9.8 Steepest descent method in ℓ_∞-norm

The steepest descent direction is defined as:

$$\Delta x_{\text{sd}} = \arg\min_{v}\{\nabla f(x)^T v \mid \|v\|_\infty \le 1\}$$

The ℓ_∞-norm ball is $\{v \mid \|v\|_\infty \le 1\} = \{v \mid -1 \le v_i \le 1,\ i = 1, \dots, n\}$, which is a hypercube.

To minimize the linear function $\nabla f(x)^T v = \sum_{i=1}^n (\nabla f(x))_i v_i$ over this hypercube, each term is minimized independently:

$$\min_{-1 \le v_i \le 1} (\nabla f(x))_i v_i = -|(\nabla f(x))_i|$$

achieved when $v_i = -\text{sign}((\nabla f(x))_i)$.

Therefore, the steepest descent direction in the ℓ_∞-norm is:

$$\boxed{(\Delta x_{\text{sd}})_i = -\text{sign}((\nabla f(x))_i), \quad i = 1, \dots, n}$$

For each coordinate, move in the opposite direction of the gradient's sign.

A7.12 Cox Proportional Hazards Model

The survival function can be expressed in terms of the hazard rate:

$$S(t) = \exp\left(-\int_0^t \lambda(\tau)\, d\tau\right)$$

Under the Cox proportional hazards model:

$$S(t) = \exp\left(-\exp(w^T x)\int_0^t \lambda_0(\tau)\, d\tau\right)$$

The density of T is:

$$p(t) = -S'(t) = \lambda(t)S(t) = \lambda_0(t)\exp(w^T x)\exp\left(-\exp(w^T x)\int_0^t \lambda_0(\tau)\, d\tau\right)$$

For observation j with event time t^j and features x^j:

$$\log p(t^j \mid x^j, w) = \log \lambda_0(t^j) + w^T x^j - \exp(w^T x^j)\int_0^{t^j} \lambda_0(\tau)\, d\tau$$

The log-likelihood over all observations is:

$$\ell(w) = \sum_{j=1}^N \left[\log \lambda_0(t^j) + w^T x^j - \exp(w^T x^j)\int_0^{t^j} \lambda_0(\tau)\, d\tau\right]$$

The negative log-likelihood is:

$$-\ell(w) = \sum_{j=1}^N \left[\exp(w^T x^j)\int_0^{t^j} \lambda_0(\tau)\, d\tau - w^T x^j\right] + \text{const}$$

Each term $\exp(w^T x^j)\int_0^{t^j} \lambda_0(\tau)\, d\tau - w^T x^j$ is:

- $\exp(w^T x^j)$ is convex in w
- $\int_0^{t^j} \lambda_0(\tau)\, d\tau \geq 0$ (since $\lambda_0 \geq 0$), so $\exp(w^T x^j)\int_0^{t^j} \lambda_0(\tau)\, d\tau$ is convex
- $-w^T x^j$ is affine (hence convex)

The sum of convex functions is convex, so $-\ell(w)$ is convex, and the maximum likelihood estimation of w is a convex optimization problem.

The problem can be written as:

$$\text{minimize} \quad \sum_{j=1}^N \left[\int_0^{t^j} \lambda_0(\tau)\, d\tau \exp(w^T x^j) - w^T x^j\right]$$

A8.16 Fitting a Sphere to Data

(a): Convex formulation

With the substitution $t = r^2 - \|c\|_2^2$, we have:

$$\begin{aligned}\|u_i - c\|_2^2 - r^2 &= \|u_i\|_2^2 - 2u_i^T c + \|c\|_2^2 - r^2 \\ &= \|u_i\|_2^2 - 2u_i^T c - (r^2 - \|c\|_2^2) \\ &= \|u_i\|_2^2 - 2u_i^T c - t.\end{aligned}$$

The loss function becomes:

$$\sum_{i=1}^m \left(\|u_i\|_2^2 - 2u_i^T c - t\right)^2,$$

so the problem is the least-squares problem in variables (c, t):

$$\text{minimize}_{c,\,t} \quad \sum_{i=1}^m \left(-2u_i^T c + \|u_i\|_2^2 - t\right)^2,$$

which is a convex quadratic optimization problem.

From $t = r^2 - \|c\|_2^2$, we have

$$r^2 = t + \|c\|_2^2.$$

After solving for (c^*, t^*), we set

$$r^* = \sqrt{t^* + \|c^*\|_2^2}.$$

(b): example

The problem in matrix form is:

$$\text{minimize} \quad \|Ax - b\|_2^2$$

where $x = \begin{bmatrix} c \\ t \end{bmatrix} \in \mathbf{R}^{n+1}$, and

$$A = \begin{bmatrix} -2u_1^T & -1 \\ \vdots & \vdots \\ -2u_m^T & -1 \end{bmatrix}, \quad b = \begin{bmatrix} -\|u_1\|_2^2 \\ \vdots \\ -\|u_m\|_2^2 \end{bmatrix}$$

The solution is $x^* = (A^T A)^{-1} A^T b$. **Results:**

$$c^* = (1.4759, -0.0440), \quad t^* = -1.1106, \quad r^* = \boxed{1.0342}$$

The optimal objective value is 2.9038.

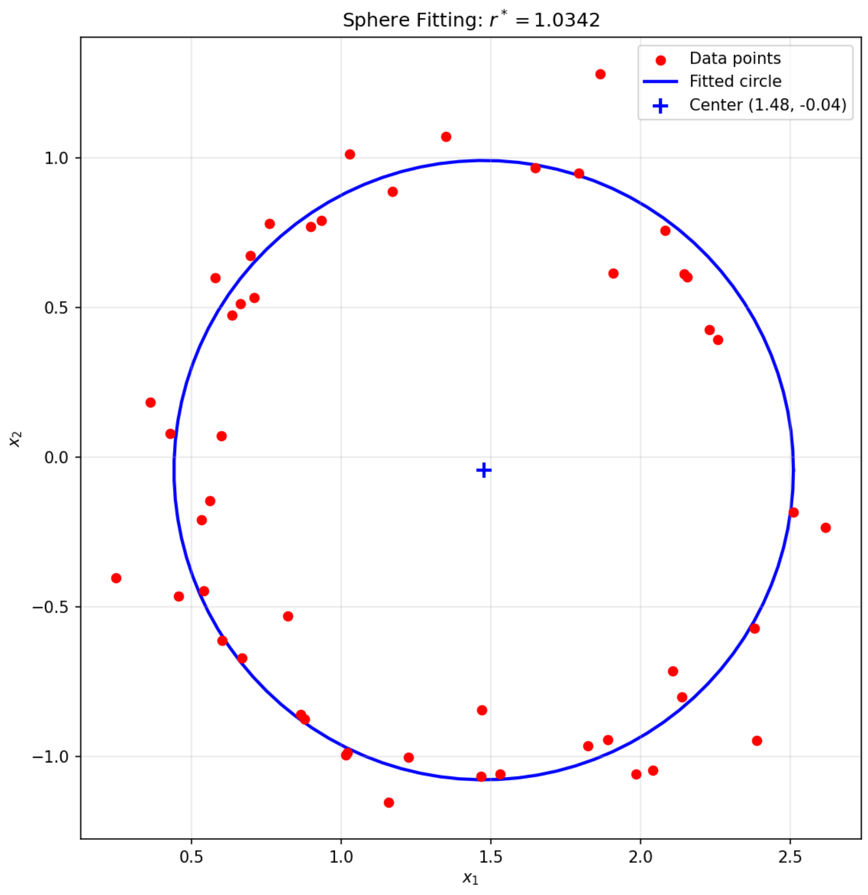

Figure 1: Fitted circle (blue) and data points (red) for the sphere fitting problem.

Code:

```python
A = np.zeros((m, n + 1))
b = np.zeros(m)
for i in range(m):
    u_i = U[:, i]
    A[i, :n] = -2 * u_i
    A[i, n] = -1
    b[i] = -np.dot(u_i, u_i)

x_star = np.linalg.lstsq(A, b, rcond=None)[0]
c_star = x_star[:n]
t_star = x_star[n]

r_star = np.sqrt(t_star + np.dot(c_star, c_star))
```

A8.26 Some Problems Involving a Polyhedron and a Point

(a) Find a point in $\mathcal{P}$ closest to a in Euclidean norm

Easy

$$\begin{aligned} \min \quad & \|x-a\|_2 \\ \text{s.t.} \quad & x \in \mathcal{P} \end{aligned}$$

The objective is convex (quadratic), the constraint set $\mathcal{P}$ is convex (polyhedron), so this is a convex optimization problem solvable efficiently.

(b) Find a point in $\mathcal{P}$ closest to a in ℓ_∞ norm

Easy

$$\min \|x-a\|_\infty = \min_x \max_i |x_i - a_i|$$

Can be be reformulated as a linear program by introducing as scalar t:

$$\begin{aligned} \min \quad & t \\ \text{s.t.} \quad & -t\mathbf{1} \preceq x - a \preceq t\mathbf{1} \\ & x \in \mathcal{P} \end{aligned}$$

The constraint $\|x-a\|_\infty \le t$ is equivalent to $-t \le x_i - a_i \le t$ for all i, which is linear.

(c) Find a point in $\mathcal{P}$ farthest from a in Euclidean norm

Hard

$$\begin{aligned} \max \quad & \|x-a\|_2 \\ \text{s.t.} \quad & x \in \mathcal{P} \end{aligned}$$

Maximizing a convex function over a convex set is generally non-convex. The maximum is at some vertex of P, and enumerating the vertices can be exponential.

(d) Find a point in $\mathcal{P}$ farthest from a in ℓ_∞ norm

Easy

$$\|x-a\|_\infty = \max_i |x_i - a_i| = \max_i \max\{x_i - a_i,\ a_i - x_i\}$$

Therefore,

$$\max_{x\in\mathcal{P}} \|x-a\|_\infty = \max_{i=1,\dots,n} \max\left\{\max_{x\in\mathcal{P}}(x_i - a_i),\ \max_{x\in\mathcal{P}}(a_i - x_i)\right\}$$

Each inner problem is a linear program (maximize or minimize x_i over $\mathcal{P}$), so the overall problem can be solved by solving $2n$ LPs and taking the best value.

A9.10 True or False

(a) In descent methods, the particular choice of search direction does not matter so much.

False

(b) In descent methods, the particular choice of line search does not matter so much.

True

(c) When the gradient method is started from a point near the solution, it will converge very quickly.

False

(d) When Newton's method is started from a point near the solution, it will converge very quickly.

True

(e) Newton's method with step size $h = 1$ always works; damping (i.e., using $h < 1$) is used only to improve the speed of convergence.

False

(f) Using the gradient method to minimize $f(Ty)$, where $Ty = x$ and T is nonsingular, can greatly improve the convergence speed when T is chosen appropriately.

True

(g) Using Newton's method to minimize $f(Ty)$, where $Ty = x$ and T is nonsingular, can greatly improve the convergence speed when T is chosen appropriately.

False

A9.13 Newton's Method in Machine Learning Problems

Answer

(d) Due to the large problem size, it is generally not practical to form or store the Hessian, let alone compute the Newton step In ML the number of parameters is huge, so forming/storing an $n \times n$ Hessian and computing the Newton step is usually too expensive in memory and time.

A13.2 True or False (Flop Counts)

(a) Algorithm flop counts allow for very accurate prediction of running time on a given computer

False

(b) Since matrix multiplication is associative, the flop count for multiplying three or more matrices doesn't depend on the order in which you multiply them

False

(c) Suppose $A \in \mathbf{R}^{n \times n}$ is lower triangular. The flop count for computing Ab is the same order as the flop count for computing $A^{-1}b$

True

A13.3 Ridge Regression

(a) Tall matrix ($m \geq n$)

mn^2

(b) Wide matrix ($m \leq n$)

m^2n

A17.16 Efficient Solution of Basic Portfolio Optimization Problem

(a)

Ignoring the low-rank-plus-diagonal structure and treating Σ as dense (assume $\lambda = 1$), the problem can be solved via the KKT system or by factoring Σ (e.g., Cholesky) and doing a few solves. The dominant cost is the factorization of an $n \times n$ dense matrix, so the flop count is

$$\boxed{O(n^3)}.$$

(b)

Assume $\lambda = 1$ and exploit $\Sigma = FQF^T + D$ with $k \ll n$ using Woodbury:

$$(D + FQF^T)^{-1} = D^{-1} - D^{-1}F\left(Q^{-1} + F^TD^{-1}F\right)^{-1}F^TD^{-1}.$$

Precompute

$$B = F^TD^{-1}F \in \mathbf{R}^{k\times k}, \qquad M = Q^{-1} + B \in \mathbf{S}^k_{++},$$

and factor M (Cholesky). To compute the optimal portfolio, use the KKT conditions

$$\Sigma w + \nu\mathbf{1} = \mu, \qquad \mathbf{1}^Tw = 1.$$

Let

$$q = \Sigma^{-1}\mu, \qquad p = \Sigma^{-1}\mathbf{1}.$$

Then

$$\nu = \frac{\mathbf{1}^Tq - 1}{\mathbf{1}^Tp}, \qquad w = q - \nu p.$$

Each application of Σ^{-1} to a vector v is computed via Woodbury:

$$\Sigma^{-1}v = D^{-1}v - D^{-1}F\,M^{-1}F^TD^{-1}v.$$

Flop count: Forming $B = F^TD^{-1}F$ costs $O(nk^2)$, factoring M costs $O(k^3)$, and each solve $\Sigma^{-1}v$ costs $O(nk + k^2)$. As two solves are needed (for $v = \mu$ and $v = \mathbf{1}$), the overall dominant cost is

$$\boxed{O(nk^2 + k^3)}.$$

(c)

Random data is generated with $n = 2500$, $k = 30$ (random F, SPD Q, positive diagonal D), and w^* is computed using both methods.

Results:

Method	Time (s)	Objective
Dense (part a)	0.167	2298.29
Structured (part b)	0.052	2298.29

Speedup: $\approx 3\times$.

Code:

```python
import scipy.sparse as sp
import scipy.sparse.linalg as spla

D_sparse = sp.diags(D_diag, format='csr')           # (n,n)
FQ_sparse = sp.csr_matrix(F @ Q)                    # (n,k)
ones_n = sp.csr_matrix(np.ones((n, 1)))             # (n,1)
zeros_k1 = sp.csr_matrix((k, 1))                    # (k,1)
zeros_1k = sp.csr_matrix((1, k))                    # (1,k)

KKT = sp.bmat([
    [D_sparse,             FQ_sparse,  ones_n],
    [sp.csr_matrix(-F.T), sp.eye(k),  zeros_k1],
    [sp.csr_matrix(np.ones((1,n))), zeros_1k, None]
], format='csr')
rhs = np.concatenate([mu, np.zeros(k), [1.0]])
w_star = spla.spsolve(KKT, rhs)[:n]
```

(d)

For M different values of $\lambda > 0$, the KKT conditions are

$$\lambda \Sigma w + \nu \mathbf{1} = \mu, \qquad \mathbf{1}^T w = 1.$$

From the first equation: $w = \frac{1}{\lambda}\Sigma^{-1}(\mu - \nu\mathbf{1}) = \frac{1}{\lambda}(q - \nu p)$, where $q = \Sigma^{-1}\mu$ and $p = \Sigma^{-1}\mathbf{1}$.

Substituting into $\mathbf{1}^T w = 1$:

$$\frac{1}{\lambda}(\mathbf{1}^T q - \nu \cdot \mathbf{1}^T p) = 1 \implies \nu = \frac{\mathbf{1}^T q - \lambda}{\mathbf{1}^T p}.$$

Thus:

$$w = \frac{1}{\lambda}\left(q - \frac{\mathbf{1}^T q - \lambda}{\mathbf{1}^T p} p\right) = \frac{p}{\mathbf{1}^T p} + \frac{1}{\lambda}\left(q - \frac{\mathbf{1}^T q}{\mathbf{1}^T p} p\right) = w_0 + \frac{1}{\lambda} w_1,$$

where

$$w_0 = \frac{p}{\mathbf{1}^T p}, \qquad w_1 = q - \frac{\mathbf{1}^T q}{\mathbf{1}^T p} p.$$

p and q are computed once (using part (b)), then $w^*(\lambda)$ is evaluated for each λ in $O(n)$ time.

Complexity comparison:

- Using part (b) M times: $O(M(nk^2 + k^3))$
- Using the affine formula: $O(nk^2 + k^3) + O(Mn)$ (one-time precomputation + M evaluations)

The affine method is much more efficient when M is large, since the expensive Σ^{-1} computation is done only once.