1 Max-Cut via SDP [Sivaraman, 35 points]

The goal of this problem is to illustrate the use of semidefinite programming for approximating NP-hard optimization problems. We will obtain a 0.87856 factor approximation to the max-cut problem.

**Max-Cut:** Given an undirected graph $G = (V, E)$, with edge weights $w : E \rightarrow Q^+$ (positive rationals), find a partition $(S, \bar{S})$ of $V$ so as to maximize the total weight of the edges in this cut, i.e., edges that have one endpoint in $S$ and one endpoint in $\bar{S}$.

The max-cut problem has applications in diverse fields like statistical physics, VLSI layout (the minimum area of a VLSI layout of a graph is at least the square of its maximum cut size), etc.

The max-cut problem is in general NP-hard. However, for certain classes of graphs, it is easy. For example, for a bipartite graph, the max-cut consists of all edges in the graph.

In the following, we will obtain an approximation to the max-cut problem for an arbitrary graph.

1.1 Integer strict QP for max-cut

1. [2 pts] Associate a variable $y_i$ with each vertex $v^i$. Restrict $y_i$ to take values $+1$ or $-1$. Express this restriction on the $y_i$’s using a quadratic constraint, and the constraint $y_i \in \mathbb{Z}$.

★ SOLUTION:

$$y_i^2 = 1$$
$$y_i \in \mathbb{Z}$$

As some of you observe, the second constraint is not needed. However, it was added to emphasize that when we do the relaxation later on, we drop the implicit integrality constraints.

■ COMMON MISTAKE : The constraint $y_i^2 \leq 1$ is not enough to ensure that $y_i$ takes the value $-1$ or $1$. It will also allow the value $0$.

Based on the values taken by the $y_i$’s, we can define a partition $(S, \bar{S})$ of vertices as follows: $S = \{v^i | y_i = 1\}$ and $\bar{S} = \{v^i | y_i = -1\}$.

2. [4 pts] Using the variables $y_i$, express the objective function for the max-cut problem as a strict quadratic function i.e. each monomial is of either degree 0 (i.e. a constant) or has degree 2 (i.e. $y_i y_j$).
★ SOLUTION: We observe that if two vertices \( i \) and \( j \) are in the same partition, then \( y_i y_j = 1 \). If they are in different partitions, \( y_i y_j = -1 \). Therefore, the objective function is given by

\[
\max_y \sum_{(i,j) \in E, i < j} w_{ij} \frac{1 - y_i y_j}{2}
\]

where we divide by 2 to account for the fact that if \( i \) and \( j \) are in different partitions, \( 1 - y_i y_j \) takes the value 2.

COMMON MISTAKE: If you do not use the constraint \( i < j \) while summing up, then you will count each edge twice, since both \( (i, j) \) and \( (j, i) \in E \). So you should either use the constraint \( i < j \), or divide by 4, instead of 2.

3. [1 pt] Combine the objective function and constraints to form a QP that solves the max-cut problem.

★ SOLUTION:

\[
\max_v \sum_{(i,j) \in E, i < j} w_{ij} \frac{1 - v_i \cdot v_j}{2}
\]

subject to

\[
\begin{align*}
v_i^2 &= 1 & & \forall i \in V \\
y_i &= 1 & & \forall i \in \mathbb{Z}
\end{align*}
\]

Notice that in this QP, both in the objective function and in the constraints, there are only constant or quadratic terms. Such a QP is called a strict QP. Moreover, since all variables are restricted to take only integral values, this is an integer strict QP.

1.2 Vector Program relaxation for max-cut

We will now relax the integer strict QP above to a vector program. A vector program is defined over \( m \) vector variables \( v_1, \ldots, v_m \in \mathbb{R}^n \). The objective function consists of a linear function of the inner products \( v_1 \cdot v_j \). The constraints consist of linear constraints on the inner products. Thus, a vector program is like an LP, with each variable in the LP being replaced by an inner product of two vectors. In this problem, \( m = n \) i.e. the number of vectors is the same as the dimension of each vector.

1. [2 pts] Associate a vector variable \( v_i \) with each integer variable \( y_i \). Obtain a vector program \( \mathcal{V} \) from the strict QP defined above by replacing each quadratic term by the inner product of the corresponding vector variables. We will no longer retain the integrality constraints on the variables \( y_i \). Instead, let each vector \( v_i \in \mathbb{R}^n \).

★ SOLUTION:

\[
\max_v \sum_{(i,j) \in E, i < j} w_{ij} \frac{1 - v_i \cdot v_j}{2}
\]

subject to

\[
\begin{align*}
v_i \cdot v_i &= 1 & & \forall i \in V \\
v_i &\in \mathbb{R}^n
\end{align*}
\]
2. [4 pts] Show that the vector program obtained above is a relaxation of the QP derived earlier. Hint: Consider any feasible solution \( y \) to the QP. Using the solution \( y \), define an appropriate set of vectors comprising of \( y_i \)'s and 0's, such that the newly defined vectors are feasible solutions for the vector program, and the value of the objective function for the vector program is the same as the value of the objective function for the QP.

\[ \text{SOLUTION:} \quad \text{We will show that for any feasible solution } \hat{y} \text{ of the QP defined in 4.1.3, we can construct a solution to the vector program above, that has the same objective function value.} \]

Define \( \hat{v}_i = (\hat{y}_i, 0 \ldots 0) \). Note that \( \forall i \in V, \hat{v}_i \cdot \hat{v}_i = \hat{y}_i^2 = 1 \) (since \( \hat{y} \) is a feasible solution). Thus, \( \hat{v}_i \) is a feasible solution for the vector program. Next, consider the objective function value for the vector program. By definition,

\[
\sum_{(i,j) \in E, i < j} w_{ij} \frac{1 - \hat{v}_i \cdot \hat{v}_j}{2} = \sum_{(i,j) \in E, i < j} w_{ij} \frac{1 - \hat{y}_i \hat{y}_j}{2}
\]

Thus, since for any feasible solution of the QP, we can construct a solution for the vector program with the same objective value, the vector program defined above is a relaxation of the QP.

1.3 Equivalence of vector programs and SDPs

We will now derive an equivalent SDP from the vector program defined above.

1. [2 pts] Associate a variable \( x_{ij} \) with each inner product \( \mathbf{v}_i \cdot \mathbf{v}_j \). Notice that now the objective function and constraints are linear in \( x_{ij} \). Additionally, require the matrix \( X \), with \( ij \)-th entry equal to \( x_{ij} \), to be positive semi-definite. Write down this SDP.

\[ \text{SOLUTION:} \quad \max \sum_{(i,j) \in E, i < j} w_{ij} \frac{1 - x_{ij}}{2} \]

subject to

\[
x_{ii} = 1 \quad \forall i \in V
\]

\[
x_{ij} \in \mathbb{R} \quad \forall i, j \in V
\]

\[
X \succeq 0
\]

2. [5 pts] Prove the equivalence of the vector program \( \mathcal{V} \) and SDP defined above by showing that corresponding to each feasible solution to \( \mathcal{V} \), there is a feasible solution to the SDP with the same objective value and vice versa. Hint: Make use of the fact that a matrix \( A \) can be decomposed as \( W^T W \) if and only if \( A \) is positive semidefinite.

Let \( a_1, \ldots, a_n \) be a feasible solution to \( \mathcal{V} \). Let \( W \) be the matrix whose columns are \( a_1, \ldots, a_n \). Using \( W \), define a matrix \( A \) that is a feasible solution to the SDP with the same objective value.

\[ \text{SOLUTION:} \quad \mathcal{V} \Rightarrow SDP \]

Let \( a_1, \ldots, a_n \) be a feasible solution to \( \mathcal{V} \). Therefore, \( a_i \cdot a_j = 1 \). Let \( W \) be the matrix whose columns are \( a_1, \ldots, a_n \). Define \( A = W^T W \). By definition, \( A \succeq 0 \) (i.e. \( A \) is positive semi-definite). Also, \( A_{ij} = a_i \cdot a_j = 1 \). Also,

\[
\sum_{(i,j) \in E, i < j} w_{ij} \frac{1 - A_{ij}}{2} = \sum_{(i,j) \in E, i < j} w_{ij} \frac{1 - a_i \cdot a_j}{2}
\]
Thus, $A$ is a feasible solution to the SDP, with the same objective value as that of $V$.

**SDP $\implies \mathcal{V}$**

Let $A$ be a feasible solution to the SDP. Therefore, $A_{ij} = 1$, and $A \succeq 0$. Since $A$ is positive semi-definite, there exists an $n \times n$ matrix $W$ such that $A = W^TW$. Let $a_1, \ldots, a_n$ be the columns of $W$. Then $a_i \cdot a_j = A_{ij} = 1$. Also,

\[
\sum_{(i,j) \in E \atop i < j} w_{ij} \frac{1 - a_i \cdot a_j}{2} = \sum_{(i,j) \in E \atop i < j} w_{ij} \frac{1 - A_{ij}}{2}
\]

Thus, $a_1, \ldots, a_n$ is a feasible solution to the vector program, with the same objective value as that of the SDP.

Therefore, the vector program and the SDP are equivalent.

### 1.4 Approximation factor for max-cut

We have now obtained an SDP relaxation for max-cut. As mentioned in class, it is possible to solve an SDP to accuracy $\epsilon$ in time polynomial in $n$ and $\frac{1}{\epsilon}$. So, let's assume we have an optimal solution to our SDP (and hence to our vector program $\mathcal{V}$).

1. [1 pt] Show that due to the form of our vector program, all the vectors $a_1, \ldots, a_n$ lie on the $n$-dimensional unit sphere centered at the origin.

   ★ **SOLUTION:** Any feasible solution $a_1, \ldots, a_n$ to the vector program satisfies the constraint $a_i \cdot a_i = 1$. Thus, all the vectors $a_1, \ldots, a_n$ lie on the $n$-dimensional unit sphere centered at the origin.

2. [2 pts] Let $\theta_{ij}$ denote the angle between the vectors $a_i$ and $a_j$. Using the fact that each vector $a_i$ is a unit vector, express the contribution of these vectors to $OPT_v$ in terms of $w_{ij}$ and $\theta_{ij}$. Show that the contribution is an increasing function of $\theta_{ij}$.

   ★ **SOLUTION:** The contribution of the vectors $a_i$ and $a_j$ to the objective function is $w_{ij} \frac{1 - a_i \cdot a_j}{2}$, since this is the only term in which both $a_i$ and $a_j$ appear. Since $a_i$ and $a_j$ are unit vectors, $a_i \cdot a_i = \cos \theta_{ij}$. Thus, the contribution of these vectors is

   \[
   w_{ij} \frac{1 - \cos \theta_{ij}}{2}
   \]

   $\theta_{ij}$ lies in the range $[0, \pi]$. In this range, $\cos \theta_{ij}$ is a decreasing function. Thus, $1 - \cos \theta_{ij}$, and hence the contribution of the vectors $a_i$ and $a_j$ to the objective function, is an increasing function of $\theta_{ij}$.

Suppose we have a bipartite graph. Then, all the vertices $v^i$ on one side of the graph will have their corresponding vector $a_i = u$, and all the vertices $v^j$ on the other side will have $a_j = -u$, where $u$ is a unit vector. Thus, all of the $\theta_{ij}$'s are either 0 or $\pi$. In this case, we will have an optimal solution to the original max-cut problem, with one cluster corresponding to $S$ and the other to $\bar{S}$.

For a general graph, the vectors $a_i$ obtained by solving $\mathcal{V}$ will not form two clusters. However, we would still like to partition all the vectors (and hence the vertices) into two sets. For this, we design a *rounding procedure* which perturbs all of the vectors $a_i$ so that they are grouped into two antipodal clusters, while losing as little of the objective value as possible.

Since the contribution of a pair of vertices to the objective function is an increasing function of the angle between them, we would like the vertices $v^i$ and $v^j$ to be separated (i.e. one is in $S$ and the other

---

1 We can obtain an optimal solution for our vector program using an eigen decomposition of the positive semi-definite matrix that is a solution to the SDP.
Figure 1: For a bipartite graph, all the vectors $a_i$ occur in one of two groups that are opposite to each other (left). For a general graph, there are many different unit vectors in the sphere (right).

is in $\bar{S}$) if $\theta_{ij}$ is large. One way to partition the vectors into the sets $S$ and $\bar{S}$ is by picking a vector $r$ that is distributed uniformly on the unit sphere in $n$-dimensions, and defining $S = \{v^i | a_i \cdot r \geq 0\}$. Intuitively, we take all the vectors on one half of the sphere, and define the set $S$ to contain exactly these vertices, as shown in Figure 2.

3. [2 pts] Compute the probability that the vertices $v^i$ and $v^j$ are separated by following the partitioning scheme illustrated above. Your answer should be in terms of $\theta_{ij}$ and $w_{ij}$.

★ SOLUTION: The vertices $v^i$ and $v^j$ will be separated if the hyperplane corresponding to the vector $r$ falls between the two vectors, so that one of the vectors falls in the shaded region (Figure 2), and the other vector does not fall in the shaded region. Since $r$ is chosen uniformly at random, this happens with probability

$$\frac{2\theta_{ij}}{2\pi} = \frac{\theta_{ij}}{\pi}$$

Therefore, our algorithm for obtaining an approximation to max-cut is as follows:

- Solve the vector program $V$ to obtain an optimal solution $a_1, \ldots, a_n$.
- Pick a vector $r$ that is distributed uniformly on the unit sphere in $n$-dimensions.
- Define $S = \{v^i | a_i \cdot r \geq 0\}$.
- All edges with one endpoint in $S$ and the other endpoint in $\bar{S}$ form the cut, and the sum of their weights $W$ is our approximation for the max-cut.
4. [5 pts] Using the definition that $\alpha = \frac{2}{\pi} \min_{0 \leq \theta \leq \pi} \frac{\theta}{1 - \cos \theta} > 0.87856$, and the answer to the above two parts, show that the expected value of the cut we obtain using this procedure $E[W]$ is greater than or equal to $\alpha OPT_v$. Thus, we have obtained a randomized algorithm that is an 0.87856 approximation to the max-cut problem.

★ SOLUTION: First, note that by definition,

\[
\alpha = \frac{2}{\pi} \min_{0 \leq \theta \leq \pi} \frac{\theta}{1 - \cos \theta} \\
\Rightarrow \alpha \leq \frac{2}{\pi} \frac{\theta}{1 - \cos \theta} \quad \forall \theta \in [0, \pi] \\
\Rightarrow \frac{\theta}{\pi} \geq \alpha \frac{1 - \cos \theta}{2}
\]
The expected value of the cut is equal to the weight of each edge, multiplied by the probability that the vertices joined by the edge are separated by the random vector $\mathbf{r}$. Thus,

$$E[W] = \sum_{(i,j) \in E, i < j} w_{ij} \Pr[\mathbf{a}_i \text{ and } \mathbf{a}_j \text{ are separated}]$$

$$= \sum_{(i,j) \in E, i < j} w_{ij} \frac{\theta_{ij}}{\pi}$$

$$\geq \sum_{(i,j) \in E, i < j} w_{ij} \frac{1 - \cos \theta_{ij}}{2}$$

$$= \alpha \sum_{(i,j) \in E, i < j} \frac{1}{2} w_{ij} (1 - \cos \theta_{ij})$$

$$= \alpha \text{OPT}_v$$

where we have used the fact that the optimal value of the vector program relaxation $\text{OPT}_v = \frac{1}{\pi} w_{ij}(1 - \cos \theta_{ij})$.

5. [5 pts] Illustrate a simple way to pick the vector $\mathbf{r}$ i.e. obtain a random vector on the unit sphere in $n$-dimensions. Make sure you include a proof of correctness for your construction.

**SOLUTION:** Here we illustrate two ways to obtain a random vector on the unit sphere in $n$-dimensions.

*Rejection Sampling:* Here, the idea is to sample a point uniformly at random from a hypercube centered at the origin with side length equal to two units, and reject the ones that fall outside the unit sphere. We can sample uniformly from the hypercube by picking $n$ numbers $x_1, \ldots, x_n$ uniformly between $[-1, 1]$. We then compute the norm $|\mathbf{x}|$ of the picked numbers as $|\mathbf{x}| = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}$. If $|\mathbf{x}| > 1$, the point lies outside the sphere, and we reject it and re-sample. If $|\mathbf{x}| \leq 1$, the point lies on or inside the unit sphere. In this case, we scale the point to obtain a point on the unit sphere i.e. we pick the point $y = \left(\frac{x_1}{|\mathbf{x}|}, \ldots, \frac{x_n}{|\mathbf{x}|}\right)$. Since we reject points that lie outside the unit sphere, the point that we pick is distributed uniformly at random in the volume inside the sphere. Also, since scaling is a spherically symmetric operation, we obtain a point on the sphere that is distributed uniformly at random.

*Sampling using a normal:* Pick $n$ numbers $x_1, \ldots, x_n$ independently from the normal distribution with zero mean and unit standard deviation. Compute $|\mathbf{x}|$ as above. Then, $y = \left(\frac{x_1}{|\mathbf{x}|}, \ldots, \frac{x_n}{|\mathbf{x}|}\right)$ is a random vector on the unit sphere. This can be seen as follows. The probability of picking the vector $(x_1, \ldots, x_n)$ is

$$Pr(x_1, \ldots, x_n) = \Pi_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-x_i^2}$$

$$= \frac{1}{(2\pi)^{n/2}} e^{-\frac{1}{2} \sum_{i=1}^{n} x_i^2}$$

$$= \frac{1}{(2\pi)^{n/2}} e^{-\frac{1}{2} |\mathbf{x}|^2}$$

Since the probability depends only on the distance of the point from the origin, the distribution of the vector $(x_1, \ldots, x_n)$ is spherically symmetric. Thus, upon normalization, we obtain a random vector on the unit sphere.

6. [Extra credit, 1 pt] Show that $\frac{2}{\pi} \min_{0 \leq \theta \leq \pi} \frac{\theta}{1 - \cos \theta} > 0.87856$. 

7.
SOLUTION: To compute the minima of the function \( \frac{\theta}{1 - \cos \theta} \), we compute its derivative and set it to 0.

\[
\frac{d}{d\theta} \left( \frac{\theta}{1 - \cos \theta} \right) = \frac{1}{1 - \cos \theta} - \frac{\theta \sin \theta}{(1 - \cos \theta)^2}
\]

Setting the derivative to 0 yields \( \cos \theta + \theta \sin \theta = 1 \), since we know that the minimum does not occur at \( \theta = 0 \). We solve this equation numerically using Matlab to obtain the minimum, which occurs at \( \theta = 2.33 \).

Evaluating the function \( 2\pi \frac{\theta}{1 - \cos \theta} \) at this point yields 0.878567, which is greater than 0.87856.

7. [Extra credit, 3 pts] Recall that we obtained only an \( \epsilon \)-optimal solution to the SDP, instead of the true optimal. Show that this does not affect the approximation factor.

SOLUTION: Consider an \( \epsilon \)-optimal solution to the SDP. Then, using the above proof, we obtain

\[
E[W] \geq \alpha (OPT_v - \epsilon) = OPT_v \left( \alpha - \frac{\epsilon \alpha}{OPT_v} \right)
\]

Now we know from above that \( \alpha \) is strictly greater than 0.87856. Thus, we choose a value of \( \epsilon \) such that \( \alpha - \frac{\epsilon \alpha}{OPT_v} \geq 0.87856 \), thus preserving the claimed approximation factor. Note that we do not preserve the true approximation factor, but since we claimed a slightly worse one so that we could write it with finitely many digits, we can pick a small enough \( \epsilon \) that maintains the 0.87856 factor.

2 Ellipsoid Algorithm [Sivaraman, 15 points]

In this problem, you’ll run through a few steps of the ellipsoid algorithm to get a better idea of how it works. I recommend using Matlab, Mathematica, or some other math environment to make the computations easier, though this doesn’t really involve programming. Make sure to write out numbers to at least 4 significant digits. (Of course, you’d have to use a lot more digits in a real implementation.)

We want to solve the linear feasibility problem:

\[
\text{find } x \text{ such that } Ax \geq b \text{ where } A = \begin{pmatrix} 1 & 0 & 8 \\ -3 & 1 & 2 \\ 8 & -1 & 4 \\ 1 & -3 & 2 \end{pmatrix} \quad \text{and } b = \begin{pmatrix} 0 \\ 3 \\ -2 \\ 2 \end{pmatrix}
\]

These constraints define a polyhedron, and we’d like to find any feasible point (i.e. a point inside this polyhedron). We’ll use the ellipsoid method, as described in Section 8.3 of Bertsimas & Tsitsiklis (and reproduced here). There are a number of equations in the algorithm here; they are derived via the intuitions given in lecture, and the detailed derivations are given in the textbook.

Note: We’ll define \( E_t \), the ellipse on iteration \( t \), as \( E_t = E(x_t, D_t) \) where its center is \( x_t \) and its matrix is \( D_t \); that is, \( E_t = \{ x : (x - x_t)^T D_t^{-1} (x - x_t) \leq 1 \} \). Note that this matrix \( D_t \) is the covariance matrix of the ellipse; it is the inverse of the precision matrix of the ellipse. In the class lectures, ellipses were defined using the precision matrix. We’ll assume that \( A \) and \( b \) are integer-valued and have entries whose absolute value is upper-bounded by an integer \( U \).

We write the \( i \)-th row of \( A \) as column vector \( a_i \). For example, \( a_1 = \begin{pmatrix} 1 \\ 0 \\ 8 \end{pmatrix} \)

Input

- \( m \)-by-\( n \) matrix \( A \), \( m \)-vector \( b \) defining polyhedron \( P = \{ x \in \mathbb{R}^n \mid a_i^T x \geq b_i, i = 1, \ldots, m \} \)
• Number \( \nu \) such that either \( \mathbf{P} \) is empty or \( Vol(\mathbf{P}) > \nu \)
  \[ \longrightarrow \quad We \ can \ use \ \nu = n^{-n} (nU)^{-n^2(n+1)} \]
  \[ \longrightarrow \quad In \ the \ real \ ellipsoid \ algorithm, \ we \ would \ need \ to \ perturb \ \mathbf{b} \ so \ that \ we \ were \ guaranteed \ a \ volume \ of \ at \ least \ \nu; \ here \ you \ may \ assume \ that \ the \ feasible \ region \ already \ has \ volume \ at \ least \ \nu \ without \ any \ perturbation. \]

• Ball \( E_0 = E(\mathbf{x}_0, \mathbf{D}_0) \) centered at \( \mathbf{x}_0 \) with matrix \( \mathbf{D}_0 = r^2 \mathbf{I} \)
  \[ \longrightarrow \quad We \ can \ use \ \mathbf{x}_0 = 0 \ and \ r^2 = n(nU)^2 \]
  \[ \longrightarrow \quad The \ volume \ of \ this \ ball \ is \ at \ most \ V = (2n)^n (nU)^n \]
  \[ \longrightarrow \quad We \ know \ that \ r \ is \ big \ enough \ to \ ensure \ that \ the \ feasible \ region \ \mathbf{P} \ lies \ inside \ of \ the \ ball \ E_0 \]

**Output**

• Feasible point \( \mathbf{x}^* \in \mathbf{P} \) if \( \mathbf{P} \) is non-empty, or show that \( \mathbf{P} \) is empty

**Algorithm**

Initialize: \( t^* = \lceil 2(n+1) \log(V/\nu) \rceil; \ t = 0 \)

*This bound \( t^* \) on the max number of iterations is chosen such that the volume of \( E_t \) is guaranteed to be less than \( \nu \). Since we know \( \mathbf{P} \) is either empty or has volume greater than \( \nu \), we know that after \( t^* \) iterations, \( \mathbf{P} \) must be empty.*

Iterate:

• If \( t = t^* \), stop; \( \mathbf{P} \) is empty.
• If \( \mathbf{x}_t \notin \mathbf{P} \), stop; \( \mathbf{P} \) is non-empty.
• If \( \mathbf{x}_t \notin \mathbf{P} \), find a violated constraint; i.e. find a constraint \( i \) such that \( \mathbf{a}_i^T \mathbf{x}_t < b_i \)
  \( This \ violated \ constraint \ would \ typically \ be \ found \ via \ some \ separation \ oracle; \ it \ gives \ us \ a \ separating \ hyperplane. \)
• Define halfspace \( \mathcal{H}_t = \{ \mathbf{x} \in \mathbb{R}^n | \mathbf{a}_i^T \mathbf{x} \geq \mathbf{a}_i^T \mathbf{x}_t \} \). Find an ellipsoid \( E_{t+1} = E(\mathbf{x}_{t+1}, \mathbf{D}_{t+1}) \) containing \( E_t \cap \mathcal{H}_t \). To do this, we can use:
  \[
  \mathbf{x}_{t+1} = \mathbf{x}_t + \frac{1}{n+1} \frac{\mathbf{D}_t \mathbf{a}_i}{\sqrt{\mathbf{a}_i^T \mathbf{D}_t \mathbf{a}_i}} \\
  \mathbf{D}_{t+1} = \frac{n^2}{n^2 - 1} \left( \mathbf{D}_t - \frac{2}{n+1} \frac{\mathbf{D}_t \mathbf{a}_i \mathbf{a}_i^T \mathbf{D}_t}{\mathbf{a}_i^T \mathbf{D}_t \mathbf{a}_i} \right)
  \]
• \( t \leftarrow t + 1 \)

Use the above definitions for \( \mathbf{A} \) and \( \mathbf{b} \) (in Equation 1) for the following questions:

1. [5 pts] Initialize the necessary values to run this algorithm. Show your values for \( \mathbf{x}_0, \mathbf{D}_0 \), the estimated volume \( V \) of the ellipse \( E_0 \), and the minimum volume \( \nu \) of \( \mathbf{P} \). Also, compute the actual volume of the ellipse. (The volume of an ellipse with radii \( a, b, c \) is \( \frac{4}{3} \pi abc \). You can find the radii of \( E_t = E(\mathbf{x}_t, \mathbf{D}_t) \) by finding the eigenvalues of \( \mathbf{D}_t \); the square roots of the eigenvalues are equal to the radii \( a, b, c \). Note: you can use the ‘eig’ function in Matlab to compute the eigenvalues of a matrix. Notice that \( E \) is defined using \( \mathbf{D}^{-1} \) whereas we are using the eigenvalues of \( \mathbf{D} \) here.)

**SOLUTION:**

\[
\mathbf{x}_0 = (0, 0, 0)
\]

\[
\mathbf{D}_0 = \begin{pmatrix}
5.7331 & 0 & 0 \\
0 & 5.7331 & 0 \\
0 & 0 & 5.7331
\end{pmatrix} \cdot 10^8
\]

estimated volume of \( E_0 \): \( V = 5.7063 \cdot 10^{14} \)

minimum volume of \( \mathbf{P} \): \( \nu = 7.6038 \cdot 10^{-52} \)

actual volume of \( E_0 \): \( 5.7501 \cdot 10^{13} \)
2. [4 pts] Run one step of the ellipsoid algorithm; you may use any violated constraint. State which constraint you choose. Write down the new ellipse $E_1$ in terms of $x_1$ and $D_1$. Compute its actual volume; how does it compare with the volume of $E_0$?

★ SOLUTION: The second and fourth constraints are violated. If you chose the second constraint, then you should have computed

\[
x_1 = (-4799.5, 1599.8, 3199.6) \\
D_1 = \begin{pmatrix} 4.3766 & 0.6910 & 1.3821 \\
0.6910 & 6.2194 & -0.4607 \\
1.3821 & -0.4607 & 5.5283 \end{pmatrix} \cdot 10^8
\]

actual volume of $E_1 = 4.8516 \cdot 10^{13}$

If, instead, you chose the fourth constraint, you should get

\[
x_1 = (1599.8, -4799.5, 3199.6) \\
D_1 = \begin{pmatrix} 6.2194 & 0.6910 & -0.4607 \\
0.6910 & 4.3766 & 1.3821 \\
-0.4607 & 1.3821 & 5.5283 \end{pmatrix} \cdot 10^8
\]

actual volume of $E_1 = 4.8516 \cdot 10^{13}$

These volumes are about 84% of the original ellipsoid’s volume.

3. [4 pts] Run another iteration of the algorithm (again using any violated constraint), and report the constraint you chose, $x_2$, $D_2$, and the actual volume of the ellipse $E_2$.

★ SOLUTION: If you chose the second constraint in the previous iteration, then in this iteration, you should find that the third and fourth constraints are violated. If you use the third constraint in this iteration, then you should get

\[
x_2 = (-133.3, 1303.1, 7137.5) \\
D_2 = \begin{pmatrix} 2.9641 & 0.9020 & -0.0989 \\
0.9020 & 6.9889 & -0.4131 \\
-0.0989 & -0.4131 & 4.8238 \end{pmatrix} \cdot 10^8
\]

actual volume of $E_2 = 4.0935 \cdot 10^{13}$

If, instead, you use the fourth constraint, then you should get

\[
x_2 = (-3459.3, -3395.1, 6854.5) \\
D_2 = \begin{pmatrix} 4.7620 & 1.3799 & 1.1140 \\
1.3799 & 4.7514 & 1.1247 \\
1.1140 & 1.1247 & 5.0172 \end{pmatrix} \cdot 10^8
\]

actual volume of $E_2 = 4.0935 \cdot 10^{13}$
If you chose the fourth constraint in the previous iteration, then in this iteration, you should find that only the second constraint is violated. Using the second constraint in this iteration, you should get

\[
x_2 = (-3395.1, -3459.3, 6854.5) \\
D_2 = \begin{pmatrix} 4.7514 & 1.3799 & 1.1247 \\ 1.3799 & 4.7620 & 1.1140 \\ 1.1247 & 1.1140 & 5.0172 \end{pmatrix} \times 10^8
\]

actual volume of \( E_2 \) = \( 4.0935 \times 10^{13} \)

These volumes are about 71% of the original ellipsoid’s volume.

4. [2 pts] Run another iteration of the algorithm; you should find that no constraint is violated. Give the feasible point in \( P \) you have computed.

★ SOLUTION: If you used constraints 2 and then 3, then \( x = (-133.3, 1303.1, 7137.5) \).

If you used constraints 2 and then 4, then \( x = (-3459.3, -3395.1, 6854.5) \).

If you used constraints 4 and then 2, then \( x = (-3395.1, -3459.3, 6854.5) \).

3 Portfolio Optimization [Yi, 50 points]

Carlos has said that one goal of taking this class is to conquer the world. In this question, we will first learn to conquer the financial world! Indeed, becoming a billionaire is the first step for world domination. Also, students who get full credits of this question can apply the chief scientist position at the C&G (Carlos&Geoff) Investment Group at careers@bankruptcy.com.

In this question, we’ll look at the problem of choosing a good set of assets (e.g., stocks, bonds, currencies) based on information about their returns. The information usually comes from historical records of asset returns as well as domain knowledge. We’ll phrase it using the classical Markowitz portfolio optimization problem framework (described on page 155 of Boyd and Vandenberghe), and study how we can improve the classical portfolio optimization to conquer the world.


CVX: For this problem, you’ll need to solve more general convex programs than the LPs and QPs which Matlab can handle. I recommend downloading and installing CVX, a Matlab package for convex optimization. The package is created by the author of our textbook. You can find the files, installation instructions, and user’s guide here: http://www.stanford.edu/~boyd/cvx/

After you install CVX, you might want to save your Matlab search path so that you can immediately use CVX whenever you run Matlab. To do this, after you run cvx_setup (from the cvx root directory, as specified in the CVX installation directions), go to the Matlab menu. In the menu, click File — Set Path, and then click Save in the dialog box to save CVX to your search path permanently.

Submit the code you use to run the solver and to generate all results and plots in this question. Please print your plots and attach in your homework submission.

3.1 Choosing a portfolio

There are \( n \) assets, say, stocks. We can buy real-valued amounts \( x_i \) of each stock \( i \) at the beginning of some time period, and we sell all of our holdings at the end of the time period. The stocks may lose or gain value; let \( p_i \) be the relative price change of stock \( i \) during the time period: if stock \( i \) changes price from \( u \) to \( v \), then \( p_i = (v - u)/u \). For example, if we buy \( x_i = $20 \) of stock \( i \) and its value doubles over the time period \( (v = 2u) \), then \( p_i = 1 \) (i.e., 100%) and our actual return from stock \( i \) is \( p_i x_i = 20 \) (and \( p_i \) is the return rate). Our return from all our investments is \( p^T x \). In this problem, we’ll assume \( x \geq 0 \) and \( 1^T x = B \) where \( B \) is
our budget. Usually we assume a unit budget $B = 1$, and in this case, $x_i$ represents the proportion of our investment on stock $i$.

To summarize:

- $n$ assets (e.g., stocks)
- $x_i = \text{the amount of investment on stock } i$
- $p_i = \text{the relative price change (i.e., return rate) of stock } i$
- $p^T x = \text{the return of the portfolio } x$
- Assume: $x \geq 0$ and $1^T x = 1$

Since $p$ is the return rates of $n$ stocks during our holding period in the future, it is an unknown vector. We assume $p$ as a Gaussian random vector with mean $\bar{p}$ and covariance $\Sigma$, where $\bar{p}$ and $\Sigma$ might be estimated from historical data or from expert knowledge. As a result, the return $p^T x$ of a portfolio $x$ is a linear combination of Gaussian variables in $p$. Therefore $p^T x$ is also a Gaussian variable with mean $\bar{p}^T x$ and variance $x^T \Sigma x$. Note that $\bar{p}$ and $\Sigma$ are our predictions about the mean and covariance of asset returns (in the future).

How can we choose $x$ to maximize our expected return while minimizing the “risk” (i.e., variance of the return)? We want to solve the following QP optimization problem:

$$\max_x \bar{p}^T x - \eta x^T \Sigma x \quad \text{s.t.} \quad 1^T x = 1, \quad x \geq 0$$

Note that the parameter $\eta$ determines the tradeoff between the two objectives: maximizing the expected return $\bar{p}^T x$ and minimizing the risk $x^T \Sigma x$.

Download the data.zip file from the homework website, and look at the pbar.txt and sigma.txt files in the zip archive; you will use these for $\bar{p}$ and $\Sigma$ in this section. Both files can be loaded into Matlab via the load function; the first has the vector $\bar{p}$, and the second has the matrix $\Sigma$.

1. [10 pts] Use Matlab and CVX to solve the above optimization problem for varying values of the tradeoff parameter $\eta$. Use at least 30 values of $\eta$, with the minimum being .01 and the maximum being 1000; use a log scale to choose the values of $\eta$. Make a plot with the standard deviation of the return (square root of $x^T \Sigma x$) on the x-axis and with the mean return ($\bar{p}^T x$) on the y-axis. Label the two endpoints of the curve you plotted with the values of $\eta$.

   ★ SOLUTION: See Figure 3, as well as the solution code available on the homework website. Plots with nice color are due to Bin Zhao, thanks Bin!

2. [5 pts] Using the portfolios $x$ you just calculated, make a plot of the setting of $\eta$ (x-axis) vs. allocation for each stock (y-axis), with $\eta$ plotted on a log scale (use the Matlab function semilogx when you want the x-axis to be log scale but y-axis to be normal scale). In other words, this plot should show how, as you require smaller standard deviation (by increasing $\eta$), your optimization will choose different sets of stocks. To see an example of such a plot, look at page 187 (Figure 4.12, second subfigure) in Boyd and Vandenberghe (but note the book’s plot has the standard deviation on the x-axis, rather than $\eta$ on the x-axis). In your plot, clearly indicate which regions of the plot correspond to which stocks.

   ★ SOLUTION: See Figure 4, as well as the solution code available on the homework website.
3.2 Short positions

In this part, we introduce short positions, or short selling in portfolio. Suppose we now allow \( x_i < 0 \) in the QP optimization \((2)\). If you choose a portfolio with \( x_i < 0 \), you will sell stock \( i \) without really having it (e.g., via borrowing this stock from a broker), and then you will buy this stock at the end of the holding period (in order to return to the broker). In this sense, by a short position on \( x_i \), you are betting that the stock \( i \) will lose value during your hold period. At the end of the period, you make a profit of \( p_i x_i \) (since \( x_i < 0 \) and you bet \( p_i < 0 \)). The profit \( p_i x_i \) has the same form as in the case without short positions, although the actual meaning differs. To allow short positions, we will introduce new variables \( x_l, x_s \) (long position, short position), and replace the constraints in \((2)\) as the following:

\[
\begin{align*}
x_l &\geq 0, \quad x_s \geq 0, \quad x = x_l - x_s, \quad 1^T x_l = 1, \quad 1^T x_s \leq \gamma 1^T x_l
\end{align*}
\]

The last constraint limits our short position to be at most a fraction \( \gamma \) (where \( 0 \leq \gamma \leq 1 \)) of our long position. We will fix \( \gamma \) as 0.5 in this homework wherever we allow short selling (note: being able to control short selling by \( \gamma \) is the reason why we include two new vectors \( x_l \) and \( x_s \) rather than just simply removing the nonnegative constraints on \( x \)). We also require that the sum of \( x_l \) (rather than \( x \)) is one, since now \( x_l \) is the real money we need to spend for buying at the beginning of the holding period.

Does allowing short positions improve our ability to trade risk against expected return? From the optimization point of view, allowing short positions relaxes the non-negative constraints on \( x \), and thus better objective values can be achieved. To see this, we’ll modify the stocks we used in the previous section and compare the optimal portfolios with and without short positions.

1. [5 pts] Change your convex program from 3.1 to allow for short positions. Find the \texttt{pbar\_short.txt} and \texttt{sigma\_short.txt} files in the \texttt{data.zip} file on the homework website; you will use these for \( \bar{p} \) and \( \Sigma \) in this question. The \( \bar{p} \) and \( \Sigma \) in this question are the same as in 3.1 but with a new stock. This new stock has an expected return \( \bar{p}_5 = 0 \). Use both the convex program from 3.1 and the modified
convex program (with short selling) to find the optimal portfolio for different choices of $\eta$. Make plots of standard deviation of the return vs. mean return for different $\eta$, just as in part 1 of 3.1. Use the same values of $\eta$, and set $\gamma = .5$. Include the plots for both methods (without short positions and with short positions) in the same figure.

\begin{center}
\textbf{SOLUTION:} See Figure 5, as well as the solution code available on the homework website.
\end{center}

2. [5 pts] Look at the portfolios you obtain from optimization; portfolios without short positions should ignore this new stock (so it gives the same portfolios as in 3.1)\footnote{Use this as a sanity check to make sure your code is correct so far.}. Look at the input covariance matrix $\Sigma$ you used in the optimization, especially the covariance between the new stock and other stocks. Then look at the portfolios you obtained (with different $\eta$), especially the position $x_5$ for the new stock. Explain \textbf{in 1-3 sentences AT MOST} why the optimization with short positions benefits from having this new zero-expected-return stock; why can the resulting portfolio achieve the same expected portfolio return with lower variance?

\begin{center}
\textbf{SOLUTION:} Returns of the new stock has zero mean, but are positively corrected with stock 1, 2 and 3. As a result, although short positions on new stock will not change the expected return of the portfolio, they will reduce the variance of the portfolio by canceling out the variance from the long positions (positive weights) of stock 1, 2, and 3.
\end{center}
3.3 Short positions or not? Real foreign exchanges data

So far we have been playing with simulated data. More importantly, the portfolio performance we obtained (e.g., mean $\bar{p}^T x$ and variance $x^T \Sigma x$) is not real performance in a future holding period. The performance is what we *expect* to happen if the future asset returns really follow a distribution described by $\bar{p}$ and $\Sigma$. This can rarely be true. Also, what really matters is the performance of our portfolio $x$ in the real future period.

Find the file `foreign_exchange.mat` in `data.zip`. The is an international exchange rates data set, which includes the daily return rates (w.r.t. US dollar) of 12 different currencies: Australian Dollar, Belgian Franc, Canadian Dollar, French Franc, German Mark, Japanese Yen, Dutch Guilder, New Zealand Dollar, Spanish Peseta, Swedish Krone, Swiss Franc, and UK Pound. The dataset contains a period of about 3 years (approximately from 1993 to 1996).

The file contains two matrices, a $1017 \times 12$ matrix “past_returns”, and a $90 \times 12$ matrix “future_returns”. Our goal is to use the historical information in the past 1017 days to construct a portfolio, hold the portfolio during the future 90 days, and see how this portfolio performs in the holding period.

1. [10 pts] Call the Matlab function “mean” and “cov” to compute the sample mean (an $12 \times 1$ vector) and sample covariance (an $12 \times 12$ matrix) of returns using the historical returns (the matrix “past_returns”). Use them as our $\bar{p}$ and $\Sigma$ to construct optimal portfolio $x$ with and without short positions. Choose different $\eta$ as in previous questions, and for short selling portfolios, fix $\gamma = 0.5$. For each portfolio $x$ you obtained (using a specific $\eta$), call the given function `calculate_returns.m` by “calculate_returns(x, future_returns)”, where $x$ should be an $12 \times 1$ vector you obtained, and future_returns is the matrix in the data file. The function will calculate the cumulative total return of the portfolio in the holding period. Now plot two curves in the same figure, for portfolios without short selling and with short selling, respectively. Each curve has the choice of $\eta$ as the x-axis (again, in log scale using “semilogx”), and the cumulative total return of the resulting portfolio as the y-axis.
**SOLUTION:** See Figure 6, as well as the solution code available on the homework website.

![Figure 6: Problem 3.3](image-url)

2. [5 pts] For each curve in the figure, locate (approximately) the \( \eta \) in the x-axis that leads to the best cumulative total return in the y-axis, and consider this point (in the y-axis) as our best performance with (or without) short positions. Is the best portfolio with short selling still better than the best portfolio without short selling? Note that the short selling portfolio, which has relaxed constraints, always has a better objective value in the QP optimization (than the portfolio without short positions). But does this advantage in the optimization problem reflect the true performance of the portfolio in the future period (as shown in your figure)? Why? Explain in at most two sentences.

**SOLUTION:** We see that the future performance of the portfolio with short positions is actually worse than the portfolio without short positions, although the portfolio with short positions solves a relaxed QP and obtains better objective value. This is because the sample mean and sample covariance of the returns are estimated from the past, and the portfolio with short positions just overfits the history.

3.4 \( \ell_1 \)-regularization on the sample covariance

Recall in the previous question we calculated the sample mean and sample covariance of historical returns, and used them directly as our \( \bar{\mu} \) and \( \Sigma \) in portfolio optimization. Now we try to improve our estimate \( \Sigma \) a little bit. Denote the sample covariance as \( S \). Now we will not directly set \( \Sigma = S \). Instead, we want to estimate \( \Sigma \) smartly: we add an \( \ell_1 \) penalty to the inverse of our estimate, i.e., \( \Sigma^{-1} \) (detailed later). The reason for such a penalty on the inverse covariance is as follows: each entry \((i, j)\) in an inverse Gaussian covariance \( \Sigma^{-1} \) represents the conditional dependence of variables \( i \) and \( j \). When the entry is zero, we know variables \( i \) and \( j \) are conditionally independent given other variables in the system. There is an active research
area in statistics and machine learning called covariance selection, which aims to find a better (regularized) estimation of the (Gaussian) covariance matrix using limited observations. Adding an ℓ1 norm penalty on the inverse covariance is very popular.

We estimate our $12 \times 12$ covariance $\Sigma$ as follows. Use $\Theta$ to denote $\Sigma^{-1}$. Given the $12 \times 12$ sample covariance $S$ (which you computed using Matlab function `cov` on the “past_returns” matrix), the ℓ1 regularized estimate of $\Theta = \Sigma^{-1}$ is the following convex optimization problem:

$$\max_{\Theta} \quad \log \det \Theta - \text{trace}(S\Theta) - \lambda \sum_{i \neq j} |\Theta_{ij}|$$  \quad (3)

s.t. $\Theta$ is positive semi-definite

Note that $\log \det \Theta$ is the natural log of the determinant of $\Theta$, $\text{trace}(S\Theta)$ is the trace of the product of $S$ and $\Theta$, $\lambda$ is a regularization parameter fixed as $10^{-5}$ in this question, and $\sum_{i \neq j} |\Theta_{ij}|$ is the ℓ1 penalty term which penalizes the sum of absolute values of all off-diagonal entries of $\Theta$.

You might be mad about this crazy optimization, but actually this problem can be solved using the CVX package with a few lines of code. I provide an example code `example_code_CVX.m` for a similar optimization problem. Note that the example code is not exactly solving eq. (3). You need to change the code (hint: we do not want to penalize the diagonal entries of $\Theta$ in (3)).

Now we estimate $\Theta$ using eq. (3) based on sample covariance $S$ (and $\lambda$ fixed as $10^{-5}$). Then, we just set $\Sigma = \Theta^{-1}$.

1. [10 pts] In this question, we only consider portfolios without short positions. First repeat your experiments in 3.3 (only the portfolios without short positions), i.e., using the sample covariance $S$ as the $\Sigma$ in portfolio optimization. Then, use the advanced technique we just introduced to estimate a new $\Sigma$ from sample covariance. Repeat the experiments using this new $\Sigma$. Note that $\bar{p}$ is always the sample mean. Again, plot two curves in the same figure, for portfolios constructed using $\Sigma = S$ and portfolios constructed using the new $\Sigma$, respectively. Each curve has the choice of $\eta$ as the x-axis (again, in log scale using “semilogx”), and the resulting total portfolio return (returned by calling `calculate_returns.m`) as the y-axis. Which curve achieves the best portfolio return (note: choosing the optimal $\eta$ in x-axis for each curve) in the y-axis?

★ SOLUTION: See Figure 7, as well as the solution code available on the homework website. We can observe that the portfolio using regularized covariance performs better than the portfolio using the sample covariance. In other words, regularized estimation from the history offers a better prediction for the dynamics of future returns.
Figure 7: Problem 3.4