x})$ subject to $c(\mathbf{x}) = 0$ lies at a point $\mathbf{x}^*$ such that the level set $\{\mathbf{x} \in \mathbb{R}^d \mid f(\mathbf{x}) = f(\mathbf{x}^*)\}$ is

tangent to the hypersurface $c(\mathbf{x}) = 0$. Indeed, moving along the hypersurface $c(\mathbf{x}) = 0$ and keeping track of the values of $f(\mathbf{x})$, the extreme values of f will be achieved at the points where the level set of f is tangent to $c(\mathbf{x}) = 0$ (see Fig. 3(a)). At such points, the gradients of f and c will be parallel to each other, i.e., they will relate via

$$\nabla f(\mathbf{x}) = \lambda \nabla c(\mathbf{x}). \tag{16}$$

The factor λ is called *the Lagrange multiplier*. The condition (16) motivates the definition of *the Lagrangian function*

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda c(\mathbf{x}). \tag{17}$$

Stationary points of $L(\mathbf{x}, \lambda)$ are those where its gradient is zero, i.e.,

$$\nabla_{\mathbf{x}} L = \nabla f(\mathbf{x}) - \lambda \nabla c(\mathbf{x}) = 0, \tag{18}$$

$$\nabla L_{\lambda} = c(\mathbf{x}) = 0, \tag{19}$$

Hence every stationary point of $f(\mathbf{x})$ where $\mathbf{x}$ is restricted to $c(\mathbf{x}) = 0$ is a stationary point of $L(\mathbf{x}, \lambda)$.

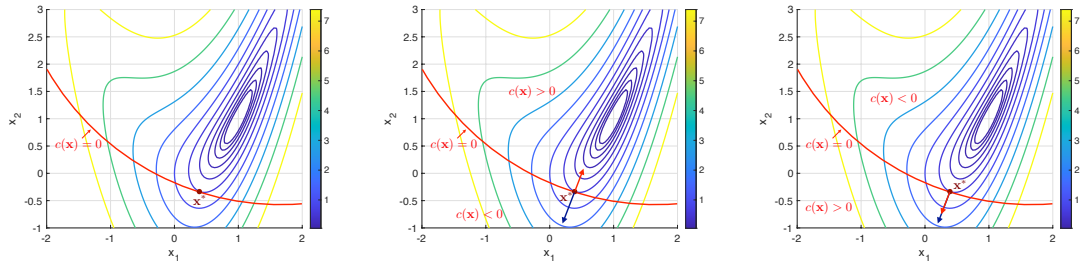


FIGURE 3. The geometric sense of Lagrange multiplier. (a): The solution to the equality-constrained minimization problem is reached at the point $\mathbf{x}^*$ where the level set of f is tangent to $c(\mathbf{x}) = 0$. (b): The unconstrained minimum of f is the solution to the inequality-constrained optimization problem $c(\mathbf{x}) \geq 0$. (c): At the constrained minimum of f, $\nabla f(\mathbf{x}^*) = \lambda \nabla c(\mathbf{x}^*)$, where $\lambda > 0$. *The level sets are those of the Rosenbrock function* $f(x_x, x_2) = (1 - x_1)^2 + (x_2 - x_1^2)^2$.

Now we replace the equality constraint $c(\mathbf{x}) = 0$ with the inequality constraint $c(\mathbf{x}) \geq 0$. For visuality, we assume that level sets of $f(\mathbf{x})$ are simple closed surfaces. This implies that f has a unique local minimizer, and it is its global minimizer.

- If the minimum of f lies in the region $c(\mathbf{x}) \geq 0$, the constrained minimum coincides with the unconstrained minimum. Apart from this global minimum, consider the point $\mathbf{x}^*$ where a level set of f touches the level set $c(\mathbf{c}) = 0$. The gradients $\nabla f(\mathbf{x}^*)$ and $\lambda \nabla c(\mathbf{x}^*)$ are antiparallel, i.e., $\nabla f(\mathbf{x}^*) = \lambda \nabla c(\mathbf{x}^*)$ for some $\lambda < 0$ (see Fig. 3(b)).

- If the global minimizer of f does not belong to the feasible set where $c(\mathbf{x}) \geq 0$ as in Fig. 3(c), the constrained minimum will be achieved at the point $\mathbf{x}^*$ lying at $c(\mathbf{x}) = 0$ where $\nabla f(\mathbf{x}^*) = \lambda \nabla c(\mathbf{x}^*)$ for some $\lambda > 0$. In this case, we say that the constraint $c(\mathbf{x}) \geq 0$ is *active* at the solution $\mathbf{x}^*$.

This illustrative example shows that the sign of the Lagrange multiplier is important and gives an intuition for the *Karush-Kuhn-Tucker* first-order optimality conditions.

3.1. **Karush-Kuhn-Tucker optimality conditions.** This section is based on J. Nocedal and S. Wright "Numerical Optimization" [6], Chapter 12.

Definition 1. *The active set $\mathcal{A}(\mathbf{x})$ at any feasible $\mathbf{x}$ consists of the equality constraint indices from $\mathcal{E}$ together with the indices of the inequality constraints i for which $c_i(\mathbf{x}) = 0$; that is,*

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(\mathbf{x}) = 0\}. \tag{20}$$

At any feasible point $\mathbf{x}$, the inequality constraint $c_i(\mathbf{x}) \geq 0$ is *active* if $c_i(\mathbf{x}) = 0$ and inactive if $c_i(\mathbf{x}) > 0$.

The *Lagrangian function* is defined by

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(\mathbf{x}). \tag{21}$$

Definition 2. *A feasible point $\mathbf{x}^*$ is a local solution of (15) if all feasible sequences $\mathbf{z}_k \to \mathbf{x}^*$ as $k \to \infty$ have the property that $f(\mathbf{z}_k) \geq f(\mathbf{x}^*)$ for all k sufficiently large.*

Definition 3. *A direction $\mathbf{d}$ at a feasible point $\mathbf{x}$ is feasible if $\nabla c_i(\mathbf{x})^\top d = 0 \ \forall i \in \mathcal{E}$ and $\nabla c_i(\mathbf{x})^\top d \geq 0 \ \forall i \in \mathcal{A}(\mathbf{x}) \cap \mathcal{I}$.*

Definition 4. *The vector $\mathbf{d}$ is a tangent vector to the feasible set Ω at a point $\mathbf{x}$ if there are a feasible sequence $\mathbf{z}_k \to \mathbf{x}$ and a sequence of positive scalars $t_k \to 0$ as $k \to \infty$ such that*

$$\lim_{k \to \infty} \frac{\mathbf{z}_k - \mathbf{x}}{t_k} = \mathbf{d}. \tag{22}$$

The set of all tangent vectors at $\mathbf{x}$ is called the tangent cone and is denoted by $T_\Omega(\mathbf{x})$.

The first-order optimality conditions known as the *Karush-Kuhn-Tucker (KKT) conditions* are stated in the following theorem.

Theorem 1. *Suppose $\mathbf{x}^*$ is a local solution of (15) where f and c_i's are continuously differentiable, and the set of active constraints gradients*

$$\{\nabla c_i(\mathbf{x}^*), \ i \in \mathcal{A}(\mathbf{x}^*)\} \tag{23}$$

is linearly independent[3]*. Then there is a Lagrange multiplier vector* $\boldsymbol{\lambda}^* = \{\lambda_i\}_{i\in(\mathcal{E}\cup\mathcal{I})}$ *such that the following conditions are satisfied at* $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = 0, \tag{24}$$

$$c_i(\mathbf{x}^*) = 0 \quad \forall i \in \mathcal{E}, \tag{25}$$

$$c_i(\mathbf{x}^*) \geq 0 \quad \forall i \in \mathcal{I}, \tag{26}$$

$$\lambda_i^* \geq 0 \quad \forall i \in \mathcal{I}, \tag{27}$$

$$\lambda_i^* c_i(\mathbf{x}^*) = 0 \quad \forall i \in \mathcal{E}\cup\mathcal{I}. \tag{28}$$

We will prove Theorem 1 for the case where the set of constrains consists only of linear inequality constraints, i.e., for the problem

$$f(\mathbf{x}) \to \min \quad \text{subject to} \tag{29}$$
$$A\mathbf{x}^* - \mathbf{b} \geq 0, \quad A \text{ is } n \times d.$$

The proof for (29) follows the same steps as the proof for (15) given in [6] but is shorter as it requires fewer technicalities. One important point is that the LICQ condition is not required if all constraints are linear.

Proof. Without the loss of generality we assume that the active set consists of the first m inequalities: $\mathcal{A}(\mathbf{x}^*) = \{1, \dots, m\}$. The matrix consisting of the first m rows of A and the vector consisting of the first m components of $\mathbf{b}$ will be denoted by A, and b respectively.

Step 1. Show that the set of feasible directions is

$$\mathcal{F}(\mathbf{x}^*) = \{\mathbf{d} \in \mathbb{R}^d \mid \mathsf{A}\mathbf{d} \geq 0\} \tag{30}$$

and coincides with the tangent cone $T_\Omega(\mathbf{x}^*)$.

Indeed, since $\nabla(A\mathbf{x} - \mathbf{b})_i$ is the ith row of A, Definition 3 implies that a direction $\mathbf{d}$ is feasible if and only if $\mathsf{A}\mathbf{d} \geq 0$.

Furthermore, for any tangent vector $\mathbf{d}$ we have a feasible sequence $\mathbf{z}_k$ and a sequence of scalars $t_k \to 0$ such that

$$\mathbf{z}_k = \mathbf{x}^* + t_k\mathbf{d} + o(t_k).$$

Multiplying this equation by A and subtracting b we get:

$$0 \leq \mathsf{A}\mathbf{z}_k - \mathsf{b} = \underbrace{\mathsf{A}\mathbf{x}^* - \mathsf{b}}_{=0} + t_k\mathsf{A}\mathbf{d} + o(t_k) = t_k\mathsf{A}\mathbf{d} + o(t_k).$$

Dividing the last equality by t_k and letting $k \to 0$, we get $\mathsf{A}\mathbf{d} \geq 0$ which means that any tangent vector is a feasible direction, i.e., $T_\Omega(\mathbf{x}^*) \subseteq \mathcal{F}(\mathbf{x}^*)$.

On the other hand, if $\mathbf{d}$ is a feasible direction then the sequence $\mathbf{z}_k = \mathbf{x}^* + t_k\mathbf{d}$ is feasible for sufficiently small t_k. Indeed, multiplying this equality by A and subtracting b we get

$$\mathsf{A}\mathbf{z}_k - \mathsf{b} = \underbrace{\mathsf{A}\mathbf{x}^* - \mathsf{b}}_{=0} + t_k\mathsf{A}\mathbf{d} = t_k\mathsf{A}\mathbf{d} \geq 0$$

which implies that $\mathbf{d}$ is a tangent vector. Hence $\mathcal{F}(\mathbf{x}^*) \subseteq T_\Omega(\mathbf{x}^*)$

[3]This condition is called the *linear independence constraint qualification (LICQ)*.

Step 2. Show that if $\mathbf{x}^*$ is a local solution of (29) then

$$\nabla f(\mathbf{x}^*)^\top \mathbf{d} \geq 0 \quad \forall \mathbf{d} \in T_\Omega(\mathbf{x}^*). \tag{31}$$

Indeed, assume the opposite: there is a tangent vector $\mathbf{d}$ such that $\nabla f(\mathbf{x}^*)^\top \mathbf{d} < 0$. Then there exist sequences $\mathbf{z}_k \to \mathbf{x}^*$, feasible, and $t_k \to 0$, $t_k > 0$, such that

$$f(\mathbf{z}_k) = f(\mathbf{x}^*) + t_k \nabla f(\mathbf{x}^*)^\top \mathbf{d} + o(t_k).$$

Hence, subtracting $f(\mathbf{x}^*)$ from both sides and dividing by t_k, we get:

$$0 \leq \frac{f(\mathbf{z}_k) - f(\mathbf{x}^*)}{t_k} = \underbrace{\nabla f(\mathbf{x}^*)^\top \mathbf{d}}_{<0} + o(1) < 0$$

for sufficiently small t_k – a contradiction. Therefore, (31) holds.

Step 3 (Farkas' lemma). Consider the cone

$$K := \left\{ \mathsf{A}^\top \lambda \mid \lambda \in \mathbb{R}^m,\ \lambda_i \geq 0\ \forall 1 \leq i \leq m \right\}. \tag{32}$$

Note that K is convex and closed. Prove that for any vector $\mathbf{g} \in \mathbb{R}^d$ the following alternative takes place:

- **either** $\mathbf{g} \in K$, i.e., there is a vector $\lambda \in \mathbb{R}^m$ with nonnegative components such that $\mathbf{g} = \mathsf{A}^\top \lambda$,
- **or** there exists a vector $\mathbf{d} \in \mathbb{R}^d$ such that the hyperplane normal to $\mathbf{d}$ separates $\mathbf{g}$ and K, i.e.,

$$\mathbf{g}^\top \mathbf{d} < 0, \quad \text{while} \quad \mathsf{A}\mathbf{d} \geq 0. \tag{33}$$

First we show that the two alternatives cannot take place simultaneously. From converse, assume that $\mathbf{g} = \mathsf{A}^\top \lambda$ for $\lambda \geq 0$ while $\mathbf{g}^\top \mathbf{d} < 0$ and $\mathsf{A}\mathbf{d} \geq 0$. Then

$$0 > \mathbf{g}^\top \mathbf{d} = \mathbf{d}^\top \mathsf{A}^\top \lambda = (\underbrace{\mathsf{A}\mathbf{d}}_{\geq 0})^\top \underbrace{\lambda}_{\geq 0} \geq 0,$$

a contradiction.

Now we show that if $\mathbf{g} \notin K$ then (33) holds. Let $\mathbf{y}$ be the point of the cone K closest to $\mathbf{g}$ in terms of the Euclidean distance. Such a point exists as K is closed. Note that $\mathbf{y}$ may be the origin or a boundary point of K different from the origin – these two cases are depicted in Fig. 4. In both cases, the argument below is valid.

By the definition of cone, the whole ray emanating from the origin and passing through $\mathbf{y}$ belongs to K:

$$\{\alpha \mathbf{y} \in K \mid \alpha \geq 0\}.$$

Since $\mathbf{y}$ is the closest point to $\mathbf{g}$ in this ray as well, the minimum of the function

$$\phi(\alpha) := \frac{1}{2}(\alpha \mathbf{y} - \mathbf{g})^\top (\alpha \mathbf{y} - \mathbf{g}) = \frac{1}{2}\alpha^2 \|\mathbf{y}\|^2 - \alpha \mathbf{y}^\top \mathbf{g} + \frac{1}{2}\|\mathbf{g}\|^2$$

is achieved at $\alpha = 1$. Therefore,

$$0 = \frac{d\phi}{d\alpha}(1) = \|\mathbf{y}\|^2 - \mathbf{y}^\top \mathbf{g} = \mathbf{y}^\top (\mathbf{y} - \mathbf{g}).$$

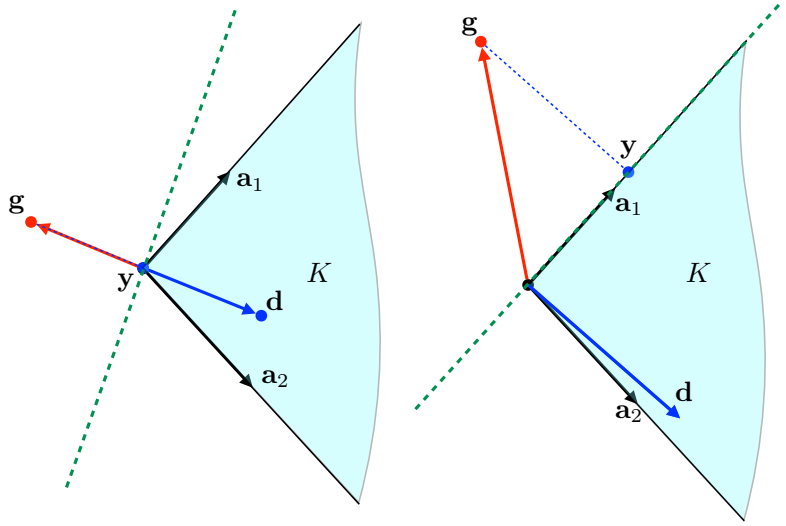


FIGURE 4. Illustration to step 3 of the proof of Theorem 1 in the case where $d = 2$ and $m = 2$. The vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ are the rows of the matrix A.

Let $\mathbf{z} \in K$, $\mathbf{z} \neq \mathbf{y}$. Since K is convex, we have

$$\mathbf{y} + \theta(\mathbf{z} - \mathbf{y}) \in K \quad \forall \theta \in [0, 1].$$

By the minimizing property of $\mathbf{y}$ we have

$$\|\mathbf{y} + \theta(\mathbf{z} - \mathbf{y}) - \mathbf{g}\|^2 \geq \|\mathbf{y} - \mathbf{g}\|^2 \quad \forall \theta \in [0, 1].$$

Hence

$$2\theta(\mathbf{z} - \mathbf{y})^\top(\mathbf{y} - \mathbf{g}) + \theta^2\|\mathbf{z} - \mathbf{y}\|^2 \geq 0.$$

Dividing this equation by θ and taking limit $\theta \downarrow 0$, we get

$$0 \leq (\mathbf{z} - \mathbf{y})^\top(\mathbf{y} - \mathbf{g}) = \mathbf{z}^\top(\mathbf{y} - \mathbf{g}) - \underbrace{\mathbf{y}^\top(\mathbf{y} - \mathbf{g})}_{=0} = \mathbf{z}^\top(\mathbf{y} - \mathbf{g}), \quad \text{i.e.}$$

$$\mathbf{z}^\top(\mathbf{y} - \mathbf{g}) \geq 0 \quad \forall \mathbf{z} \in K. \tag{34}$$

Now we set

$$\mathbf{d} := \mathbf{y} - \mathbf{g}$$

and show that $\mathbf{d}$ satisfies (33). Note that $\mathbf{d} \neq 0$ as $\mathbf{g} \notin K$. Indeed, (34) means that

$$\mathbf{d}^\top \mathsf{A}^\top \lambda \geq 0 \quad \forall \lambda \geq 0,$$

which is true only if $\mathbf{d}^\top \mathsf{A}^\top \geq 0$, i.e, $\mathsf{A}\mathbf{d} \geq 0$. On the other hand,

$$\mathbf{d}^\top \mathbf{g} = \mathbf{d}^\top(\mathbf{y}-\mathbf{d}) = (\mathbf{y}-\mathbf{g})^\top(\mathbf{y}-\mathbf{d}) = \underbrace{(\mathbf{y}-\mathbf{g})^\top \mathbf{y}}_{=0} - \|\mathbf{d}\|^2 < 0.$$

Step 4. Now we set $\mathbf{g} = \nabla f(\mathbf{x}^*)$. By Farkas' lemma proven in Step 3, either $\nabla f(\mathbf{x}^*) \in K$, i.e., there is a vector λ with nonnegative components such that

$$\nabla f(\mathbf{x}^*) = \mathsf{A}^\top \lambda, \tag{35}$$

or there is a feasible direction $\mathbf{d}$ such that $\mathsf{A}\mathbf{d} \geq 0$ and $\nabla f(\mathbf{x}^*)^\top \mathbf{d} < 0$ which means that $\mathbf{x}^*$ is not a local solution. Therefore, since $\mathbf{x}^*$ is a local solution, (35) takes place.

Finally, we set the Lagrange multipliers with indices in the active set ($1 \leq i \leq m$) to be equal to λ and we set to zero the rest of them, i.e.,

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda \\ 0 \end{bmatrix}.$$

The conditions (24)–(28) are readily verified. □

3.2. **Active-set method.** This section is based on J. Nocedal and S. Wright "Numerical Optimization" [6], Chapter 16. We consider a convex quadratic program (QP)

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top G\mathbf{x} + \mathbf{c}^\top \mathbf{x} \;\to\; \min \quad \text{subject to} \tag{36}$$

$$\mathbf{a}_i^\top \mathbf{x} = \mathbf{b}_i, \quad i \in \mathcal{E} \tag{37}$$

$$\mathbf{a}_i^\top \mathbf{x} \geq b_i, \quad i \in \mathcal{I}. \tag{38}$$

Convexity of the QP means that the matrix G is positive definite.

A quite natural method to solve (36)–(38) is *the active-set method*. At every step of this algorithm, we solve a QP

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top G\mathbf{x} + \mathbf{c}^\top \mathbf{x} \;\to\; \min \quad \text{subject to} \tag{39}$$

$$\mathbf{a}_i^\top \mathbf{x} = \mathbf{b}_i, \quad i \in \mathcal{W}, \tag{40}$$

(41)

where $\mathcal{W}$ is the current set of active constraints. We will denote by $A_\mathcal{W}$ the matrix whose rows at $\mathbf{a}_i$, $i \in \mathcal{W}$, and $\mathbf{b}_\mathcal{W}$ the vector of b_i's, $i \in \mathcal{W}$. Then (40) becomes $A_\mathcal{W}\mathbf{x} = \mathbf{b}_\mathcal{W}$.

We do so as follows. Let $\mathbf{x}_k$ be the current iterate. We want to find a step $\mathbf{p}_k$ to obtain the next iterate

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k.$$

In the case if $\mathbf{x}_{k+1}$ is not feasible, we shorten the step length just enough to make $\mathbf{x}_{k+1}$ remain feasible. Plugging $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$ into (39) we get

$$\frac{1}{2}(\mathbf{x}_k+\mathbf{p}_k)^\top G(\mathbf{x}_k+\mathbf{p}_k) + \mathbf{c}^\top(\mathbf{x}_k+\mathbf{p}_k) = \frac{1}{2}\mathbf{p}_k^\top G\mathbf{p}_k + (G\mathbf{x}_k+\mathbf{c})^\top \mathbf{p}_k + f(\mathbf{x}_k).$$

We observe that $G\mathbf{x}_k + \mathbf{c} \equiv \nabla f(\mathbf{x}_k)$ and $f(\mathbf{x}_k)$ is independent of $\mathbf{p}_k$ and hence does not affect the minimizer. Moreover, $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$ into (40) we get the following constraint for $\mathbf{p}_k$:

$$A_{\mathcal{W}}(\mathbf{x}_k + \mathbf{p}_k) - \mathbf{b}_{\mathcal{W}} = \underbrace{A_{\mathcal{W}}\mathbf{x} - \mathbf{b}_{\mathcal{W}}}_{=0} + A_{\mathcal{W}}\mathbf{p}_k = A_{\mathcal{W}}\mathbf{p}_k = 0.$$

Therefore, the minimization problem for $\mathbf{p}_k$ reduces to

$$\frac{1}{2}\mathbf{p}^\top G\mathbf{p} + \nabla f(\mathbf{x}_k)^\top \mathbf{p} \;\to\; \min \quad \text{subject to} \tag{42}$$

$$A_{\mathcal{W}}\mathbf{p}_k = 0. \tag{43}$$

(44)

The KKT system for (42)–(43) is

$$G\mathbf{p}_k + \nabla f(\mathbf{x}_k) - A_{\mathcal{W}}^\top \boldsymbol{\lambda} = 0, \quad A_{\mathcal{W}}\mathbf{p}_k = 0.$$

It can be rewritten in the matrix form:

$$\begin{bmatrix} G & A_{\mathcal{W}}^\top \\ A_{\mathcal{W}} & 0 \end{bmatrix} \begin{bmatrix} -\mathbf{p}_k \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \nabla f(\mathbf{x}_k) \\ 0 \end{bmatrix}. \tag{45}$$

Exercise Show that the matrix in (45) with G $d \times d$ being symmetric positive definite and A $m \times d$ having linearly independent rows, is of *saddle-point type*, i.e., it has d positive eigenvalues and m negative ones. *Hint: Omit the subscript $\mathcal{W}$ for brevity. Find matrices X and S (S is called the **Schur compliment**) such that*

$$\begin{bmatrix} G & A^\top \\ A & 0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ X & I \end{bmatrix} \begin{bmatrix} G & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & X^\top \\ 0 & I \end{bmatrix}.$$

Then use Sylvester's Law of Inertia (look it up!) to finish the proof.

Exercise Consider an equality-constrained QP (G is symmetric)

$$\frac{1}{2}\mathbf{x}^\top G\mathbf{x} + \mathbf{c}^\top \mathbf{x} \;\to\; \min \quad \text{subject to} \tag{46}$$

$$A\mathbf{x} = \mathbf{b}. \tag{47}$$

(48)

Assume that A is full rank (i.e., its rows are linearly independent) and $Z^\top GZ$ is positive definite where Z is a basis for the null-space of A, i.e., $AZ = 0$.

(1) Write the KKT system for this case in the matrix form.
(2) Show that the matrix of this system K is invertible. *Hint: assume that there is a vector $\mathbf{z} := (\mathbf{x}, \mathbf{y})^\top$ such that $K\mathbf{z} = 0$. Consider the form $\mathbf{z}^\top K\mathbf{z}$, and so on ... . You should arrive at the conclusion that then $\mathbf{z} = 0$.*
(3) Conclude that there exists a unique vector $(\mathbf{x}^*, \boldsymbol{\lambda}^*)^\top$ that solves the KKT system. Note that since we have only equality constraints, positivity of $\boldsymbol{\lambda}$ is irrelevant.

According to the claims in the exercises, (45) has a unique solution $(\mathbf{p}, \boldsymbol{\lambda})$. We consider two cases.

- If $\mathbf{p} \neq 0$, we can make a step in the direction $\mathbf{p}$. Let us show that this is a descend direction for $f(\mathbf{x})$, i.e., $\nabla f(\mathbf{x}_k)^\top p < 0$. Indeed, we have

$$G\mathbf{p} + \nabla f(\mathbf{x}_k) = A^\top \boldsymbol{\lambda}$$
$$A\mathbf{p} = 0.$$

  Multiplying the first equation by $\mathbf{p}^\top$ we get

$$\mathbf{p}^\top (G\mathbf{p} + \nabla f(\mathbf{x}_k)) = \mathbf{p}^\top G\mathbf{p} + \nabla f(\mathbf{x}_k)^\top \mathbf{p} = \mathbf{p}^\top A^\top \boldsymbol{\lambda} = (\underbrace{A\mathbf{p}}_{=0})^\top \boldsymbol{\lambda} = 0.$$

  Since G is positive definite and $\mathbf{p} \neq 0$ by assumption, we conclude that $\nabla f(\mathbf{x}_k)^\top \mathbf{p} < 0$ which means that $\mathbf{p}$ is a descend direction for f.

  Next, we start moving from $\mathbf{x}_k$ along the direction $\mathbf{p}$ until we either travel the full distance $\|\mathbf{p}\|$ or activate another constraint:

$$\mathbf{a}_i \mathbf{x} = b_i \quad \text{for some} \quad i \notin \mathcal{W}.$$

  Hence, in order to find step length α, we consider

$$\mathbf{a}_i^\top (\mathbf{x}_k + \alpha \mathbf{p}_k) = \mathbf{a}_i \mathbf{x}_k + \alpha \mathbf{a}_i^\top \mathbf{p}_k = b_i \quad \text{for all} \quad i \notin \mathcal{W}.$$

  Since $\mathbf{a}_i^\top \mathbf{x}_k > b_i \; \forall i \notin \mathcal{W}$, if $\mathbf{a}_i^\top \mathbf{p} \geq 0$, we can only reinforce this constraint by moving along $\mathbf{p}$. In contrast, if $\mathbf{a}_i^\top \mathbf{p} < 0$, we may hit the constraint prior to traveling the full distance. Hence, the step length α is given by

$$\alpha = \min \left\{ 1, \min_{i \notin \mathcal{W}, \mathbf{a}_i^\top \mathbf{p} < 0} \frac{b_i - \mathbf{a}_i^\top \mathbf{p}}{\mathbf{a}_i^\top \mathbf{p}} \right\}. \tag{49}$$

  Finally, we set $\alpha_k = \alpha$ and $\mathbf{p}_k = \mathbf{p}$.
- If $\mathbf{p} = 0$, we cannot move anywhere from $\mathbf{x}_k$ given the set of constraints $\mathcal{W}$. Hence, there are two possibilities.
  - We have found the solution to (36)–(38). To check if this is the case, we look at the vector $\boldsymbol{\lambda}$ of Lagrange's multipliers, and check their signs. If all Lagrange multipliers corresponding to inequality constrains are nonnegative, a solution is found, and we terminate the iterations.
  - If there is a negative Lagrange multiplier corresponding to an inequality constraint, we remove it from $\mathcal{W}$ and proceed with iterations.

A Matlab function `ASM` implements the active-set method for the case where f is allowed to be nonconvex and nonquadratic, and the set of constraints consists of inequality constraints only. A driver for it with the Rosenbrock function and the feasible region being a hexagon is the function `ASMdriver`.

```
 function ASMdriver()
%%  the Rosenbrock function
a = 5;
func = @(x,y)(1-x).^2 + a*(y - x.^2).^2;  % Rosenbrock's function
gfun = @(x)[-2*(1-x(1))-4*a*(x(2)-x(1)^2)*x(1);2*a*(x(2)-x(1)^2)]; % gradient of f
Hfun = @(x)[2 + 12*a*x(1)^2 - 4*a*x(2), -4*a*x(1); -4*a*x(1), 2*a]; % Hessian of f
```

```
lsets = exp([-3:0.5:2]);
%% constraints
Nv = 6;
t = linspace(0,2*pi,Nv+1);
t(end) = [];
t0 = 0.1;
verts = [0.1+cos(t0+t);0.1+sin(t0+t)];
R = [0,-1;1,0];
A = (R*(circshift(verts,[0,-1])-verts))';
b = verts(1,:)'.*A(:,1) + verts(2,:)'.*A(:,2); % b_i = a_i*verts(:,i)
x = [-0.5;0.5];
W = [];
[xiter,lm] = ASM(x,gfun,Hfun,A,b,W);
%% graphics
close all
fsz = 16;
figure(1);
hold on;
n = 100;
txmin = min(verts(1,:))-0.2;
txmax = max(verts(1,:))+0.2;
tymin = min(verts(2,:))-0.2;
tymax = max(verts(2,:))+0.2;
tx = linspace(txmin,txmax,n);
ty = linspace(tymin,tymax,n);
[txx,tyy] = meshgrid(tx,ty);
ff = func(txx,tyy);
contour(tx,ty,ff,lsets,'Linewidth',1);
edges = [verts,verts(:,1)];
plot(edges(1,:),edges(2,:),'Linewidth',2,'color','k');
plot(xiter(1,:),xiter(2,:),'Marker','.','Markersize',20,'Linestyle','-',...
    'Linewidth',2,'color','r');
xlabel('x','Fontsize',fsz);
ylabel('y','Fontsize',fsz);
set(gca,'Fontsize',fsz);
colorbar;
grid;
daspect([1,1,1]);
end

function [xiter,lm] = ASM(x,gfun,Hfun,A,b,W)
%% minimization using the active set method (Nocedal & Wright, Section 16.5)
% Solves f(x) --> min subject to Ax >= b
```

```
% x = initial guess, a column vector
TOL = 1e-10;
dim = length(x);
g = gfun(x);
H = Hfun(x);
iter = 0;
itermax = 1000;
m = size(A,1); % the number of constraints
% W = working set, the set of active constrains
I = (1:m)';
Wc = I; % the compliment of W
xiter = x;
while iter < itermax
    % compute step p: solve 0.5*p'*H*p + g'*p --> min subject to A(W,:)*p = 0
    AW = A(W,:); % LHS of active constraints
    % fix H if it is not positive definite
    ee = sort(eig(H),'ascend');
    if ee(1) < 1e-10
        lam = -ee(1) + 1;
    else
        lam = 0;
    end
    H = H + lam*eye(dim);
    if ~isempty(W)
        M = [H, -AW';AW,zeros(size(W,1))];
        RHS = [-g;zeros(size(W,1),1)];
    else
        M = H;
        RHS = -g;
    end
    aux = M\RHS;
    p = aux(1:dim);
    lm = aux(dim+1:end);
    if  norm(p) < TOL  % if step == 0
        if ~isempty(W)
            lm = AW'\g; % find Lagrange multipliers
            if min(lm) >= 0 % if Lagrange multipliers are positive, we are done
                % the minimizer is one of the corners
                fprintf('A local solution is found, iter = %d\n',iter);
                fprintf('x = [\n'); fprintf('%d\n',x);fprintf(']\n');
                break;
            else % remove the index of the most negative multiplier from W
                [lmin,imin] = min(lm);
```

```
                W = setdiff(W,W(imin));
                Wc = setdiff(I,W);
            end
        else
            fprintf('A local solution is found, iter = %d\n',iter);
            fprintf('x = [\n'); fprintf('%d\n',x);fprintf(']\n');
            break;
        end
    else % if step is nonzero
        alp = 1;
        % check for blocking constraints
        Ap = A(Wc,:)*p;
        icand = find(Ap < -TOL);
        if ~isempty(icand)
            % find step lengths to all possible blocking constraints
            al = (b(Wc(icand)) - A(Wc(icand),:)*x)./Ap(icand);
            % find minimal step length that does not exceed 1
            [almin,kmin] = min(al);
            alp = min(1,almin);
        end
        x = x + alp*p;
        g = gfun(x);
        H = Hfun(x);
        if alp < 1
            W = [W;Wc(icand(kmin))];
            Wc = setdiff(I,W);
        end
    end
    iter = iter + 1;
    xiter = [xiter,x];
end
if iter == itermax
    fprintf('Stopped because the max number of iterations %d is performed\n',iter);
end
end
```

3.3. **Finding the dividing hyperplane for classification problems.** Now we return to the constrained minimization problem (10)–(11), a QP with a convex but not strictly convex objective function and linear inequality constraints. We rewrite it in a form convenient for

feeding into the active-set solver:

$$\mathsf{w} := \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}, \quad \mathsf{A} = \begin{bmatrix} y_1 & \dots & y_1 \\ \vdots & & \vdots \\ y_n & \dots & y_n \end{bmatrix} \odot \begin{bmatrix} \mathbf{x}_1^\top & \to, & 1 \\ \vdots & & \vdots \\ \mathbf{x}_n^\top & \to, & 1 \end{bmatrix}, \quad \mathsf{b} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix},$$

where the symbol $\odot$ denotes the componentwise matrix multiplication. The vector w is $(d+1) \times 1$, the matrix $\mathsf{A} = [X, 1_{n\times 1}]$ is $n \times (d+1)$, where X is the matrix whose rows are $\mathbf{x}_i^\top$, $i = 1, \dots, d$, and the right-hand side vector b is $n \times 1$; n is the number of data points and d is the dimension of the data space. Hence, the optimization problem becomes:

$$f(\mathsf{w}) = \frac{1}{2}\mathsf{w}^\top \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & 0 \end{bmatrix} \mathsf{w} \to \min \quad \text{subject to} \tag{50}$$

$$\mathsf{A}\mathsf{w} = \mathsf{b}. \tag{51}$$

An example of the active-set algorithm applied to find the optimal line dividing samples from two 2D Gaussian distributions with means $(0.5, 0.5)^\top$ and $(1.5, 1.5)^\top$ and variances 1 is shown in Fig. 2. There are two complications with this approach.

- The active-set method needs a feasible point to start with. It is hard-to-find by inspection even in 2D. In order to find the initial guess for w, I wrote an extremely simple routine shown below. I define a loss function

  $$l(\mathsf{w}, \mathsf{A}, \mathsf{b}) := \sum_{i=1}^{n} \mathsf{ReLU}(\mathsf{b}_i - (\mathsf{A}\mathsf{w})_i), \quad \text{where} \quad \mathsf{ReLU}(x) = \max\{0, x\},$$

  and organize a gradient descend with constant step length. Note that $l = 0$ and $\nabla l = 0$ if w is a feasible point.

```
function [w,l,lcomp] = FindInitGuess(w,A,b)
relu = @(w)max(w,0);
drelu = @(w)ones(size(w)).*sign(relu(w));
l = sum(relu(b - A*w));
iter = 0;
itermax = 10000;
step = 0.1;
while l > 0 && iter < itermax
    dl = sum(-drelu(b - A*w)'*A,1)';
    if norm(dl) > 1
        dl = dl/norm(dl);
    end
    w = w - step*dl;
    l = sum(relu(b - A*w));
    iter = iter + 1;
```

```
end
fprintf('%d iterations: x = [%d,%d],f = %d\n',iter, w(1),w(2),l);
lcomp = relu(b - A*w);
end
end
```

The output `x` of this function is the desired initial guess. If the output `l` is nonzero, this means that the algorithm failed to find a feasible point. Why I need the output `lcomp` you will see shortly – I need it to address the issue in the next item.

- As you can easily imagine, samples from the mentioned two Gaussian distributions may not be separable by a line. This is a very common situation in real-life problems. For example, screening medical tests give results that indicate a risk for some condition but do not diagnose it exactly for the reason that data from normal and affected patients are not separable, though do exhibit certain trends (see e.g. this example). A way to address this issue proposed in [1] is to introduce *soft margins*, i.e., a set of n penalty variables $\xi_i \geq 0$, $i = 1, \ldots, n$. The new problem becomes:

$$f(\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i, \tag{52}$$

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \tag{53}$$

$$\xi_i \geq 0, \quad i = 1 \ldots, n. \tag{54}$$

To write this problem in the matrix form we define $\mathsf{x} := [\mathbf{w}, b, \boldsymbol{\xi}]^\top \in \mathbb{R}^{d+1+n}$ and get:

$$f(\mathsf{x}) = \frac{1}{2}\mathsf{x}^\top \begin{bmatrix} I_{d\times d} & 0_{d\times(n+1)} \\ 0_{(n+1)\times d} & 0_{(n+1)\times(n+1)} \end{bmatrix} \mathsf{x} + C[0_{1\times(d+1)}, 1_{1\times n}]\mathsf{x} \to \min, \tag{55}$$

$$\begin{bmatrix} [\mathbf{y}\cdot 1_{1\times(d+1)}] \odot [X, 1_{n\times 1}] & I_{n\times n} \\ 0_{n\times d+1} & I_{n\times n} \end{bmatrix} \mathsf{x} \geq \begin{bmatrix} 1_{n\times 1} \\ 0_{n\times 1} \end{bmatrix}. \tag{56}$$

The positive constant C is chosen depending on what what goal is being pursued. I chose $C = 10^3$, a large number. To find an initial guess, I run the same routine for finding the initial guess. the initial guess for the vector of errors $\boldsymbol{\xi}$ is the output `lcomp`. An example of separating samples from the same two Gaussian distributions with soft marging is shown in Fig. 5

4. Duality

The solution proposed in [1] to address the issue of finding the initial guess for the QP arising in the classification problem is to consider the dual problem [6] (Section 12.9). We will first discuss what it is in general, and then set it up for the classification problem.

Recall the definition of the Lagrangian function:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \boldsymbol{\lambda}^\top \mathbf{c}(\mathbf{x}), \tag{57}$$

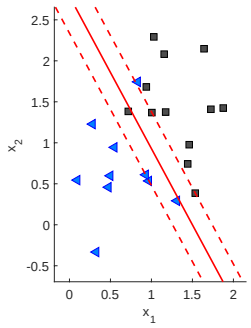


FIGURE 5. Separating samples from two Gaussian distributions with soft margins (55)–(56)

.

where $\mathbf{c}(\mathbf{x}) = [c_1(\mathbf{x}), \ldots, c_n(\mathbf{x})]^\top$ is the vector of the left-hand sides of the constraints $c_i(\mathbf{x}) \geq 0$. The solution to the KKT system maximizes L with respect to $\boldsymbol{\lambda}$ and while minimizes it with respect to $\mathbf{x}$. This suggests to consider the *dual problem* defined by

$$q(\boldsymbol{\lambda}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}) \to \max \quad \text{subject to} \quad \boldsymbol{\lambda} \geq 0. \tag{58}$$

The domain of q is defines at the set of those $\boldsymbol{\lambda}$ for which q is finite, i.e.,

$$\mathcal{D}_q := \{\boldsymbol{\lambda} \mid q(\boldsymbol{\lambda}) > -\infty\}.$$

In general, the dual problem might be easier or harder than the original problem.

Now we will derive the dual problem for the classification problem (50)–(51) (see [1], Appendix A). The Lagrangian for (50)–(51) is

$$L(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{i=1}^{n} \lambda_i \left[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \right]. \tag{59}$$

Let $\mathbf{w}^*\mathbf{x} + b^* = 0$ be the optimal hyperplane and $\boldsymbol{\lambda}^*$ be the corresponding vector of Lagrange multipliers. Taking the gradient of L with respect to $\mathbf{w}$ and b and evaluating it at $(\mathbf{w}^*, b^*, \boldsymbol{\lambda}^*)$ we obtain:

$$\frac{\partial L}{\partial \mathbf{w}}(\mathbf{w}^*, b^*, \boldsymbol{\lambda}^*) = \mathbf{w}^* - \sum_{i=1}^{n} \lambda_i y_i \mathbf{x}_i = 0, \tag{60}$$

$$\frac{\partial L}{\partial b}(\mathbf{w}^*, b^*, \boldsymbol{\lambda}^*) = -\sum_{i=1}^{n} \lambda_i y_i = 0. \tag{61}$$

Therefore, the optimal vector $\mathbf{w}^*$ relates to the optimal $\boldsymbol{\lambda}$ via

$$\mathbf{w}^* = \sum_{i=1}^{n} \lambda_i y_i \mathbf{x}_i, \tag{62}$$

and the optimal $\boldsymbol{\lambda}^*$ also satisfies the following equality condition

$$\mathbf{y}^\top \boldsymbol{\lambda} = 0. \tag{63}$$

Substituting (62) and (63) into (59) we obtain the dual function $q(\boldsymbol{\lambda})$:

$$\begin{aligned} q(\boldsymbol{\lambda}) &= \frac{1}{2}\left(\sum_{i=1}^n \lambda_i y_i \mathbf{x}_i^\top\right)\left(\sum_{j=1}^n \lambda_j y_j \mathbf{x}_j\right) - \sum_{i=1}^n \lambda_i \left[y_i \left\{ \left(\sum_{j=1}^n \lambda_j y_j \mathbf{x}_j^\top\right) \mathbf{x}_i + b \right\} - 1 \right] \\ &= \frac{1}{2}\boldsymbol{\lambda}^\top D \boldsymbol{\lambda} - \boldsymbol{\lambda}^\top D \boldsymbol{\lambda} - b\mathbf{y}^\top \boldsymbol{\lambda} + 1_{1\times n}\boldsymbol{\lambda} \\ &= -\frac{1}{2}\boldsymbol{\lambda}^\top D \boldsymbol{\lambda} + 1_{1\times n}\boldsymbol{\lambda}, \text{where} \end{aligned} \tag{64}$$

$$D = \left(\mathbf{y}\mathbf{y}^\top\right) \odot \left(\begin{bmatrix} \mathbf{x}_1^\top & \rightarrow \\ \vdots & \\ \mathbf{x}_n^\top & \rightarrow \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ \downarrow & & \downarrow \end{bmatrix} \right) = \left(\mathbf{y}\mathbf{y}^\top\right) \odot \left(XX^\top\right). \tag{65}$$

In summary, we obtain the following constrained optimization problem for $\boldsymbol{\lambda}$:

$$q(\boldsymbol{\lambda}) = -\frac{1}{2}\boldsymbol{\lambda}^\top D \boldsymbol{\lambda} + 1_{1\times n}\boldsymbol{\lambda} \to \max \quad \text{subject to} \tag{66}$$

$$\mathbf{y}^\top \boldsymbol{\lambda} = 0, \tag{67}$$

$$\boldsymbol{\lambda} \geq 0. \tag{68}$$

As The vector $\boldsymbol{\lambda}$ is found, the optimal $\mathbf{w}^*$ is found from (62). Then we select two support vectors

$$\begin{aligned} &\mathbf{x}_{i_+} \quad \text{such that} \quad i_+ = \arg\max\{\lambda_i \mid 1 \leq i \leq n,\ y_i = 1\} \quad \text{and} \\ &\mathbf{x}_{i_-} \quad \text{such that} \quad i_- = \arg\max\{\lambda_i \mid 1 \leq i \leq n,\ y_i = -1\}, \end{aligned}$$

and find b from the identities $(\mathbf{w}^*)^\top \mathbf{x}_{i_+} + b = 1$ and $(\mathbf{w}^*)^\top \mathbf{x}_{i_-} + b = -1$:

$$b = -\frac{1}{2}(\mathbf{w}^*)^\top \left(\mathbf{x}_{i_+} + \mathbf{x}_{i_-}\right).$$

Exercise Is the matrix D positive definite? Either prove the the statement, or give a counterexample.

The dual problem (66)–(70) has the following advantages.

- The vector $\boldsymbol{\lambda} = 0$ is always a feasible point for (66)–(70) as all constraints hold at it. Hence, we do not need to search for an initial guess.
- If the number of constraints is large, we can solve the problem iteratively where the dimensionality of the problem at each iteration is as modest as desired. We take a subset of data points $1, \ldots, m$, $m < n$, and solve (66)–(70) for it. Most of the inequality constrains will be inactive, i.e., most of λ_i's will be zeros. Those data points $\mathbf{x}_i$ corresponding to $\lambda_i > 0$ are called *the support vectors* – this is why this approach is referred to as the *support-vector network* or the *support-vector machine*. Only these few support vectors are carried to the next iteration. The next m data

points are added to them to form the constrains for the next problem-to-solve. And so on until there are no more data points to add.

- The dual approach is also generalizable for the case of soft margins.

  Exercise Derive the dual problem for (52)–(54).

- Finally, the dual approach is generalizable for nonlinear separating hypersurfaces by means of replacing the dot product with a kernel function: $\mathbf{x}_i^\top \mathbf{x}_j$ is replaced with $K(\mathbf{x}_i, \mathbf{x}_j)$ in the definition of the matrix D in (65) [1]:

$$D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j).$$

  For example, $K(\mathbf{x}_i, \mathbf{x}_j)$ can be

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + 1)^p \quad \text{or} \quad K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right\}.$$

Let us make a few implementational remarks.

(1) The dual problem is a constrained maximization problem. To convert it to a more accustomed constrained minimization problem, just take $-q(\boldsymbol{\lambda})$ as an objective function: $q(\boldsymbol{\lambda}) \mapsto -q(\boldsymbol{\lambda})$.

(2) We can get rid of the equality constraint $\mathbf{y}^\top \boldsymbol{\lambda} = 0$ by incorporating it into the objective function. Let us express λ_n via λ_i's, $1 \le i \le n-1$:

$$\lambda_n = -\frac{1}{y_n} \sum_{i=1}^{n-1} y_i \lambda_i = -\frac{1}{y_n} \mathbf{y}_{1:(n-1)}^\top \lambda, \tag{69}$$

where $\mathbf{y}_{1:(n-1)} := (y_1, \ldots, y_{n-1})^\top$ and $\lambda := (\lambda_1, \ldots, \lambda_{n-1})^\top$). Next, we write D as a block matrix separating its last column and last row:

$$D = \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix},$$

or, in Matlab notaion:

```
D00 = D(1:end-1,1:end-1);
D01 = D(1:end-1,end);
D10 = D(end,1:end-1);
D11 = D(end,end);
```

Plugging (69) into the quadratic form $\boldsymbol{\lambda}^\top D \boldsymbol{\lambda}$ we get:

$$\begin{aligned}\boldsymbol{\lambda}^\top D \boldsymbol{\lambda} &= [\lambda^\top, -\frac{1}{y_n}\mathbf{y}_{1:(n-1)}^\top \lambda] \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix} \begin{bmatrix} \lambda \\ -\frac{1}{y_n}\mathbf{y}_{1:(n-1)}^\top \lambda \end{bmatrix} \\ &= \lambda^\top D_{00} \lambda - \frac{1}{y_n} \lambda^\top \mathbf{y}_{1:(n-1)} D_{10} \lambda - \frac{1}{y_n} \lambda^\top D_{01} \mathbf{y}_{1:(n-1)}^\top \lambda + \frac{1}{y_n^2} \lambda^\top \mathbf{y}_{1:(n-1)} D_{11} \mathbf{y}_{1:(n-1)}^\top \lambda.\end{aligned}$$

Hence the matrix of the quadratic form in the new constrained minimization problem is:

$$H = D_{00} + \frac{1}{y_n^2} \mathbf{y}_{1:(n-1)} \mathbf{y}_{1:(n-1)}^\top D_{11} - \frac{1}{y_n} \mathbf{y}_{1:(n-1)} D_{10} - \frac{1}{y_n} D_{01} \mathbf{y}_{1:(n-1)}^\top. \tag{70}$$

The linear term of $q(\boldsymbol{\lambda})$ is modified to:

$$\mathbf{c} := 1_{(n-1)\times 1} - \frac{1}{y_n}\mathbf{y}_{1:(n-1)}. \tag{71}$$

The resulting problem for $\lambda := (\lambda_1, \ldots, \lambda_{n-1})^\top$ becomes:

$$q(\lambda) = \frac{1}{2}\lambda^\top H\lambda - \mathbf{c}^\top\lambda \to \min \quad \text{subject to} \tag{72}$$

$$\lambda \geq 0, \tag{73}$$

$$-\frac{1}{y_n}\mathbf{y}_{1:(n-1)}^\top\lambda \geq 0. \tag{74}$$

(3) The matrix H is symmetric but not, in general, positive definite. To make sure that every step of the active-set method reduces the objective function, we add a multiple of the identity matrix to H. Let $-\mu_0$ is the most negative eigenvalue of H. Then the matrix $\tilde{H} := H + \mu I$ where $\mu > \mu_0$ (I set $\mu = \mu_0 + 1$ in my code) is symmetric positive definite. Then the modified KKT matrix is invertible, and its inverse is [7] (see Section 3.3):

$$K^{-1} = \begin{bmatrix} \tilde{H} & A^\top \\ A & 0 \end{bmatrix}^{-1} = \begin{bmatrix} \tilde{H}^{-1} + \tilde{H}^{-1}A^\top S^{-1}A\tilde{H}^{-1} & -\tilde{H}^{-1}A^\top S^{-1} \\ -S^{-1}A\tilde{H}^{-1} & S^{-1} \end{bmatrix}, \tag{75}$$

$$S = -A\tilde{H}^{-1}A^\top \quad \text{is the Schur compliment.} \tag{76}$$

Exercise Let $(\mathbf{p}^*, \boldsymbol{\lambda}^*)$ be the solution to the modified KKT system

$$\begin{bmatrix} \tilde{H} & A^\top \\ A & 0 \end{bmatrix} = \begin{bmatrix} -\mathbf{p} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \nabla f \\ 0 \end{bmatrix}.$$

Show that $\mathbf{p}^*$ is a descend direction i.e., $\nabla f^\top \mathbf{p} < 0$ provided that columns of $A^\top$ are linearly independent and $n < d$. *Hint: First try to get it yourself. If you get stuck, look into* [7].

5. Unconstrained optimization problem for classification problems

Classification problems are often much more complicated than those where two sample sets can be separated by a hyperplane. Often one first needs to map the input data to a feature space using a nonlinear map, find a dividing hyperplane there, and then use its pre-image in the input space as the classifier. A simple nonlinear benchmark classification problem is the Swiss roll shown in Fig. 6(a). This example is set so that it is suffices to define the feature space by taking polar angle of each data point $\mathbf{x}_i$, multiply by three (this operation will collapses all blue petals and all black petals, and take sines and cosines of these angles to avoid the issue with discontinuity and periodicity. In Matlab syntax, the map to the feature space is:

$$f(\mathbf{x}_i) := [\sin(3\phi(\mathbf{x})), \cos(3\phi(\mathbf{x}))]^\top, \tag{77}$$

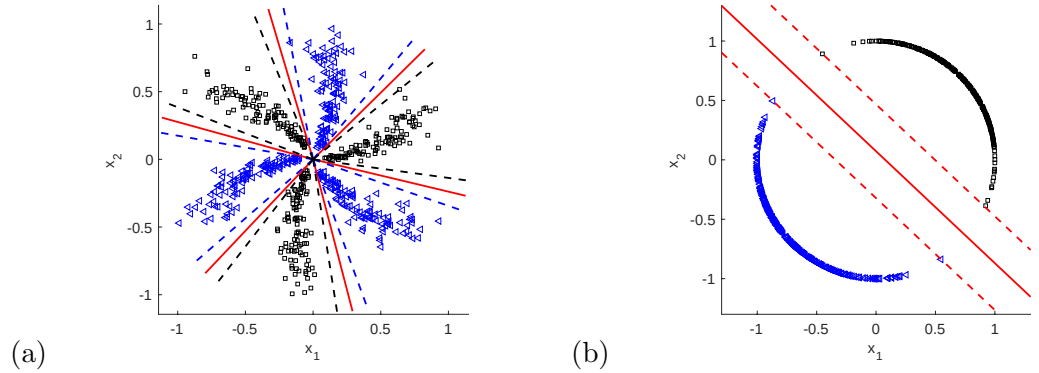


FIGURE 6. (a): Swiss roll dataset of 600 points and dividing rays (solid) together with margins (dashed). For "petal" k, the data are generated by adding a 2D Gaussian random variable with mean 0 and variance $0.1r$ to the curves $\phi_k(r) = {}^{k\pi}/_6 + 0.5r$. Here, (r, ϕ) are the polar coordinates. (b): The data mapped to the feature space by (77) where they are linearly separable.

where $\phi(\mathbf{x})$ is the polar angle of $\mathbf{x}$. Fig. 6(b) shows that this map to a 2D feature space is sufficient for making the data linearly separable. Finally, the SVM [1] is used to find the dividing lines and the margins that are mapped back to the data space (Fig. 6(a)).

While the solution in Fig. 6 looks nice, it is tuned only for this particular problem. Moreover, if we increase the twist in the curves around which we sample data points, e.g., we define $\phi_k(r) = {}^{k\pi}/_6 + r$ instead of $\phi_k(r) = {}^{k\pi}/_6 + 0.5r$, this classification method will break. We can do SVN with soft margins, but this will not help to adjust the shape of dividing surfaces. Adding r to the feature space won't help either (see Fig. 7).

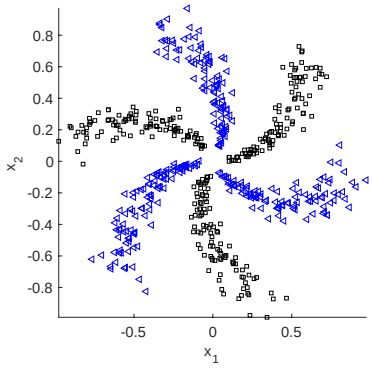
We can custom-design a special feature space specifically for this problem, but this will not help to solve other problems. This problem is just a toy problem that is only of interest as a test problem for generic techniques.

The approach that proved its power is based on the use of deep neural networks (NN). One can set up a DNN with three hidden layers to solve this problem[3] as follows. The labels for class i will be the standard basis vector $\mathbf{e}_i \in \mathbb{R}^K$ where K is the number of classes; $K = 2$ in our case. Let the nonlinear mapping to the feature space be

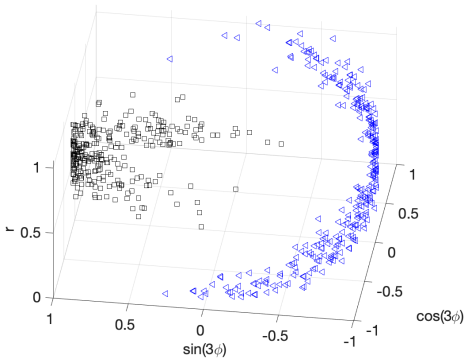
$$\Phi(\mathbf{x}) = \sigma(A_3\sigma(A_2\sigma(A_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3). \tag{78}$$

The nonlinear function σ (applied entrywise) can be chosen to be ReLU which is popular in image recognition, or a smooth function like $(1 + e^{-x})^{-1}$ or $\tanh(x)$ that are more preferable in other applications like solving PDEs and MD sampling. We expect the data in the feature space to be linearly separable by $A_4\Phi(\mathbf{x})+\mathbf{b}_4$ where A_4 is $2\times d_3$, and $\mathbf{b}_4 \in \mathbb{R}^2$.

[3] I thank Avi Schwarzschild (AMSC, UMD) for this example who provided this example in his final report for AMSC664.



(a)



(b)

FIGURE 7. (a): Swiss roll dataset of 600 points. For "petal" k, the data are generated by adding a 2D Gaussian random variable with mean 0 and variance $0.1r$ to the curves $\phi_k(r) = {}^{k\pi}/_6 + r$. Here, (r, ϕ) are the polar coordinates. (b): The data mapped to the feature space: $\mathbf{x} \mapsto [\sin(3\phi(\mathbf{x})), \cos(3\phi(\mathbf{x})), r]^\top$ are not separable by a hyperplane.

The matrices A_1, A_2, A_3 have dimensions $d_1 \times 2$, $d_2 \times d_1$, and $d_3 \times d_2$ respectively, and vectors $\mathbf{b}_i \in \mathbb{R}^{d_i}$, $i = 1, 2, 3$. The numbers d_i can be large (e.g., between 100 and 1000). The model described here is called a *multilayer perceptron*. The loss function can be chosen as

$$\mathcal{L}(w) := \frac{1}{N} \sum_{i=1}^{n} \|h(\mathbf{x}_i; w) - \mathbf{y}_i\|^2, \tag{79}$$

where $w := \{(A_j, \mathbf{b}_j)\}_{i=1}^4$ is the vector of parameters comprising all matrices A_j and shifts $\mathbf{b}_j$, and $h(\mathbf{x}; w) := A_4\phi(\mathbf{x}; \tilde{w}) + \mathbf{b}_4$ is the prediction function, $\tilde{w} := \{(A_j, \mathbf{b}_j)\}_{i=1}^3$.

This example demonstrates how the classification problem can be reduced to an unconstrained minimization problem. Moreover, the objective function is of the form of sum of squares of nonlinear functions taking advantages of which can be exploited.

6. An overview of methods for unconstrained optimization

Key references for this section are [6] and [2]. The most widely used methods for unconstrained optimization are summarized in Fig. 8. Methods for unconstrained optimization can be divided into three categories described in the three respective subsections below.

6.1. **Derivative-free methods.**

Derivative-free methods only require function evaluations. A well-known method in this category is the Nedler-Mead downhill simplex method [6] (Section 9.5). Matlab's function `fminsearch`, very handy for small problems, employs a derivative-free method.

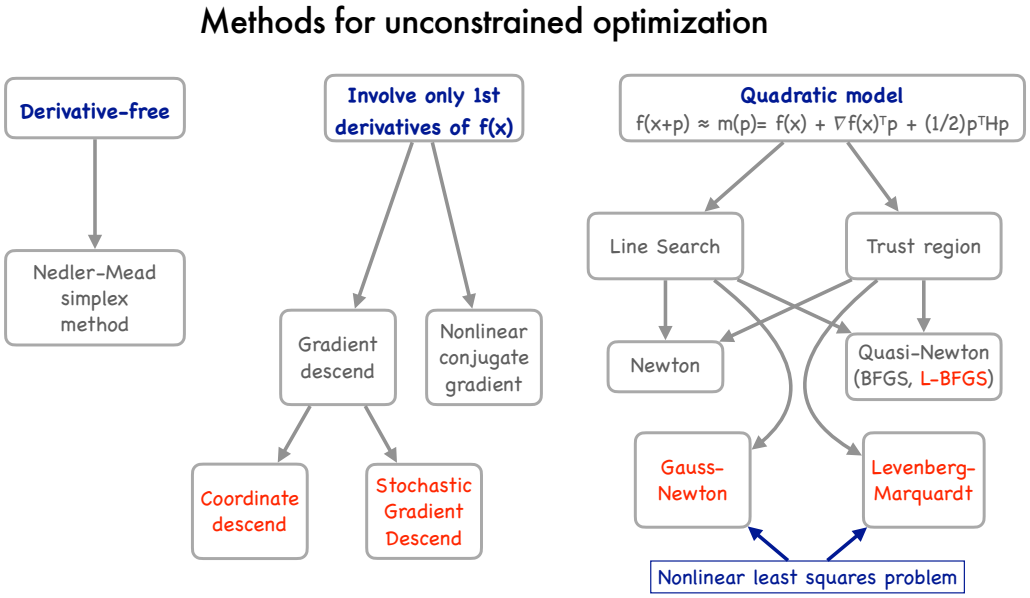


FIGURE 8. The most widely used methods for unconstrained optimization and their classification.

6.2. **Methods involving only first derivative evaluation.** The pivotal method in this category is *gradient descend* a.k.a. *steepest descend*. **Please read** D. Bindel's notes (Sections 8, 9, and 10) about gradient descend and related issues. This is the most straightforward minimization method (as the forward Euler method for solving ODEs), which, in the context of solving practical optimization problems can be attacked from two opposite directions. On one hand, its convergence is slow:

$$\|\mathbf{x}_k - \mathbf{x}^*\| = O(\rho^k) \quad \text{where} \quad \rho = 1 - \frac{2}{1+\kappa(A)},$$

where $A = \nabla f^2(\mathbf{x}^*)$ is the Hessian matrix of f at the local minimizer $\mathbf{x}^*$ and $\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$. (Note that if $\mathbf{x}^*$ is a nondegenerate local minimizer, then A is symmetric positive definite). An example depicting how the gradient descend requires many iterations to converge when the problem is poorly scaled is shown in Fig. 9. Methods for unconstrained optimization can be divided into three categories described in the three respective subsections below. Methods that use or build second derivative information converge significantly faster. Another group of methods in this category is the one of *nonlinear conjugate gradient* methods whose iterations are almost as cheap but they tend to perform better, especially on badly scaled problems. This group is inspired by the conjugate gradient method for solving large linear systems of algebraic equations with symmetric positive definite matrices.

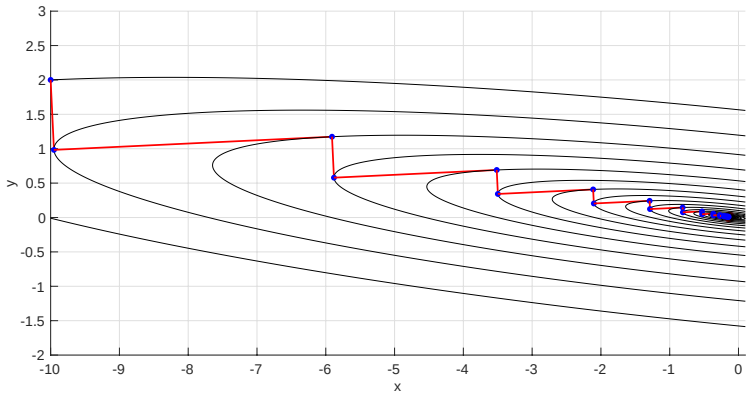


FIGURE 9. Iterations of the gradient descend starting from $\mathbf{x}_0 = (-10, 2)^\top$ and $f(\mathbf{x}) = \frac{1}{2}x_1^2 + 4x_1x_2 + 20x_2^2 + 0.1x_1 + 0.2x_2$.

However, slow convergence of gradient descend may be advantageous in some situations (see Section 9 in Bindel's notes linked above). On the other hand, the gradient of a function may be to costly to evaluate in large-scale optimization problems arising in machine learning. *Stochastic gradient descend* became a method of choice in TensorFlow and PyTorch. It is designed for case where the objective function is a sum of functions for each sample, it gives a good and cheap estimate for the gradient at each step, and its efficiency is independent of the number of samples. The other method used in large-scale problems is *coordinate descend*. While this method, in contrast with gradient descend, is not guaranteed to converge to a stationary point of $f(\mathbf{x})$ (i.e., a point $\mathbf{x}^*$ such that $\nabla f(\mathbf{x}^*) = 0$ if f is nonconvex [8] (there is an example of a nonconvex continuously differentiable function of three variables for which a cyclic coordinate descend method, with step lengths chosen by exact one-dimensional minimization, cycles without converging to a stationary point). The appeal of coordinate descend is its easy implementation and cheap iterations. These facts made this method a subject of active contemporary research.

6.3. **Methods using quadratic models.** Finally, there are methods that a based on approximating f by a quadratic model at each step:

$$f(\mathbf{x} + \mathbf{p}) \sim m(\mathbf{p}) := f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top H \mathbf{p},$$

where H is some approximation to the Hessian $\nabla^2 f(\mathbf{x})$. These methods are divided into two families according the strategy they use for choosing step direction and step length at their iterations: *line search methods* and *trust region methods*. Line search methods first pick search direction $\mathbf{p}$ and then choose step length. Certainly, all methods in the previous item belong to this family. Trust region methods iterate the other way around: they first pick the upper bound for step length Δ and then solve constraint minimization problem with the objective function being the current model $m(\mathbf{p})$ and the constraint $\|\mathbf{p}\| \le \Delta$.

Next, for each of these strategies, methods are classified according to the way they obtain the matrix H to build the quadratic model. If the dimensionality of the problem is not too large and the true Hessian is available and not too expensive to evaluate, *Netwon's method* is a good method to try. For finding local minima of Lennard-Jones clusters (for dimensions $\lesssim 45$), I evaluated the Hessian by finite differences: the exact Hessian was available but its evaluations was taking flops. For higher-dimensional and/or more complicated problems, evaluation of the Hessian becomes impractical. Then a good choice is *BFGS*, a quasi-newton method motivated by the secant method for solving 1D nonlinear equations. BFGS updates its approximate for the Hessian at each iteration incorporating the gained knowledge regarding the change in the gradient with the change in the argument (see [6], Section 2.2):

$$\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) \approx H(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k).$$

There is a version of BFGS called *L-BFGS* (L stands for limited memory) that is designed for handling large-scale problems.

If the objective function has a special structure, for example

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{n} \|f_i(\mathbf{x})\|^2,$$

it can be exploited for a cheap approximation for the Hessian. We will discuss two methods that do it: it Gauss-Newton (line search) and *Levenberg-Marquardt* (trust region).

7. Stochastic gradient descend

The key reference for this section is [2]. *Stochastic gradient descend (SG)* originates from the work by Robbins and Monroe (1951)"A stochastic approximation method" [9]. SG is designed for minimizing functions of the form

$$f(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}), \quad \text{N is large.} \tag{80}$$

One simple version of SG is runs as follows. One picks a *butch*, a random subset of indices $\mathcal{S}_k \subset \{1, \dots, N\}$ and makes a step:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \left[\frac{1}{|\mathcal{S}_k|}\sum_{j\in\mathcal{S}_k} \nabla f_j(\mathbf{x}_k)\right], \tag{81}$$

A 1D example with batch size 1 is shown in Fig. 10. The expression in the square brackets is a stochastic approximation to $\nabla f(\mathbf{x}_k)$. Why is this a reasonable approximation? Often, there are many samples $\mathbf{x}_i$ in the training set (each sample $\mathbf{x}_i$ defines the corresponding f_i), and these samples can be split into groups consisting of similar samples. In this case, using just a subset of samples for estimating the gradient will give almost as good result but will be cheaper by the factor m/N. Moreover, typically, the dimensionality of machine learning optimization problems is very large. Hence, evaluating all N gradients may cause a computer memory problem.

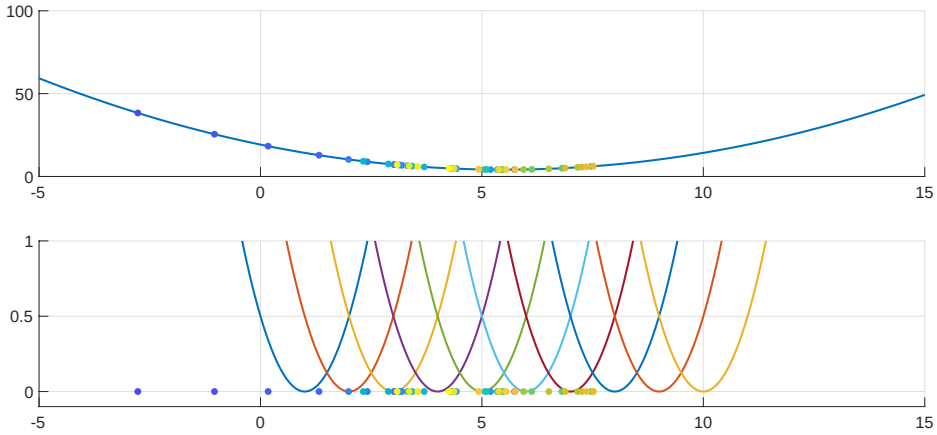


FIGURE 10. An example illustrating how stochastic gradient descend with a constant step length first rushes to the region where the minima of the individual functions are and then bounces around forever. Here, $f(x) = {}^1/_{20}\sum_{i=1}^{10}(x-i)^2$, step length is 0.3, batch size is 1. The iteration numbers are indicated by the `parula` colormap going from blue to yellow.

7.1. **Expected decrease of f under SG iterations: assumptions and basic lemmas.** The SG algorithm offers a lot of flexibility for choosing batch sizes $|\mathcal{S}_k|$ and stepsizes α_k. These can be fixed or variable and chosen according to some strategy enhancing performance. Moreover, the choice the recipe for generating the direction for the step gives an additional flexibility. For example, one can calculate the direction of a step from scratch at each step, or incorporate the previously used directions, or even build up a stochastic estimate for the inverse Hessian and make the method a stochastic quasi-Newton. We will denote the stochastic vector in the direction opposite to the direction of step k by $g(\mathbf{x}_k;\xi_k)$ where ξ_k is a random variable (generally, a vector random variable). For a simple SG with batch size 1,

$$\mathbf{g}(\mathbf{x}_k;\xi_k) = \nabla f_{\xi_k}(\mathbf{x}_k).$$

For SG with batch size n_k we have

$$\mathbf{g}(\mathbf{x}_k;\xi_k) = \frac{1}{n_k}\sum_{i=1}^{n_k}\nabla f_{\xi_k,i}(\mathbf{x}_k).$$

For a stochastic quasi-Newton version,

$$\mathbf{g}(\mathbf{x}_k;\xi_k) = H_k\frac{1}{n_k}\sum_{i=1}^{n_k}\nabla f_{\xi_k,i}(\mathbf{x}_k).$$

In broad strokes, the SG algorithm goes as follows:

Algorithm 1: SG algorithm

Initialization: Choose an initial vector $\mathbf{x}_0$.
for $k = 1, 2, ...$ **do**
  Generate a realization of the random variable ξ_k;
  Compute a stochastic vector $\mathbf{g}(\mathbf{x}_k, \xi_k)$;
  Choose a stepsize α_k;
  Set a new iterate as $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}(\mathbf{x}_k, \xi_k)$;
end

In order to analyze SG we need to make some assumptions on niceness of the objective function f.

Assumption 1. *$f : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable and $\nabla f : \mathbb{R}^d \to \mathbb{R}^d$ is Lipschitz-continuous with constant L, i.e.,*

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \le L\|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d, \tag{82}$$

where $\|\cdot\|$ is the 2-norm.

Using (82) we can obtain a quadratic bound for the growth of f:

$$\begin{aligned} f(\mathbf{x}) &= f(\mathbf{y}) + \int_0^1 \frac{df}{d\alpha}(\mathbf{y} + \alpha(\mathbf{x} - \mathbf{y}))d\alpha = f(\mathbf{y}) + \int_0^1 \nabla f(\mathbf{y} + \alpha(\mathbf{x} - \mathbf{y}))^\top (\mathbf{x} - \mathbf{y})d\alpha \\ &= f(\mathbf{y}) + \int_0^1 \left(\nabla f(\mathbf{y}) + \nabla f(\mathbf{y} + \alpha(\mathbf{x} - \mathbf{y})) - \nabla f(\mathbf{y})\right)^\top (\mathbf{x} - \mathbf{y})d\alpha \\ &\le f(\mathbf{y}) + \nabla f(\mathbf{y})^\top (\mathbf{x} - \mathbf{y}) + \int_0^1 L\alpha\|\mathbf{x} - \mathbf{y}\|^2 d\alpha. \end{aligned}$$

Performing integration in the last identity, we obtain:

$$\boxed{f(\mathbf{x}) \le f(\mathbf{y}) + \nabla f(\mathbf{y})^\top (\mathbf{x} - \mathbf{y}) + \frac{1}{2}L\|\mathbf{x} - \mathbf{y}\|^2 \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d.} \tag{83}$$

This identity allows us to establish the following

Lemma 1. *Under Assumption 1, iterates of Algorithm 1 satisfy*

(84)

$$\boxed{\mathbb{E}_{\xi_k}[f(\mathbf{x}_{k+1})] - f(\mathbf{x}_k) \le -\alpha_k \nabla f(\mathbf{x}_k)^\top \mathbb{E}_{\xi_k}[\mathbf{g}(\mathbf{x}_k, \xi_k)] + \frac{L\alpha_k^2}{2}\mathbb{E}_{\xi_k}[\|\mathbf{g}(\mathbf{x}_k, \xi_k)\|^2] \quad \forall k \in \mathbb{Z}_+.}$$

Proof. By (83), the iterates satisfy

$$\begin{aligned} f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) &\le \nabla f(\mathbf{x}_k)^\top (\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{1}{2}L\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2 \\ &\le -\alpha_k \nabla f(\mathbf{x}_k)^\top \mathbf{g}(\mathbf{x}_k, \xi_k) + \frac{L\alpha_k^2}{2}\|\mathbf{g}(\mathbf{x}_k, \xi_k)\|^2. \end{aligned}$$

Taking expectations with respect to ξ_k and noting that $\mathbf{x}_k$ is independent of ξ_k, we obtain (84). □

A good news is that if $-\mathbb{E}_{\xi_k}[\mathbf{g}(\mathbf{x}_k, \xi_k)]$ is a descend direction for f, i.e.,

$$\nabla f(\mathbf{x}_k)^\top \mathbb{E}_{\xi_k}[\mathbf{g}(\mathbf{x}_k, \xi_k)] > 0,$$

then for sufficiently small step length α we expect that f will decrease as a result the step $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{g}(\mathbf{x}_k, \xi_k)$. To ensure that the random directions guarantee some minimal decrease we state

Assumption 2. *There exists a constant $\mu > 0$ such that for all $k \in \mathbb{Z}_+$*

$$\nabla f(\mathbf{x}_k)^\top \mathbb{E}_{\xi_k}[\mathbf{g}(\mathbf{x}_k, \xi_k)] \geq \mu \|\nabla f(\mathbf{x}_k)\|^2. \tag{85}$$

A bad news is that there is a harmful effect of the second term that is always positive. In order to limit its effect, we make one more

Assumption 3. *There exist constants $M \geq 0$ and $M_G \geq \mu^2 > 0$ (μ is from (85)) such that for all $k \in \mathbb{Z}_+$*

$$\mathbb{E}_{\xi_k}[\|\mathbf{g}(\mathbf{x}_k, \xi_k)\|^2] \leq M + M_G \|\nabla f(\mathbf{x}_k)\|^2. \tag{86}$$

Example Let us see how Assumptions 2 and 3 apply to the 1D example in Fig. 10 where

$$f(x) = \frac{1}{20} \sum_{i=1}^{10} (x - i)^2 \quad \text{and} \quad g = -(x - i) \quad \text{where} \quad i = \texttt{randi}(10).$$

Let us start with Assumption 2. We have: $\nabla f(x) = x - 5.5$. The distribution of g is: each of the values -4.5, -3.5, $\ldots$, 3.5, 4.5 is taken with the same probability 0.1. Hence $\mathbb{E}_i[g(x, i)] = x - 5.5$ as well, hence $\mu = 1$ works.

The stationary point is $x^* = 5.5$. The distribution of g is: each of the values -4.5, -3.5, $\ldots$, 3.5, 4.5 is taken with the same probability 0.1. Hence, the expectation of g^2 is

$$\mathbb{E}[g(5.5, i)] = 0.2(4.5^2 + 3.5^2 + 2.5^2 + 1.5^2 + 0.5^2) = 8.25.$$

Hence, $M = 8.25$. Now, $\nabla f(x) = x - 5.5$ and $\|\nabla f(x)\|^2 = x^2 - 11x + 30.25$, while $g(x, i) = x - i$ and

$$\mathbb{E}[\|g(x, i)\|^2] = 0.1 \sum_{i=1}^{10} x^2 - 2ix + i^2 = x^2 - 11x + 38.5.$$

Plugging this into (86) we get:

$$x^2 - 11x + 38.5. \leq 8.25 + M_G(x^2 - 11x + 30.25).$$

Hence, it suffices to pick $M_G = 1$. Note that $M_G = \mu^2$, hence the requirement that $M_G \geq \mu^2$ is satisfied.

Assumptions 2 and 3 allow us further elaborate the expected decrease of f under SG iterations.

Lemma 2. *Under Assumptions 1, 2, and 3, the iterates of SG satisfy*

$$\boxed{\mathbb{E}_{\xi_k}[f(\mathbf{x}_{k+1})] - f(\mathbf{x}_k) \leq -\left(\mu - \frac{1}{2}\alpha_k L M_G\right)\alpha_k \|\nabla f(\mathbf{x}_k)\|^2 + \frac{1}{2}\alpha_k^2 L M.} \tag{87}$$

Proof. Plugging (85) and (86) to (84), we git:

$$\begin{aligned}\mathbb{E}_{\xi_k}[f(\mathbf{x}_{k+1})] - f(\mathbf{x}_k) &\leq -\mu\alpha_k\|\nabla f(\mathbf{x}_k)\|^2 + \frac{1}{2}\alpha_k^2 L\left(M + M_G\|\nabla f(\mathbf{x}_k)\|^2\right)\\ &= -\left(\mu - \frac{1}{2}\alpha_k L M_G\right)\alpha_k\|\nabla f(\mathbf{x}_k)\|^2 + \frac{1}{2}\alpha_k^2 L M\end{aligned}$$

as desired. □

Note that if α_k is small enough, the first term in the right-hand side of (87) is negative. The second term is always positive. These considerations are crucial for designing SG methods.

7.2. Convergence of SG for strongly convex objective functions.

Assumption 4. *The objective function $f : \mathbb{R}^d \to \mathbb{R}$ is strongly convex, i.e., there exists a constant $c > 0$ such that*

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) + \frac{c}{2}\|\mathbf{y} - \mathbf{x}\|^2 \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d. \tag{88}$$

Hence, f has a unique minimizer $\mathbf{x}^* \in \mathbb{R}^d$ with $f^* := f(\mathbf{x}^*)$.

Assumption 4 guarantees that f grows away from its minimizer $\mathbf{x}^*$ at least as fast as the convex quadratic function $\frac{c}{2}\|\mathbf{x} - \mathbf{x}^*\|^2$. Note that the requirement of strong convexity is stronger than the one of strict convexity. If f is twice continuously differentiable, strong convexity means that its Hessian is positive definite everywhere, and its eigenvalues are bounded from below by $c > 0$, while positive definiteness of the Hessian only will suffice to guarantee strict convexity. For example, the function $y = \sqrt{x^2 + 1}$ whose graph is the hyperbola lying above its two slant asymptotes $y = \pm x$ is strictly convex but not strongly convex. Its second derivative $y'' = (1 + x^2)^{-3/2}$ is positive everywhere but is approaches 0 as $|x| \to \infty$.

Comparing (89) and (83) we observe that $c < L$ (L is the Lipschitz constant for ∇f).

Assumption 4 also allows us to bound so called *optimality gap*.

Proposition 1. *Let f satisfy Assumption 4. Then*

$$2c\,(f(\mathbf{x}) - f^*) \leq \|\nabla f(\mathbf{x})\|^2 \quad \forall \mathbf{x} \in \mathbb{R}^d. \tag{89}$$

Proof. Let us fix $\mathbf{x}$ and consider the quadratic model

$$q(\mathbf{y}) := f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) + \frac{c}{2}\|\mathbf{y} - \mathbf{x}\|^2.$$

$q(\mathbf{y})$ has the unique minimizer $\hat{\mathbf{y}} := \mathbf{x} - \frac{1}{c}\nabla f(\mathbf{x})$ with

$$q(\hat{\mathbf{y}}) = f(\mathbf{x}) - \frac{1}{2c}\|\nabla f(\mathbf{x})\|^2.$$

Therefore, setting $\mathbf{y} = \mathbf{x}^*$ in (88), for any $\mathbf{x} \in \mathbb{R}^d$ we have:

$$f^* \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{x}^* - \mathbf{x}) + \frac{1}{2}\|\mathbf{x}^* - \mathbf{x}\| \geq f(\mathbf{x}) - \frac{1}{2c}\|\nabla f(\mathbf{x})\|^2.$$

Hence, (89) readily follows. □

7.2.1. *Fixed stepsize.* Now we are ready to establish the first convergence result for SG with fixed stepsize for a strongly convex objective function. It is clear from (87) that the iterates will not be able to converge to the minimizer but will bounce around in its neighborhood as the first term in (87) tends to zero as we approach $\mathbf{x}^*$ while the second term remains constant. We will denote the *total expectation* of $f(\mathbf{x})$ for any $k \in \mathbb{Z}_+$ by

$$\mathbb{E}[f(\mathbf{x}_k)] := \mathbb{E}_{\xi_1}\mathbb{E}_{\xi_2}\ldots\mathbb{E}_{\xi_{k-1}}[f(\mathbf{x}_k)].$$

Theorem 2. *Under Assumptions 1, 2, 3, and 4, suppose that the SG method (Algorithm 1) is run with a fixed stepsize α satisfying*

$$0 < \alpha \leq \frac{\mu}{LM_G}. \tag{90}$$

Then the expected optimality gap satisfies the following inequality for all $k \in \mathbb{Z}_+$:

$$\boxed{\mathbb{E}[f(\mathbf{x}_k) - f^*] \leq \frac{\alpha LM}{2c\mu} + (1 - \alpha c\mu)^{k-1}\left(f(\mathbf{x}_0) - f^* - \frac{\alpha LM}{2c\mu}\right) \to \frac{\alpha LM}{2c\mu}.} \tag{91}$$

Roughly speaking, this theorem says that the SG iterates with fixed stepsize will reach a certain neighborhood of the optimal point and bounce there forever provided that the stepsize is not too large.

Proof. Plugging the bound (90) on the stepsize to (87) (the result of Lemma 2) and then inserting the optimality gap (89) we get:

$$\begin{aligned}
\mathbb{E}_{\xi_k}[f(\mathbf{x}_{k+1})] - f(\mathbf{x}_k) &\leq -\left(\mu - \frac{1}{2}\alpha LM_G\right)\alpha\|\nabla f(\mathbf{x}_k)\|^2 + \frac{1}{2}\alpha^2 LM \\
&\leq -\frac{\alpha\mu}{2}\|\nabla f(\mathbf{x}_k)\|^2 + \frac{1}{2}\alpha^2 LM \\
&\leq -\alpha c\mu(f(\mathbf{x}_k) - f^*) + \frac{1}{2}\alpha^2 LM.
\end{aligned}$$

Subtracting f^* from both sides, rearranging terms, and taking total expectations, we get:

$$\mathbb{E}[f(\mathbf{x}_{k+1}) - f^*] \leq (1 - \alpha c\mu)\mathbb{E}[f(\mathbf{x}_k) - f^*] + \frac{1}{2}\alpha^2 LM.$$

Subtracting $\frac{\alpha LM}{2\mu c}$ from both sides we get:

$$\mathbb{E}[f(\mathbf{x}_{k+1}) - f^*] - \frac{\alpha LM}{2\mu c} \leq (1 - \alpha c\mu)\left(\mathbb{E}[f(\mathbf{x}_k) - f^*] - \frac{\alpha LM}{2\mu c}\right). \tag{92}$$

Note that

$$0 < \alpha c\mu \leq \frac{c\mu^2}{LM_G} \leq \frac{c\mu^2}{L\mu^2} = \frac{c}{L} \leq 1.$$

Here we used the assumption that $M_G \geq \mu^2$(see Assumption 3) you might have been wondering about what is it for. Therefore,

$$0 \leq (1 - \alpha c\mu) < 1.$$

Applying (92) repeatedly we obtain (91). This completes the proof. □

7.2.2. *Decreasing step size.* As we have proven, SG with fixed step size does not converge to the minimizer. In order to achieve convergence, we need to reduce stepsize as we progress but not too fast: a condition for stepsizes α_k proven in [9] is

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty. \tag{93}$$

Note that $\alpha_k = k^{-1}$ satisfies (93). Moreover, if we want to reduce stepsize by factor of 2, we need to do it on a certain schedule. One such a schedule is: do m_0 steps of size α, then do $m_1 = m_0$ steps of size $2^{-1}\alpha$, ..., then do $m_k = 2^k/k$ steps of size $2^{-k}\alpha$, and so on. Then

$$\sum_{k=0}^{\infty} m_0 \frac{2^k}{k} \frac{\alpha}{2^k} = \infty, \quad \sum_{k=0}^{\infty} m_0 \frac{2^k}{k} \frac{\alpha^2}{2^{2k}} = \sum_{k=0}^{\infty} m_0 \frac{\alpha^2}{k 2^k} < \infty.$$

The following theorem offers another schedule for stepsize reduction for which error decay goes as $O(1/k)$.

Theorem 3. *Under Assumptions 1, 2, 3,and 4, suppose that the SG method 1 is run with a stepsize sequence α_k, $k \in \mathbb{Z}_+$, such that*

$$\alpha_k = \frac{\beta}{\gamma + k} \quad \textit{for some} \quad \beta > \frac{1}{c\mu} \quad \text{and} \quad \gamma > 1 \quad \textit{such that} \quad \alpha_0 \leq \frac{\mu}{LM_G}. \tag{94}$$

Then for all $k \in \mathbb{Z}_+$, the expected optimality gap satisfies

$$\boxed{\mathbb{E}[f(\mathbf{x}_k) - f^*] \leq \frac{\nu}{\gamma + k},} \tag{95}$$

where

$$\nu := \max\left\{ \frac{\beta^2 LM}{2(\beta c\mu - 1)}, (\gamma - 1)(f(\mathbf{x}_0) - f^*) + \frac{1}{2}\frac{\gamma M \mu^2}{LM_G^2} \right\}. \tag{96}$$

Proof. Repeating the start of the proof of Theorem 2 we obtain the inequality:

$$\mathbb{E}_{\xi_k}[f(\mathbf{x}_{k+1})] - f(\mathbf{x}_k) \leq -\alpha_k c\mu (f(\mathbf{x}_k) - f^*) + \frac{1}{2}\alpha_k^2 LM$$

Subtracting f^* from both sides, rearranging terms, and taking total expectations, we get:

$$\mathbb{E}[f(\mathbf{x}_{k+1}) - f^*] \leq (1 - \alpha_k c\mu)\mathbb{E}[f(\mathbf{x}_k) - f^*] + \frac{1}{2}\alpha_k^2 LM.$$

Now we proceed by induction. It follows from definition of ν that (95) holds for $k = 0$:

$$\mathbb{E}[f(\mathbf{x}_1) - f^*] \leq \frac{\gamma - 1}{\gamma}[f(\mathbf{x}_0) - f^*] + \frac{1}{2}\frac{M\mu^2}{LM_G^2} \leq \frac{\nu}{\gamma}$$

Induction assumption: (95) holds for k:

$$\mathbb{E}[f(\mathbf{x}_k) - f^*] \le \frac{\nu}{\gamma + k}.$$

Induction step:

$$\begin{aligned}
\mathbb{E}[f(\mathbf{x}_{k+1}) - f^*] &\le (1 - \alpha_k c\mu)\mathbb{E}[f(\mathbf{x}_k) - f^*] + \frac{1}{2}\alpha_k^2 LM \\
&= \left(1 - \frac{\beta c\mu}{\gamma + k}\right)\frac{\nu}{\gamma + k} + \frac{1}{2}\frac{\beta^2}{(\gamma + k)^2}LM \\
&= \frac{\gamma + k - 1}{(\gamma + k)^2}\nu \underbrace{- \frac{\beta c\mu - 1}{(\gamma + k)^2}\nu + \frac{1}{2}\frac{\beta^2}{(\gamma + k)^2}LM}_{< 0 \text{ by definition of } \nu} \\
&\le \frac{\gamma + k - 1}{(\gamma + k)^2}\nu \\
&< \frac{\nu}{\gamma + k + 1} \quad \text{as} \quad (\gamma + k)^2 > (\gamma + k + 1)(\gamma + k - 1).
\end{aligned}$$

This completes the proof. □

Example Fig. 11 shows an example of application of the SG algorithm to

$$f(x) = \frac{1}{2n}\sum_{i=1}^{n}(x - i)^2 \tag{97}$$

with $n = 10$, initial guess $x_0 = -5$, batch size 1, and stepsizes reduced according to the following schedule:

```
step = 0.3;
N = 15;
NN = 10;
for ii = 1 : NN
    s = step/2^ii;
    nsteps = ceil(N*2^ii/ii);
    for i = 1 : nsteps
        k = randi(n);
        g = grad(k,x);
        x = x - g*s;
    end
end
```

Then the expectations $\mathbb{E}[f(x_k) - f^*]$ is estimates as the average over 10^5 runs of this algorithm.

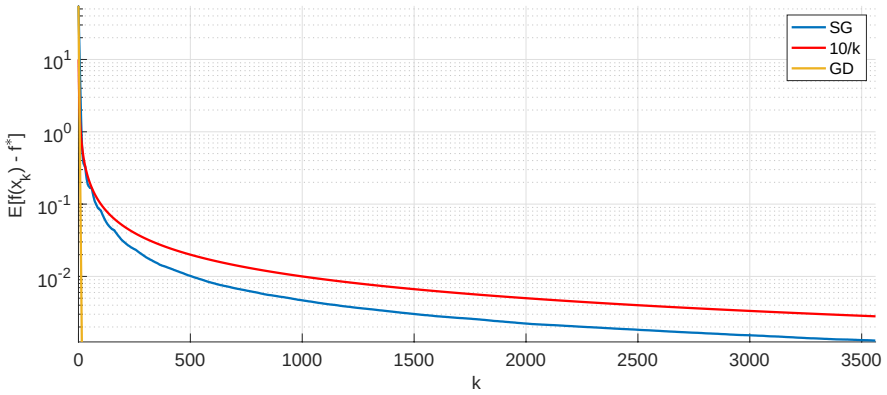


FIGURE 11. Decay of the optimality gap for SG applied to (97) with $n = 10$ and batch size 1. The graph of $10/k$ is shown in red for comparison. The regular gradient descend reduces the optimality gap to the same value in 16 iterations (see the yellow plot).

7.3. **SG for nonconvex objective functions.** If the objective function is not convex, then, under the rest of assmptions that we have made for the strongly convex case, there is a subsequence of iterates that at which the expected norm squared of the gradient approaches zero, i.e., a subsequence of iterates approaches a stationary point. If we amplify smoothness requirements for f, the iterates will approach a stationary point. This is summarized in the following

Theorem 4. *Under Assumptions 1, 2, and 3, suppose that the SG algorithm is run with stepsizes satisfying (93). Then*

$$\liminf_{k\to\infty} \mathbb{E}[\|\nabla f(\mathbf{x}_k)\|^2] = 0. \tag{98}$$

If, in addition f is twice continuously differentiable, and the mapping $\mathbf{x} \mapsto \|\nabla f(\mathbf{x})\|^2$ has Lipschitz-continuous derivatives, then

$$\lim_{k\to\infty} \mathbb{E}[\|\nabla f(\mathbf{x}_k)\|^2] = 0. \tag{99}$$

I am referring an interested reader to [2] (Section 4.3) to look up the proof.

8. Second order methods

8.1. **Motivation: scale invariance.** One might wonder, what is the sense to use stochastic optimization methods making use of second derivatives approximations, while the rate of convergence is going to be sublinear anyway? Well, these methods tend to improve rate constants. More importantly, these methods are able to reduce adverse effects of ill-conditioning and poor scaling. Let us illustrate an adverse effect of poor scaling on an example.

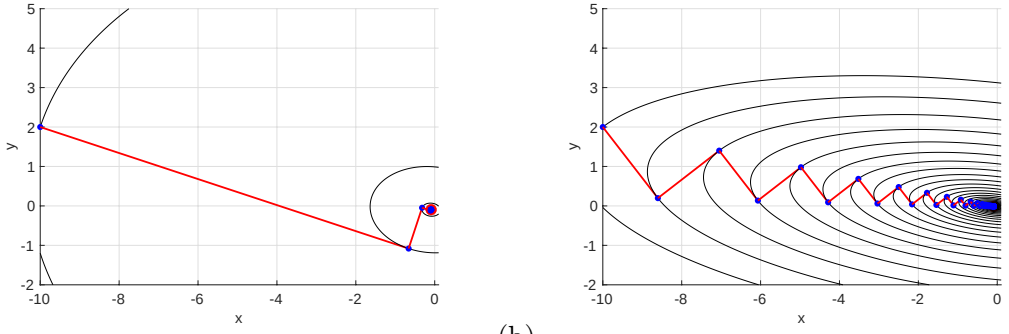
Example Suppose we would like to minimize two 2D quadratic functions

$$f_i(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A_i \mathbf{x} + \mathbf{b}^\top \mathbf{x}, \quad i = 1, 2, \quad \text{where} \tag{100}$$
$$A_1 = \begin{bmatrix} 1 & 0.1 \\ 0.1 & 2 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 1 & 1 \\ 1 & 10 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}.$$

We set the initial guess to $\mathbf{x}_0 = [-10, 2]^\top$ and iterate the gradient descend method with exact line search

$$x_{k+1} = x_k - \alpha_k \nabla f_i(\mathbf{x}_k) \quad \text{where} \quad \alpha_k = \frac{\nabla f(\mathbf{x}_k)^\top \nabla f(\mathbf{x}_k)}{\nabla f_i(\mathbf{x}_k)^\top A_i \nabla f_i(\mathbf{x}_k)}$$

until $\|\nabla f_i(x_k)\| < 10^{-10}$. Note that the eigenvalues of A_1 are 0.9901 and 2.0099 while the eigenvalues of A_2 are 0.8902 and 10.1098. Hence, none of the minimization problems is poorly scaled. Nonetheless, it takes 14 and 144 iterations to meet the convergence criterion for f_1 and f_2 respectively — see Fig. 12. In contrast, Newton's iteration proceeds



(a) (b)

FIGURE 12. Convergence of gradient descend for the functions given by (100) with tolerance set to 10^{-10}. (a): $i = 1$: 14 iterations. (b): $i = 2$: 144 iterations.

by making steps to the minimizer of the quadratic model (in our case, the model is the function itself)

$$\mathbf{x}_1 = \mathbf{x}_0 - A_i^{-1} \nabla f_i(\mathbf{x}_0) = \mathbf{x}_0 - A_i^{-1}(A_i \mathbf{x}_0 + \mathbf{b}) = -A_i^{-1}\mathbf{b}$$

meaning that it converges in one iteration in both cases. The conjugate gradient algorithm (see [6], Chapter 5) will converge in two iterations in both cases.

The conjugate gradient (CG) method is an iterative solver for linear systems $A\mathbf{x} - \mathbf{b}$, or, equivalently, optimization method for $f(\mathbf{x}) := \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{b}^\top \mathbf{x}$, where A is symmetric positive definite. It generates a sequence of directions $\mathbf{p}_k$ conjugate with respect the matrix A, i.e., $\mathbf{p}_k^\top A \mathbf{p}_j = 0$ for all $j \neq k$ starting from the steepest descend direction: $\mathbf{p}_0 = -\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$. All directions $\mathbf{p}_k$, $k \geq 1$, are linear combinations of $\mathbf{p}_{k-1}$ and the steepest descend direction $-\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ with weights chosen to ensure conjugacy of $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$. Stepsizes α_k are chosen to minimize the function along the ray $\mathbf{x}_k + \alpha \mathbf{p}_k$,

i.e., so that $\nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^\top \mathbf{p}_k = 0$. One can show that the sequence of directions $\mathbf{p}_k$ generated this way is conjugate with respect to A. Moreover, the following identities facilitate the implementation of CG:

$$\alpha_k = -\frac{\mathbf{p}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top A \mathbf{p}_k} = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top A \mathbf{p}_k},$$
$$\mathbf{r}_{k+1} = A\mathbf{x}_{k+1} - \mathbf{b} = \mathbf{r}_k + \alpha_k A \mathbf{p}_k,$$
$$\beta_{k+1} = \frac{\mathbf{p}_k^\top A \mathbf{r}_{k+1}}{\mathbf{p}_k^\top A \mathbf{p}_k} = \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}.$$

Here is a Matlab CG code:

```
function x = CG(x,A,b,kmax,rho)
r =A*x - b;
p = r;
k = 0;
rerr = 1;
normb = norm(b);
while k < kmax &&  rerr > rho
    Ap = A*p;
    a = r'*r/(Ap'*p);
    x = x + a*p;
    rr = r'*r;
    r = r + a*Ap;
    bet = r'*r/rr;
    p = -r + bet*p;
    k = k + 1;
    rerr = norm(r)/normb;
end
end
```

One can show (see [6], chapter 5) that the directions generated by CG are linearly independent, and each CG iteration minimizes $f(\mathbf{x})$ in the subspace spanned by all directions used so far. In particular, this means that CG converges in d iterations in $\mathbb{R}^d$ in exact arithmetic. Typically, CG algorithm is used to solve large and sparse linear systems with symmetric positive definite matrices and is terminated after much fewer iterations than the dimension of the space d.

Let us return to the gradient descend (GD) and Newton's method. GD is not scale-invariant. This means that if we apply a nondegenerate linear transformation to $\mathbb{R}^d$: $\mathbf{x} \mapsto \mathbf{z} := C\mathbf{x}$, $\det C \neq 0$, and run GD in the original space X and the transformed space Z starting with $\mathbf{x}_0$ and $\mathbf{z}_0$ such that $\mathbf{z}_0 = C\mathbf{x}_0$, then $\mathbf{z}_k \neq \mathbf{x}_k$ for $k > 0$. Let us demonstrate this on a quadratic function.

In the original space $X = \mathbb{R}^d$ and the transformed space $Z = \mathbb{R}^d$ we have:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A \mathbf{x} + \mathbf{b}^\top \mathbf{x}, \quad g(\mathbf{z}) := f(C^{-1}\mathbf{z}) = \frac{1}{2}\mathbf{z}^\top C^{-\top} A C^{-1} \mathbf{x} + \mathbf{b}^\top C^{-1} \mathbf{z}.$$

The first iterates starting from $\mathbf{x}_0$ and $\mathbf{z}_0 := C\mathbf{x}_0$ iterates are,

$$\mathbf{x}_1 = \mathbf{x}_0 - \frac{(A\mathbf{x}_0 + \mathbf{b}_0)^\top (A\mathbf{x}_0 + \mathbf{b}_0)}{(A\mathbf{x}_0 + \mathbf{b}_0)^\top A (A\mathbf{x}_0 + \mathbf{b}_0)} (A\mathbf{x}_0 + \mathbf{b}_0),$$

$$\mathbf{z}_1 = \mathbf{z}_0 - \frac{(C^{-\top} A C^{-1} \mathbf{z}_0 + C^{-\top}\mathbf{b}_0)^\top (C^{-\top} A C^{-1} \mathbf{z}_0 + C^{-\top}\mathbf{b}_0)}{(C^{-\top} A C^{-1} \mathbf{z}_0 + C^{-\top}\mathbf{b}_0)^\top C^{-\top} A C^{-1} (C^{-\top} A C^{-1} \mathbf{z}_0 + C^{-\top}\mathbf{b}_0)} (C^{-\top} A C^{-1} \mathbf{z}_0 + C^{-\top}\mathbf{b}_0).$$

Multiplying $\mathbf{x}_1$ by C we do not obtain $\mathbf{z}_1$.

Newton's iteration is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \nabla\nabla f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k).$$

In the transformed space, $g(\mathbf{z}) = f(C^{-1}\mathbf{z})$, hence

$$\nabla_{\mathbf{z}} g = C^{-\top} \nabla_{\mathbf{x}} f \quad \text{and} \quad \nabla_{\mathbf{z}} \nabla_{\mathbf{z}} g = C^{-\top} \nabla_{\mathbf{x}} \nabla_{\mathbf{x}} f C^{-1}.$$

Exercise: verify these formulas. Hence, Newton's iterations in the transformed space starting from $\mathbf{z}_0 = C\mathbf{x}_0$ are:

$$\begin{aligned}\mathbf{z}_{k+1} &= \mathbf{z}_k - \nabla_{\mathbf{z}}\nabla_{\mathbf{z}} g(\mathbf{z}_k)^{-1}\nabla_{\mathbf{z}} g(\mathbf{z}_k)\\ &= \mathbf{z}_k - \left[C^{-\top}\nabla_{\mathbf{x}}\nabla_{\mathbf{x}} f(\mathbf{x}_k)C^{-1}\right]^{-1} C^{-\top}\nabla_{\mathbf{x}} f(\mathbf{x}_k)\\ &= \mathbf{z}_k - \left[C\nabla_{\mathbf{x}}\nabla_{\mathbf{x}} f(\mathbf{x}_k)^{-1}C^{\top}\right] C^{-\top}\nabla_{\mathbf{x}} f(\mathbf{x}_k)\\ &= \mathbf{z}_k - C\nabla_{\mathbf{x}}\nabla_{\mathbf{x}} f(\mathbf{x}_k)^{-1}\nabla_{\mathbf{x}} f(\mathbf{x}_k).\end{aligned}$$

Multiplying $\mathbf{x}_{k+1}$ by C we obtain $\mathbf{z}_{k+1}$ provided that $\mathbf{z}_k = C\mathbf{x}_k$. This proves the induction step. Hence, we have shown scale invariance of Newton's method.

8.2. **Subsampled Inexact Hessian-free Newton's method.** To take an advantage of second-order methods, batch sizes for evaluating the Hessian or its approximations should not be too small. A good batch size is e.g. 256 [2]. In my relatively small-scale test problem with $n \sim 1000$, I used 64. Taking this into account, keeping Hessian in memory and inverting it might be undesirable. An alternative to it is to define a function with a vector argument $\mathbf{v}$ that evaluates the matrix-vector product $\nabla\nabla f(\mathbf{x}_k)\mathbf{v}$. Then one can use the CG algorithm for the problem

$$\nabla\nabla f(\mathbf{x}_k)\mathbf{p} = -\nabla f(\mathbf{x}_k)$$

to compute a good approximation to Newton's direction $\mathbf{p}$. Once the direction $\mathbf{p}$ is obtained, one can use backtracking line search to determine step length. The idea is the following. The objective function f along the ray $\mathbf{x}_k + \alpha\mathbf{p}$ is given by

$$f(\mathbf{x}_k + \alpha\mathbf{p}) = f(\mathbf{x}_k) + \alpha\nabla f(\mathbf{x}_k)^{\top}\mathbf{p} + O(\alpha^2).$$

Suppose that $\mathbf{p}$ is a descend direction of f, i.e., $\nabla f(\mathbf{x}_k)^{\top}\mathbf{p} < 0$. Then, for any constant $\eta \in (0,1)$, there is a sufficiently small interval $(0,\hat{\alpha})$ such that for all $\alpha \in (0,\hat{\alpha})$ we have

$$f(\mathbf{x}_k + \alpha\mathbf{p}) < f(\mathbf{x}_k) + \eta\alpha\nabla f(\mathbf{x}_k)^{\top}\mathbf{p}.$$

Note that multiplying by η we decrease the slope – see Fig. 13. We pick a constant $\gamma \in (0,1)$ (I usually set $\gamma = 0.9$) and evaluate f at $\mathbf{x}_k + \gamma^j\mathbf{p}$ for $j = 0, 1, \ldots$ until either we get

$$f(\mathbf{x}_k + \gamma^j\mathbf{p}) < f(\mathbf{x}_k) + \eta\gamma^j\nabla f(\mathbf{x}_k)^{\top}\mathbf{p}$$

or $\gamma^j <$ `tol` (I usually set `tol`$= 10^{-14}$).

All these ideas are incorporated to the Matlab code below. The resulting method is called *subsampled inexact Hessian-free Newton's method*. Here the vector with respect to which we are solving optimization problem is denoted by w. The input arguments are:
`fun` is the objective function,
`gfun` is its gradient,
`Hvec` is the function returning the product Hessian times vector,
`Y` is the data matrix which is the argument for all of these functions,
`w` is the initial guess for the solution.
This routine calls the function `CG`, the previous Matlab code.

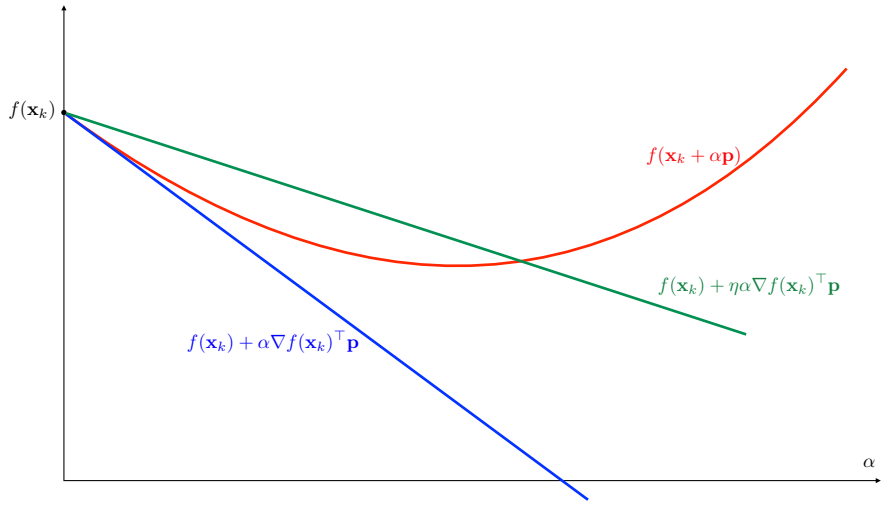


FIGURE 13. An illustration to backtracking line search.

```
function [w,f,normgrad] = SINewton(fun,gfun,Hvec,Y,w)
rho = 0.1;
gam = 0.9;
jmax = ceil(log(1e-14)/log(gam)); % max # of iterations in line search
eta = 0.5;
CGimax = 20; % max number of CG iterations
n = size(Y,1);
bsz = min(n,64); % batch size
kmax = 1e3;
[n,~] = size(Y);
I = 1:n;
f = zeros(kmax + 1,1);
f(1) = fun(I,Y,w);
normgrad = zeros(kmax,1);
nfail = 0;
nfailmax = 5*ceil(n/bsz);
for k = 1 : kmax
    Ig = randperm(n,bsz); % batch for evaluating stochastic gradient
    IH = randperm(n,bsz); % batch for evaluating Hessian times vector
    Mvec = @(v)Hvec(IH,Y,w,v);
    b = gfun(Ig,Y,w);
    normgrad(k) = norm(b);
```

```
    s = CG(Mvec,-b,-b,CGimax,rho);
    a = 1;
    f0 = fun(Ig,Y,w);
    aux = eta*b'*s;
    for j = 0 : jmax
        wtry = w + a*s;
        f1 = fun(Ig,Y,wtry);
        if f1 < f0 + a*aux
            fprintf('Linesearch: j = %d, f1 = %d, f0 = %d\n',j,f1,f0);
            break;
        else
            a = a*gam;
        end
    end
    if j < jmax
        w = wtry;
    else
        nfail = nfail + 1;
    end
    f(k + 1) = fun(I,Y,w);
    fprintf('k = %d, a = %d, f = %d\n',k,a,f(k+1));
    if nfail > nfailmax
        f(k+2:end) = [];
        normgrad(k+1:end) = [];
        break;
    end
end
end
```

8.3. **BFGS.** BFGS (Broyden-Fletcher-Goldfarb-Shanno) is, perhaps, the most successful quasi-newton method [6] (Section 2.2). At each step of optimizing a function $f(\mathbf{x})$, any quasi-newton method updates a quadratic model for it

$$m(\mathbf{p}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top B_k \mathbf{p}, \tag{101}$$

where B_k is a matrix approximating the Hessian of f. The step direction $\mathbf{p}$ is the minimizer of (101):

$$\mathbf{p}_k = -B_k^{-1}\nabla f(\mathbf{x}_k).$$

The matrix B_k is constructed as follows. The initial matrix is often set to identity: $B_0 = I$. Then, at each step, is it updated to match the action of the actual Hessian of f on the actual step. For brevity, we will denote $f_k \equiv f(\mathbf{x}_k)$, $\nabla f_k \equiv \nabla f(\mathbf{x}_k)$, and $\nabla\nabla f(\mathbf{x}_k) \equiv \nabla\nabla f_k$. Taylor expansion at $\mathbf{x}_k$ yields the following identity:

$$\nabla f_{k+1} = \nabla f_k + \nabla\nabla f_k(\mathbf{x}_{k+1} - \mathbf{x}_k) + o(\|\mathbf{x}_{k+1} - \mathbf{x}_k\|). \tag{102}$$

Hence

$$\nabla f_{k+1} - \nabla f_k \approx \nabla\nabla f_k(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{103}$$

We introduce notation

$$\mathbf{s}_k := \mathbf{x}_{k+1} - \mathbf{x}_k \quad \text{and} \quad \mathbf{y}_k := \nabla f_{k+1} - \nabla f_k$$

and define the update requirement for the matrix B motivated by (103):

$$B_{k+1}\mathbf{s}_k = \mathbf{y}_k. \tag{104}$$

Note that B is $d \times d$ while (104) gives only d equations. Therefore, (104) is an underdetermined system that has a d-dimensional solution space. The *BFGS update formula* for B_k adds a matrix of rank 2 to it designed so that B_k remains symmetric positive definite provided that B_0 is symmetric positive definite and $\mathbf{s}_k^\top \mathbf{y}_k > 0$ for all k.

Exercise Prove that all matrices B_k generated by BFGS

$$B_{k+1} = B_k - \frac{B_k\mathbf{s}_k\mathbf{s}_k^\top B_k}{\mathbf{s}_k^\top B_k \mathbf{s}_k} + \frac{\mathbf{y}_k\mathbf{y}_k^\top}{\mathbf{y}_k^\top \mathbf{s}_k} \tag{105}$$

are symmetric positive definite provided that such is B_0 and $\mathbf{s}_k^\top \mathbf{y}_k > 0$ for all $k \in \mathbb{N}$.

A convenient feature of BFGS is that the inverse matrices $H_k \equiv B_k^{-1}$ can be generated automatically instead of B_k:

$$H_{k+1} = V_k^\top H_k V_k + \rho_k \mathbf{s}_k\mathbf{s}_k^\top, \quad \text{where} \quad \rho_k = \frac{1}{\mathbf{y}_k^\top \mathbf{s}_k}, \quad V_k = I - \rho_k \mathbf{y}_k\mathbf{s}_k^\top. \tag{106}$$

Exercise Prove that the matrices H_k given by (106) are inverses of B_k given by (105).

Fig. 14 displays iterates produced by three line search methods: Newton's (blue), BFGS (dark green), and gradient descend (dark red) applied to the Rosenbrock function

$$f(\mathbf{x}) = (1 - x_1)^2 + 5(x_2 - x_1^2)^2 \quad \text{with initial guess} \quad [-1.3, 1.5]^\top. \tag{107}$$

This function has a unique local minimum at $[1; 1]^\top$ but with the given parameter values it is not convex. For Newton's method, the Hessian is modified in the case if it is not positive definite – see the code below. The stopping criterion is $\|\nabla f\| < 10^{-10}$. Newton's method converges in 10 iterations, BFGS – in 18, while gradient descend takes 270 iterations most of which are in a small neighborhood of the solution. The main lesson for us is that *BFGS converges almost as fast as Newton as iterates get to a neighborhood of the local minimum in which the objective function is well-approximated by a convex quadratic.* Here is the Matlab code that generates Fig. 14.

```
function Newton_BFGS_GD()
%%  the Rosenbrock function and parameters
a = 5;
func = @(x,y)(1-x).^2 + a*(y - x.^2).^2;  % Rosenbrock's function
gfun = @(x)[-2*(1-x(1))-4*a*(x(2)-x(1)^2)*x(1);2*a*(x(2)-x(1)^2)]; % gradient of f
Hfun = @(x)[2 + 12*a*x(1)^2 - 4*a*x(2), -4*a*x(1); -4*a*x(1), 2*a]; % Hessian of f
gam = 0.9; % line search step factor
```

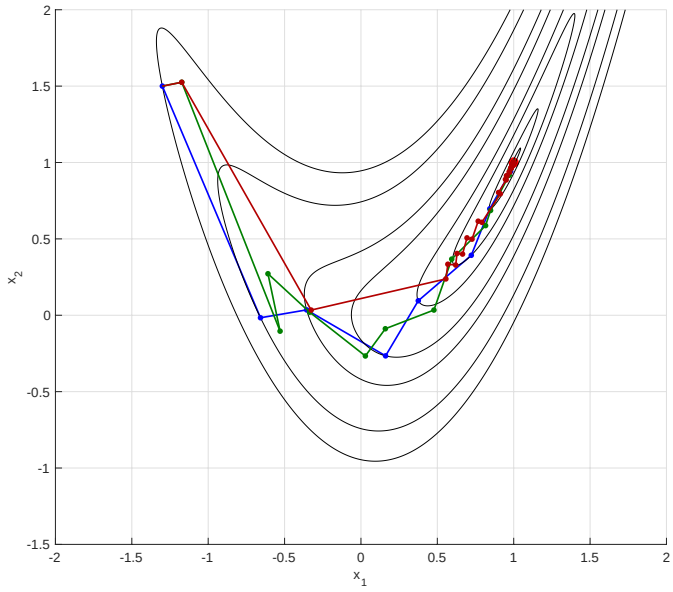


FIGURE 14. Iterates of Newton's (blue), BFGS (dark green), and gradient descend (dark red) applied to (107).

```
jmax = ceil(log(1e-14)/log(gam)); % max # of iterations in line search
eta = 0.5; % backtracking stopping criterion factor
tol = 1e-10;
%%
close all
figure;
hold on; grid;
x0 = [-1.3;1.5];  %initial guess
xstar = [1;1]; % the global minimizer
[xx,yy]=meshgrid(linspace(-2,2,1000),linspace(-1.5,2,1000));
ff = func(xx,yy);
plot(xstar(1),xstar(2),'r.','Markersize',40);
daspect([1,1,1])
col = [0,0,1;0,0.5,0;0.7,0,0]; % colors RGB
% method = 1: Newton
% method = 2: BFGS
% method = 3: gradient descend
str = ["Newton","BFGS","GD"];
for method = 1 : 3
```

```
x = x0;
g = gfun(x);
nor = norm(g);
if method == 1 % Newton
    H = Hfun(x);
else
    if method == 2 % BFGS
        H = eye(2);
    end
end
plot(x(1),x(2),'b.','Markersize',20);
fx = func(x(1),x(2));
contour(xx,yy,ff,[fx,fx],'k','Linewidth',1);
iter = 0;
while nor > tol
    switch method
        case 1
            emin = min(eig(H));
            if emin < 0
                H = H + (-emin + 1)*eye(2);
            end
            p = -H\g;
        case 2
            p = -H*g;
        case 3
            p = -g;
        otherwise
            return
    end
    a = 1;
    f0 = func(x(1),x(2));
    aux = eta*g'*p;
    for j = 0 : jmax
        xtry = x + a*p;
        f1 = func(xtry(1),xtry(2));
        if f1 < f0 + a*aux
            break;
        else
            a = a*gam;
        end
    end
    s = a*p;
    xnew = x + s;
```

```
        plot([x(1),xnew(1)],[x(2),xnew(2)],'Linewidth',2,'color',col(method,:));
        gnew = gfun(xnew);
        if method == 1 % Newton
            H = Hfun(xnew);
        else if method == 2 % BFGS
                y = gnew - g;
                r = 1/(y'*s);
                rys = eye(2) - r*y*s';
                H = rys'*H*rys + r*s*s';
            end
        end
        x = xnew;
        g = gnew;
        fx = func(x(1),x(2));
        if method == 1
            contour(xx,yy,ff,[fx,fx],'k','Linewidth',1);
        end
        plot(x(1),x(2),'.','color',col(method,:),'Markersize',20);
        nor = norm(g);
        iter = iter + 1;
    end
    fprintf('%s: %d iterations, norm(g) = %d\n',str(method),iter,nor);
end
set(gca,'Fontsize',16);
xlabel('x_1','Fontsize',16);
ylabel('x_2','Fontsize',16);
end
```

8.4. **L-BFGS.** L-BFGS stands for *limited memory BFGS* [6] (Section 7.2). It is often the method of choice for large-scale problems where the Hessian cannot be computed at a reasonable cost or is not sparse. L-BFGS (as other limited-memory quasi-Newton methods) stores a small fixed number of vectors (e.g., $m = 5$) that represent an approximation to the Hessian implicitly, i.e., it stores the pairs $(\mathbf{s}_i, \mathbf{y}_i)$ for $i = k-m, \ldots, k-1$. At each iteration k, an initial approximation H_k^0 to the inverse Hessian is chosen. One such approximation that has proven effective in practice is (see [6] Section 6.1)

$$H_k^0 = \frac{\mathbf{s}_{k-1}^\top \mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^\top \mathbf{y}_{k-1}}. \tag{108}$$

Then this approximation is updated by

$$H_k^{j+1} = V_{k-m+j}^\top H_k^{j-1} V_{k-m+j} + \rho_{k-m+j} \mathbf{s}_{k-m+j} \mathbf{s}_{k-m+j}^\top, \quad j = 0, \ldots, m-1, \tag{109}$$

where ρ_i and V_i are defined in (106). Equation (109) suggests that the matrix-vector multiplication defining the direction of the step $\mathbf{p}_k = -H_k^m \nabla f_k$ can be performed in two for-loops implemented in the Matlab code below.

```
function p = finddirection(g,s,y,rho)
% input: g = gradient dim-by-1
% s = matrix dim-by-m, s(:,i) = x_{k-i+1}-x_{k-i}
% y = matrix dim-by-m, y(:,i) = g_{k-i+1}-g_{k-i}
% rho is 1-by-m, rho(i) = 1/(s(:,i)'*y(:,i))
m = size(s,2);
a = zeros(m,1);
for i = 1 : m
    a(i) = rho(i)*s(:,i)'*g;
    g = g - a(i)*y(:,i);
end
gam = s(:,1)'*y(:,1)/(y(:,1)'*y(:,1)); % H0 = gam*eye(dim)
g = g*gam;
for i = m :-1 : 1
    aux = rho(i)*y(:,i)'*g;
    g = g + (a(i) - aux)*s(:,i);
end
p = -g;
end
```

L-BFGS keeps in memory pairs of vectors $(\mathbf{s}_k, \mathbf{y}_k)$ from the most recent m steps and replaces the least recent pair with the new pair at each step. The Matlab program below encodes L-BFGS and testing it on the Rosenbrock function (107). The convergence is achieved in 20 iterations which is comparable with Newton's and close to BFGS (10 and 18, respectively) and which is much fewer than gradient descend (270). The majority of these iterates are done in a small neighborhood of the minimizer. Fig. 15 is generated by this routine. The norm of the gradient versus iteration number for all four methods are plotted in Fig. 16.

```
function LBFGS()
%%  the Rosenbrock function and parameters
a = 5;
func = @(x,y)(1-x).^2 + a*(y - x.^2).^2;  % Rosenbrock's function
gfun = @(x)[-2*(1-x(1))-4*a*(x(2)-x(1)^2)*x(1);2*a*(x(2)-x(1)^2)]; % gradient of f
Hfun = @(x)[2 + 12*a*x(1)^2 - 4*a*x(2), -4*a*x(1); -4*a*x(1), 2*a]; % Hessian of f
gam = 0.9; % line search step factor
jmax = ceil(log(1e-14)/log(gam)); % max # of iterations in line search
eta = 0.5; % backtracking stopping criterion factor
tol = 1e-10;
m = 5; % the number of steps to keep in memory
%%
close all
```

```
figure;
hold on; grid;
x0 = [-1.3;1.5];  %initial guess
xstar = [1;1]; % the global minimizer
[xx,yy]=meshgrid(linspace(-2,2,1000),linspace(-1.5,2,1000));
ff = func(xx,yy);
plot(xstar(1),xstar(2),'r.','Markersize',40);
daspect([1,1,1])
col = [0.4,0.2,0];
%
s = zeros(2,m);
y = zeros(2,m);
rho = zeros(1,m);
%
x = x0;
g = gfun(x);
plot(x(1),x(2),'.','color',col,'Markersize',20);
fx = func(x(1),x(2));
contour(xx,yy,ff,[fx,fx],'k','Linewidth',1);
% first do steepest decend step
a = linesearch(x,-g,g,func,eta,gam,jmax);
xnew = x - a*g;
gnew = gfun(xnew);
s(:,1) = xnew - x;
y(:,1) = gnew - g;
rho(1) = 1/(s(:,1)'*y(:,1));
plot([x(1),xnew(1)],[x(2),xnew(2)],'Linewidth',2,'color',col);
x = xnew;
g = gnew;
nor = norm(g);
plot(x(1),x(2),'.','color',col,'Markersize',20);
fx = func(x(1),x(2));
contour(xx,yy,ff,[fx,fx],'k','Linewidth',1);
iter = 1;
while nor > tol
    if iter < m
        I = 1 : iter;
        p = finddirection(g,s(:,I),y(:,I),rho(I));
    else
        p = finddirection(g,s,y,rho);
    end
    [a,j] = linesearch(x,p,g,func,eta,gam,jmax);
    if j == jmax
```

```
        p = -g;
        [a,j] = linesearch(x,p,g,func,eta,gam,jmax);
    end
    step = a*p;
    xnew = x + step;
    plot([x(1),xnew(1)],[x(2),xnew(2)],'Linewidth',2,'color',col);
    gnew = gfun(xnew);
    s = circshift(s,[0,1]);
    y = circshift(y,[0,1]);
    rho = circshift(rho,[0,1]);
    s(:,1) = step;
    y(:,1) = gnew - g;
    rho(1) = 1/(step'*y(:,1));
    x = xnew;
    g = gnew;
    fx = func(x(1),x(2));
    if nor > 1e-1
        contour(xx,yy,ff,[fx,fx],'k','Linewidth',1);
    end
    plot(x(1),x(2),'.','color',col,'Markersize',20);
    nor = norm(g);
    iter = iter + 1;
end
fprintf('L-BFGS: %d iterations, norm(g) = %d\n',iter,nor);
set(gca,'Fontsize',16);
xlabel('x_1','Fontsize',16);
ylabel('x_2','Fontsize',16);
end

%%
function [a,j] = linesearch(x,p,g,func,eta,gam,jmax)
    a = 1;
    f0 = func(x(1),x(2));
    aux = eta*g'*p;
    for j = 0 : jmax
        xtry = x + a*p;
        f1 = func(xtry(1),xtry(2));
        if f1 < f0 + a*aux
            break;
        else
            a = a*gam;
        end
    end
```

```
end
```

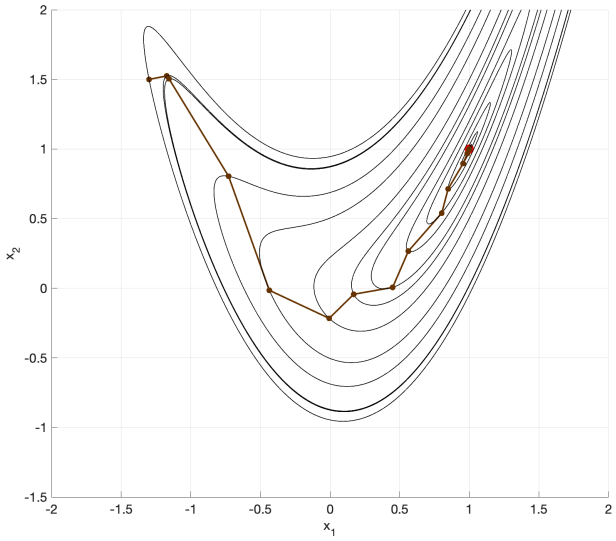


FIGURE 15. Iterates of L-BFGS applied to (107).

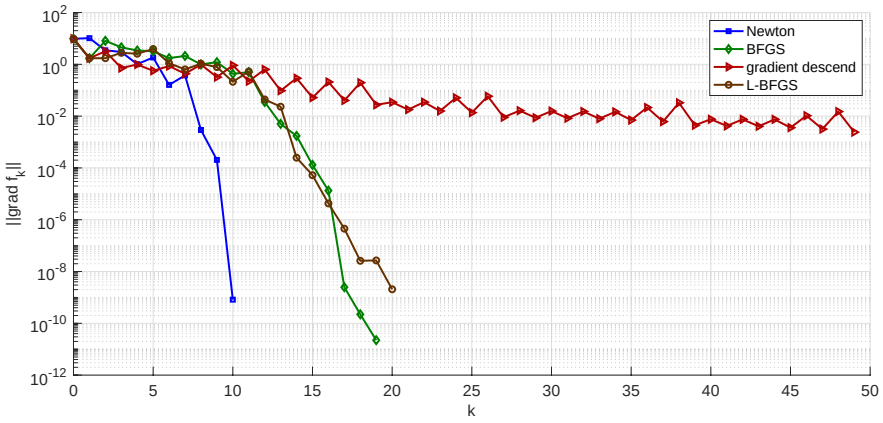


FIGURE 16. Decay of $\|\nabla f(\mathbf{x}_k)\|$ for various methods applied to (107).

8.5. **Stochastic L-BFGS.** Stochastic L-BFGS [2] (Section 6.2) like stochastic Newton's method makes the performance less sensitive to poor scaling and improves convergence

prefactor in the sublinear convergence rate. Our goal is to extend L-BFGS for large-scale optimization problems and make it competitive and, ideally, more efficient than SG. Performance of L-BFGS is sensitive to the quality of approximation of the Hessian. Note that I reset the direction to the steepest descend direction in `LBFGS.m` (the code above) for a good reason. Without this, the routine wastes about 70 iterations in one unfortunate spot where the produced direction is bad. To address this issue, it is advisable to use larger batches for stochastic approximation for the inverse Hessian than those for approximation of the gradient. This measure would make stochastic L-BFGS much more expensive than SG. A countermeasure against the drastic increase of cost per iteration is to update the inverse Hessian only once per 10 or 20 iterations. Here is a pseudocode depicting stochastic L-BFGS without specifying implementational details. This routine leaves you some important

Algorithm 2: Stochastic L-BFGS

Initialization:
Choose an initial vector $\mathbf{x}_0$ and set $\mathcal{P} = \emptyset$;
Choose m, the limited memory constant in L-BFGS;
Choose N_g and N_H, the batch sizes for the gradient and the inverse Hessian;
Choose M, the number of steps between every update of the inverse Hessian;
for $k = 1, 2, ...$ **do**
  Generate a realization of the random variable ξ_k;
  Compute the stochastic gradient $\mathbf{g}(\mathbf{x}_k, \xi_k)$;
  Compute a stochastic direction $\mathbf{p}_k$ by a routine similar to `finddirection`;
  Choose a stepsize α_k;
  Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}(\mathbf{x}_k, \xi_k)$;
  if $k \mod M = 0$ **then**
    Generate a realization of the random variable ξ_k^H;
    Compute $\mathbf{s}_k$ and $\mathbf{y}_k$ based on sample $\mathcal{S}_k^H$;
    If $|\mathcal{P}| > m$, then remove the eldest pair from it;
    Add a new pair $(\mathbf{s}_k, \mathbf{y}_k)$ to $\mathcal{P}$;
  end
end

choices to make. First of all, you can use some preset stepsize reducing strategy or you can do backtracking line search as it is done in the stochastic inexact Hessian-free Newton's method. Second, you can compute $\mathbf{y}_k$ using the stochastic gradients evaluated at $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ or you can use the stochastic gradient $\mathbf{x}_{k+1}$ and the stochastic gradient evaluated under the `if`-statement M steps ago. What works the best should be determined by numerical experiments.

9. Methods for nonlinear least-squares problem

In least-squares problems, the objective function has the following special form:

$$f(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{n} r_j^2(\mathbf{x}). \tag{110}$$

Objective functions of this form arise in many applications wherever there is a nonlinear model and experimental noisy data. The assumption that the noise is Gaussian leads to the objective function (110) [6] (Chapter 10). Loss functions in classification problem can also be chosen of the form (110). Another important application is solving partial differential equations with the aid of neural networks. The solution to the PDE is represented via a neural network and then the least-squares optimization problem is set up to fit the PDE at some number of mesh points. While this approach dates back to 1990s (see e.g. Lagaris, Likas, and Fotiadis, 1998 [10]), it became one of the hottest areas or research within the last couple of years. In comparison with the traditional PDE solvers (finite difference methods, finite element methods, finite volume methods, etc), the approach based on the representation of the solution via a neural network (NN) overcomes the "curse of dimensionality" and hence opens new horizons [11, 12, 13, 14].

In this section, we will discuss two methods for solving nonlinear least squares problem: Gauss-Newton and Levenberg-Marquardt [6] (Chapter 10). We will demonstrate how they work on an example of solving the Poisson PDE with nonhomogeneous Dirichlet boundary conditions by means on NNs from [10].

9.1. The gradient and a handy approximation to the Hessian.

We will denote by $\mathbf{r}(\mathbf{x})$ and $J(\mathbf{x})$, respectively, the vector-function with components $r_j(\mathbf{x})$ in (110) and its Jacobian matrix:

$$\mathbf{r}(\mathbf{x}) := \begin{bmatrix} r_1(\mathbf{x}) \\ \vdots \\ r_n(\mathbf{x}) \end{bmatrix}, \quad J(\mathbf{x}) = \begin{bmatrix} \nabla r_1(\mathbf{x})^\top & \to \\ \vdots & \\ \nabla r_n(\mathbf{x})^\top & \to \end{bmatrix}. \tag{111}$$

Then f and its gradient and Hessian are:

$$f(\mathbf{x}) = \frac{1}{2}\|\mathbf{r}(\mathbf{x})\|^2, \tag{112}$$

$$\nabla f(\mathbf{x}) = J(\mathbf{x})^\top \mathbf{r}(\mathbf{x}), \tag{113}$$

$$\nabla\nabla f(\mathbf{x}) = J(\mathbf{x})^\top J(\mathbf{x}) + \sum_{j=1}^{n} r_j(\mathbf{x})\nabla\nabla r_j(\mathbf{x}). \tag{114}$$

The second term in (114) can be small in two cases:

- if the residuals r_j are small which is the case if the exact solution is zero of $\mathbf{r}$, and/or
- if r_j are nearly linear in the neighborhood of the solution, i.e., $\nabla\nabla r_j$ are small.

Whether this is the case or not, both Gauss-Newton and Levenberg-Marquardt methods approximate the Hessian of f with the first term in (114) only. Then Gauss-Newton follows the line-search strategy, while Levenberg-Marquardt employs the trust-region strategy.

9.2. **The Gauss-Newton method.** The Gauss-Newton method defines the search direction at step k by solving

$$J_k^\top J_k \mathbf{p}_k = -J_k^\top \mathbf{r}_k, \tag{115}$$

where the subscript k replaces the argument $\mathbf{x}_k$, and proceeds as the line-search methods in the routine in Section 8.3. This choice of direction has several advantages.

- No computation of Hessians of r_j is required which is very important in the context of NNs.
- If r_j are small or if $\nabla\nabla r_j$ are small, the Gauss-Netwon method converges almost as fast as Newton's method.
- If J_k has linearly independent columns then the Gauss-Newton direction is a descend direction ,i.e., $\mathbf{p}_k^\top J^\top \mathbf{r}_k < 0$. Indeed, from (115) we have:

$$\mathbf{p}_k^\top J^\top \mathbf{r}_k = -\mathbf{p}_k^\top J_k^\top J_k \mathbf{p}_k = -\|J_k^\top \mathbf{p}_k\|^2 \le 0. \tag{116}$$

  If columns of J_k are linearly independent, equality takes place if and only if $\mathbf{r}_k = 0$. Hence, if $\mathbf{x}_k$ is not the solution, we have strict inequality in (116) which means that $\mathbf{p}_k$ is a descend direction. Note that this is not true, in general, for the regular Newton's method unless it is applied to a strictly convex function.
- Equation (115) is the *normal equation* for the *linear least squares problem* $J_k\mathbf{p}_k = -\mathbf{r}_k$, i.e., its solution is the minimizer of

$$\min_{\mathbf{p}} \|J_k\mathbf{p} + \mathbf{r}_k\|^2. \tag{117}$$

  This allows us to use methods for solving linear least squares problem such as QR decomposition via Householder reflections [15] for finding the search direction.

Convergence of the Gauss-Newton method to a stationary point under nonrestrictive conditions is guaranteed by the following

Theorem 5. *Suppose that all $r_j(\mathbf{x})$ are Lipschitz continuously differentiable in a neighborhood of the level set*

$$\{\mathbf{x} \mid f(\mathbf{x}) \le f(\mathbf{x}_0)\}$$

and $J(\mathbf{x})$ satisfies the uniform full-rank condition

$$\|J(\mathbf{x})\mathbf{z}\| \ge \gamma\|\mathbf{z}\| \quad \text{for some} \quad \gamma > 0$$

in this neighborhood. Then the iterates of the Gauss-Newton method with stepsizes satisfying the Wolfe conditions [6] (Section 3.1) converge to a stationary point of $f(\mathbf{x})$, i.e.,

$$\lim_{k\to\infty} J_k^\top \mathbf{r}_k = 0.$$

The proof of this theorem follows from Theorem 3.2 in [6] which I prove in AMSC660. I will not repeat it here. The Wolfe conditions for stepsizes are satisfied if you you do backtracking line search implemented in the code in Section 8.3.

9.3. **The Levenberg-Marquardt method.** The Levenberg-Marquardt method follows the trust-region strategy. This means that at each step k, a trust region, typically of the form

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \equiv \|\mathbf{p}\| \leq \Delta_k, \tag{118}$$

is given, an a constrained minimization problem for a quadratic model

$$m(\mathbf{p}) := f_k + \mathbf{p}^\top \mathbf{g}_k + \frac{1}{2}\mathbf{p}^\top B_k \mathbf{p} \tag{119}$$

is solved. Then the quality of the model is assessed by computing the ratio of the actual reduction to the expected reduction

$$\rho = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{p}_k)}{f(\mathbf{x}_k) - m(\mathbf{p}_k)} \tag{120}$$

and, depending on its value, the trust region radius for the next step is increased, left the same, or decreased. Finally, if ρ is smaller than a user-prescribed threshold, the proposed step $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$ is accepted or rejected. If the step is rejected, a new $\mathbf{p}_{k+1}$ is obtained at the next step as the solution to the constrained optimization problem in a smaller trust region. A pseudocode giving a template for any trust region method is outlined in Algorithm 3 I usually set $\eta = 0.1$ and $\Delta_0 = 0.2\Delta_{\max}$. There are several approaches to solving the constrained minimization problem (118)–(119). The one used in Levenberg-Marquard gives the exact solution. Note that the quadratic model for Levenberg-Marquardt is of the form

$$m(\mathbf{p}) := \frac{1}{2}\|J_k\mathbf{p} + \mathbf{r}_k\|^2 = \frac{1}{2}\|\mathbf{r}_k\|^2 + \mathbf{p}^\top J^\top \mathbf{r}_k + \frac{1}{2}\mathbf{p}^\top J_k^\top J_k \mathbf{p}. \tag{121}$$

Therefore, the quadratic model is convex but it might be not strictly convex if J_k has linearly dependent columns.

Recall the KKT optimality conditions and apply them to (118)–(119). The Lagrangian function is

$$\mathcal{L}(\mathbf{p}, \lambda) = f + \mathbf{p}^\top \mathbf{g} + \frac{1}{2}\mathbf{p}^\top B\mathbf{p} - \frac{\lambda}{2}\left(\Delta^2 - \|\mathbf{p}\|^2\right). \tag{122}$$

Here, we omit the subscripts k for brevity. Hence, if $(\mathbf{p}, \lambda)$ is a solution to (118)–(119), then:

$$\nabla_{\mathbf{p}}\mathcal{L} = B\mathbf{p} + \mathbf{g} + \lambda\mathbf{p} = 0, \tag{123}$$

$$\left(\Delta^2 - \|\mathbf{p}\|^2\right) \geq 0, \tag{124}$$

$$\lambda \geq 0, \tag{125}$$

$$\lambda\left(\Delta^2 - \|\mathbf{p}\|^2\right) = 0. \tag{126}$$

Therefore, if the unconstrained minimizer $-B^{-1}\mathbf{g}$ lies outside the trust region, i.e, $\|B^{-1}\mathbf{g}\| > \Delta$, the solution to (123)–(126) lies on the trust region boundary, i.e., $\|\mathbf{p}\| = \Delta$. Let us find discuss how to find $\mathbf{p}$ in this case. We have:

$$(B + \lambda I)\mathbf{p} = -\mathbf{g}, \quad \|\mathbf{p}\| = \Delta. \tag{127}$$

Algorithm 3: Trust region template

Input:
Choose minimal and maximal radii of trust region $\Delta_{\min}$ and $\Delta_{\max}$;
Choose the initial trust region radius $\Delta_0 \in [\Delta_{\min}, \Delta_{\max}]$;
Choose threshold $\eta \in [0, 1/4)$ for accepting proposed step;
Choose initial approximation $\mathbf{x}_0$;
for $k = 1, 2, \ldots$ **do**
  Compute unconstrained minimizer $\mathbf{p}_k = -B_k^{-1}\mathbf{g}$ of $m(\mathbf{p})$;
  if $\|\mathbf{p}_k\| > \Delta_k$ **then**
    Solve constrained minimization problem (118)–(119) and get $\mathbf{p}_k$;
  end
  Compute the ratio ρ (120);
  if $\rho < 1/4$ **then**
    Reduce the trust region radius: $\Delta_{k+1} = 0.25\Delta_k$;
  else
    if $\rho > 3/4$ *and* $\|\mathbf{p}_k\| = \Delta_k$ **then**
      Increase trust region radius $\Delta_{k+1} = \min(2\Delta_k, \Delta_{\max})$
    end
  end
  if $\rho_k > \eta$ **then**
    Accept step: $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$;
  else
    Reject step: $\mathbf{x}_{k+1} = \mathbf{x}_k$;
  end
end

Since B is symmetric and nonnegative definite as $B = J^\top J$, its spectral decomposition is

$$B = Q^\top \Lambda Q,$$

where $Q = [\mathbf{q}_1, \ldots, \mathbf{q}_d]$ is orthogonal, i.e. its columns are orthonormal, and Λ is diagonal. We always can order the eigenvalues in the nondecreasing order, i.e,

$$\Lambda = \operatorname{diag}\{\lambda_1, \ldots, \lambda_d\}, \quad \text{where} \quad 0 \le \lambda_1 \le \ldots \le \lambda_d.$$

Using the spectral decomposition, the identity $\mathbf{p} = -(B + \lambda I)^{-1}\mathbf{g}$ can be rewritten as

$$\mathbf{p} = -\sum_{j=1}^{d} \frac{\mathbf{q}_j^\top \mathbf{g}}{\lambda_j + \lambda}\mathbf{q}_j. \tag{128}$$

Since columns of Q are orthonormal, the identity $\|\mathbf{p}\| = \Delta$ becomes:

$$\sqrt{\sum_{j=1}^{d} \frac{(\mathbf{q}_j^\top \mathbf{g})^2}{(\lambda_j + \lambda)^2}} = \Delta. \tag{129}$$

Equation (129) is a 1D nonlinear equation that can be solved using Newton's method. Note that if $\lambda = 0$, then $\mathbf{p}$ is unconstrained minimizer with $\|\mathbf{p}\| > \Delta$ by our assumption. Hence, since all λ_j are nonnegative, the solution λ to (129) must be positive. On the other hand, since the left-hand side of (129) strictly decreases and tends to zero as $\lambda \to \infty$, we conclude that there exists a unique solution λ^* to (129). Also, the difference between the left-and right-hand side of (129) behaves approximately as $C\lambda^{-1}$ while the difference between their reciprocals behaves approximately as a linear function which is beneficial for rapid convergence of Newton's iterations. So, we will solve numerically the equation

$$\phi(\lambda) := \frac{1}{\Delta} - \left[\sum_{j=1}^{d} \frac{(\mathbf{q}_j^\top \mathbf{g})^2}{(\lambda_j + \lambda)^2} \right]^{-1/2} = 0. \tag{130}$$

The Newton iteration is

$$\lambda^{(l+1)} = \lambda^{(l)} - \frac{\phi(\lambda^{(l)})}{\phi'(\lambda^{(l)})}. \tag{131}$$

The derivative of ϕ is given by:

$$\phi'(\lambda) = -\left[\sum_{j=1}^{d} \frac{(\mathbf{q}_j^\top \mathbf{g})^2}{(\lambda_j + \lambda)^3} \right]^{-3/2} \left[\sum_{j=1}^{d} \frac{(\mathbf{q}_j^\top \mathbf{g})^2}{(\lambda_j + \lambda)^3} \right]. \tag{132}$$

Left $\mathbf{p}_l$ be the solution to $(B + \lambda^{(l)})\mathbf{p} = -\mathbf{g}$. Then $\phi(\lambda^{(l)}) = \Delta^{-1} - \|\mathbf{p}\|^{-1}$, while the first factor in (132) is $\|\mathbf{p}_l\|^3$. The second factor in (132) is the squared norm of the solution to $(B + \lambda I)^{3/2}\mathbf{q} = -\mathbf{g}$. These considerations lead to the following subroutine for computing the solution to the constrained minimization problem in Levenberg-Marquardt.

```
% solve the constrained minimization problem using dogleg strategy
% do Tikhonov regularization for the case J is rank-deficient
B = J'*J + (1e-12)*I;
pstar = -B\g; % unconstrained minimizer
if norm(pstar) <= R
    p = pstar;
else % solve constrained minimization problem
    lam = 1; % initial guess for lambda
    while 1
        B1 = B + lam*I;
        C = chol(B1); % do Cholesky factorization of B
        p = -C\(C'\g);  % solve B1*p = -g
        np = norm(p);
```

```
            dd = abs(np - R); % R is the trust region radius
            if dd < 1e-6
                break
            end
            q = C'\p; % solve C^\top q = p
            nq = norm(q);
            lamnew = lam + (np/nq)^2*(np - R)/R;
            if lamnew < 0
                lam = 0.5*lam;
            else
                lam = lamnew;
            end
        end
    end
```

This Matlab routine does not take advantage of the fact that $(B+\lambda I)\mathbf{p} = -\mathbf{g}$ is really the normal equation for the linear least squares problem

$$\min_{\mathbf{p}} \frac{1}{2} \left\| \begin{bmatrix} J \\ \sqrt{\lambda} I \end{bmatrix} \mathbf{p} + \begin{bmatrix} \mathbf{g} \\ 0 \end{bmatrix} \right\|^2 .$$

To take an advantage of it, one needs to use low-level language. Then $(B + \lambda I)\mathbf{p} = -\mathbf{g}$ can be solved by QR decomposition implemented via a clever combination of Householder reflections and Givens rotations (see [6] (Section 10.3) and [15]).

For convergence theorems for Levenberg-Marquard consult Section 4 in [6]. There are many nuances, but in short, iterates of Levenberg-Marquardt converge to a stationary point provided that certain nonrestrictive conditions hold.

9.4. **An application to solving PDEs using NNs.** As an example of a nonlinear least squares problem, we consider the one from [10] arising in solving the boundary-value problem for the Poisson equation:

$$u_{xx} + u_{yy} = \phi(x, y), \quad (x, y) \in \Omega = [0, 1]^2, \tag{133}$$

$$u(0, y) = f_0(y), \quad u(1, y) = f_1(y), \quad u(x, 0) = g_0(x), \quad u(x, 1) = g_1(x). \tag{134}$$

Lagaris, Likas, and Fotiadis (1998) [10] proposed to look for the solution to (133)–(134) in the following form:

$$\Psi(x, y, \mathsf{w}) = A(x, y) + h(x)h(y)\mathcal{N}(x, y, \mathsf{w}). \tag{135}$$

Here $A(x, y)$ is a function satisfying boundary conditions (134) which can be written out analytically:

$$\begin{aligned} A(x, y) = {} & (1 - x)f_0(y) + xf_1(y) + (1 - y)\left[g_0(x) - \{(1 - x)f_0(0) + xf_1(0)\}\right] \\ & + y\left[g_1(x) - \{(1 - x)f_0(1) + xf_1(1)\}\right]. \end{aligned} \tag{136}$$

The function h is chosen to guarantee that the second term in the right-hand side in (135) is zero on the boundary $\partial\Omega$:

$$h(t) = t(1-t). \tag{137}$$

The function $\mathcal{N}(x, y, \mathsf{w})$ is a neural network with one hidden layer and a single linear output unit:

$$\mathcal{N}(x, y, \mathsf{w}) = \mathbf{v}^\top \sigma(W\mathbf{x} + \mathbf{u}), \quad \mathsf{w} \equiv (\mathbf{v}, W, \mathbf{u}), \tag{138}$$

where

$$\mathbf{x} \equiv \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{v} \in \mathbb{R}^N, \quad W = (w_{ij}) \in \mathbb{R}^{N\times 2}, \quad \mathbf{u} \in \mathbb{R}^N,$$

and σ is a nonlinear function applied entry-wise. We will experiment with

$$\sigma(t) = (1 + \exp(-t))^{-1} \text{ (sigmoid, as in [10]) and } \sigma(t) = \tanh(t).$$

I chose this example due to its simplicity. A one-layer NN is sufficient to achieve quite impressive accuracy using just a few training points, and the derivatives of $\mathcal{N}$ with respect to x, y, and parameters packed in w can be computed analytically without the use of *automatic differentiation*.

As the form of the solution is set, the proposed parameter-dependent solution is plugged into the differential operator, evaluated at a number of training points, and equated to the corresponding values of the right-hand side of the differential equation. Then the nonlinear least-squares problem (NLLS) is set up to minimize the sum of the squares of the discrepancies by choosing an optimal set of the parameters. This approach was proposed in [10] to solve ODEs and PDEs. The nonlinear least-squares problem for the PDE above is

$$f(\mathsf{w}) := \frac{1}{2}\sum_{j=1}^{n} |\Psi_{xx}(x_j, y_j, \mathsf{w}) + \Psi_{yy}(x_j, y_j, \mathsf{w}) - f(x_j, y_j)|^2 \to \min. \tag{139}$$

Plugging Ψ into PDE (133) we get:

$$\begin{aligned} r_j(\mathsf{w}) &= \Psi_{xx}(x_j, y_j, \mathsf{w}) + \Psi_{yy}(x_j, y_j, \mathsf{w}) - f(x_j, y_j) \\ &= A_{xx} + A_{yy} + \left[h(x)h''(y) + h''(x)h(y)\right]\mathcal{N} \\ &\quad + 2\left[h'(x)h(y)\mathcal{N}_x + h(x)h'(y)\mathcal{N}_y\right] \\ &\quad + h(x)h(y)\left[\mathcal{N}_{xx} + \mathcal{N}_{yy}\right] - f(x_j, y_j). \end{aligned} \tag{140}$$

Equation (140) shows that in order to solve the NLLS (139) using Gauss-Newton or Levenberg-Marquardt, one needs to calculate the first derivatives of $\mathcal{N}$, $\mathcal{N}_x$, $\mathcal{N}_y$, $\mathcal{N}_{xx}$, and $\mathcal{N}_{yy}$ with respect to the components of $\mathbf{v}$, W, and $\mathbf{u}$. This is done in the Matlab routine `NN.m`:

```
function [f,fx,fy,fxx,fyy,df,dfx,dfy,dfxx,dfyy] = NN(x,v,W,u,fun,dfun,d2fun,d3fun)
%% derivatives of the network
z = W*x + u;
s0 = fun(z); % sigma(z)
f = v'*s0;
```

```
W2 = W.*W;
s1 = dfun(z); % sigma'(z)
s2 = d2fun(z); % sigma''(z)
s3 = d3fun(z); % % sigma'(z)
fx = v'*(W(:,1).*s1); % Psi_x
fy = v'*(W(:,2).*s1); % Psi_y
fxx = v'*(W2(:,1).*s2); % Psi_{xx}
fyy = v'*(W2(:,2).*s2); % Psi_{yy}
%% derivatives with respect to parameters
[nv1,nv2] = size(v); % nv2 must be 1
[nw1,nw2] = size(W);
[nu1,nu2] = size(u); % nu2 must be 1
dim = nv1 + nw1*nw2 + nu1;
df = zeros(dim,1);
dfx = zeros(dim,1);
dfy = zeros(dim,1);
dfxx = zeros(dim,1);
dfyy = zeros(dim,1);
% df
df(1:nv1) = s0;
df(nv1+1 : nv1+nw1*nw2) = reshape((v.*s1)*(x'),[nw1*nw2,1]);
df(nv1+nw1*nw2+1 : end) = v.*s1;
% dfx
dfx(1:nv1) = W(:,1).*s1;
dfx(nv1+1 : nv1+nw1*nw2) = ...
    reshape((v.*W(:,1).*s2)*(x') + (v.*s1)*[1,0],[nw1*nw2,1]);
dfx(nv1+nw1*nw2+1 : end) = v.*W(:,1).*s2;
% dfy
dfy(1:nv1) = W(:,2).*s1;
dfy(nv1+1 : nv1+nw1*nw2) = ...
    reshape((v.*W(:,2).*s2)*(x') + (v.*s1)*[0,1],[nw1*nw2,1]);
dfy(nv1+nw1*nw2+1 : end) = v.*W(:,2).*s2;
% dfxx
dfxx(1:nv1) = W2(:,1).*s2;
dfxx(nv1+1 : nv1+nw1*nw2) = ...
    reshape((v.*W2(:,1).*s3)*(x') + 2*(v.*W(:,1).*s2)*[1,0],[nw1*nw2,1]);
dfxx(nv1+nw1*nw2+1 : end) = v.*W2(:,1).*s3;
% dfyy
dfyy(1:nv1) = W2(:,2).*s2;
dfyy(nv1+1 : nv1+nw1*nw2) = ...
    reshape((v.*W2(:,2).*s3)*(x') + 2*(v.*W(:,2).*s2)*[0,1],[nw1*nw2,1]);
dfyy(nv1+nw1*nw2+1 : end) = v.*W2(:,2).*s3;
end
```

As in an example in [10], we set the right-hand side and the boundary functions in (133)–(134) to:

$$\phi(x,y) = e^{-x}(x - 2 + y^3 + 6y),$$
$$f_0(y) = y^3, \qquad f_1(y) = (1+y^3)e^{-1},$$
$$g_0(x) = xe^{-x}, \qquad g_1(x) = e^{-1}(x+1).$$

Then the exact solution is

$$u(x,y) = e^{-x}(x + y^3).$$

The training set is a 5×5 set of mesh points (see Fig. 17). The number of hidden units N is set to 10. This makes the dimensional lity of the parameter space equal to 40. The function $\sigma = \tanh$:

```
fun = @(x)tanh(x);
dfun = @(x)1./cosh(x).^2;
d2fun = @(x)-2*sinh(x)./cosh(x).^3;
d3fun = @(x)(4*sinh(x).^2-2)./cosh(x).^4;
```

The initial guess for parameter values was the vector of all ones. We solve the resulting nonlinear least squares problem using three methods: gradient descend (GD), Gauss-Newton (GN), and Levenberg-Marquardt (LM). Note that then the matrix J has more columns than rows. Hence $J^\top J$ has zero eigenvalues. The test points are the 101×101 mesh points. Stopping criteria were: either the number of iterations exceeds 120, or the norm of the gradient of the objective function decays to 10^{-4}. Fig. 17 shows that GN and LM perform notably better than GD – see also printouts below. Overall, LM tends to be more robust.

```
GD:
iter # 120: f = 0.17110610126100, |df| = 8.9608e-02
CPUtime = 4.136553e+00, iter = 120
max|err| = 4.205708e-03, L2 err = 2.067841e-01

GN:
iter # 114: f = 0.00000000002963, |df| = 9.2399e-05
CPUtime = 5.718491e+00, iter = 114
max|err| = 1.442690e-07, L2 err = 8.296356e-06

LM:
iter # 109: f = 0.00000002701552, |df| = 4.4645e-05, rho = 9.9390e-01, R = 1.5625e-02
CPUtime = 1.260592e-01, iter = 109
max|err| = 1.301953e-06, L2 err = 4.449413e-05
```

In the final experiment, the function σ was replaced with the sigmoid:

```
fun = @(x)1./(1+exp(-x));
fun = @(x)1./(1+exp(-x));
dfun = @(x)exp(-x)./(1+exp(-x)).^2;
```

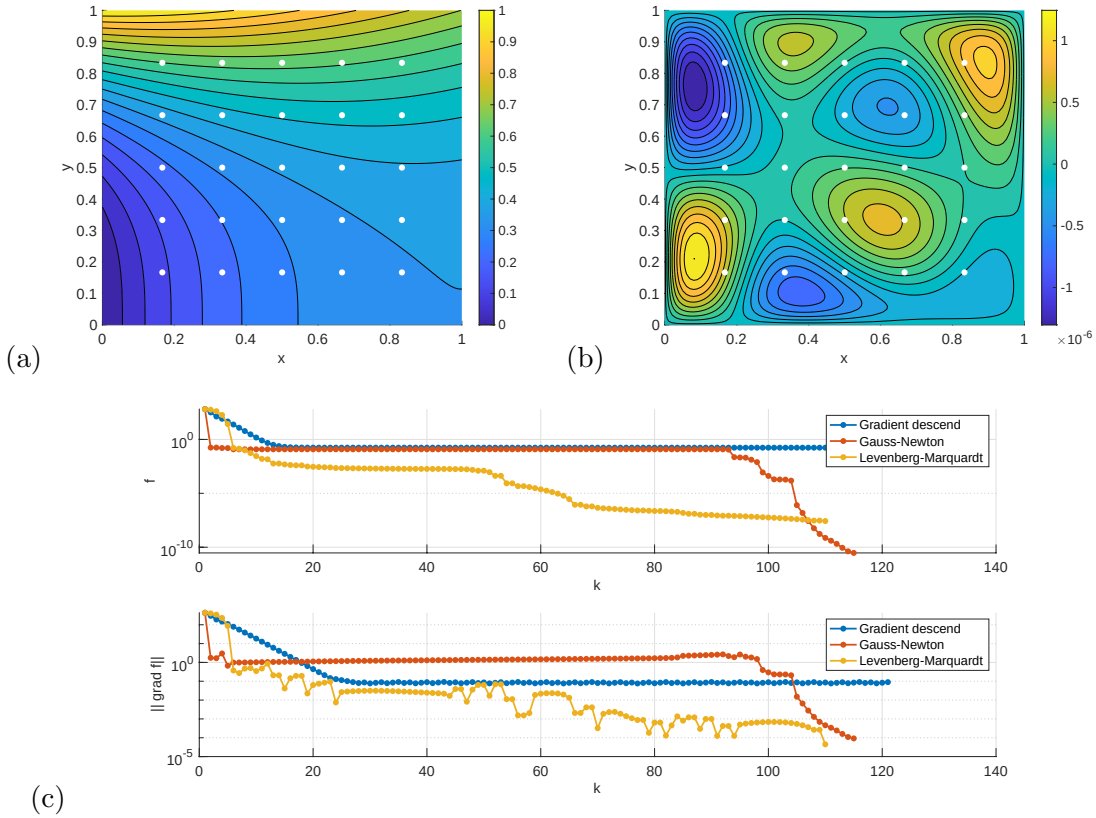


FIGURE 17. The solution (a) to the NLLS (139) and the error (b) committed by Levenberg-Marquardt. While dots are the training points. (c): Comparison of the performance of three methods.

```
d2fun = @(x)-exp(-x)./(1+exp(-x)).^2 + 2*exp(-2*x)./(1+exp(-x)).^3;
d3fun = @(x)exp(-x)./(1+exp(-x)).^2 - 6*exp(-2*x)./(1+exp(-x)).^3 ...
 + 6*exp(-3*x)./(1+exp(-x)).^4;
```

Here are the results:

```
GD:
iter # 120: f = 0.16857469843620, |df| = 7.7421e-02
CPUtime = 4.010367e+00, iter = 120
max|err| = 4.200602e-03, L2 err = 2.059516e-01

GN:
iter # 15: f = 0.00000000000483, |df| = 2.5831e-05
CPUtime = 1.795551e-01, iter = 15
max|err| = 1.589962e-04, L2 err = 4.956789e-03
```

```
LM:
iter # 44: f = 0.00000375382680, |df| = 6.5900e-05, rho = 1.0037e+00, R = 6.2500e-02
CPUtime = 5.234713e-02, iter = 44
max|err| = 1.869000e-05, L2 err = 6.464585e-04
```

To conclude this section, I would like to make a few remarks. The approach considered in this section is currently a hot area of research. For recent advances in solving forward and inverse problems for PDEs, see works of G. Karniadakis' group on physics-informed neural networks[5]. It is clear from the example discussed in this section that an accurate solution of a 2D PDE can be obtained using a very coarse mesh of training points. Nevertheless, the number of mesh points grows exponentially with the increase of dimensionality. Therefore, using a mesh of training points is infeasible for promoting NN-based solvers to high dimensions. An approach in which the importance sampling of training points and training the NN are interweaved which allowed to solve a PDE in $\mathbb{R}^{100}$ is developed in works of G. Rostkoff, E. Vanden-Eijnden, and collaborators.

10. Regularization, sparsity, and coordinate descend

In this section, we establish how the *Tikhonov* and *lasso* (least absolute shrinkage and selection operator) regularization affect the solution to linear least squares problem, show how the lasso promotes sparsity, and go over the *coordinate descend* method that has been revived largely due to its convenience for handling lasso regularizers.

10.1. Geometry of linear least squares problems.

The linear least squares problem reads:

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\|A\mathbf{x}-\mathbf{b}\|_2^2, \quad \text{where} \quad A\in\mathbb{R}^{n\times d}, \quad \mathbf{b}\in\mathbb{R}^n. \tag{141}$$

- First we consider the case where $n \geq d$, i.e., the system $A\mathbf{x}=\mathbf{b}$ is overdetermined if $n > d$, and columns of A are linearly independent (i.e., A is full rank). A geometric interpretation of (141) is that we need to find a vector in the span of columns of A (this vector is $A\mathbf{x}$) minimizing the Euclidean distance to $\mathbf{b}$. Let $\hat{\mathbf{b}}$ be the orthogonal projection of $\mathbf{b}$ on to $\mathsf{span}(A)$:

  $$\hat{\mathbf{b}} = QQ^\top\mathbf{b},$$

  where $A = QR$ is the QR-decomposition of A. Since the vector $\mathbf{d} := \mathbf{b} - A\mathbf{x}$ can be decomposed into the sum of $\mathbf{b}-\hat{\mathbf{b}} = (I-QQ^\top)\mathbf{b}$ and a vector in $\mathsf{span}(A)$, and the squared length $\|\mathbf{d}\|_2^2$ is the sum of squares of these two vectors, the minimum of $\|\mathbf{d}\|_2^2$ is reached if the second vector is zero, i.e., if $A\mathbf{x}=\hat{\mathbf{b}}=QQ^\top\mathbf{b}$, i.e., if

  $$\mathbf{x}^* = R^{-1}Q^\top\mathbf{b}.$$

[5] I thank AMSC graduate students Alex Papados and Jiajing Guan for bringing these works to my attention.

- Next, we consider the case where $n < d$, i.e., the system $A\mathbf{x} = \mathbf{b}$ is underdetermined, and rows of A are linearly independent. In this case, there are infinitely many vectors $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$. Indeed, let $\mathsf{null}(A)$ be the null-space of A; $\dim \mathsf{null}(A) = n - d$. A solution to $A\mathbf{x} = \mathbf{b}$ can be found by selecting a subset $\hat{A}$ of n linearly independent columns of A, solving $\hat{A}\hat{\mathbf{x}} = \mathbf{b}$, and setting the entries of $\mathbf{x}$ corresponding to the columns $\hat{A}$ to $\hat{\mathbf{x}}$ and the rest of the entries to zero. Then for any $\mathbf{z} \in \mathsf{null}(A)$, $\mathbf{x} + \mathbf{z}$ is also a solution to $A\mathbf{x} = \mathbf{b}$. Often, out of all solutions to $A\mathbf{x} = \mathbf{b}$, we would like to find some special solution. Quite often this special solution is the minimum 2-norm solution, i.e., the solution to

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{x}\|_2^2 \quad \text{subject to} \quad A\mathbf{x} = \mathbf{b}.$$

  The vector $A\mathbf{x}$ is the projection of $\mathbf{x}$ onto the row space of A (which is $\mathsf{span}(A^\top)$) written in the basis of rows of A. We can decompose $\mathbf{x}$ into $\mathbf{x} = \mathbf{x}^* + \mathbf{y}$ where $\mathbf{x}^* \in \mathsf{span}(A^\top)$ and $\mathbf{y} \in \mathsf{span}(A^\top)^\perp$. Since $\mathbf{x}^*$ and $\mathbf{y}$ are orthogonal the minimum of the norm of $\mathbf{x}$ will be achieved if $\mathbf{y} = 0$. Since $\mathbf{x}^* \in \mathsf{span}(A^\top)$, $\mathbf{x}^* = A^\top\mathbf{q}$ for some $\mathbf{q} \in \mathbb{R}^n$. Therefore,

$$AA^\top\mathbf{q} = \mathbf{b}.$$

  Since rows of A are linearly independent,the matrix $AA^\top$ is invertible. Hence

$$\mathbf{q} = (AA^\top)^{-1}\mathbf{b} \quad \text{and} \quad \mathbf{x}^* = A^\top(AA^\top)^{-1}\mathbf{b}.$$

- Finally, we consider the case where columns of A are linearly dependent and so are rows of A. In this case, our goal is also to find the minimum 2-norm solution to (141), though, in contrast to the previous case, the minimum in (141) is not necessarily zero. To handle this case, we consider the singular-value decomposition of A:

$$A = [U, \tilde{U}] \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V^\top \\ \tilde{V}^\top \end{bmatrix} = U\Sigma V^\top,$$

  where Σ is diagonal $k \times k$ matrix collecting all nonzero singular values of A; U is orthogonal $n \times k$, its columns constitute an orthonormal basis in the column space of A (in $\mathsf{span}(A)$); V is orthogonal $k \times k$, its columns constitute an orthonormal basis in the row space of A (in $\mathsf{span}(A^\top)$).

  Exercise Prove that the minimum 2-norm solution to (141) is given by

$$\mathbf{x}^* = V\Sigma^{-1}U^\top\mathbf{b}.$$

For greater visuality, let us consider the case where A is $n \times 2$ and $\sigma_1 \gg \sigma_2$. Let $\mathbf{x}^*$ be the least squares solution to (141). Since σ_2 is small compared to σ_1, the function $f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2$ varies slowly along the line parallel to $\mathbf{v}_2$ passing through $\mathbf{x}^*$. In Fig. 18, $\sigma_1 = 2$, $\sigma_2 = 0.2$, $\mathbf{v}_1 = (1/\sqrt{5})[1, 2]^\top$, $\mathbf{v}_2 = (1/\sqrt{5})[2, -1]^\top$, and $\mathbf{u}_1$, $\mathbf{u}_2$ are chosen as the basis in $\mathsf{range}(A)$.

If we fix $\mathbf{x}^*$ while set σ_2 to zero, the set of solutions will be the line parallel to $\mathbf{v}_2$ and passing through $\mathbf{x}^*$. The minimum 2-norm solution will be the intersection of this line with the line parallel to $\mathbf{v}_1$ and passing through to origin. It will have both components nonzero

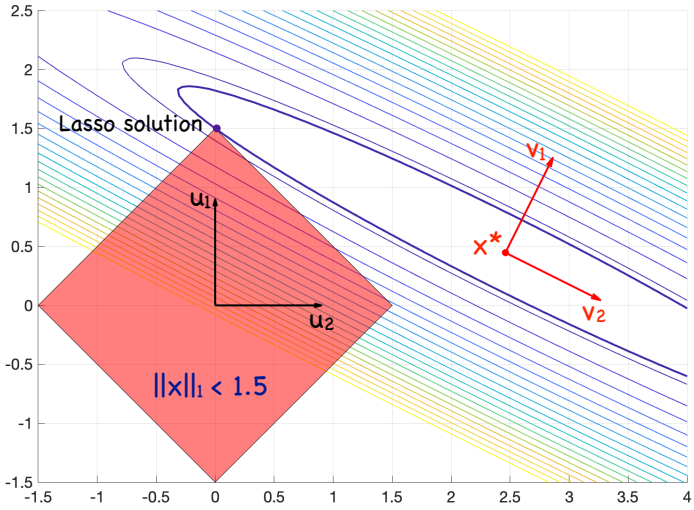


FIGURE 18. An illustration for the lasso regularization.

unless $\mathbf{v}_2$ is parallel to $\mathbf{u}_1$ or $\mathbf{u}_2$. On the other hand, if we will look for the minimum 1-norm solution, it will have one of the components zero unless the entries of $\mathbf{v}_2$ are equal in absolute value.

However, since σ_2 is small but positive, we want to have some balance between the function $f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2$ being small and some norm of $\mathbf{x}$ being small.

10.2. **Tikhonov and lasso regularization.** Now consider two problems:

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2 + \frac{\lambda}{2}\|\mathbf{x}\|_2^2 \quad \text{(Tikhonov regularized)}, \tag{142}$$

and

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_1 \quad \text{(lasso regularized)}, \tag{143}$$

If we let λ tend to infinity, then the second term in these objective functions will dominate, and the minimizer will tend to zero. One the other hand, if λ is very small and positive, it will hardly affect the minimizer of the first term provided that the minimizer is unique. However, if the family of minimizers constitute a line, a plane, or a hyperplane, the regularizing terms in (142) and (143) will select the solutions with minimal 2-norm and 1-norm, respectively. If the columns of A are nearly linearly dependent, i.e., if it has small singular values, then the solution to the regularized problems will affect only the components of the solution along the vectors $\mathbf{v}_j$ corresponding to small singular values.

Now note that the objective functions in (142) and (143) are the Lagrangian functions plus λt, respectively, for the constrained minimization problems

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\|A\mathbf{x}-\mathbf{b}\|_2^2 \quad \text{subject to} \quad t-\frac{1}{2}\|\mathbf{x}\|_2^2 \geq 0, \tag{144}$$

and

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\|A\mathbf{x}-\mathbf{b}\|_2^2 \quad \text{subject to} \quad t-\|\mathbf{x}\|_1 \geq 0, \tag{145}$$

where t is some user-supplied parameter. If no unconstrained minimizer of (144) or (145) lies in the corresponding feasible set, then the solution is located on the boundary of the feasible set resulting at a positive λ. The solution to the lasso constrained minimization problem for the example that we discussed above is shown in Fig. 18. Note that it lies at the corner of the constrained region. This gives a visual clue how lasso regularization picks out sparse solutions.

The Tikhonov regularization was introduced by A. N. Tikhonov in 1963 in the context of solving inverse problems which are often ill-posed.

The lasso regularization was proposed by R. Tibshirani in 1996 [16] as a sparsity-promoting regression method. The problem posed in [16] was of the form (145). Two methods for solving it numerically were discussed. The simpler one of them, which also has much better worst-case scenario performance guarantees, is based on the decomposition $\mathbf{x}=\mathbf{x}_+-\mathbf{x}_-$ where $\mathbf{x}_+$ and $\mathbf{x}_-$ are vectors with nonnegative components. Then (145) is equivalent to a quadratic programming problem with $2d$ variables and $2d+1$ linear constraints.

Exercise Set up this quadratic programming problem: write out the objective function and the constraints.

10.3. **Coordinate descend.** The key reference for this section is [2] (Section 7.3). The coordinate descend method (CD) proceeds by selecting a coordinate i_k at each step k and updating $\mathbf{x}$ according to:

$$\mathbf{x}_{k+1}(i_k) = \mathbf{x}_k(i_k) - \alpha_k \nabla_{i_k} f(\mathbf{x}_k), \tag{146}$$

where $\mathbf{x}(i)$ is the ith coordinate of $\mathbf{x}$ and $\nabla_i f(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}(i)}(\mathbf{x})$. Specific versions of CD are distinguished in the way the sequences α_k and i_k are defined. The sequence of coordinates can be cyclic or randomized. The randomized version is preferable since it is less likely to choose an unfortunate sequence of coordinates.

As we have mentioned, contrary to GD, CD may fail to converge to a stationary point even on a continuously differentiable function – such an example of a function in three variables was offered by Powell [8]. However, if f is strongly convex, one can establish linear rate of convergence for CD. Recall the definition and an important property of strong convexity:

$$f(\mathbf{x}+\mathbf{p}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{p} + \frac{c}{2}\|\mathbf{p}\|^2 \;\text{ implying }\; 2c[f(\mathbf{x})-f(\mathbf{x}^*)] \leq \|\nabla f(\mathbf{x})\|_2^2 \;\; \forall \mathbf{x}\in\mathbb{R}^d.$$

Theorem 6. *Suppose that the objective function $f : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable, strongly convex with constant c, and ∇f is coordinate-wise Lipschitz, i.e.,*

$$|\nabla_i f(\mathbf{x} + \Delta\mathbf{x}(i)\mathbf{e}_i) - \nabla_i f(\mathbf{x})| \leq L_i |\Delta\mathbf{x}(i)|, \quad i = 1, \ldots, d. \tag{147}$$

In addition, suppose that $\alpha_k = L_{\max}^{-1}$ ($L_{\max} = \max_i L_i$) and that i_k is chosen independently and uniformly from $\{1, \ldots, d\}$ for all $k \in \mathbb{Z}_+$. Then for all $k \in \mathbf{Z}_+$, the iteration (146) yields

$$\mathbb{E}[f(\mathbf{x}_k)] - f^* \leq \left(1 - \frac{c}{dL_{\max}}\right)^k (f(\mathbf{x}_0) - f^*). \tag{148}$$

Proof. Coordinate-wise Lipschitz continuity of ∇f yields:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \nabla_{i_k} f(\mathbf{x}_k)[\mathbf{x}_{k+1}(i_k) - \mathbf{x}_k(i_k)] + \frac{1}{2} L_{i_k} [\mathbf{x}_{k+1}(i_k) - \mathbf{x}_k(i_k)]^2.$$

Hence, with stepsize $\alpha_k = L_{\max}^{-1}$, it follows that

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \leq -\frac{1}{L_{\max}} \nabla_{i_k} f(\mathbf{x}_k)^2 + \frac{1}{2L_{\max}} \nabla_{i_k} f(\mathbf{x}_k)^2 = -\frac{1}{2L_{\max}} \nabla_{i_k} f(\mathbf{x}_k)^2.$$

Taking expectations with respect to the distribution of i_k we get:

$$\begin{aligned}
\mathbb{E}_{i_k}\left[f(\mathbf{x}_{k+1})\right] - f(\mathbf{x}_k) &\leq -\frac{1}{2L_{\max}} \mathbb{E}_{i_k}\left[\nabla_{i_k} f(\mathbf{x}_k)^2\right] \\
&= -\frac{1}{2L_{\max}} \frac{1}{d} \sum_{i=1}^{d} \nabla_i f(\mathbf{x}_k)^2 = -\frac{1}{2dL_{\max}} \|\nabla f(\mathbf{x}_k)\|_2^2.
\end{aligned}$$

Then it follows that

$$\begin{aligned}
\mathbb{E}_{i_k}\left[f(\mathbf{x}_{k+1})\right] - f^* &\leq \left[f(\mathbf{x}_k) - f^*\right] - \frac{1}{2dL_{\max}} \|\nabla f(\mathbf{x}_k)\|_2^2 \\
&\leq \left[f(\mathbf{x}_k) - f^*\right] - \frac{c}{dL_{\max}} \left[f(\mathbf{x}_k) - f^*\right] \\
&= \left(1 - \frac{c}{dL_{\max}}\right) \left[f(\mathbf{x}_k) - f^*\right].
\end{aligned}$$

Taking total expectation and applying the last inequality for all $k \in \mathbb{Z}_+$ we obtain the desired result. □

This simple randomized CD method is competitive with GD for objective functions of the form

$$f(\mathbf{w}) = \frac{1}{2} \|X\mathbf{w} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|_1,$$

where X is a sparse matrix of data. Another advantage of CD is that it is parallelizable, and this feature of it is being actively explored.

References

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[2] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[3] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. Computer Vision and Pattern Recognition (CVPR)*, vol. 6, pp. 770–778, 2016.

[6] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 2 ed., 2006.

[7] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems," *Acta Numerica*, pp. 1–137, 2005.

[8] M. J. D. Powell, "On search directions for minimization algorithms," *Mathematical Programming*, vol. 4, pp. 193–201, 1973.

[9] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 1951.

[10] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transcactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[11] W. E and B. Yu, "The deep ritz method: A deep learning-based numerical algorithm for solving variational problems," *Communications in Mathematics and Statistics*, vol. 6, pp. 1–12, 2018.

[12] Y. Khoo, J. Lu, and L. Ying, "Solving for high-dimensional committor functions using artificial neural networks," *Research in the Mathematical Sciences*, vol. 6, no. 1, p. 1, 2019.

[13] Q. Li, B. Lin, and W. Ren, "Computing committor functions for the study of rare events using deep learning," *Journal of Machine Learning Research*, vol. 151, p. 054112, 2019.

[14] G. M. Rostkoff and E. Vanden-Eijnden, "Learning with rare data: Using active importance sampling to optimize objectives dominated by rare events." arXiv:2008.06334v1.

[15] J. W. Demmel, *Applied Numerical Linear Algebra*. SIAM, 1997.

[16] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society*, vol. 58, no. 1, pp. 267–288, 1996.