19 Interior Point II: Quadratic Programs

Lab Objective: *Interior point methods originated as an alternative to the Simplex method for solving linear optimization problems. However, they can also be adapted to treat convex optimization problems in general. In this lab, we implement a primal-dual Interior Point method for convex quadratic constrained optimization and explore applications in elastic membrane theory and finance.*

Quadratic Optimization Problems

A *quadratic constrained optimization problem* differs from a linear constrained optimization problem only in that the objective function is quadratic rather than linear. We can pose such a problem as follows:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \succeq \mathbf{b}, \\ & G\mathbf{x} = \mathbf{h}. \end{aligned}$$

We will restrict our attention to quadratic programs involving positive semidefinite quadratic terms (in general, indefinite quadratic objective functions admit many local minima, complicating matters considerably). Such problems are called *convex*, since the objective function is convex. To simplify the exposition, we will also only allow inequality constraints (generalizing to include equality constraints is not difficult). Thus, we have the problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \succeq \mathbf{b} \end{aligned}$$

where Q is an $n \times n$ positive semidefinite matrix, $\mathbf{x}, \mathbf{c} \in \mathbb{R}^n$, A is an $m \times n$ matrix, and $\mathbf{b} \in \mathbb{R}^m$.

The Lagrangian function for this problem is:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} - \boldsymbol{\mu}^T(A\mathbf{x} - \mathbf{b}), \tag{19.1}$$

where $\boldsymbol{\mu} \in \mathbb{R}^m$ is the (as of yet unknown) Lagrange multiplier.

We also introduce a nonnegative slack vector $\mathbf{y} \in \mathbb{R}^m$ to change the inequality

$$A\mathbf{x} - \mathbf{b} \succeq \mathbf{0}$$

into the equality

$$A\mathbf{x} - \mathbf{b} - \mathbf{y} = \mathbf{0} \qquad \Longrightarrow \qquad \mathbf{y} = A\mathbf{x} - \mathbf{b} \tag{19.2}$$

Equations 19.1 and 19.2, together with complementary slackness, gives us our complete set of KKT conditions:

$$\begin{aligned} Q\mathbf{x} - A^T\boldsymbol{\mu} + \mathbf{c} &= \mathbf{0}, \\ A\mathbf{x} - \mathbf{y} - \mathbf{b} &= \mathbf{0}, \\ y_i\mu_i &= 0, \qquad i = 1, 2, \ldots, m, \\ \mathbf{y}, \boldsymbol{\mu} &\succeq \mathbf{0}. \end{aligned}$$

Quadratic Interior Point Method

The Interior Point method we describe here is an adaptation of the method we used with linear programming. Define $Y = \operatorname{diag}(y_1, y_2, \ldots, y_m)$, $M = \operatorname{diag}(\mu_1, \mu_2, \ldots, \mu_m)$, and let $\mathbf{e} \in \mathbb{R}^m$ be a vector of all ones. Then the roots of the function

$$F(\mathbf{x}, \mathbf{y}, \boldsymbol{\mu}) = \begin{bmatrix} Q\mathbf{x} - A^T\boldsymbol{\mu} + c \\ A\mathbf{x} - \mathbf{y} - \mathbf{b} \\ YM\mathbf{e} \end{bmatrix} = \mathbf{0}, \qquad (\mathbf{y}, \boldsymbol{\mu}) \succeq \mathbf{0}$$

satisfy the KKT conditions. The derivative matrix of this function is given by

$$DF(\mathbf{x}, \mathbf{y}, \boldsymbol{\mu}) = \begin{bmatrix} Q & 0 & -A^T \\ A & -I & 0 \\ 0 & M & Y \end{bmatrix},$$

and the duality measure ν for this problem is

$$\nu = \frac{\mathbf{y}^T\boldsymbol{\mu}}{m}.$$

Search Direction

We calculate the search direction for this algorithm the same way that we did in the linear programming case. That is, we solve the system:

$$DF(\mathbf{x}, \mathbf{y}, \boldsymbol{\mu}) \begin{bmatrix} \triangle\mathbf{x} \\ \triangle\mathbf{y} \\ \triangle\boldsymbol{\mu} \end{bmatrix} = -F(\mathbf{x}, \mathbf{y}, \boldsymbol{\mu}) + \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \sigma\nu\mathbf{e} \end{bmatrix}, \tag{19.3}$$

where $\sigma \in [0, 1)$ is the centering parameter.

Problem 1. Copy your `interiorPoint()` function from the previous lab into your solutions file for this lab, renaming it `qInteriorPoint()`. This new function should accept the arrays $Q, \mathbf{c}, A$, and $\mathbf{b}$, a tuple of arrays `guess` giving initial estimates for $\mathbf{x}, \mathbf{y}$, and $\boldsymbol{\mu}$ (this will be explained later), along with the keyword arguments `niter=20` and `tol=1e-16`.

Modify your code to match the F and DF described above, and calculate the search direction $(\triangle\mathbf{x}^T, \triangle\mathbf{y}^T, \triangle\boldsymbol{\mu}^T)$ by solving Equation 19.3. Use $\sigma = \frac{1}{10}$ for the centering parameter.

Hint: What are the dimensions of F and DF?

Step Length

Now that we have our search direction, we select a step length. We want to step nearly as far as possible without violating the nonnegativity constraints. We back off slightly from the maximum allowed step length, however, because an overly greedy step at one iteration may prevent a decent step at the next iteration. Thus, we choose our step size

$$\alpha = \max\{a \in (0, 1] \mid \tau(\mathbf{y}, \boldsymbol{\mu}) + a(\triangle\mathbf{y}, \triangle\boldsymbol{\mu}) \succeq \mathbf{0}\},$$

where $\tau \in (0, 1)$ controls how much we back off from the maximal step length. For now, choose $\tau = 0.95$. In general, τ can be made to approach 1 at each successive iteration, and this may speed up convergence in some cases.

This is equivalent to the method of choosing a step direction used in the previous lab. In this case, however, we will use a single step length for all three of the parameters.

$$\begin{aligned}\beta_{\max} &= \min\{1, \min\{-\mu_i/\triangle\mu_i \mid \triangle\mu_i < 0\}\} \\ \delta_{\max} &= \min\{1, \min\{-y_i/\triangle y_i \mid \triangle y_i < 0\}\}\end{aligned}$$

Since $\boldsymbol{\mu}, \mathbf{y} \geq \mathbf{0}$. If all of the entries of $\triangle\mu$ are positive, we let $\beta_{\max} = 1$, and likewise for $\delta_{\max}$. Next, we back off from these maximum step lengths slightly:

$$\begin{aligned}\beta &= \min(1, \tau\beta_{\max}) \\ \delta &= \min(1, \tau\delta_{\max}) \\ \alpha &= \min(\beta, \delta)\end{aligned}$$

This α is our final step length. Thus, the next point in the iteration is given by:

$$(\mathbf{x}_{k+1}, \mathbf{y}_{k+1}, \boldsymbol{\mu}_{k+1}) = (\mathbf{x}_k, \mathbf{y}_k, \boldsymbol{\mu}_k) + \alpha(\triangle\mathbf{x}_k, \triangle\mathbf{y}_k, \triangle\boldsymbol{\mu}_k).$$

This completes one iteration of the algorithm.

Initial Point

As usual, the starting point $(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\mu}_0)$ has an important effect on the convergence of the algorithm. The code listed below will calculate an appropriate starting point:

```
def startingPoint(G, c, A, b, guess):
    """
    Obtain an appropriate initial point for solving the QP
```

```
    .5 x^T Gx + x^T c s.t. Ax >= b.
    Inputs:
        G -- symmetric positive semidefinite matrix shape (n,n)
        c -- array of length n
        A -- constraint matrix shape (m,n)
        b -- array of length m
        guess -- a tuple of arrays (x, y, l) of lengths n, m, and m, resp.
    Returns:
        a tuple of arrays (x0, y0, l0) of lengths n, m, and m, resp.
    """
    m,n = A.shape
    x0, y0, l0 = guess

    # initialize linear system
    N = np.zeros((n+m+m, n+m+m))
    N[:n,:n] = G
    N[:n, n+m:] = -A.T
    N[n:n+m, :n] = A
    N[n:n+m, n:n+m] = -np.eye(m)
    N[n+m:, n:n+m] = np.diag(l0)
    N[n+m:, n+m:] = np.diag(y0)
    rhs = np.empty(n+m+m)
    rhs[:n] = -(G.dot(x0) - A.T.dot(l0)+c)
    rhs[n:n+m] = -(A.dot(x0) - y0 - b)
    rhs[n+m:] = -(y0*l0)

    sol = la.solve(N, rhs)
    dx = sol[:n]
    dy = sol[n:n+m]
    dl = sol[n+m:]

    y0 = np.maximum(1, np.abs(y0 + dy))
    l0 = np.maximum(1, np.abs(l0+dl))

    return x0, y0, l0
```

Notice that we still need to provide a tuple of arrays `guess` as an argument. Do your best to provide a reasonable guess for the array $\mathbf{x}$, and we suggest setting $\mathbf{y}$ and $\boldsymbol{\mu}$ equal to arrays of ones. We summarize the entire algorithm below.

---

procedure Interior Point Method for QP
  Choose initial point $(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\mu}_0)$.
  while $k <$ `niters` and $\nu <$ `tol`: **do**
    Calculate the duality measure ν.
    Solve 19.3 for the search direction $(\triangle\mathbf{x}_k, \triangle\mathbf{y}_k, \triangle\boldsymbol{\mu}_k)$.
    Calculate the step length α.
    $(\mathbf{x}_{k+1}, \mathbf{y}_{k+1}, \boldsymbol{\mu}_{k+1}) = (\mathbf{x}_k, \mathbf{y}_k, \boldsymbol{\mu}_k) + \alpha(\triangle\mathbf{x}_k, \triangle\mathbf{y}_k, \triangle\boldsymbol{\mu}_k)$.

Problem 2. Complete the implementation of `qInteriorPoint()`. Return the optimal point $\mathbf{x}$ as well as the final objective function value. You may want to print out the duality measure ν to check the progress of the iteration.

Test your algorithm on the simple problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2 \\ \text{subject to} \quad & -x_1 - x_2 \geq -2, \\ & x_1 - 2x_2 \geq -2, \\ & -2x_1 - x_2 \geq -3, \\ & x_1, x_2 \geq 0. \end{aligned}$$

In this case, we have for the objective function matrix Q and vector $\mathbf{c}$,

$$Q = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} -2 \\ -6 \end{bmatrix}.$$

The constraint matrix A and vector $\mathbf{b}$ are given by:

$$A = \begin{bmatrix} -1 & -1 \\ 1 & -2 \\ -2 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} -2 \\ -2 \\ -3 \\ 0 \\ 0 \end{bmatrix}.$$

Use $\mathbf{x} = [.5, .5]$ as the initial guess. The correct minimizer is $\left[\frac{2}{3}, \frac{4}{3}\right]$.

NOTE

The Interior Point methods presented in this and the preceding labs are only special cases of the more general Interior Point algorithm. The general version can be used to solve many convex optimization problems, provided that one can derive the corresponding KKT conditions and duality measure ν.

Application: Optimal Elastic Membranes

The properties of elastic membranes (stretchy materials like a thin rubber sheet) are of interest in certain fields of mathematics and various sciences. A mathematical model for such materials can be used by biologists to study interfaces in cellular regions in an organism, or by engineers to design tensile structures. Often we can describe configurations of elastic membranes as a solution to an optimization problem. As a simple example, we will find the shape of a large circus tent by solving a quadratic constrained optimization problem using our Interior Point method.

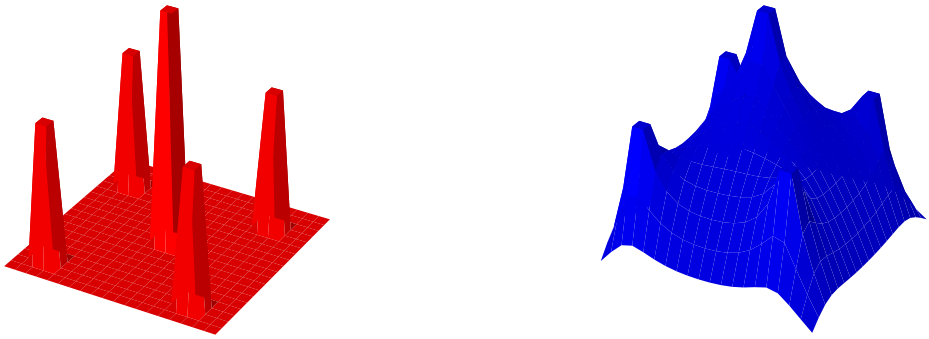

Figure 19.1: Tent pole configuration (left) and optimal elastic tent (right).

Imagine a large circus tent held up by a few poles. We can model the tent by a square two-dimensional grid, where each grid point has an associated number that gives the height of the tent at that point. At each grid point containing a tent pole, the tent height is constrained to be at least as large as the height of the tent pole. At all other grid points, the tent height is simply constrained to be greater than zero (ground height). In Python, we can store a two-dimensional grid of values as a simple two-dimensional array. We can then flatten this array to give a one-dimensional vector representation of the grid. If we let $\mathbf{x}$ be a one-dimensional array giving the tent height at each grid point, and L be the one-dimensional array giving the underlying tent pole structure (consisting mainly of zeros, except at the grid points that contain a tent pole), we have the linear constraints:

$$\mathbf{x} \succeq L.$$

The theory of elastic membranes claims that such materials tend to naturally minimize a quantity known as the *Dirichlet energy*. This quantity can be expressed as a quadratic function of the membrane. Then since we have modeled our tent with a discrete grid of values, this energy function has the form

$$\frac{1}{2}\mathbf{x}^T H\mathbf{x} + \mathbf{c}^T\mathbf{x},$$

where H is a particular positive semidefinite matrix closely related to Laplace's Equation, $\mathbf{c}$ is a vector whose entries are all equal to $-(n-1)^{-2}$, and n is the side length of the grid. Our circus tent is therefore given by the solution to the quadratic constrained optimization problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^T H\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{x} \succeq L. \end{aligned}$$

See Figure 19.1 for an example of a tent pole configuration and the corresponding tent.

We provide the following function for producing the Dirichlet energy matrix H.

```
from scipy.sparse import spdiags
def laplacian(n):
    """Construct the discrete Dirichlet energy matrix H for an n x n grid."""
    data = -1*np.ones((5, n**2))
    data[2,:] = 4
    data[1, n-1::n] = 0
    data[3, ::n] = 0
    diags = np.array([-n, -1, 0, 1, n])
    return spdiags(data, diags, n**2, n**2).toarray()
```

Now we initialize the tent pole configuration for a grid of side length n, as well as initial guesses for $\mathbf{x}$, $\mathbf{y}$, and $\boldsymbol{\mu}$.

```
# Create the tent pole configuration.
>>> L = np.zeros((n,n))
>>> L[n//2-1:n//2+1,n//2-1:n//2+1] = .5
>>> m = [n//6-1, n//6, int(5*(n/6.))-1, int(5*(n/6.))]
>>> mask1, mask2 = np.meshgrid(m, m)
>>> L[mask1, mask2] = .3
>>> L = L.ravel()

# Set initial guesses.
>>> x = np.ones((n,n)).ravel()
>>> y = np.ones(n**2)
>>> mu = np.ones(n**2)
```

We leave it to you to initialize the vector $\mathbf{c}$, the constraint matrix A, and to initialize the matrix H with the `laplacian()` function. We can solve and plot the tent with the following code:

```
>>> from matplotlib import pyplot as plt
>>> from mpl_toolkits.mplot3d import axes3d

# Calculate the solution.
>>> z = qInteriorPoint(H, c, A, L, (x,y,mu))[0].reshape((n,n))

# Plot the solution.
>>> domain = np.arange(n)
>>> X, Y = np.meshgrid(domain, domain)
>>> fig = plt.figure()
>>> ax1 = fig.add_subplot(111, projection='3d')
>>> ax1.plot_surface(X, Y, z,  rstride=1, cstride=1, color='r')
>>> plt.show()
```

Problem 3. Solve the circus tent problem with the tent pole configuration given above, for grid side length $n = 15$. Plot your solution.

Application: Markowitz Portfolio Optimization

Suppose you have a certain amount of money saved up, with no intention of consuming it any time soon. What will you do with this money? If you hide it somewhere in your living quarters or on your person, it will lose value over time due to inflation, not to mention you run the risk of burglary or accidental loss. A safer choice might be to put the money into a bank account. That way, there is less risk of losing the money, plus you may even add to your savings through interest payments from the bank. You could also consider purchasing bonds from the government or stocks from various companies, which come with their own sets of risks and returns. Given all of these possibilities, how can you invest your money in such a way that maximizes the return (i.e. the wealth that you gain over the course of the investment) while still exercising caution and avoiding excessive risk? Economist and Nobel laureate Harry Markowitz developed the mathematical underpinnings of and answer to this question in his work on modern portfolio theory.

A *portfolio* is a set of investments over a period of time. Each investment is characterized by a financial asset (such as a stock or bond) together with the proportion of wealth allocated to the asset. An asset is a random variable, and can be described as a sequence of values over time. The variance or spread of these values is associated with the risk of the asset, and the percent change of the values over each time period is related to the return of the asset. For our purposes, we will assume that each asset has a positive risk, i.e. there are no *riskless* assets available.

Stated more precisely, our portfolio consists of n risky assets together with an allocation vector $\mathbf{x} = (x_1, \dots, x_n)^T$, where x_i indicates the proportion of wealth we invest in asset i. By definition, the vector $\mathbf{x}$ must satisfy

$$\sum_{i=1}^{n} x_i = 1.$$

Suppose the i^{th} asset has an expected rate of return μ_i and a standard deviation σ_i. The total return on our portfolio, i.e. the expected percent change in our invested wealth over the investment period, is given by

$$\sum_{i=1}^{n} \mu_i x_i.$$

We define the risk of this portfolio in terms of the covariance matrix Q of the n assets:

$$\sqrt{\mathbf{x}^T Q \mathbf{x}}.$$

The covariance matrix Q is always positive semidefinite, and captures the variance and correlations of the assets.

Given that we want our portfolio to have a prescribed return R, there are in general many possible allocation vectors $\mathbf{x}$ that make this possible. It would be wise to choose the vector minimizing the risk. We can state this as a quadratic program:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^T Q \mathbf{x} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = 1 \\ & \sum_{i=1}^{n} \mu_i x_i = R. \end{aligned}$$

Note that we have slightly altered our objective function for convenience, as minimizing $\frac{1}{2}\mathbf{x}^T Q\mathbf{x}$ is equivalent to minimizing $\sqrt{\mathbf{x}^T Q\mathbf{x}}$. The solution to this problem will give the portfolio with least risk having a return R. Because the components of $\mathbf{x}$ are not constrained to be nonnegative, the solution may have some negative entries. This indicates short selling those particular assets. If we want to disallow short selling, we simply include nonnegativity constraints, stated in the following problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^T Q\mathbf{x} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = 1 \\ & \sum_{i=1}^{n} \mu_i x_i = R \\ & \mathbf{x} \succeq \mathbf{0}. \end{aligned}$$

Each return value R can be paired with its corresponding minimal risk σ. If we plot these risk-return pairs on the risk-return plane, we obtain a hyperbola. In general, the risk-return pair of any portfolio, optimal or not, will be found in the region bounded on the left by the hyperbola. The positively-sloped portion of the hyperbola is known as the *efficient frontier*, since the points there correspond to optimal portfolios. Portfolios with risk-return pairs that lie to the right of the efficient frontier are inefficient portfolios, since we could either increase the return while keeping the risk constant, or we could decrease the risk while keeping the return constant. See Figure 19.2.

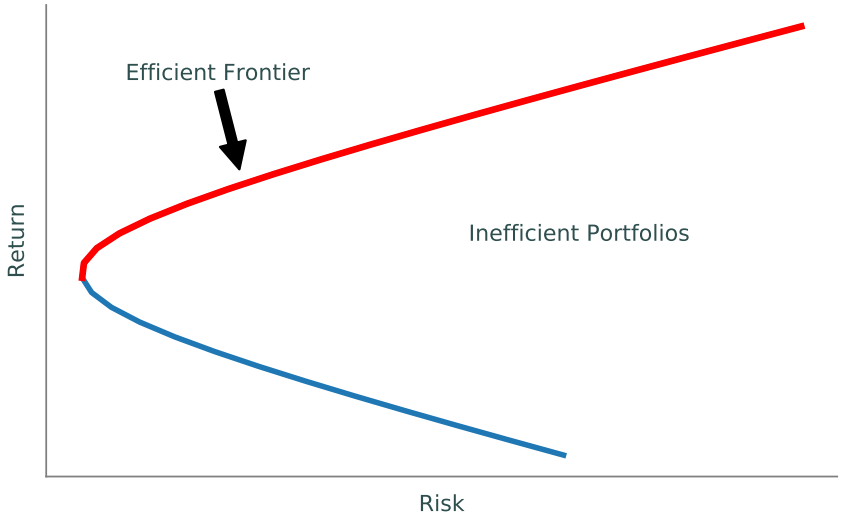

Figure 19.2: Efficient frontier on the risk-return plane.

One difficulty of this model is that the risk and return of each asset is in general unknown. After all, no one can predict the stock market with complete certainty. There are various ways of estimating these values given past stock prices, and we take a very straightforward approach. Suppose for each asset we have k previous return values of the asset. That is, for asset i, we have the data vector

$$y^i = [y_1^i, \ \ldots, \ y_k^i]^T.$$

We estimate the expected rate of return for asset i by simply taking the average of $y_1, \dots, y_k$, and we estimate the variance of asset i by taking the variance of the data. We can estimate the covariance matrix for all assets by taking the covariance matrix of the vectors $y^1, \dots, y^n$. In this way, we obtain estimated values for Q and each μ_i.

Problem 4. The text file `portfolio.txt` contains historical stock data for several assets (U.S. bonds, gold, S&P 500, etc). In particular, the first column gives the years corresponding to the data, and the remaining eight columns give the historical returns of eight assets over the course of these years. Use this data to estimate the covariance matrix Q as well as the expected rates of return μ_i for each asset. Assuming that we want to guarantee an expected return of $R = 1.13$ for our portfolio, find the optimal portfolio both with and without short selling.

Since the problem contains both equality and inequality constraints, use the QP solver in CVXOPT rather than your `qInteriorPoint()` function.

Hint: Use `numpy.cov()` to compute Q.