7 Numerical Opimization Techniques

7.1 Minimization of a Real-valued Function

In many real-world problems, we would like to construct mathematical models that can predict the observable outcomes of some natural phenomena. Searching for a general model without some guidance is usually intractable. We thus often rely on our intuition or a specific mathematical framework that can provide a sense of what such a model should look like, yielding a parametric family of candidate models, from which we need to choose the best one that can explain the previously obtained observations.

Given a parametric model, we can construct a loss function (a.k.a. a cost function) $\mathcal{L}(\beta|\text{data})$ that assesses how well the current model with parameter $\beta \in \mathbb{R}^n$ explains the observed data; the smaller the loss function, the better the model. Hence, the problem of choosing a "best" model amounts to minimizing the loss function over all possible values of β:

$$\beta_* = \arg\min_{\beta\in\Omega} \mathcal{L}(\beta|\text{data}),$$

where $\Omega \subseteq \mathbb{R}^n$, called the feasibility set, may constrain the possible values of β that the models may take.

In this subsection, we will study methods for minimizing a function without any constraints, i.e. assuming $\Omega = \mathbb{R}^n$. We are usually interested in finding a global minimum, because we are interested in finding an optimal model that is as best as possible, but there is really no good algorithm for finding the global extrema of a general function. There are, however, many iterative algorithms that yield local minima. Luckily, imposing the loss function and feasibility set to be convex provides a sufficient condition to ensure that a local minimum is also a global minimum. We will thus discuss the notion of convexity first and then describe some well-known iterative algorithms for finding local minima.

7.1.1 Convex Sets and Convex Functions

7.1.1.i Definitions and Criteria. We are often interested in minimizing a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ restricted to some subset $A \subseteq \mathbb{R}^n$. It turns out the geometry of the search space A and certain functional properties of f both have a profound consequence on the nature of possible minima of f in A. We will now review that the property of convexity plays a key role in ensuring the uniqueness of a solution obtained via numerical optimization algorithms. Let us first define this notion of convexity.

Definition 7.1 (Convex Set)**.** *A set $A \subseteq \mathbb{R}^n$ is called convex if $\forall\, x, y \in A$ and $\forall\, \tau \in [0, 1]$,*

$$(1-\tau)\,x + \tau\,y \in A.$$

REMARK 7.1. *That is, A is convex if given any two points in A, the entire line segment connecting the two points is contained in A.*

Example 7.1. *Some examples of a convex set in $\mathbb{R}^n$ are: $\mathbb{R}^n$, an open ball of radius $r > 0$, a simplex, and a hyperplane.*

Definition 7.2 (Convex Function)**.** *Let $A \subseteq \mathbb{R}^n$ be a convex set. A real-valued function $f : A \to \mathbb{R}$ is called convex if $\forall\, x, y \in A$ and $\forall\, \tau \in [0, 1]$,*

$$(1-\tau)f(x) + \tau f(y) \geq f((1-\tau)x + \tau y). \tag{7.1}$$

If the strict inequality holds for $\tau \in (0, 1)$ and for all $x, y \in A$, $x \neq y$, then f is called strictly convex.

REMARK 7.2. *One can equivalently define a function to be convex if its epigraph (the region above the graph) is convex.*

REMARK 7.3. *Instead of $(1-\tau)x+\tau y$, one could write $x+\tau(y-x)$, which makes it more apparent that it parameterizes a line segment from x to y as τ increases from 0 to 1.*

Example 7.2. *Any norm $\| \ \|$ defined on a vector space V is a convex function, since the triangle inequality implies that $\forall v, w \in V$ and $\tau \in [0, 1]$,*

$$\|\tau v + (1-\tau)w\| \leq \tau\|v\| + (1-\tau)\|w\|.$$

Example 7.3. *Let $\| \ \| : V \to \mathbb{R}$ be a norm on vector space V. Then, the solid ball $B(R) = \{v \in V \mid \|v\| \leq R\}$ of radius R is a convex set, since $\forall v, w \in B(R)$ and $\tau \in [0, 1]$, we have*

$$\|\tau v + (1-\tau)w\| \leq \tau\|v\| + (1-\tau)\|w\| \leq \tau R + (1-\tau)R = R,$$

implying that $\tau v + (1-\tau)w \in B(R)$. This fact explains why the ℓ_p-norm has the restriction $p \geq 1$.

Example 7.4. *Let $\| \ \| : V \to \mathbb{R}$ be a function satisfying the homogeneity and non-negativity conditions of a norm. We claim that $\| \ \|$ satisfies the triangle inequality, i.e. $\| \ \|$ is a norm, iff the associated unit ball $B(1) = \{v \in V \mid \|v\| \leq 1\}$ is convex. In Example 7.3, we have already proven the "only if" part.*

To see the "if" part of this claim, first note that the triangle equality

$$\|x + y\| \leq \|x\| + \|y\|$$

always holds if either $x = 0$ or $y = 0$. Hence, we only need to prove that the triangle inequality holds for non-zero x and y. For any $x, y \in V \setminus \{0\}$, consider

$$z(\tau) \equiv \frac{y}{\|y\|}\tau + (1-\tau)\frac{x}{\|x\|}\,, \quad \tau \in [0, 1].$$

Since $y/\|y\| \in B(1)$ and $x/\|x\| \in B(1)$, the assumed convexity of $B(1)$ implies that $z \in B(1)$; hence, we must have

$$\|z(\tau)\| \leq 1\,, \quad \forall \tau \in [0, 1].$$

In particular, choosing $\tau = \|y\|/(\|x\| + \|y\|)$ and multiplying both sides of the inequality by $\|x\| + \|y\|$ establish the triangle inequality for $x \neq 0$ and $y \neq 0$.

For a differentiable function, there is an equivalent definition for checking whether it is convex:

Theorem 7.1 (First Order Convexity Condition). *Let $A \subseteq \mathbb{R}^n$ be a convex set. A differentiable function $f : A \to \mathbb{R}$ is convex iff $\forall\, x, y \in A$,*

$$f(y) \geq f(x) + (y - x) \cdot \nabla f(x).$$

Similarly, f is strictly convex iff $\forall\, x, y \in A, x \neq y$,

$$f(y) > f(x) + (y - x) \cdot \nabla f(x).$$

Proof. Let us prove the convexity claim first.
($\Rightarrow$) If f is convex, then $\forall\, x, y \in A$, we can arrange (7.1) as

$$\tau f(y) \geq \tau f(x) + f(x + \tau(y - x)) - f(x).$$

For $\tau > 0$, we can divide the inequality by τ and get

$$f(y) \geq f(x) + \frac{f(x + \tau(y - x)) - f(x)}{\tau}.$$

In the limit $\tau \to 0^+$, the second term on the right-hand side becomes the directional derivative $(y - x) \cdot f(x)$, proving the implication.
($\Leftarrow$) For any $\tau \in [0, 1]$ and any $x, y \in A$, let $z = (1 - \tau)x + \tau y$; note that $z \in A$, since A is convex. Then, the assumed inequality condition implies that

$$f(x) \geq f(z) + (x - z) \cdot \nabla f(z) \quad \text{and} \quad f(y) \geq f(z) + (y - z) \cdot \nabla f(z)$$

Multiplying the first inequality by $1 - \tau$ and the second inequality by τ and then adding them, we get

$$(1 - \tau) f(x) + \tau f(y) \geq f(z) + ((1 - \tau)x + \tau y - z) \cdot \nabla f(z).$$

Since $z \equiv (1 - \tau)x + \tau y$, the second term is actually 0, and we get (7.1), thereby proving the claim.

The second claim regarding strict convexity can be proved using the exact same approach upon replacing the inequality with strict inequality. $\square$

If a function is twice differentiable, then one can use the mean value theorem to prove the following useful theorem which plays a central role in optimization:

Theorem 7.2 (Second Order Convexity Condition). *Let $A \subseteq \mathbb{R}^n$ be a convex set. A twice differentiable function $f : A \to \mathbb{R}$ is convex iff its Hessian matrix $Q(x)$ is positive semi-definite for all $x \in A$. Similarly f is strictly convex iff $Q(x)$ is positive definite for all $x \in A$.*

Proof. Exercise (Hint: Use Theorem 7.1 and Corollary A.3). $\square$

7.1.1.ii Consequences of Convexity.

We now show why the notion of convexity is useful for numerical optimization.

Theorem 7.3. *Let $A \subseteq \mathbb{R}^n$ be a convex set. If $f : A \to \mathbb{R}$ is a convex function, then any local minimum of f is also a global minimum. The set of minima is convex.*

Proof. Suppose the contrary, and assume that $x \in A$ is a local minimum of f, but there exists $y \in A$ such that $f(y) < f(x)$. Since A is convex, we have $z(\tau) \equiv (1-\tau)x + \tau y \in A$, $\forall \tau \in [0,1]$, and

$$(1-\tau)f(x) + \tau f(y) \geq f(z(\tau)).$$

Since $f(x) > f(y)$, this inequality implies that

$$f(x) = (1-\tau)f(x) + \tau f(x) > (1-\tau)f(x) + \tau f(y) \geq f(z(\tau)).$$

Hence, $\forall \tau \in [0,1]$, we have the strict inequality

$$f(x) > f(z(\tau)).$$

But, for any $\epsilon > 0$, choosing $\tau = \frac{\epsilon}{2\|y-x\|_2}$ shows that

$$\|x - z(\tau)\|_2 = \tau \|y-x\|_2 = \frac{\epsilon}{2} < \epsilon$$

and, thus, that $z(\tau)$ is in the ϵ-neighborhood of x. Hence, $f(x)$ cannot be a local minimum, contradicting the initial assumption.

To show that the set of minima is convex, let x and y be any minima points in A. Then, since they are both global minima, we must have $f(x) = f(y)$. For all $\tau \in [0,1]$, the convexity of f implies that it must satisfy

$$f(x) = (1-\tau)f(x) + \tau f(x) = (1-\tau)f(x) + \tau f(y) \geq f(z(\tau)).$$

Since x is a global minimum, the above inequality must be saturated, and $z(\tau)$ lies in the set of minima. □

Hence, if a function f is convex on a convex search space, then we only need to find one local minimum to find the global minimum value of f. But, the convexity condition alone does not exclude the possibility of having many degenerate global minima. For example, the constant function $f(x) \equiv 0$ is convex, but all points in $\mathbb{R}^n$ are global minima. Luckily, strict convexity does guarantee that there cannot be any degenerate global minima:

Theorem 7.4. *Let $A \subseteq \mathbb{R}^n$ be a convex set. If $f : A \to \mathbb{R}$ is a strictly convex function, then it has at most one local minimum which, if it exists, must also be the unique global minimum.*

Proof. Suppose f has two distinct local minima x and y. By Theorem 7.3, we see that both x and y must be global minima; as a result, we must have $f(x) = f(y)$. For any $\tau \in (0,1)$, the convexity of A implies that $(1-\tau)x + \tau y \in A$, and the strict convexity of f implies that

$$f(x) = (1-\tau)f(x) + \tau f(x) = (1-\tau)f(x) + \tau f(y) > f((1-\tau)x + \tau y),$$

contradicting the fact that x is a global minimum. □

REMARK 7.4. *You might be wondering whether a convex function always has a minimum. The answer actually depends on the domain $A \subseteq \mathbb{R}^n$.*

For example, if A is open, then a convex function may not have a minimum; e.g., both $f(x) = e^{-x}$ on $A = \mathbb{R}$ and $f(x) = -\log(x)$ on $A = (0, \infty)$ are strictly convex functions, but they have no minimum in the respective domain.

By contrast, if A is compact (closed and bounded), then the Weierstraß Extreme Value Theorem implies that a continuous strictly convex function always has a unique minimum and maximum in A.

If A is closed but not bounded, then the Weierstraß Extreme Value Theorem applied to the intersection of A and a sufficiently large closed ball again implies that a continuous strictly convex function satisfying the coercive condition,

$$\lim_{\|x\| \to \infty} f(x) = \infty,$$

achieves a unique minimum in A.

REMARK 7.5. *A convex function is always continuous in the interior of its domain, but it might be discontinuous on the boundary.*

Given these important considerations, we will assume in the remainder of this subsection, unless stated otherwise, that the Hessian of a function to be optimized is positive definite, thereby guaranteeing that the function is strictly convex and that its local minimum, which we will assume exists in the domain, is the unique global minimum.

7.1.2 Gradient Descent Algorithms

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. At any given point $x_0 \in \mathbb{R}^n$, the directional derivative $\nabla_v f(x_0) \equiv v^t \nabla f(x_0)$ along a vector $v \in \mathbb{R}^n$ with unit ℓ_2-norm yields the rate of change in the direction specified by v. Since the Cauchy-Schwarz inequality implies that

$$|\nabla_v f(x_0)| \leq \|\nabla f(x_0)\|_2 \|v\|_2 = \|\nabla f(x_0)\|_2,$$

we see that $f(x_0)$ increases the most along the $\nabla f(x_0)$ direction (steepest ascent) and decreases the most along the $-\nabla f(x_0)$ direction (steepest descent). These gradient directions are also orthogonal to the level set passing through x_0:

EXERCISE 7.1. *Show that $\nabla f(x_0)$ is orthogonal to any vector that is tangent to the level set $\{x \in \mathbb{R}^n \mid f(x) = f(x_0)\}$ at x_0.*

This observation motivates us to search for a minimum of f by updating the coordinate in the direction of steepest descent, $-\nabla f(x_0)$, which we assume to be non-zero; if $\nabla f(x_0) = 0$, then we have found a critical point and need to test whether this is a local minimum that we can keep as a candidate solution. Thus, the proposed update scheme is

$$\boxed{x_{k+1} = x_k - \alpha_k \, g_k, \;\; \alpha_k > 0}\,, \tag{7.2}$$

where $g_k \equiv \nabla f(x_k)$. Many algorithms exist to guide the choice of α_k, which is called the step size or the learning rate. There are trade-offs between small and large step sizes.

7.1.2.i Fixed Step Size. One simple approach is to specify a fixed step size α for all updates.

```
                Gradient Descent Method Pseudocode:  Fixed Step Size

Given:  A convex differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
        Step size $\alpha$.
        Tolerance value $\epsilon$ for terminating the algorithm.

function Gradient_Descent_Fixed_Step_Size($f, \alpha, \epsilon$):
   Initialize $x_1 \in \mathbb{R}^n$
   $k = 0$
   do {
        $k$++
        $g_k = \nabla f(x_k)$
        $x_{k+1} = x_k - \alpha g_k$
   } while convergence test $h(x_k, x_{k+1}) > \epsilon$
   $x_* = x_{k+1}$
   return $x_*$
```

REMARK 7.6 (Advantages)**.** *This approach is very simple to implement and has little computational overhead in terms of complexity and memory usage.*

REMARK 7.7 (Disadvantages)**.** *One severe limitation of this approach is that the step size is not adaptive, thereby leading to several disadvantages:*

1. *The number of required iteration may be large; see Figure 7.1 (top).*
2. *If the step size is too small, it is possible to get trapped in a local minimum and never be able to escape from the basin of attraction. For a non-convex cost function, this local minimum may not be the global minimum.*
3. *If the step size is too large, then the algorithm might overshoot the location of the true minimum and fail to converge; see Figure 7.1 (bottom). In fact, for a quadratic function, the fixed-step-size algorithm converges for all initialization iff*

$$0 < \alpha < \frac{2}{\lambda_{\max}(Q)},$$

   where $\lambda_{\max}(Q)$ is the largest eigenvalue of the Hessian Q.

Moreover, the algorithm cannot be applied to non-differentiable functions.

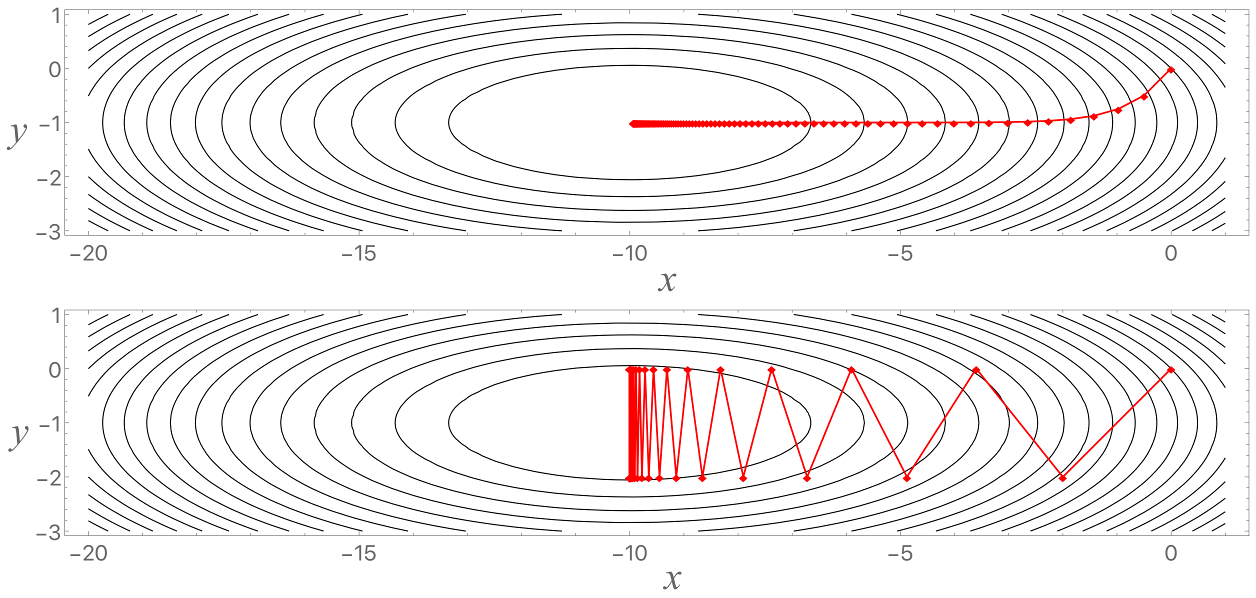

Figure 7.1: Implementation of gradient descent with a fixed step size α for minimizing $f(x,y) = -\begin{pmatrix} -1 & -1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \frac{1}{2}\begin{pmatrix} x & y \end{pmatrix}\begin{pmatrix} 0.1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$. The initial starting point is at the origin. (Top Figure) $\alpha = 0.5$ is chosen. Each small update steadily decreases f, but it takes a large number of steps to approach the global minimum at $(-10, -1)$. (Bottom Figure) $\alpha = 2$ is chosen. It takes fewer steps to get near the minimum, but then it oscillates between $(-10, 0)$ and $(-10, -2)$ and does not converge to the global minimum.

7.1.2.ii Exact Line Search. Another well-known method is to choose α_k at each iteration step by minimizing the 1D function $\phi(\alpha) \equiv f(x_k - \alpha\, g_k)$:

$$\alpha_k = \arg\min_{\alpha \geq 0} f(x_k - \alpha\, g_k).$$

For differentiable f, one usually finds the critical points of $\phi(\alpha)$ by solving for the values of α satisfying the condition $\phi'(\alpha) = 0$ and finds the minimum among them. For a differentiable convex function, such as the quadratic function

$$f(x) = a - b^t x + \frac{1}{2} x^t Q x \tag{7.3}$$

with a positive definite Hessian matrix Q, there will be a unique minimum that is the sole critical point of $\phi(\alpha)$. Substituting $p_k = -g_k = b - Qx_k$ in (1.13), we see that the optimal value of α_k for the quadratic function (7.3) is

$$\boxed{\alpha_k = \frac{g_k^t g_k}{\langle g_k, g_k \rangle_Q}}. \quad \text{(for quadratic functions)}$$

In general, we obtain α_k by imposing that the directional derivative of f along $-g_k$ vanishes at x_{k+1}, i.e. that

$$0 = \left.\frac{df(x_k - \alpha g_k)}{d\alpha}\right|_{\alpha_k} = -g_k^t\, \nabla f(x_{k+1}) = -g_k^t\, g_{k+1}.$$

This condition proves the following important result:

Theorem 7.5. *In the exact line search algorithm for gradient descent, all consecutive updates are orthogonal to each other.*

Taylor expanding f around x_k shows that, for α sufficienlty small such that $0 < \alpha \ll 1$ and $\alpha < \alpha_k$,

$$f(x_k - \alpha_k g_k) - f(x_k) \leq f(x_k - \alpha g_k) - f(x_k) \approx -\alpha \|g_k\|_2^2\,.$$

Thus, as long as $g_k \neq 0$, each update will decrease the value of f. In numerical implementations, gradients may become very small but still non-zero, because of rounding-off errors intrinsic to digital computers. Thus, to terminate the algorithm, one usually checks whether $\|g_k\|_2$, the absolute updates $\|f(x_{k+1}) - f(x_k)\|_2$, $\|x_{k+1} - x_k\|_2$, or the relative updates $\|f(x_{k+1}) - f(x_k)\|_2 / \|f(x_k)\|_2$, $\|x_{k+1} - x_k\|_2 / \|x_k\|_2$ are sufficiently small. Let us call this choice of convergence test function $h(x_k, x_k + 1)$. This algorithm is summarized as follows:

```
Gradient Descent Method Pseudocode:  Exact Line Search

Given:  A convex differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
        Tolerance value $\epsilon$ for terminating the algorithm.

function Gradient_Descent_Exact_Line_Search($f, \epsilon$):
```

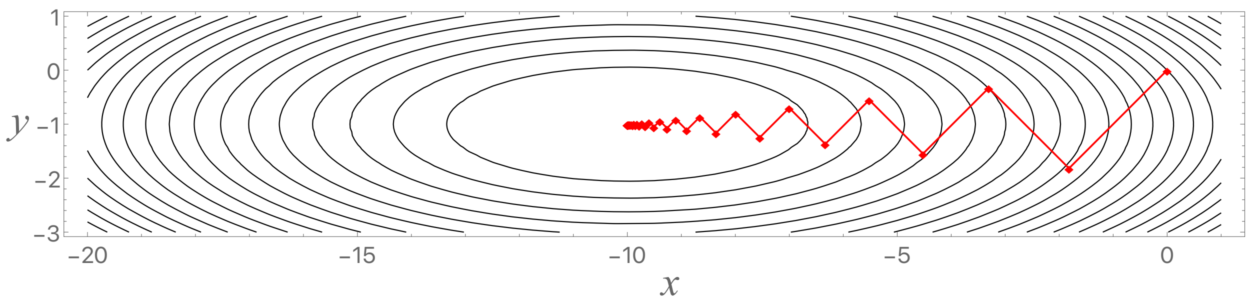

Figure 7.2: Implementation of gradient descent with an exact line search for minimizing $f(x,y) = -\begin{pmatrix} -1 & -1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \frac{1}{2}\begin{pmatrix} x & y \end{pmatrix}\begin{pmatrix} 0.1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$. The initial starting point is at the origin as before. It can be seen that adjacent steps are orthogonal to each other and lead to a zigzag path along the long major axis, thereby reducing the efficiency of search, typical of the situation when the condition number of the Hessian is large (10 in this example). The step size is seen to become smaller at each iteration, and the algorithms converges to the global minimum at $(-10, -1)$ relatively quickly. (Note: if the condition number of the Hessian were 1 instead, then the contours of f would have been circles, and the line search would have terminated in a single step.)

```
Initialize $x_1 \in \mathbb{R}^n$
$k = 0$
do {
    $k$++
    $g_k = \nabla f(x_k)$
    calculate $\alpha_k \geq 0$ s.t.  $-g_k^t \nabla f(x_k - \alpha_k g_k) = 0$
        (if $f$ is quadratic, then $\alpha_k = \frac{g_k^t g_k}{\langle g_k, g_k \rangle_Q}$)
    $x_{k+1} = x_k - \alpha_k g_k$
} while convergence test $h(x_k, x_{k+1}) > \epsilon$
$x_* = x_{k+1}$
return $x_*$
```

REMARK 7.8 (Advantages)**.** *Some advantages are*

1. *For a quadratic function, the gradient descent method with exact line search converges to the true minimum for all initialization.*

2. *The algorithm is simple and easy to implement.*

REMARK 7.9 (Disadvantages)**.** *Similar to the fixed-step-size approach, some disadvantages are*

1. *The steps might oscillate when the Hessian matrix Q has a large condition number.*

2. *The algorithm cannot be applied to non-differentiable functions.*

7.1.3 Conjugate Gradient Methods

7.1.3.i Conjugate Gradient Method for Quadratic Functions. Advantages of the gradient descent method are that it is simple to implement and provides a guideline for choosing a general direction of coordinate update for minimizing a quadratic function $f : \mathbb{R}^n \to \mathbb{R}$. A notable disadvantage of the gradient descent method, however, is that it can lead to oscillations in path and slow convergence to the optimal solution when the Hessian matrix Q has a large condition number. By contrast, we have seen in Section 1.11.1 that the conjugate direction method always terminates in at most n line searches along directions conjugate w.r.t. to the Hessian. When $n \gg 1$, however, choosing a random set of conjugate directions can lead to unnecessary calculations of all n steps and slow down the performance.

The conjugate gradient method combines the ideas of gradient descent and conjugate direction methods to achieve an efficient and reliable optimization method that derives mutually conjugate directions from iterative gradient descent directions. More precisely, we iteratively apply the Gram-Schmidt orthogonalization process (Theorem 1.25) to the gradient descent directions $-g_1, -g_2, -g_3 \dots$ The pseudocode for this algorithm is thus:

```
        Conjugate Gradient Method Pseudocode for Quadratic Functions

Given:  A quadratic function $f : \mathbb{R}^n \to \mathbb{R}$ with positive definite Hessian $Q$.
        Tolerance value $\epsilon$ for terminating the algorithm.
Objective:  Apply the conjugate direction method by iteratively learning
        a new conjugate direction $p_k$ from $-\nabla f(x_k)$.

function Conjugate_Gradient($f, \epsilon$):
    Initialize $x_1 \in \mathbb{R}^n$
    for $k = 1, \dots, n$:
        $g_k = \nabla f(x_k)$
        if $\|g_k\|_2 < \epsilon$, break
        if $k = 1$
            $p_1 = -g_1$
        else:
            $p_k = -g_k + \frac{g_k^t g_k}{g_{k-1}^t g_{k-1}} p_{k-1}$ (Gram-Schmidt $Q$-orthogonalization of $-g_k$ to
                    $p_1, \dots, p_{k-1}$ using Theorem 1.25, Corollary 7.1 and (7.6))
        $\alpha_{*k} = \arg\min_{\alpha_k} f(x_k + \alpha_k p_k) = -p_k^t g_k / \langle p_k, p_k \rangle_Q$ (from (1.13))
        $x_{k+1} = x_k + \alpha_{*k} p_k$
        $x_* = x_{k+1}$
    return $x_*$
```

The Gram-Schmidt Q-orthogonalization of the negative gradient $-g_k$ to the previous conjugate directions $p_1, \dots, p_{k-1}$ is simplified by the following two theorems:

Theorem 7.6. *The gradients at distinct coordinate updates are orthogonal in the conjugate gradient method for a quadratic function. That is*

$$g_k^t\, g_i = 0, \;\; k \neq i.$$

Proof. Let Q denote the Hessian of the quadratic function. Without loss of generality, assume that $k > i$. Then, since the Gram-Schmidt orthogonalization of $-g_i$ for obtaining p_i is

$$p_i = -g_i + \sum_{j=1}^{i-1} \frac{\langle p_j, g_i \rangle_Q}{\langle p_j, p_j \rangle_Q} p_j,$$

we have

$$g_k^t g_i = g_k^t \left(-p_i + \sum_{j=1}^{i-1} \frac{\langle p_j, g_i \rangle_Q}{\langle p_j, p_j \rangle_Q} p_j \right) = 0,$$

since g_k is orthogonal to $p_i, i < k$, by Lemma 1.2. □

Theorem 7.7. *In the conjugate gradient method for a quadratic function with the Hessian Q,*

$$\langle p_j, g_k \rangle_Q = 0, \;\; j < k - 1.$$

That is, g_k is already conjugate to $p_1, \ldots, p_{k-2}$ w.r.t. the Hessian matrix Q.

Proof. By Lemma 1.1, we have $Qp_j = \alpha_{*j}^{-1}(g_{j+1} - g_j)$. Hence,

$$\langle p_j, g_k \rangle_Q \equiv p_j^t Q g_k = \alpha_{*j}^{-1} \, (g_{j+1} - g_j)^t g_k = 0,$$

where the last equality follows from Theorem 7.6, as the condition $j < k - 1$ implies that $j + 1 \neq k$ and $j \neq k$. □

This theorem implies that we only need to make $-g_k$ conjugate to p_{k-1} when constructing p_k. We thus have:

Corollary 7.1 (Conjugate Gradient Gram-Schmidt)**.** *In the conjugate gradient method for a quadratic function, the k-th conjugate direction, for $k \geq 2$, is*

$$p_k = -g_k + \beta_{k-1} \, p_{k-1}, \quad \textit{where } \beta_{k-1} = \frac{\langle p_{k-1}, g_k \rangle_Q}{\langle p_{k-1}, p_{k-1} \rangle_Q}.$$

EXERCISE 7.2. *Use the fact that $\alpha_{*k-1} Q p_{k-1} = (g_k - g_{k-1})$ to show that β_{k-1} can be also expressed as*

$$\boxed{\beta_{k-1} = \frac{g_k^t (g_k - g_{k-1})}{p_{k-1}^t (g_k - g_{k-1})}} \qquad \text{(Hestenes-Stiefel Formula)} \qquad (7.4)$$

$$\boxed{\beta_{k-1} = \frac{g_k^t (g_k - g_{k-1})}{g_{k-1}^t g_{k-1}}} \qquad \text{(Polak-Ribière Formula)} \qquad (7.5)$$

$$\boxed{\beta_{k-1} = \frac{g_k^t g_k}{g_{k-1}^t g_{k-1}}} \qquad \text{(Fletcher-Reeves Formula)} \qquad (7.6)$$

The above three expressions are equivalent for a quadratic function, but they may be different for a non-quadratic function, as we will discuss below in Section 7.1.3.ii. These formulae *are useful, because they readily yield the new conjugate direction at each iteration without performing matrix multiplications involving the Hessian. In fact, we do not even need to know what the Hessian is in order to compute the new direction. In Section 7.1.3.ii, we will discuss how these formulae may be used in conjugate gradient methods for non-quadratic functions, for which the Hessian generally changes after each coordinate update.*

The following theorem tells us how many line searches along adaptive conjugate directions are needed in general before we obtain the global minimum:

Theorem 7.8. *Using the same notation as above, the number m of exact line searches needed in the conjugate gradient method for minimizing a quadratic function $f : \mathbb{R}^n \to \mathbb{R}$ is the maximum number m such that*

$$g_1, Qg_1, Q^2 g_1, \ldots, Q^{m-1} g_1$$

are linearly independent. (Note that $m \leq n$, since $\mathbb{R}^n$ cannot have more than n linearly independent vectors.)

Proof. By Theorem 7.6, the set $\{g_1, g_2, \ldots, g_m\}$ forms an orthogonal basis of an m-dimensional subspace $V \subset \mathbb{R}^n$. Note that by construction, p_k itself is a linear combination of $g_1, \ldots, g_k$, as it arises from the Gram-Schmidt orthogonalization of these vectors w.r.t. Q. We claim that each g_k, $k \leq m$, is a linear combination of $g_1, \ldots, Q^{k-1} g_1$; we will prove this claim by induction. The claim is clearly true for $k = 1$. Assume that the claim is true for k, $1 \leq k < m$, which thus also assumes that p_k is a linear combination of $g_1, Qg_1, \ldots, Q^{k-1} g_1$. But, for a quadratic function, we have $g_{k+1} = g_k + \alpha_{*k} Q p_k$; since the induction hypothesis assumes that both g_k and p_k are linear combinations of $g_1, Qg_1, \ldots, Q^{k-1} g_1$, we see that g_{k+1} is a linear combination of $g_1, \ldots, Q^k g_1$, thereby proving the claim. This claim has two immediate consequences:

1. For $k \leq m$, the coefficient of $Q^{k-1} g_1$ in the expansion $g_k = \sum_{i=0}^{k-1} \gamma_i \, Q^i g_1$ must be non-zero, since g_k is orthogonal to $\mathrm{Span}\{g_1, \ldots, g_{k-1}\} = \mathrm{Span}\{g_1, Qg_1 \ldots, Q^{k-2} g_1\}$. In particular, the expansion coefficient of $Q^{m-1} g_1$ in g_m, and thus p_m, is non-zero.

2. $g_1, Qg_1, \ldots, Q^{m-1} g_1$ must be linearly independent since they must span V.

Since we assume that $g_{m+1} = 0 = g_m + \alpha_{*m} Q p_m$, and the expansion of $Q p_m$ contains $Q^m g_1$ with a non-zero coefficient, we conclude that $Q^m g_1$ can be written as a linear combination of $g_1, \ldots, Q^{m-1} g_1$. Induction now shows that for any $k > m$, $Q^k g_1$ can be expressed as a linear combination of $g_1, \ldots, Q^{m-1} g_1$. □

EXERCISE 7.3. *Suppose the Hessian matrix Q has ℓ distinct eigenvalues $\lambda_1, \ldots, \lambda_\ell$. We can thus decompose $\mathbb{R}^n = V_{\lambda_1} \oplus V_{\lambda_2} \oplus \cdots \oplus V_{\lambda_\ell}$, where V_{λ_i} denotes the possibly degenerate eigenspace of Q corresponding to the eigenvalue λ_i. In this decomposition, g_1 can be uniquely expressed as*

$$g_1 = v_1 + v_2 + \cdots + v_\ell \,, \quad \text{where } v_i \in V_{\lambda_i}, i = 1, \ldots, \ell. \tag{7.7}$$

Show that the number m of required line searches in Theorem 7.8 is equal to the number of non-zero v_i in (7.7). (Hint: Consider a Vandermonde matrix constructed using the eigenvalues corresponding to the nonzero v_i).

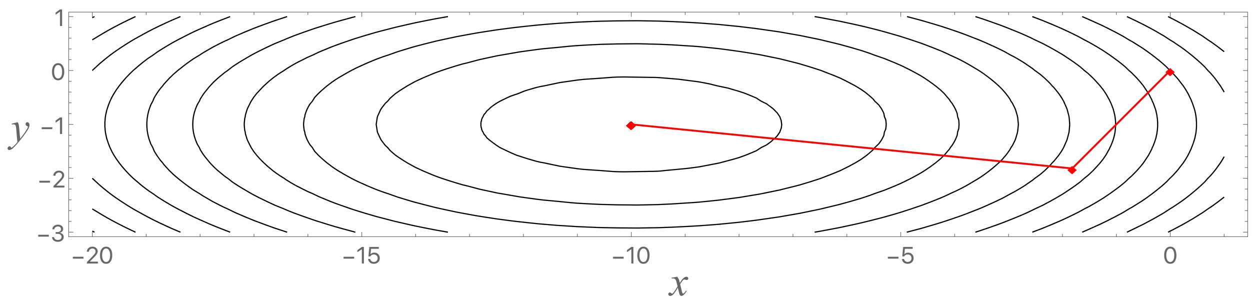

Figure 7.3: Implementation of the conjugate gradient descent algorithm for minimizing $f(x,y) = -\begin{pmatrix} -1 & -1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \frac{1}{2}\begin{pmatrix} x & y \end{pmatrix}\begin{pmatrix} 0.1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$. The initial starting point is at the origin as before. It can be seen that the algorithm converges to the minimum at $(-10,-1)$ in two steps. (Note: if the condition number of the Hessian were 1 instead, then the contours of f would have been circles, and the algorithm would have terminated in a single step.)

7.1.3.ii Conjugate Gradient Method for Non-quadratic Functions. For a non-quadratic function, we are faced with the difficulty that, in general, the Hessian changes from point to point. Hence, we cannot impose all search directions to be conjugate w.r.t. a single Hessian matrix. Moreover, even if we try to make consecutive search directions, p_k and p_{k+1}, be mutually conjugate w.r.t. the Hessian at x_{k+1}, recomputing the Hessian at each iteration may be too slow for practical applications. An approximate conjugate gradient method for non-quadratic functions is thus to update the search directions using

$$p_1 = -g_1 \quad \text{and} \quad p_k = -g_k + \beta_{k-1}\, p_{k-1}, \;\; k \geq 2,$$

where $g_k \equiv \nabla f(x_k)$ is the full gradient of f, not just its quadratic approximation at x_{k-1}, and β_{k-1} can be obtained from any of the formulae (7.4), (7.5), (7.6) not requiring the Hessian. These formulae for β_{k-1} are not equivalent for a general non-quadratic function and will thus yield different optimization results and performance. We can also avoid computing the Hessian when calculating the step size α_{*k} by applying a line search algorithm to estimate

$$\alpha_{*k} = \arg\min_{\alpha_k} f(x_k + \alpha_k\, p_k).$$

7.1.4 Newton-Raphson Method

Gradient-based methods rely only on the "rate-of-change" information contained in first order derivatives. It is thus plausible that also incorporating the "curvature" information about a function contained in its second order derivatives might provide both conceptual and algorithmic benefits. This idea motivates the Newton-Raphson method that utilizes the Hessian of a function together with the gradient.

7.1.4.i Newton-Raphson Method in 1D: Finding the Zeros of a Differentiable Function. Suppose you have a 1D function $f : \mathbb{R} \to \mathbb{R}$ satisfying the following criteria:

i f has at least one zero x_* that is isolated (i.e. there exists a neighborhood of x_* in which

x_* is the only zero);

ii $f'(x)$ and $f''(x)$ exist near x_0 (i.e. 2nd order differentiable);

iii $f'(x) \neq 0$ for all x near x_* and $x \neq x_*$ (i.e. no stationary point near x_*, except possible for x_*);

iv $f''(x)$ is continuous near x_*.

Then, the Newton's method provides an iterative method for approximating x_* starting from some initial point x_0 that is sufficiently close to x_*. The convergence is quadratic in successive error $\|x_k - x_*\|_2$, if the above criteria are satisfied. The main ideas are quite simple:

1. At each iteration step with the current approximation x_k of x_*, find a linear approximation of $f(x)$ near x_k via the 1st order Taylor expansion:

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \mathcal{O}(x - x_k)^2. \tag{7.8}$$

2. Find the x-axis crossing x_{k+1} of the linear approximation by setting $f(x) = 0$ in (7.8), ignoring higher order terms:

$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k) \Rightarrow \boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}.$$

3. Repeat Steps 1 and 2 until $\|x_{k+1} - x_k\|_2 < \epsilon$, where ϵ is some small *a priori* chosen error tolerance.

Example 7.5 (Finding $f'(x_*) = 0$). *One important application of the Newton's method in optimization problems is for finding the stationary points of f, i.e. the zeros of the gradient of f. In 1D, Newton's method for this problem amounts to calculating:*

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

7.1.4.ii Newton-Raphson Method for Minimizing a Function in Higher Dimensions. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth function with a minimum at x_*. Many problems in machine learning involve attempting to find such a minimum, which may unfortunately be only a local minimum. Ideally, we want to compute all stationary points of f

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid \nabla f(x) = 0\},$$

which is usually very difficult to accomplish in high dimensions. A more feasible approach is to apply Newton's method to search for stationary points. In this iterative scheme, to update x_{k+1} from x_k, we Taylor expand f around x_k to get

$$f(x) = f(x_k) + (x - x_k)^t g_k + \frac{1}{2}(x - x_k)^t Q_k (x - x_k) + \mathcal{O}(\|x - x_k\|_2^3),$$

where $g_k \equiv \nabla f(x_k)$ is the gradient of f at x_k, and $Q_k \equiv \left(\frac{\partial^2 f}{\partial x_i \partial x_j}\right)\Big|_{x_k}$ is the Hessian matrix of f at x_k. From this expansion, we obtain a linear approximation to the gradient $\nabla f(x)$ near x_k as

$$\nabla f(x) = g_k + Q_k(x - x_k) + \mathcal{O}(\|x - x_k\|_2^2).$$

Ignoring higher order terms and setting the left-hand side to 0, we get

$$Q_k\, x_{k+1} = Q_k\, x_k - g_k.$$

If we assume that Q_k is positive definite, so that Q_k is invertible and x_{k+1} is a local minimum in the approximation, then

$$\boxed{x_{k+1} = x_k - Q_k^{-1} g_k}. \tag{7.9}$$

In practical applications, inverting the Hessian may be too computationally demanding, and we may need to solve $Q_k d_k = -g_k$ for $d_k \equiv x_{k+1} - x_k$ using other indirect approaches such as the conjugate gradient method.

There are several options for deciding when to terminate the iteration. In addition to the ones already discussed above, one could impose

$$\|g_k\|_2 < \epsilon$$

for some small ϵ. Another option is to compare the current value $f(x_k)$ with the minimum of the quadratic approximation of $f(x)$ at x_k:

$$\begin{aligned} f(x_k) - \min_x \left(f(x_k) + (x - x_k)^t g_k + \frac{1}{2}(x - x_k)^t Q_k (x - x_k) \right) &= \frac{1}{2}\, g_k^t\, Q_k^{-1}\, g_k \\ &\equiv \frac{N_k^2}{2}. \end{aligned}$$

The term $N_k \equiv \sqrt{g_k^t\, Q_k^{-1}\, g_k} = \sqrt{\langle d_k, d_k \rangle_{Q_k}} = \|d_k\|_{Q_k}$ is called the Newton decrement and measures the length of each coordinate update w.r.t. the norm defined by the Hessian Q_k.

Newton-Raphson Method Pseudocode

```
Given:  A convex differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
        Tolerance value $\epsilon$ for terminating the algorithm.

function Newton_Raphson($f, \epsilon$):
   Initialize $x_1 \in \mathbb{R}^n$
   $k = 0$
   do {
        $k$++
        $g_k = \nabla f(x_k)$
        $Q_k$ = Hessian matrix of $f$ at $x_k$
        Solve $Q_k d_k = -g_k$ for $d_k$ (e.g.  via the conjugate gradient method)
        $N_k = \sqrt{d_k\, Q_k\, d_k}$
        $x_{k+1} = x_k + d_k$
    } **while** $N_k > \epsilon$
    $x_* = x_{k+1}$
    **return** $x_*$
```

REMARK 7.10 (Advantages)**.** *The convergence is faster than the gradient method if the initialization point is near the true minimum point. For a convex quadratic function, it takes only one iteration to find the global minimum.*

REMARK 7.11 (Disadvantages)**.** *The main idea is simple, but the update in* (7.9) *requires calculating the gradient g_k and the Hessian Q_k^{-1} at each iteration step. Calculating the first and second order derivatives can be computationally costly in high dimensions. Moreover, inverting the Hessian matrix Q_k may be both computationally costly in high dimensions and numerically unstable when the condition number is high (Section A.3), in which case small rounding-off errors intrinsic to digital computers may lead to large errors in approximating the inverse.*

The initial point needs to be sufficiently close to the ideal solution. If the Hessian is not positive definite, then the algorithm may not converge to the solution.

7.1.4.iii Affine Invariance of the Newton-Raphson Method.

The main reason why the Newton-Raphson method is so efficient for a quadratic function is that the method is invariant under affine coordinate transformations and that in some appropriately chosen coordinate system, the Hessian of the cost function f becomes spherically symmetric and the Newton-Raphson step updates to the center of the spherical contours of the quadratic approximation of f in a single step. In fact, if all contours were concentric spheres, then even the gradient descent method with exact line search and the conjugate gradient method would just take a single step to converge to the minimum residing at the center of the contours; however, the gradient-based methods are not invariant under general affine coordinate transformations, and the gradient-based updates in the original coordinate system do not necessarily correspond to gradient-based updates in the desired coordinate system that renders the spherical symmetry manifest.

Let us now discuss more precisely what affine invariance means.

Definition 7.3. *Let $x = (x^1, \ldots, x^n)^t$ represent a column vector in $\mathbb{R}^n$ in some fixed basis. An affine transform y of x is defined as*

$$y = Mx + a,$$

where $M \in \mathbb{R}^{n\times n}$ is a constant invertible matrix and $a \in \mathbb{R}^n$.

In the following, we will use the notation $\partial_{x^i} \equiv \frac{\partial}{\partial x^i} \equiv (\nabla_x)_i$. We will apply the next lemma to show that the Newton-Raphson method is invariant under affine transformations:

Lemma 7.1. *Under the affine transformation $y = Mx + a$, the gradient ∇_x and the Hessian $Q_x \equiv (\partial_{x^i}\partial_{x^j})$ operators transform as*

$$\nabla_y = (M^t)^{-1}\nabla_x \quad \textit{and} \quad Q_y = (M^t)^{-1}Q_x M^{-1}.$$

Proof. Under the affine transformation, applying the chain rule yields

$$(\nabla_x)_i = \sum_{j=1}^{n} \frac{\partial y^j}{\partial x^i} (\nabla_y)_j = \sum_{j=1}^{n} M_i^j (\nabla_y)_j \Rightarrow \nabla_x = M^t \nabla_y \Rightarrow \nabla_y = (M^t)^{-1} \nabla_x.$$

Similarly, we have

$$(\nabla_x)_i (\nabla_x)_j = \sum_{k=1}^{n} \sum_{\ell=1}^{n} M_i^k M_j^\ell (\nabla_y)_k (\nabla_y)_\ell \Rightarrow Q_x = M^t Q_y M \Rightarrow Q_y = (M^t)^{-1} Q_x M^{-1}.$$

□

We can now prove:

Theorem 7.9. *The Newton-Raphson update $x_{k+1} = x_k - Q_{x,k}^{-1} g_{x,k}$ is invariant under the affine transformation $y = Mx + a$.*

Proof. Acting on the left of the update rule $x_{k+1} = x_k - Q_{x,k}^{-1} g_{x,k}$ with M and adding a to both sides of the equation, we get

$$y_{k+1} = Mx_{k+1} + a = Mx_k + a - MQ_{x,k}^{-1} g_{x,k} = y_k - MQ_{x,k}^{-1} M^t (M^t)^{-1} g_{x,k} = y_k - Q_{y,k}^{-1} g_{y,k},$$

where the last equality follows from Lemma 7.1. □

Hence, if v_i and λ_i, $i = 1, \ldots, n$, are the orthonormal eigenvectors and corresponding eigenvalues of Q_x, then choosing

$$M^{-1} = \begin{pmatrix} \frac{1}{\sqrt{\lambda_1}} v_1 & \cdots & \frac{1}{\sqrt{\lambda_n}} v_n \end{pmatrix}$$

will transform Q_x into $Q_y = I_{n \times n}$, and the Newton-Raphson step in the transformed coordinate system amounts to a single-step gradient descent to the center of the spherical contours of the quadratic approximation (Figure 7.4):

$$y_2 = y_1 - g_{y,1}.$$

REMARK 7.12. *Lemma 7.1 also shows that the Newton decrement $N_k \equiv \sqrt{g_k^t Q_k^{-1} g_k}$ is invariant under affine transformations, suggesting that it may provide a useful coordinate-invariant convergence test.*

7.1.4.iv Levenberg-Marquardt Modification of the Newton-Raphson Method.

The above formulation of the Newton-Raphson method assumes that the Hessian matrix Q_k is positive definite at each x_k. This assumption guarantees that

$$g_k^t d_k = -g_k^t Q_k^{-1} g_k < 0;$$

that is, the update direction d_k is a descent direction. In real-world problems, this assumption may be too strong; and, the Newton-Raphson method may leads to updates not in a descent

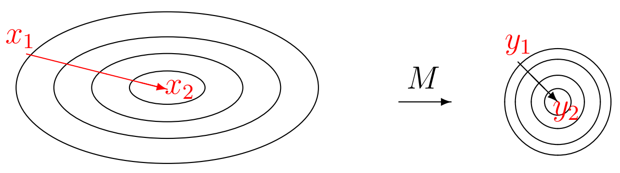

Figure 7.4: Effect of "whitening" the Hessian matrix. In the transformed coordinate system, the Newton-Raphson step is just a single-step gradient descent update orthogonal to the circular contours.

direction. The Levenberg-Marquardt algorithm thus modifies the Newton-Raphson method by adding positive diagonal entries to Q_k, thereby forcing the resulting matrix $\tilde{Q}_k$ to become positive definite:

$$x_{k+1} = x_k - \tilde{Q}_k^{-1} g_k \,, \;\; \tilde{Q}_k = Q_k + \mu_k I$$

where $\mu_k \geq 0$ is tuned to a value as small as possible to make $\tilde{Q}_k$ positive definite. The directional derivative of f along the new direction $\tilde{d}_k \equiv -\tilde{Q}_k^{-1} g_k$ now satisfies

$$g_k^t \, \tilde{d}_k = -g_k^t \, \tilde{Q}_k^{-1} g_k < 0;$$

and is thus a descent direction.

7.1.4.v Nonlinear Least Square Optimization: Gauss-Newton Algorithm. In Section 3.5, we formulated (mean-centered) linear regression as a linear least square optimization problem of minimizing

$$L(\beta) = \sum_{i=1}^{m} (y_i - \beta_1 x_{i1} - \cdots - \beta_n x_{in})^2 \tag{7.10}$$

over $\beta_1, \ldots, \beta_n$. An important variation of this problem is when each penalty term is some general residual function $r_i(\beta)$, which may not be linear in $\beta \equiv (\beta_1, \ldots, \beta_n)^t$:

$$\hat{\beta} = \arg\min_{\beta} L(\beta) \,, \quad \text{where } L(\beta) = \sum_{i=1}^{m} r_i(\beta)^2.$$

To apply the Newton-Raphson method to this problem, we need to compute the gradient and Hessian of L:

$$\nabla_\beta L = 2\sum_{i=1}^{m} r_i\,\nabla_\beta r_i,$$

and

$$Q_\beta(L) = 2\sum_{i=1}^{m}(\nabla_\beta r_i)\,(\nabla_\beta r_i)^t + 2\sum_{i=1}^{m} r_i\,Q_\beta(r_i),$$

where Q_β denotes the matrix of second order partial derivatives. To simplify the notation, define

$$\boldsymbol{r} = \begin{pmatrix} r_1 \\ r_2 \\ \cdots \\ r_m \end{pmatrix} \quad \text{and} \quad \boldsymbol{J} = \begin{pmatrix} \frac{\partial r_1}{\partial \beta_1} & \cdots & \frac{\partial r_1}{\partial \beta_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial r_m}{\partial \beta_1} & \cdots & \frac{\partial r_m}{\partial \beta_n} \end{pmatrix}$$

and rewrite

$$\nabla_\beta L = 2\,\boldsymbol{J}^t\boldsymbol{r} \quad \text{and} \quad Q_\beta(L) = 2\,\boldsymbol{J}^t\boldsymbol{J} + 2\sum_{i=1}^{m} r_i\,Q_\beta(r_i).$$

In the above product-rule expansion of $Q_\beta(L)$, the Gauss-Newton algorithm assumes that the term multiplying the residual by second order derivatives is much smaller than the first order derivatives term containing the Jacobian $\boldsymbol{J}$, especially near the critical point. That is, we will approximate

$$\boxed{Q_\beta(L) \approx 2\,\boldsymbol{J}^t\boldsymbol{J}}.$$

Hence, the Gauss-Newton update is

$$\boxed{\beta_{k+1} = \beta_k - (2\,\boldsymbol{J}^t\boldsymbol{J})^{-1}(2\,\boldsymbol{J}^t\boldsymbol{r}) = \beta_k - (\boldsymbol{J}^t\boldsymbol{J})^{-1}(\boldsymbol{J}^t\boldsymbol{r})}. \tag{7.11}$$

Example 7.6 (Nonlinear Curve Fitting)**.** *Suppose you have noisy data points (x_i, y_i), $i = 1, \ldots, m$. For each value of x, we model y as being sampled from the conditional distribution $y|_x = f(x|\beta) + \epsilon$, where $f(x|\beta)$ is some function parameterized by $\beta \in \mathbb{R}^n$ and ϵ is modeled by some known noise distribution. Then, the residual of the model for the i-th data pair is*

$$r_i(\beta) = f(x_i|\beta) - y_i,$$

and the cost function to be minimized over β is

$$L(\beta) = \sum_{i=1}^{m} r_i(\beta)^2.$$

Figure 7.5 shows the result of running the Gauss-Newton algorithm for $f(x|A,\omega,\phi) = A\,\sin(\omega x + \phi)$, where the data points were simulated using the values $(A,\omega,\phi) = (2, 3, \pi/4)$ and a uniform random variable ϵ in $[-0.5, 0.5]$.

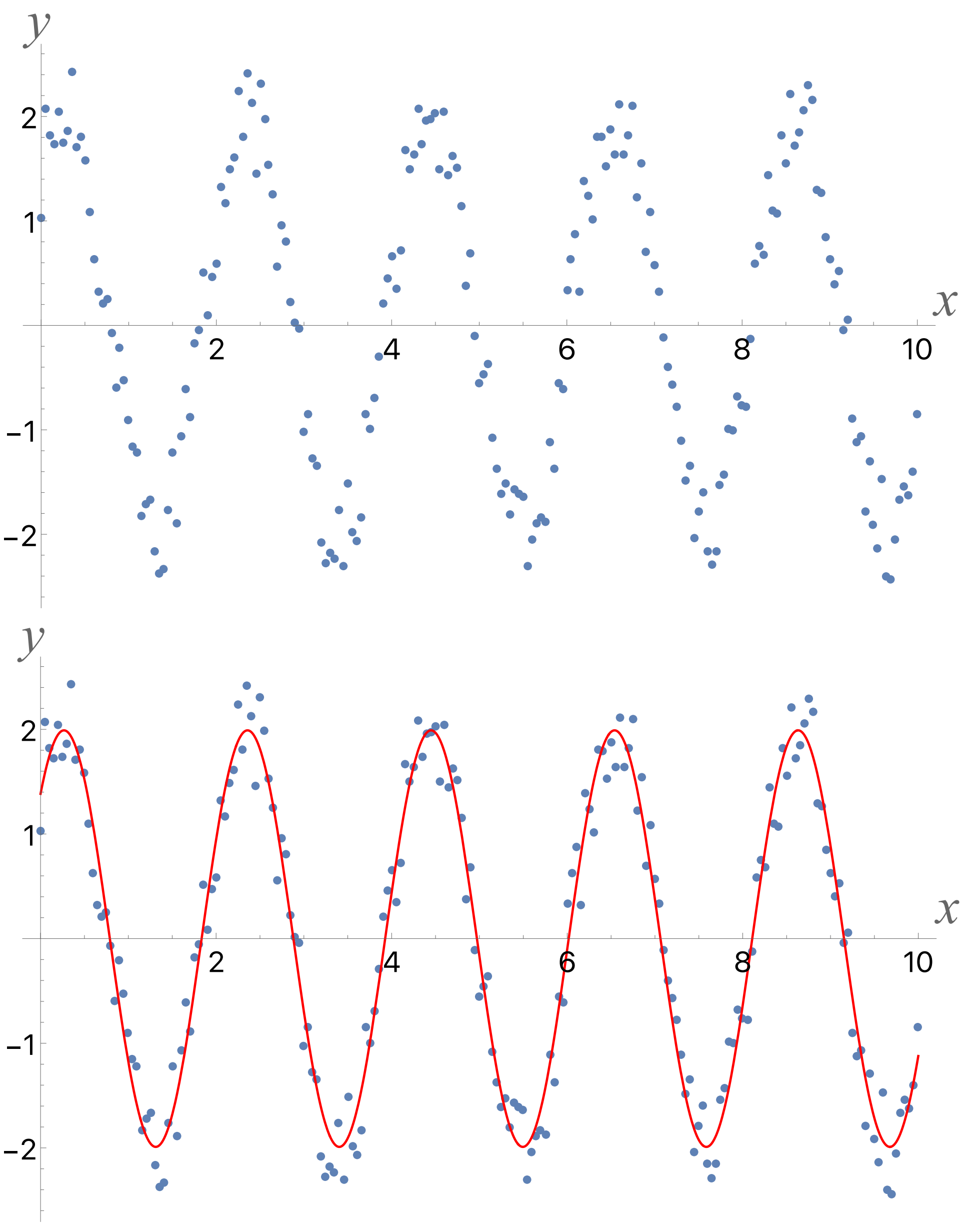

Figure 7.5: Let $y = A\sin(\omega x + \phi) + \epsilon$, where ϵ is a random error uniformly distributed in $[-0.5, 0.5]$. (Top) 201 samples of y for $(A, \omega, \phi) = (2, 3, \pi/4)$ at x evenly spaced between 0 and 10 in steps of 0.05. (Bottom) Red curve shows the result of the Gauss-Newton algorithm initialized at $(A_1, \omega_1, \phi_1) = (1.6, 3.2, \pi/3)$. The algorithm converged to $(2.0, 3.0, 0.77)$ in six iterations.

REMARK 7.13 (Levenberg-Marquardt Algorithm). *The modification*

$$\boldsymbol{J}^t\boldsymbol{J} \to \boldsymbol{J}^t\boldsymbol{J} + \mu_k I.$$

eliminates potential zero eigenvalues of the Gram matrix and represents a compromise between gradient descent and Gauss-Newton methods.

7.2 Cross Entropy Method

7.2.1 Kullback-Leibler Divergence (Relative Entropy)

Definition 7.4 (Kullback-Leibler Divergence). *Let $P_1(x)$ and $P_2(x)$ be probability functions defined on a common set $\mathcal{X}$, such that $\forall x \in \mathcal{X}$, $P_2(x) = 0 \Rightarrow P_1(x) = 0$. The Kullback-Leibler divergence $D(P_1||P_2)$ between P_1 and P_2 is*

$$D(P_1||P_2) = \sum_{x\in\mathcal{X}} P_1(x)\log_2 \frac{P_1(x)}{P_2(x)}.$$

REMARK 7.14. *In this notation, we have assumed that $\mathcal{X}$ is a discrete set. Upon replacing the sum with an integral and replacing the probability functions with their densities, this definition and the ensuing discussion also hold true for a continuous probability space $\mathcal{X}$.*

Note that in general, $D(P_1||P_2) \neq D(P_2||P_1)$, so Kullback-Leibler divergence is not a distance between P_1 and P_2. However, it does roughly capture how different P_1 and P_2 are. In particular,

Theorem 7.10. *$D(P_1||P_2) \geq 0$, where the equality holds if and only if $P_1 = P_2$.*

Proof. Since $-\log$ is a strictly convex function, we have

$$D(P_1||P_2) = -\sum_{x\in\mathcal{X}} P_1(x)\log_2\frac{P_2(x)}{P_1(x)} \geq -\log_2\left(\sum_{x\in\mathcal{X}} P_1(x)\frac{P_2(x)}{P_1(x)}\right) = 0.$$

The equality holds if and only if $P_2(x)/P_1(x)$ is constant, i.e. $P_2(x) = cP_1(x)$; summing both sides over x yields $1 = c$. Hence, the equality holds if and only if $P_1(x) = P_2(x)$ for all x. □

Example 7.7. *Let X be a discrete random variable taking n possible values. Suppose P_2 is a uniform probability function, i.e. $P_2(x) = 1/n$ for all values x of X. If P_1 is any probability function of X, then*

$$D(P_1||P_2) = -H(X) + \log_2 n \geq 0 \ \Rightarrow \log_2 n \geq H(X),$$

where

$$H(X) = -\sum_{x\in\mathcal{X}} P(x)\log_2 P(x)$$

is the Shannon entropy of X. Thus, the entropy $H(X)$ of X is bounded above by the entropy of a uniformly distributed random variable, and the equality is saturated if and only if X itself is uniformly distributed.

One disadvantage of the Kullback-Leibler divergence $D(P_2||P_1)$ is that it is not symmetric in P_1 and P_2. We can formulate a symmetric measure of difference between probability distributions as follows: define a mixture probability function P as $P(x) = \frac{1}{2}P_1(x) + \frac{1}{2}P_2(x)$. The intuition behind this mixture probability is that you flip a fair coin, so that if you get heads, then you sample from P_1, and if you get tails, then you sample from P_2. In other words, we introduce a latent random variable $Z \in \{1, 2\}$, such that $P(x|z = 1) = P_1(x)$ and $P(x|z = 2) = P_2(x)$; the marginal distribution of X is then

$$P(x) = P(x|z = 1)P(z = 1) + P(x|z = 2)P(z = 2) = \frac{1}{2}P_1(x) + \frac{1}{2}P_2(x).$$

Definition 7.5 (Jensen-Shannon Divergence)**.** *The Jensen-Shannon Divergence between two probability functions P_1 and P_2 is*

$$J(P_1, P_2) = \frac{1}{2}D(P_1||P) + \frac{1}{2}D(P_2||P)$$

where $P(x) = \frac{1}{2}P_1(x) + \frac{1}{2}P_2(x)$.

N.B. One can show that $0 \leq J(P_1, P_2) \leq 1$. In particular, $J(P_1, P_2) = 0$ iff $P_1 = P_2$; and, the more divergent P_1 and P_2 are, the larger the value $J(P_1, P_2)$.

7.2.2 Cross Entropy (CE)

Definition 7.6 (Cross Entropy)**.** *The cross entropy $H_c(P, Q)$ of probability functions P and Q defined on a common set $\mathcal{X}$ is defined as*

$$H_c(P, Q) := -\sum_{x\in\mathcal{X}} P(x)\log_2 Q(x) = \sum_{x\in\mathcal{X}} P(x)\log_2 \frac{1}{Q(x)} = H(P) + D(P||Q).$$

REMARK 7.15. *In terms of coding theory, $H_c(P, Q)$ is the average number of bits required to represent the random variable X that is distributed as P but encoded as Q. Because $D(P||Q) \geq 0$ for any Q, we see that*

$$\min_Q H_c(P, Q) = H(P).$$

Hence, the smallest average number of bits required to encode X is $H(P)$.

REMARK 7.16. *Suppose Q is a member of a fixed parametric family of probability distributions. Given P, finding Q that minimizes $D(P||Q)$ is equivalent to finding Q that minimizes the cross entropy $H_c(P, Q)$.*

7.2.3 Rare Event Sampling and Motivation for the Cross-Entropy Method

Let us revisit the concept of importance sampling described in Section 6.4. For z a large positive number, we wanted to reduce the variance of the naive MC estimate of

$$f(z) = \int_z^\infty \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\, dx = \int_{-\infty}^\infty I(x \geq z)p(x)dx,$$

where $p(x)$ is the standard normal density. The main idea behind importance sampling is to sample from an alternate distribution $\pi(x)$ and weight each sample by the importance weight $p(x)/\pi(x)$. We have seen in lecture that the minimum-variance solution is

$$\pi(x) = \frac{I(x \geq z)p(x)}{C}$$

where $C = f(z)$ is the very integral that we want to evaluate. So, this ideal importance distribution does not directly help us. However, suppose we would like to consider an exponential family $q(x; \lambda) = \lambda e^{-\lambda(x-z)}$. In Section 6.4, we simply used

$$\lambda^* = \frac{z + \sqrt{z^2 + 4}}{2}$$

from (6.2) that gave the optimal rejection sampling. Here, let us try tuning λ such that the cross entropy $H_c(\pi, q)$ is minimized. That is, we need to solve

$$\arg\min_q \left(-\int_{-\infty}^{\infty} \pi(x) \log_2 q(x; \lambda)\, dx \right) = \arg\max_q \int_{-\infty}^{\infty} I(x \geq z)p(x) \log q(x; \lambda)\, dx.$$

Finding a critical point of the integral with respect to λ yields

$$\int_{-\infty}^{\infty} I(x \geq z)p(x) \left(\frac{1}{\lambda} - (x - z) \right) dx = 0.$$

Solving for λ, we get

$$\lambda_c = \frac{\int_{-\infty}^{\infty} I(x \geq z)p(x)\, dx}{\int_{-\infty}^{\infty} xI(x \geq z)p(x)\, dx - z \int_{-\infty}^{\infty} I(x \geq z)p(x)\, dx} \tag{7.12}$$

$$= \frac{\frac{1}{2} \operatorname{erfc}\left(\frac{z}{\sqrt{2}}\right)}{\frac{1}{\sqrt{2\pi}} e^{-z^2/2} - \frac{z}{2} \operatorname{erfc}\left(\frac{z}{\sqrt{2}}\right)}, \tag{7.13}$$

where $\operatorname{erfc}(z) = 1 - \operatorname{erf}(z)$ is the complementary error function. To evaluate λ_c, we need to evaluate the error function, so it seems we are back to the starting point; but, that's okay for two reasons:

1. we are just trying to see whether this variance-minimizing choice for the exponential family will actually reduce the estimation variance, and
2. we will learn how to approximate λ_c using an iterative sampling scheme shortly.

Figure 7.6 shows the comparison between importance sampling using λ^* and λ_c. We can indeed see that the importance sampling estimates using λ_c has a slightly smaller variance than λ^*.

REMARK 7.17. *Numerical calculation shows that λ^* and λ_c are actually very close to each other (Figure 7.7). This finding is not very surprising, because λ^* has been already optimized in a different way to resemble the standard normal tail distribution.*

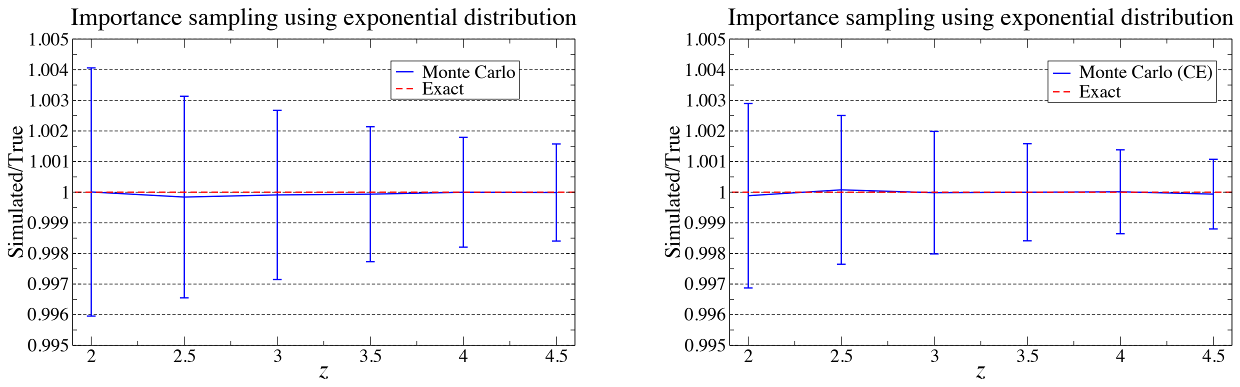

Figure 7.6: Monte Carlo estimation of the tail probability $\int_z^\infty \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\,dx$ by sampling from a shifted exponential distribution with $\lambda^* = (z + \sqrt{z^2+4})/2$ (left) and the cross-entropy solution λ_c (right).

Let us now try to approximate the solution λ_c that minimizes the CE among the exponential functions. The main idea is to approximate (7.13) by performing another importance sampling MC approximation of (7.12). That is, set an initial rate λ_0, say $\lambda_0 = 1$. Then, perform the following importance sampling using $q(x;\lambda_0)$:

$$\int_{-\infty}^{\infty} I(x \geq z) p(x)\,dx \approx \frac{1}{N}\sum_{i=1}^{N} I(x_i \geq z)\frac{p(x_i)}{q(x_i;\lambda_0)} = \frac{1}{N}\sum_{i=1}^{N}\frac{p(x_i)}{q(x_i;\lambda_0)}$$

and

$$\int_{-\infty}^{\infty} (x-z) I(x \geq z) p(x)\,dx \approx \frac{1}{N}\sum_{i=1}^{N} (x_i - z) I(x_i \geq z)\frac{p(x_i)}{q(x_i;\lambda_0)} = \frac{1}{N}\sum_{i=1}^{N}(x_i - z)\frac{p(x_i)}{q(x_i;\lambda_0)},$$

where x_i are samples from $q(x;\lambda_0)$ and thus always satisfy $x_i \geq z$. We use these values to estimate λ_1. Iterating this scheme, we need to keep updating

$$\lambda_{t+1} = \frac{\sum_{i=1}^{N} \frac{p(x_i)}{q(x_i;\lambda_t)}}{\sum_{i=1}^{N} \frac{p(x_i)}{q(x_i;\lambda_t)}(x_i - z)}.$$

Figure 7.7 shows the result of this iterative algorithm for $t_{\max} = 100$ and $N = 1000$.

REMARK 7.18. *Recall that the maximum likelihood estimate of the parameter $1/\lambda = E[X]$ for $X \sim Exp(\lambda)$ is just the sample average of the random variable. Similarly, $1/\lambda_{t+1}$ is the average of the exponentially distributed $x_i - z$ weighted by the importance weight $p(x_i)/q(x_i;\lambda_t)$.*

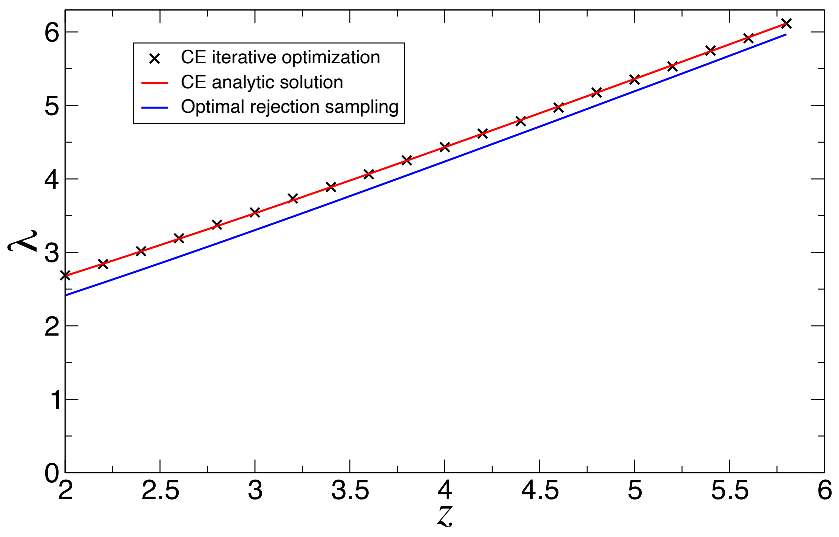

Figure 7.7: Comparison of parameters for the exponential importance sampling distribution.

7.2.4 Cross-Entropy Method for Rare Event Simulation

Here is the general problem that we would like to solve: Given a random vector X distributed according to density $p(x)$ and some function $S(X)$, compute

$$P(S(X) \geq \gamma) = E_p[I(S(X) \geq \gamma)],$$

where the subscript p indicates taking expectation with respect to the density p and the event $\{X|S(X) \geq \gamma\}$ is assumed to be a rare event, say with probability $< 10^{-5}$. The optimal distribution for importance sampling is

$$q^*(x) \propto I(S(X) \geq \gamma)p(x).$$

We would like to estimate this distribution using a parametric family $q_\theta(x)$ by minimizing the cross entropy

$$H_c(q^*, q_\theta) \propto -E_p[I(S(X) \geq \gamma) \log q_\theta(X)] = -E_{q_{\theta_0}}[I(S(X) \geq \gamma)W(X, \theta_0) \log q_\theta(X)] \quad (7.14)$$

where $W(X, \theta_0)$ is the importance weight $p(X)/q_{\theta_0}(X)$ for the distribution with a guessed value θ_0. We need to iteratively update this value until our MC estimate of the expectation (7.14) becomes good. However, we already know that for large γ, our initial guess of θ_0 may not yield enough samples in the event $\{X|S(X) \geq \gamma\}$; we address this situation by decreasing the cutoff γ to $\gamma_1 < \gamma$ and iterate between optimizing θ_t and increasing γ_t. Here is the pseudo-code:

```
Cross-Entropy Method (CEM) for Rare Event Sampling (RES) Pseudocode
```

```
Given: distribution $p$, function $S$, threshold $\gamma$, parametric family $q_\theta$.
Want:  Estimate $E_p[I(S(X) \geq \gamma)]$ using $E_{q_\theta}[I(S(X) \geq \gamma)W(X,\theta)]$,
       where $W(X,\theta) = p(X)/q_\theta(X)$.
Problem:  Find optimal $\theta$ that minimizes the CE $-E_p[I(S(X) \geq \gamma) \log q_\theta(X)]$.

function CEM_RES($p$, $S$, $\gamma$, $q_\theta$, nSample $N$, quantile $\rho$, stopping $\epsilon$):
    Initialize $\theta_0$
    Set $t = 1$
    While convergence not reached :
        Sample $X_1, \ldots, X_N$ from $q_{\theta_{t-1}}$
        Set $\gamma_t = (1-\rho)$-quantile of $\{S(X_1), \ldots, S(X_N)\}$
        If $\gamma_t > \gamma$, set $\gamma_t = \gamma$
        Solve $\theta_t = \arg\max_\theta \frac{1}{N} \sum_{i=1}^{N} I(S(X_i) \geq \gamma_t) W(X_i; \theta_{t-1}) \log q_\theta(X_i)$
        If $\gamma_t < \gamma$ or $\|\theta_t - \theta_{t-1}\| > \epsilon$ :   $t$++
        else:  break out of the while loop
    Sample $X_1, \ldots, X_N$ from $q_{\theta_t}$
    Compute $\hat{E}_{q_{\theta_t}}[I(S(X) \geq \gamma)W(X,\theta_t)] = \frac{1}{N} \sum_{i=1}^{N} I(S(X_i) \geq \gamma) W(X_i, \theta_t)$
    Return $\hat{E}_{q_{\theta_t}}[I(S(X) \geq \gamma)W(X,\theta_t)]$
```

REMARK 7.19. *The number of samples N should be chosen such that θ_t can be robustly estimated. Experience shows that choosing $N \propto n/\rho$, where n is the number of parameters in θ, yields good results.*

REMARK 7.20. *ρ is typically chosen to be between 0.01 and 0.1.*

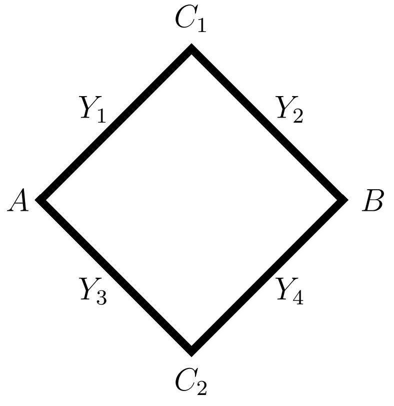

Figure 7.8: Distribution of shortest path from A to B. The Y_i's are exponentially distributed random variables with mean λ_i^{-1} and are independent of each other. $S(Y_1, Y_2, Y_3, Y_4) = \min(Y_1 + Y_2, Y_3 + Y_4)$. Find $P(S(Y_1, Y_2, Y_3, Y_4) \geq \gamma)$ for some large value $\gamma \gg \max(\lambda_1^{-1} + \lambda_2^{-1}, \lambda_3^{-1} + \lambda_4^{-1})$.

Example 7.8. *Suppose you are a passenger at an airport. You need to go from entrance A to gate B by going through either security check-point C_1 or check-point C_2 (Figure 7.8). Suppose the waiting/travel time distributions for Y_1, Y_2, Y_3, Y_4 are independent exponential distributions with mean $\lambda_1^{-1}, \lambda_2^{-1}, \lambda_3^{-1}, \lambda_4^{-1}$, respectively. Let $X = (Y_1, Y_2, Y_3, Y_4)$. We are interested in estimating the probability that $S(X) = \min(Y_1 + Y_2, Y_3 + Y_4) \geq \gamma$, for some fixed $\gamma \gg \max(\lambda_1^{-1} + \lambda_2^{-1}, \lambda_3^{-1} + \lambda_4^{-1})$.*

Choose the importance sampling distribution with $\theta = (\eta_1, \eta_2, \eta_3, \eta_4)$ as

$$q_\theta(y_1, y_2, y_3, y_4) = \prod_{j=1}^{4} \eta_j e^{-\eta_j y_j}.$$

Then, at each iteration step, we need to solve

$$\theta_t = \arg\max_\theta \frac{1}{N}\left[\sum_{i=1}^{N} I(S(X_i) \geq \gamma_t) W(X_i; \theta_{t-1}) \sum_{j=1}^{4} (\log \eta_j - \eta_j Y_{ij})\right].$$

Taking derivative with respect to η_i and setting equal to 0, we get $\theta_t = (\eta_1, \ldots, \eta_4)$, where

$$\frac{1}{\eta_j} = \frac{\sum_{i=1}^{N} I(S(X_i) \geq \gamma_t) W(X_i; \theta_{t-1}) Y_{ij}}{\sum_{i=1}^{N} I(S(X_i) \geq \gamma_t) W(X_i; \theta_{t-1}),}$$

which is similar to the usual MLE estimate of an exponential parameter, but scaled by the importance weight and restricted to only those extreme samples passing the criterion $S(X_i) \geq \gamma_t$.

7.2.5 Cross-Entropy Method for Combinatorial Optimization

Let Ω denote a discrete configuration space and $S : \Omega \to \mathbb{R}$ a reward function; equivalently, we can think of $-S$ as a cost function. We would like to solve the following optimization problem

$$X^* = \arg\max_{X \in \Omega} S(X) = \arg\min_{X \in \Omega} [-S(X)],$$

assuming that a unique solution exists. As it stands, there is no probabilistic component to the problem. However, suppose we associate the following stochastic problem to the optimization. Let us define a parametric family of sample-generating distributions $P_\theta(X)$ on Ω. The cross-entropy method can then be used to tune θ to θ^* such that the distribution $P_{\theta^*}(X)$ is sharply centered around the optimal solution X^*. As before, we will guess an initial value θ_0 and iteratively update θ. Because we do not *a priori* know where the optimal solution X^* lies, we need to choose θ_0 such that any state $X \in \Omega$ has non-zero probability of being sampled. Then, the associated stochastic problem is to estimate

$$P_{\theta_{t-1}}(S(X) \geq \gamma_t) = E_{\theta_{t-1}}[I(S(X) \geq \gamma_t)]$$

via importance sampling

$$E_{\theta_t}\left[I(S(X)\geq\gamma_t)\frac{P_{\theta_{t-1}}(X)}{P_{\theta_t}(X)}\right],$$

where γ_t is the $(1-\rho)$-quantile of the empirical samples from $P_{\theta_{t-1}}$ and θ_t is chosen to minimize the cross entropy $H_c(I(S(X)\geq\gamma_t)P_{\theta_{t-1}}(X), P_\theta(X))$. Hence, at each iteration, P_θ is optimized to put much of its probability mass to regions in Ω where $S(X)$ takes extreme values. In many concrete applications, P_{θ_t} will converge towards a delta function distribution with probability 1 at X^*.

To minimize the cross entropy $H_c(I(S(X)\geq\gamma_t)P_{\theta_{t-1}}(X), P_\theta(X))$, we first obtain a Monte Carlo estimate

$$\begin{aligned}H_c(I(S(X)\geq\gamma_t)P_{\theta_{t-1}}(X), P_\theta(X)) &= -E_{P_{\theta_{t-1}}}[I(S(X)\geq\gamma_t)\log_2 P_\theta(X)]\\ &\approx -\frac{1}{N}\sum_{i=1}^{N} I(S(X_i)\geq\gamma_t)\log_2 P_\theta(X_i),\end{aligned}$$

where X_i are sampled from $P_{\theta_{t-1}}$. Then, differentiate this equation with respect to θ to obtain θ_t that solves

$$\frac{\partial}{\partial\theta}\left[\sum_{i=1}^{N} I(S(X_i)\geq\gamma_t)\log P_\theta(X_i)\right]\Bigg|_{\theta=\theta_t} = 0,$$

where we have dropped all irrelevant constants.

REMARK 7.21. *In contrast to the rare event sampling case, there is no importance factor here, because our target distribution changes at every iteration, and our previous importance sampling distribution for estimating the cross-entropy is the target distribution.*

```
        Cross-Entropy Method for Combinatorial Optimization Pseudocode

Given: configuration space $\Omega$, function $S$, parametric family $P_\theta$.
Problem:  Estimate $\arg\max_{X\in\Omega} S(X)$.

function CE_CO($\Omega$, $S$, $P_\theta$, nSample $N$, quantile $\rho$, interp $\alpha$, convergence $d$):
    Initialize $\theta_0$
    Set $t = 1$
    While convergence not reached :
        Sample $X_1,\ldots,X_N$ from $P_{\theta_{t-1}}$
        Set $\gamma_t = (1-\rho)$-quantile of $\{S(X_1),\ldots,S(X_N)\}$
        Solve $\eta_t = \arg\max_\theta \sum_{i=1}^N I(S(X)\geq\gamma_t)\log P_\theta(X_i)$
        Set $\theta_t = \alpha\,\eta_t + (1-\alpha)\theta_{t-1}$
        If $\theta_t = \theta_{t-1} = \cdots = \theta_{t-d}$:  break out of while loop
        else:  $t$++
    Set $X^* =$ most probable state according to $P_{\theta_t}$
    Return $X^*$ and $S(X^*)$
```

Example 7.9. *Let Y be an unknown binary string of length L. Even though the precise nature of this string is not known, suppose there is a black box telling us the value*

$$S(X) = L - |Y \veebar X|$$

where $\veebar$ indicates XOR operation and $|\ |$ is counting the number of bits flipped to 1. That is, $S(X)$ is telling us in how bit locations X agrees with Y. For $L = 100$, the configuration space Ω has roughly 10^{30} elements, so a brute force enumeration approach will not work. To infer the nature of this string Y via the cross-entropy method, generate $X = (Z_1, \ldots, Z_L)$ as L independent Bernoulli random variables with success probability $P(Z_j = 1) = p_j$. Thus, $\theta = (p_1, \ldots, p_L)$, and

$$P_\theta(X) = \prod_{j=1}^{L} p_j^{Z_j}(1-p_j)^{1-Z_j}.$$

Minimizing the cross entropy yields

$$(p_t)_j = \frac{\sum_{i=1}^{N} I(S(X_i) \geq \gamma_t) Z_{ij}}{\sum_{i=1}^{N} I(S(X_i) \geq \gamma_t)}$$

which is just MLE of p_j using extreme samples satisfying $S(X_i) \geq \gamma_t$. The reason behind using the interpolation parameter α in updating $\theta_t = \alpha\,\eta_t + (1-\alpha)\theta_{t-1}$ is that some p_j may get quickly set to either 0 or 1 before the configuration space is adequately explored, thereby always yielding the same sample values for Z_j and trapping the exploration in a sub-optimal region.

REMARK 7.22. *Find a deterministic algorithm that requires only $L+1$ steps to solve the above example.*

7.3 Simulated Annealing

We have previously discussed that MCMC can get trapped in local maxima of the target distribution π_s and exhibit poor mixing. Simulated annealing is based on the idea that at high temperature all microstates in thermal equilibrium become almost equiprobable; i.e. differences in probability landscape on state space become negligible at very high temperature T. Specifically, if we define a new target distribution π_s^β / C, where $\beta = 1/T$ and C is a normalizing constant, then at $\beta \ll 1$, we have $\pi_s^\beta(x) \approx \pi_s^\beta(y)$. Thus, the acceptance ratio become

$$\frac{(\pi_s^\beta)_y Q_{yx}}{(\pi_s^\beta)_x Q_{xy}} \approx \frac{Q_{yx}}{Q_{xy}} \quad \text{for } \beta \ll 1.$$

In particular, if we choose a symmetric proposal distribution, then the acceptance ratio will be roughly 1 for any proposed move, allowing us to explore the entire state space efficiently at high temperature. Hence, we can iterate between taking N MCMC steps at a fixed β and gradually raising β towards 1. Empirical evidence shows that if we raise β sufficiently slowly and take sufficiently large N steps at each β, then this iteration process ultimately leads to

MCMC samples lying in high probability regions when β has reached the value 1. In this way, we can obtain good initialization states to start the final MCMC simulations at $\beta = 1$.

REMARK 7.23. *Note that every time we change β, MCMC has a different steady state distribution. Hence, even if the MCMC at a fixed β has converged to equilibrium, updating β will take the system out of equilibrium. This disadvantage can be addressed by simulating multiple MCMC chains at several different temperature and allowing random swaps between samples from different chains while preserving the detailed balance condition. This approach is called parallel tempering, which we will discuss in the next section.*

A more typical application of simulated annealing is to optimize a multimodal function $f(x)$ defined on a state space S. Without loss of generality, let us assume that we wish to find the global minimum of $f(x)$ and that this minimum is unique.

REMARK 7.24. *To maximize $f(x)$, simply replace $f(x)$ with $-f(x)$ in the discussion below.*

Define the 1-parameter family of probability distributions

$$p_\beta(x) = \frac{\exp(-\beta f(x))}{Z_\beta},$$

where Z_β is a normalizing constant, a.k.a. the partition function, which we assume exists. As before, we think of β as the inverse temperature, so $0 < \beta < \infty$. At each fixed β, if we construct an MCMC sampler with p_β as the target distribution and use a symmetric proposal distribution, then the acceptance probability for a move from x to y is

$$r_\beta = \min\{1, \exp(-\beta(f(y) - f(x)))\}.$$

At small β, we have $r_\beta \approx 1$, and the MCMC states will explore the entire state space efficiently. At finite β, any move that decreases f will still be accepted with probability 1, but any move that increases f will be accepted with probability

$$\exp(-\beta \Delta f(x))$$

which can be a small number if $\beta \gg 1/\Delta f(x)$. Hence, at high β, most hill-climbing moves will be rejected. As described above, simulated annealing gradually raises β from approximately 0 to a very high value, and at each value of β, we perform N MCMC moves. In several practical applications, experience shows that by the time β has reached $\beta_{\max} \gg 1$, the MCMC states will have found the global mininum-containing valley and will descend to the global minimum. Theoretically, however, only if β is raised infinitesimally slowly and the number N of MCMC samples is infinite at each β, it is guaranteed that simulated annealing will find the global minimum of f.

REMARK 7.25. *A typical cooling schedule for $\beta_0 < \beta_1 \ldots < \beta_n = \beta_{\max}$ is*

$$\beta_t = \beta_0 \left(\frac{\beta_{\max}}{\beta_0}\right)^{\frac{t}{n}}.$$

7.4 Metropolis Coupled MCMC (Parallel Tempering)

Similar to simulated annealing, parallel tempering flattens the target distribution in order to facilitate transitions between modes. Also like simulated annealing, parallel tempering admits simulations at both high and low temperature. There are, however, two key differences:

1. Unlike simulated annealing, parallel tempering simultaneously performs several Markov chains at different temperature and combines them into a single master Markov chain.
2. Unlike simulated annealing, the steady state is not taken out of equilibrium upon changing temperature.

Let us now examine these two key differences in detail. As before, denote our state space by S and the desired target distribution as π. Flattening this target distribution by raising it to power $0 < \beta < 1$ yields a tempered density. Denote these tempered densities by $\pi_k(y) \propto \pi(y)^{\beta_k}$, for $k = 1, \ldots, N$, where $0 < \beta_1 < \beta_2 < \ldots < \beta_N = 1$. As before, for each tempered density π_k, we can construct an MCMC on S with steady state distribution π_k. We would like to combine these N chains into a single chain so as to allow swapping states between them. The new state space is then $\Omega := S^N$, and the desired master steady state distribution on Ω is

$$\pi_{PT}(x) = \prod_{k=1}^{N} \pi_k(y_k), \tag{7.15}$$

where $x = (y_1, \ldots, y_N) \in \Omega$.

There are two allowed moves on Ω:

1. Parallel Markov Random Walk: Updating $x^{(t)}$ from $x^{(t-1)}$ is done element-wise using the Metropolis-Hastings algorithm on each sub-chain separately.
2. State Swap: Choose k uniformly from $\{1, \ldots, N-1\}$. Propose swapping $y_k^{(t)}$ with $y_{k+1}^{(t)}$. We need to accept this proposed swap so that the detailed balance condition is still satisfied. That is, we need to balance between

$$(y_1^{(t)}, \ldots, y_k^{(t)}, y_{k+1}^{(t)}, \ldots, y_N^{(t)}) \to (y_1^{(t)}, \ldots, y_{k+1}^{(t)}, y_k^{(t)}, \ldots, y_N^{(t)})$$

   and the reverse move

$$(y_1^{(t)}, \ldots, y_{k+1}^{(t)}, y_k^{(t)}, \ldots, y_N^{(t)}) \to (y_1^{(t)}, \ldots, y_k^{(t)}, y_{k+1}^{(t)}, \ldots, y_N^{(t)}).$$

   Using the Metropolis-Hasting methods, we thus need to accept the proposal with probability

$$r = \min\left\{1, \frac{\pi_{PT}(y_1^{(t)}, \ldots, y_{k+1}^{(t)}, y_k^{(t)}, \ldots, y_N^{(t)})}{\pi_{PT}(y_1^{(t)}, \ldots, y_k^{(t)}, y_{k+1}^{(t)}, \ldots, y_N^{(t)})}\right\}.$$

Using (7.15), we can simplify this expression as

$$r = \min\left\{1, \frac{\pi_k(y_{k+1}^{(t)})\pi_{k+1}(y_k^{(t)})}{\pi_k(y_k^{(t)})\pi_{k+1}(y_{k+1}^{(t)})}\right\}.$$

REMARK 7.26. *There are different ways of alternating between these two moves. For example, one can simply iterate between the two moves one by one. Or, one can first take $M > 1$ Parallel Markov Random Walks and then attempt a State Swap.*

REMARK 7.27. *In the case of minimizing an objective function f on S, we have*

$$\pi_k(y) \propto \exp(-\beta_k f(y)).$$

Hence, the acceptance probability is

$$r = \min\left\{1, \exp\left[(\beta_{k+1} - \beta_k)\Delta f\right]\right\},$$

where $\Delta f = f(y_{k+1}^{(t)}) - f(y_k^{(t)})$, which implies that if the objective function has a higher value at the lower temperature β_{k+1} than at β_k, then the proposed swap is accepted with probability 1. Otherwise, there is a finite probability that the swap will be rejected; and, the lower the value of $f(y_{k+1})$ compared to $f(y_k)$, the higher the rejection probability.

7.5 Hamiltonian Monte Carlo

As previously discussed, poor mixing can arise from having several local modes in the probability landscape of target distribution. A similar situation arises in classical mechanics when potential energy U has several local minima that can trap a particle carrying low momentum. Physical intuition tells us that such local minima can be escaped by increasing the momentum of the particle. Hamiltonian Monte Carlo (HMC) utilizes this intuition by augmenting the original state space to its phase space and allowing a jump in momentum at each MCMC step. Forgetting about the momentum coordinate, i.e. marginalizing it, then yields samples on the original state space.

We will briefly sketch the HMC approach in this subsection. We know from classical mechanics that a classical particle evolves according to Hamilton's equations

$$\dot{q}_i = \frac{\partial H}{\partial p_i} \text{ and } \dot{p}_i = -\frac{\partial H}{\partial q_i},$$

where p_i is the momentum conjugate to coordinate q_i, and

$$H(q,p) = \frac{p^2}{2m} + U(q)$$

is the Hamiltonian. If $\pi_s(q)$ is the target distribution from which we ultimately wish to sample, then we take

$$U(q) = -\log \pi_s(q).$$

We know that

1. the Hamiltonian evolution satisfies time reversal symmetry,
2. H is conserved when U does not explicitly depend on time (Noether's theorem), and
3. the Hamiltonian flow preserves volume in phase space (Liouville's theorem).

Hence, if we use the integral curve of Hamiltonian flow for time step T on the phase space as the Markov transition kernel, then the canonical distribution

$$\pi(q,p) \propto e^{-\left(\frac{\mathbf{p}^2}{2m}+U(q)\right)}$$

on the phase space is stationary. However, the chain does not converge to $\pi(q,p)$, because the moves are confined to constant energy surfaces. To make the chain ergodic, we update the momentum by sampling from the normal distribution $\mathcal{N}(0,mI)$ before evolving the Hamiltonian flow by time interval T. Hence, each HMC step of moving from (q_n,p_n) involves

1. Sample $p' \sim \mathcal{N}(0,mI)$.
2. Approximate Hamiltonian evolution from (q_n,p') to $(\tilde{q},\tilde{p})$ by time interval T.
3. Sample $u \sim \text{Unif}(0,1)$.
4. if $u \leq \min(1, e^{-H(\tilde{q},-\tilde{p})+H(q_n,p')})$: $q_{n+1} = \tilde{q}$ and $p_{n+1} = -\tilde{p}$; else: $q_{n+1} = q_n$ and $p_{n+1} = p'$.

REMARK 7.28. *If the Hamiltonian evolution could be performed exactly, then the Hamiltonian would be constant along the path, and the acceptance probability would be 1. In practical applications, however, the evolution arises from numerical integration, and the Hamiltonian may change slightly, still yielding a high acceptance probability.*

REMARK 7.29. *The final momentum was flipped in sign to account for the initial condition of the time-reversed evolution.*

REMARK 7.30. *In the above algorithm, a new momentum value will be sampled at each iteration from $\mathcal{N}(0,mI)$, and it does not depend on the previous momentum. Hence, we do not need to keep track of p_n.*

REMARK 7.31. *In practice, we usually cannot analytically solve Hamilton's equations and need to numerically integrate the equations, typically using the leapfrog integration method.*