Convex-concave functions and saddle-points. We say the function \( f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R} \) is convex-concave if \( f(x, z) \) is a concave function of \( z \), for each fixed \( x \), and a convex function of \( x \), for each fixed \( z \). We also require its domain to have the product form \( \text{dom} \ f = A \times B \), where \( A \subseteq \mathbb{R}^n \) and \( B \subseteq \mathbb{R}^m \) are convex.

(a) Give a second-order condition for a twice differentiable function \( f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R} \) to be convex-concave, in terms of its Hessian \( \nabla^2 f(x, z) \).

(b) Suppose that \( f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R} \) is convex-concave and differentiable, with \( \nabla f(\tilde{x}, \tilde{z}) = 0 \). Show that the saddle-point property holds: for all \( x, z \), we have

\[
f(\tilde{x}, z) \leq f(\tilde{x}, \tilde{z}) \leq f(x, \tilde{z}).
\]

Show that this implies that \( f \) satisfies the strong max-min property:

\[
\sup_z \inf_x f(x, z) = \inf_x \sup_z f(x, z)
\]

(and their common value is \( f(\tilde{x}, \tilde{z}) \)).

**Solution.**

(a) The condition follows directly from the second-order conditions for convexity and concavity: it is

\[
\nabla^2_{xx} f(x, z) \succeq 0, \quad \nabla^2_{zz} f(x, z) \preceq 0,
\]

for all \( x, z \). In terms of \( \nabla^2 f \), this means that its 1,1 block is positive semidefinite, and its 2,2 block is negative semidefinite.

(b) Let us fix \( \tilde{z} \). Since \( \nabla_x f(\tilde{x}, \tilde{z}) = 0 \) and \( f(x, \tilde{z}) \) is convex in \( x \), we conclude that \( \tilde{x} \) minimizes \( f(\tilde{x}, \tilde{z}) \) over \( x \), i.e., for all \( z \), we have

\[
f(\tilde{x}, z) \leq f(\tilde{x}, \tilde{z}).
\]

This is one of the inequalities in the saddle-point condition. We can argue in the same way about \( \tilde{z} \). Fix \( \tilde{x} \), and note that \( \nabla_z f(\tilde{x}, \tilde{z}) = 0 \), together with concavity of this function in \( z \), means that \( \tilde{z} \) maximizes the function, i.e., for any \( x \) we have

\[
f(\tilde{x}, \tilde{z}) \geq f(\tilde{x}, z).
\]

A3.31 Stochastic optimization via Monte Carlo sampling. In (convex) stochastic optimization, the goal is to minimize a cost function of the form \( F(x) = \mathbb{E} f(x, \omega) \), where \( \omega \) is a random variable on \( \Omega \), and \( f : \mathbb{R}^n \times \Omega \to \mathbb{R} \) is convex in its first argument for each \( \omega \in \Omega \). (For simplicity we consider the unconstrained problem; it is not hard
to include constraints.) Evidently $F$ is convex. Let $p^*$ denote the optimal value, \( i.e., p^* = \inf_x F(x) \) (which we assume is finite).

In a few very simple cases we can work out what $F$ is analytically, but in general this is not possible. Moreover in many applications, we do not know the distribution of $\omega$; we only have access to an oracle that can generate independent samples from the distribution.

A standard method for approximately solving the stochastic optimization problem is based on Monte Carlo sampling. We first generate $N$ independent samples, $\omega_1, \ldots, \omega_N$, and form the empirical expectation

$$
\hat{F}(x) = \frac{1}{N} \sum_{i=1}^{N} f(x, \omega_i).
$$

This is a random function, since it depends on the particular samples drawn. For each $x$, we have $E \hat{F}(x) = F(x)$, and also $E(\hat{F}(x) - F(x))^2 \propto 1/N$. Roughly speaking, for $N$ large enough, $\hat{F}(x) \approx F(x)$.

To (approximately) minimize $F$, we instead minimize $\hat{F}(x)$. The minimizer, $\hat{x}^*$, and the optimal value $\hat{p}^* = \hat{F}(\hat{x}^*)$, are also random variables. The hope is that for $N$ large enough, we have $\hat{p}^* \approx p^*$. (In practice, stochastic optimization via Monte Carlo sampling works very well, even when $N$ is not that big.)

One way to check the result of Monte Carlo sampling is to carry it out multiple times. We repeatedly generate different batches of samples, and for each batch, we find $\hat{x}^*$ and $\hat{p}^*$. If the values of $\hat{p}^*$ are near each other, it’s reasonable to believe that we have (approximately) minimized $F$. If they are not, it means our value of $N$ is too small.

Show that $E \hat{p}^* \leq p^*$.

This inequality implies that if we repeatedly use Monte Carlo sampling and the values of $\hat{p}^*$ that we get are all very close, then they are (likely) close to $p^*$.

**Hint.** Show that for any function $G : \mathbb{R}^n \times \Omega \to \mathbb{R}$ (convex or not in its first argument), and any random variable $\omega$ on $\Omega$, we have

$$
\inf_z E G(x, \omega) \geq E \inf_x G(x, \omega).
$$

**Solution.** Let’s show the hint. For each $\omega$ and any $z \in \mathbb{R}^n$, we have $G(z, \omega) \geq \inf_x G(x, \omega)$. Since expectation is monotone, we can take expectection and get

$$
E G(z, \omega) \geq E \inf_x G(x, \omega).
$$

This holds for all $z \in \mathbb{R}^n$, so we conclude that

$$
\inf_z E G(z, \omega) \geq E \inf_x G(x, \omega),
$$

which is what we wanted to show.
Then we have

\[ p^* = \inf_x F(x) = \inf_x \mathbf{E} \hat{F}(x) \]

\[ = \inf_x \mathbf{E} \frac{1}{N} \sum_{i=1}^{N} f(x, \omega_i) \]

\[ \geq \mathbf{E} \inf_x \frac{1}{N} \sum_{i=1}^{N} f(x, \omega_i) = \mathbf{E} \hat{p}^*. \]

A4.4 **Source localization from range measurements** [Beck, Stoica, and Li]. A signal emitted by a source at an unknown position \( x \in \mathbb{R}^n \) (\( n = 2 \) or \( n = 3 \)) is received by \( m \) sensors at known positions \( y_1, \ldots, y_m \in \mathbb{R}^n \). From the strength of the received signals, we can obtain noisy estimates \( d_k \) of the distances \( \|x - y_k\|_2 \). We are interested in estimating the source position \( x \) based on the measured distances \( d_k \).

In the following problem the error between the squares of the actual and observed distances is minimized:

\[
\text{minimize } f_0(x) = \sum_{k=1}^{m} \left( \|x - y_k\|_2^2 - d_k^2 \right)^2.
\]

Introducing a new variable \( t = x^T x \), we can express this as

\[
\text{minimize } \sum_{k=1}^{m} \left( t - 2y_k^T x + \|y_k\|_2^2 - d_k^2 \right)^2 \quad \text{subject to } x^T x - t = 0.
\]

(1)

The variables are \( x \in \mathbb{R}^n \), \( t \in \mathbb{R} \). Although this problem is not convex, it can be shown that strong duality holds. (It is a variation on the problem discussed on page 229 and in exercise 5.29 of *Convex Optimization*.)

Solve (1) for an example with \( m = 5 \),

\[
y_1 = \begin{bmatrix} 1.8 \\ 2.5 \end{bmatrix}, \quad y_2 = \begin{bmatrix} 2.0 \\ 1.7 \end{bmatrix}, \quad y_3 = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix}, \quad y_4 = \begin{bmatrix} 1.5 \\ 2.0 \end{bmatrix}, \quad y_5 = \begin{bmatrix} 2.5 \\ 1.5 \end{bmatrix},
\]

and

\[
d = (2.00, 1.24, 0.59, 1.31, 1.44).
\]

The figure shows some contour lines of the cost function \( f_0 \), with the positions \( y_k \) indicated by circles.
To solve the problem, you can note that $x^*$ is easily obtained from the KKT conditions for (1) if the optimal multiplier $\nu^*$ for the equality constraint is known. You can use one of the following two methods to find $\nu^*$:

- Derive the dual problem, express it as an SDP, and solve it using CVX.
- Reduce the KKT conditions to a nonlinear equation in $\nu$, and pick the correct solution (similarly as in exercise 5.29 of Convex Optimization).

**Solution.**


Define

\[
A = \begin{bmatrix}
-2y_1^T & 1 \\
-2y_2^T & 1 \\
\vdots & \vdots \\
-2y_5^T & 1
\end{bmatrix}, \quad b = \begin{bmatrix}
d_1^2 - \|y_1\|_2^2 \\
d_2^2 - \|y_2\|_2^2 \\
\vdots \\
d_5^2 - \|y_5\|_2^2
\end{bmatrix}, \quad C = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}, \quad f = \begin{bmatrix}
0 \\
0 \\
-1/2
\end{bmatrix}
\]

and $z = (x_1, x_2, t)$. With this notation, the problem is

\[
\begin{aligned}
& \text{minimize} & & \|Az - b\|_2^2 \\
& \text{subject to} & & z^T C z + 2f^T z = 0.
\end{aligned}
\]

The Lagrangian is

\[
L(z, \nu) = z^T (A^T A + \nu C) z - 2(A^T b - \nu f)^T z + \|b\|_2^2,
\]

which is bounded below as a function of $z$ only if

\[
A^T A + \nu C \succeq 0, \quad A^T b - \nu f \in \mathcal{R}(A^T A + \nu C).
\]

The KKT conditions are therefore as follows.
• Primal feasibility. 

\[ z^T C z + 2 f^T z = 0. \]

• Dual feasibility.

\[ A^T A + \nu C \succeq 0, \quad A^T b - \nu f \in \mathcal{R}(A^T A + \nu C). \]

• Gradient of Lagrangian is zero.

\[(A^T A + \nu C)z = A^T b - \nu f.\]

(Note that this implies the range condition for dual feasibility.)

Method 1. We derive the dual problem. If \( \nu \) is feasible, then

\[ g(\nu) = -(A^T b - \nu f)^T (A^T A + \nu C) (A^T b - \nu f) + \|b\|^2, \]

so the dual problem can be expressed as an SDP

\[
\begin{align*}
\text{maximize} & \quad -t + \|b\|^2 \\
\text{subject to} & \quad \begin{bmatrix} A^T A + \nu C & A^T b - \nu f \\ (A^T b - \nu f)^T & t \end{bmatrix} \succeq 0.
\end{align*}
\]

Solving this in CVX gives \( \nu^* = 0.5896 \). From \( \nu^* \), we get

\[ z^* = (A^T A + \nu C)^{-1}(A^T b - \nu f) = (1.33, 0.64, 2.18). \]

Hence \( x^* = (1.33, 0.64) \).

Method 2. Alternatively, we can solve the KKT equations directly. To simplify the equations, we make a change of variables

\[ w = Q^T L^T z, \]

where \( L \) is the Cholesky factor in the factorization \( A^T A = LL^T \), and \( Q \) is the matrix of eigenvectors of \( L^{-1} CL^{-T} = Q\Lambda Q^T \). This transforms the KKT equations to

\[ w^T \Lambda w + 2g^T w = 0, \quad I + \nu \Lambda \succeq 0, \quad (I + \nu \Lambda)w = h - \nu g \]

where

\[ g = Q^T L^{-1} f, \quad h = Q^T L^{-1} A^T b. \]

We can eliminate \( w \) from the last equation in the KKT conditions to obtain an equation in \( \nu \):

\[ r(\nu) = \sum_{k=1}^{n+1} \left( \frac{\lambda_k (h_k - \nu g_k)^2}{(1 + \nu \lambda_k)^2} + \frac{2g_k (h_k - \nu g_k)}{1 + \nu \lambda_k} \right) = 0 \]

In our example, the eigenvalues are

\[ \lambda_1 = 0.5104, \quad \lambda_2 = 0.2735, \quad \lambda_3 = 0. \]

The figure shows the function \( r \) on two different scales.
The correct solution of \( r(\nu) = 0 \) is the one that satisfies \( 1 + \nu \lambda_k \geq 0 \) for \( k = 1, 2, 3 \), i.e., the solution to the right of the two singularities. This solution can be determined using Newton’s method by repeating the iteration

\[
\nu := \nu - \frac{r(\nu)}{r'(\nu)}
\]

a few times, starting at a value close to the solution. This gives \( \nu^* = 0.5896 \). From \( \nu^* \), we determine \( x^* \) as in the first method.

The last figure shows the contour lines and the optimal \( x^* \).

A5.15 Learning a quadratic pseudo-metric from distance measurements. We are given a set of \( N \) pairs of points in \( \mathbb{R}^n \), \( x_1, \ldots, x_N \), and \( y_1, \ldots, y_N \), together with a set of distances \( d_1, \ldots, d_N > 0 \).
The goal is to find (or estimate or learn) a quadratic pseudo-metric $d$,

$$d(x, y) = \left( (x - y)^T P (x - y) \right)^{1/2},$$

with $P \in S^n_+$, which approximates the given distances, i.e., $d(x_i, y_i) \approx d_i$. (The pseudo-metric $d$ is a metric only when $P \succ 0$; when $P \succeq 0$ is singular, it is a pseudo-metric.)

To do this, we will choose $P \in S^n_+$ that minimizes the mean squared error objective

$$\frac{1}{N} \sum_{i=1}^{N} (d_i - d(x_i, y_i))^2.$$

(a) Explain how to find $P$ using convex or quasiconvex optimization. If you cannot find an exact formulation (i.e., one that is guaranteed to minimize the total squared error objective), give a formulation that approximately minimizes the given objective, subject to the constraints.

(b) Carry out the method of part (a) with the data given in `quad_metric_data.m`. The columns of the matrices $X$ and $Y$ are the points $x_i$ and $y_i$; the row vector $d$ gives the distances $d_i$. Give the optimal mean squared distance error.

We also provide a test set, with data $X_{test}$, $Y_{test}$, and $d_{test}$. Report the mean squared distance error on the test set (using the metric found using the data set above).

Solution.

(a) The problem is

$$\text{minimize } \frac{1}{N} \sum_{i=1}^{N} (d_i - d(x_i, y_i))^2$$

with variable $P \in S^n_+$. This problem can be rewritten as

$$\text{minimize } \frac{1}{N} \sum_{i=1}^{N} (d_i^2 - 2d_i d(x_i, y_i) + d(x_i, y_i)^2),$$

with variable $P$ (which enters through $d(x_i, y_i)$). The objective is convex because each term of the objective can be written as (ignoring the $1/N$ factor)

$$d_i^2 - 2d_i \left( (x_i - y_i)^T P (x_i - y_i) \right)^{1/2} + (x_i - y_i)^T P (x_i - y_i),$$

which is convex in $P$. To see this, note that the first term is constant and the third term is linear in $P$. The middle term is convex because it is the negation of the composition of a concave function (square root) with a linear function of $P$.

(b) The following code solves the problem for the given instance. We find that the optimal mean squared error on the training set is 0.887; on the test set, it is 0.827 (in MATLAB). This tells us that we probably haven’t overfit. In fact,

\[\text{You will get 0.899 and 0.655 in python; 0.900 and 0.964 in Julia instead. Because we use different random seeds.}\]
the optimal $P$ is singular; it has one zero eigenvalue. This is correct; the positive semidefinite constraint is active.
Here is the solution in Matlab.

```matlab
%% learning a quadratic metric

quad_metric_data;

Z = X-Y;
cvx_begin
    variable P(n,n) symmetric
    % objective
    f = 0;
    for i = 1:N
        f = f + d(i)^2 -2*d(i)*sqrt(Z(:,i)'*P*Z(:,i)) + Z(:,i)'*P*Z(:,i);
    end
    minimize (f/N)
    subject to
        P == semidefinite(n);
end

Z_test = X_test-Y_test;
d_hat = norms(sqrtm(P)*Z_test);
obj_test = sum_square(d_test - d_hat)/N_test
```

Here it is in Python.

```python
from quad_metric_data import *
import numpy as np
from scipy import linalg as la
import cvxpy as cvx

Z = X - Y
P = cvx.Variable(n,n)
f = 0
for i in range(N):
    f += d[i]**2
    f += -2*d[i]*cvx.sqrt(cvx.quad_form(Z[:,i],P)) + Z[:,i]'*P*Z[:,i]
prob = cvx.Problem(cvx.Minimize(f/N),[P == cvx.Semidef(n)])
train_error = prob.solve()
print(train_error)
```
Z_test = X_test-Y_test

d_hat = np.linalg.norm(la.sqrtm(P.value).dot(Z_test), axis=0)

obj_test = (np.linalg.norm(d_test - d_hat)**2)/N_test

print(obj_test)

And finally, Julia.

# learning a quadratic metric
include("quad_metric_data.jl");
using Convex, SCS

Z = X-Y;
P = Semidefinite(n, n);

f = 0;
for i = 1:N
    f += d[i]^2 - 2*d[i]*sqrt(Z[:,i]'*P*Z[:,i]) + Z[:,i]'*P*Z[:,i];
end
p = minimize(f/N);
solve!(p, SCSSolver(max_iters=100000));

Z_test = X_test-Y_test;
d_hat = sqrt(sum((real(sqrtm(P.value))*Z_test).^2, 1));
obj_test = vecnorm(d_test - d_hat)^2/N_test;
println("train error: ", p.optval)
println("test error: ", obj_test)
eig(P.value)

A14.13 Lightest structure that resists a set of loads. We consider a mechanical structure in 2D (for simplicity) which consists of a set of m nodes, with known positions p_1,...,p_m \in \mathbb{R}^2, connected by a set of n bars (also called struts or elements), with cross-sectional areas a_1,...,a_n \in \mathbb{R}^+, and internal tensions t_1,...,t_n \in \mathbb{R}.

Bar j is connected between nodes r_j and s_j. (The indices r_1,...,r_n and s_1,...,s_n give the structure topology.) The length of bar j is L_j = ||p_{r_j} - p_{s_j}||_2, and the total volume of the bars is V = \sum_{j=1}^n a_j L_j. (The total weight is proportional to the total volume.)

Bar j applies a force (t_j/L_j)(p_{r_j} - p_{s_j}) \in \mathbb{R}^2 to node s_j and the negative of this force to node r_j. Thus, positive tension in a bar pulls its two adjacent nodes towards each other; negative tension (also called compression) pushes them apart. The ratio of the tension in a bar to its cross-sectional area is limited by its yield strength, which is symmetric in tension and compression: |t_j| \leq \sigma a_j, where \sigma > 0 is a known constant that depends on the material.

The nodes are divided into two groups: free and fixed. We will take nodes 1,...,k to be free, and nodes k+1,...,m to be fixed. Roughly speaking, the fixed nodes are
firmly attached to the ground, or a rigid structure connected to the ground; the free ones are not.

A loading consists of a set of external forces, \( f_1, \ldots, f_k \in \mathbb{R}^2 \) applied to the free nodes. Each free node must be in equilibrium, which means that the sum of the forces applied to it by the bars and the external force is zero. The structure can resist a loading (without collapsing) if there exists a set of bar tensions that satisfy the tension bounds and force equilibrium constraints. (For those with knowledge of statics, these conditions correspond to a structure made entirely with pin joints.)

Finally, we get to the problem. You are given a set of \( M \) loadings, i.e., \( f_1^{(i)}, \ldots, f_k^{(i)} \in \mathbb{R}^2, i = 1, \ldots, M \). The goal is to find the bar cross-sectional areas that minimize the structure volume \( V \) while resisting all of the given loadings. (Thus, you are to find one set of bar cross-sectional areas, and \( M \) sets of tensions.) Using the problem data provided in `lightest_struct_data.m`, report \( V^* \) and \( V^{\text{unif}} \), the smallest feasible structure volume when all bars have the same cross-sectional area. The node positions are given as a \( 2 \times m \) matrix \( P \), and the loadings as a \( 2 \times k \times M \) array \( F \). Use the code included in the data file to visualize the structure with the bar cross-sectional areas that you find, and provide the plot in your solution.

**Hint.** You might find the graph incidence matrix \( A \in \mathbb{R}^{m \times n} \) useful. It is defined as

\[
A_{ij} = \begin{cases} 
+1 & i = r_j \\
-1 & i = s_j \\
0 & \text{otherwise}
\end{cases}
\]

**Remark.** You could reasonably ask, ‘Does a mechanical structure really solve a convex optimization problem to determine whether it should collapse?’ It sounds odd, but the answer is, yes it does.

**Solution.** We can use the graph incidence matrix \( A \) to express the force exerted by the bars on each node. For a loading \( f_1, \ldots, f_k \), define \( G \in \mathbb{R}^{2 \times m} \) such that \( g_i \), the \( i \)th column of \( G \), is the sum of the forces from each of the bars connected to node \( i \). Then \( G = -PADA^T \), where \( P \in \mathbb{R}^{2 \times m} \) is a matrix whose \( i \)th column is \( p_i \), and \( D \in \mathbb{R}^{n \times n} \) is a diagonal matrix such that \( D_{jj} = t_j/L_j \). (To see this, note that the \( i \)th column of \( PAD \) is \( (t_i/L_i)(p_{r_i} - p_{s_i}) \), the force that bar \( i \) applies to the adjacent node \( r_i \).) The force equilibrium constraints for the loading can then be written as \( g_i + f_i = 0 \), for \( i = 1, \ldots, k \). A single set of bar cross-sectional areas must satisfy the equilibrium constraints for each of the \( M \) loadings. The remaining constraints are easily formulated.

We find that \( V^* = 188.55 \), and \( V^{\text{unif}} = 492.00 \).
The following code solves the problem.

```matlab
% lightest structure that resists a set of loads
lightest_struct_data;

% form incidence matrix
A = zeros(m, n);
for i = 1:n
    A(r(i), i) = +1;
    A(s(i), i) = -1;
end

L = norms(P*A)';

% solve with all bars having same cross-sectional area
cvx_begin quiet
    variables a(n) t(n, M)
    expression G(2, m, M) % force due to bars
    minimize (a'*L)
    subject to
        a == mean(a);
        for i = 1:M
            abs(t(:, i)) <= sigma.*a;
            G(:, :, i) = -P*A*diag(t(:, i)./L)*A';
            G(:, 1:k, i) + F(:, :, i) == 0;
        end
end
cvx_end
fprintf('V^unif = %f
', cvx_optval);

% plot
clf;
```

11
subplot(1,2,1); hold on;
for i = 1:n
  p1 = r(i); p2 = s(i);
  plt_str = 'b-';
  if a(i) < 0.001
    plt_str = 'r--';
  end
  plot([P(1, p1) P(1, p2)], [P(2, p1) P(2, p2)], ...
       plt_str, 'LineWidth', a(i));
end
axis([-0.5 N-0.5 -0.1 N-0.5]); axis square; box on;
sel(gca, 'xtick', [], 'ytick', []);
hold off;

% solve with bars having different cross-sectional areas
cvx_begin quiet
  variables a(n) t(n, M)
  expression G(2, m, M) % force due to bars
  minimize (a'*L)
  subject to
    for i = 1:M
      abs(t(:, i)) <= sigma.*a;
      G(:, :, i) = -P*A*diag(t(:, i)./L)*A';
      G(:, 1:k, i) + F(:, :, i) == 0;
    end
  cvx_end
fprintf('V^star = %f
', cvx_optval);

% plot
subplot(1,2,2); hold on;
for i = 1:n
  p1 = r(i); p2 = s(i);
  plt_str = 'b-';
  width = a(i);
  if a(i) < 0.001
    plt_str = 'r--';
    width = 1;
  end
  plot([P(1, p1) P(1, p2)], [P(2, p1) P(2, p2)], ...
       plt_str, 'LineWidth', width);
end
axis([-0.5 N-0.5 -0.1 N-0.5]); axis square; box on;
A16.1 **Power flow optimization with ‘N − 1’ reliability constraint.** We model a network of power lines as a graph with \( n \) nodes and \( m \) edges. The power flow along line \( j \) is denoted \( p_j \), which can be positive, which means power flows along the line in the direction of the edge, or negative, which means power flows along the line in the direction opposite the edge. (In other words, edge orientation is only used to determine the direction in which power flow is considered positive.) Each edge can support power flow in either direction, up to a given maximum capacity \( P_{j \text{max}} \), i.e., we have \( |p_j| \leq P_{j \text{max}} \).

Generators are attached to the first \( k \) nodes. Generator \( i \) provides power \( g_i \) to the network. These must satisfy \( 0 \leq g_i \leq G_{i \text{max}} \), where \( G_{i \text{max}} \) is a given maximum power available from generator \( i \). The power generation costs are \( c_i > 0 \), which are given; the total cost of power generation is \( c^T g \).

Electrical loads are connected to the nodes \( k + 1, \ldots, n \). We let \( d_i \geq 0 \) denote the demand at node \( k + i \), for \( i = 1, \ldots, n - k \). We will consider these loads as given. In this simple model we will neglect all power losses on lines or at nodes. Therefore, power must balance at each node: the total power flowing into the node must equal the sum of the power flowing out of the node. This power balance constraint can be expressed as

\[
Ap = \begin{bmatrix} -g \\ d \end{bmatrix},
\]

where \( A \in \mathbb{R}^{n \times m} \) is the node-incidence matrix of the graph, defined by

\[
A_{ij} = \begin{cases} +1 & \text{edge } j \text{ enters node } i, \\ -1 & \text{edge } j \text{ leaves node } i, \\ 0 & \text{otherwise.} \end{cases}
\]

In the basic power flow optimization problem, we choose the generator powers \( g \) and the line flow powers \( p \) to minimize the total power generation cost, subject to the constraints listed above. The (given) problem data are the incidence matrix \( A \), line capacities \( P_{\text{max}} \), demands \( d \), maximum generator powers \( G_{\text{max}} \), and generator costs \( c \).

In this problem we will add a basic (and widely used) reliability constraint, commonly called an ‘\( N − 1 \) constraint’. (\( N \) is not a parameter in the problem; ‘\( N − 1 \)’ just means ‘all-but-one’.) This states that the system can still operate even if any one power line goes out, by re-routing the line powers. The case when line \( j \) goes out is called ‘failure contingency \( j \)’; this corresponds to replacing \( P_{j \text{max}} \) with 0. The requirement is that there must exist a contingency power flow vector \( p^{(j)} \) that satisfies all the constraints above, with \( p_j^{(j)} = 0 \), using the same given generator powers. (This corresponds to
the idea that power flows can be re-routed quickly, but generator power can only be changed more slowly.) The ‘$N-1$ reliability constraint’ requires that for each line, there is a contingency power flow vector. The ‘$N-1$ reliability constraint’ is (implicitly) a constraint on the generator powers.

The questions below concern the specific instance of this problem with data given in rel_pwr_flow_data.*. (Executing this file will also generate a figure showing the network you are optimizing.) Especially for part (b) below, you must explain exactly how you set up the problem as a convex optimization problem.

(a) **Nominal optimization.** Find the optimal generator and line power flows for this problem instance (without the $N-1$ reliability constraint). Report the optimal cost and generator powers. (You do not have to give the power line flows.)

(b) **Nominal optimization with $N-1$ reliability constraint.** Minimize the nominal cost, but you must choose generator powers that meet the $N-1$ reliability requirement as well. Report the optimal cost and generator powers. (You do not have to give the nominal power line flows, or any of the contingency flows.)

**Solution.**

(a) To find the optimal generators and line power flows we solve the LP

$$\begin{align*}
\text{minimize} & \quad c^T g \\
\text{subject to} & \quad Ap = \begin{bmatrix} -g \\ d \end{bmatrix} \\
& \quad -P_{\text{max}} \preceq p \preceq P_{\text{max}} \\
& \quad 0 \preceq g \preceq G_{\text{max}},
\end{align*}$$

with variables $g$ and $p$.

(b) To handle the additional $N-1$ reliability constraint, we must introduce a set of power flow vectors for each contingency. We then solve the LP

$$\begin{align*}
\text{minimize} & \quad c^T g \\
\text{subject to} & \quad Ap^{(j)} = \begin{bmatrix} -g \\ d \end{bmatrix}, \quad j = 1, \ldots, m \\
& \quad p^{(j)}_j = 0, \quad j = 1, \ldots, m \\
& \quad -P_{\text{max}} \preceq p^{(j)} \preceq P_{\text{max}}, \quad j = 1, \ldots, m \\
& \quad 0 \preceq g \preceq G_{\text{max}},
\end{align*}$$

with variables $g \in \mathbb{R}^k$ and $p^{(1)}, \ldots, p^{(m)} \in \mathbb{R}^m$.

The optimal costs are 44.60 and 56.20 for parts (a) and (b) respectively. The optimal generator powers are

$$g_{\text{nom}} = \begin{bmatrix} 3.0 \\ 0.0 \\ 2.3 \\ 7.0 \end{bmatrix}, \quad g_{\text{rel}} = \begin{bmatrix} 1.9 \\ 1.9 \\ 4.0 \\ 4.5 \end{bmatrix}.$$
We can say a little about these results. In the nominal case, it turns out that the line capacities are not tight; that is, we have no congestion on the transmission lines. So we must select a set of generator powers that deliver the required power, which is the sum of the demands (12.32 power units). To do this most efficiently, we start with generator 4, which has the lowest cost, and we set it to its maximum, which gives us a total of 7 power units. We then go the second cheapest generator, generator 1, and set it to its maximum (3 power units), which gives us a total of 10 power units. Finally, we go to the third cheapest generator, generator 3, and use it to satisfy the remaining demand.

When we impose the additional $N-1$ reliability constraint, we are forced to shift some generation to more expensive generators.

The following MATLAB code solves parts (a) and (b).

```matlab
rel_pwr_flow_data;

% nominal case
cvx_begin
    variables p(m) g_nom(k)
    minimize (c'*g_nom)
    subject to
        A*p == [-g_nom;d];
        abs(p) <= Pmax;
        g_nom <= Gmax;
        g_nom >= 0;
cvx_end
nom_cost = cvx_optval;

% N-1 case
cvx_begin
    variables P(m,m) g_rel(k)
    minimize (c'*g_rel)
    subject to
        A*P == [-g_rel;d]*ones(1,m);
        diag(P) == 0;
        abs(P) <= Pmax*ones(1,m);
        g_rel <= Gmax;
        g_rel >= 0;
    cvx_end
rel_cost = cvx_optval;

% show nominal optimal and reliable optimal cost
[nom_cost rel_cost]
```

15
The following Python code solves parts (a) and (b).

```python
import cvxpy as cvx
from rel_pwr_flow_data import *

# nominal case
p = cvx.Variable(m)
g_nom = cvx.Variable(k)
nom_cost = cvx.Problem(cvx.Minimize(c.T*g_nom),
                         [A[:k,:]*p == -g_nom,
                          A[k:,:]*p == d.T,  
                          cvx.abs(p) <= Pmax.T,  
                          g_nom <= Gmax,  
                          g_nom >= 0.]).solve()

# N-1 case
P = cvx.Variable(m,m)
g_rel = cvx.Variable(k)
rel_cost = cvx.Problem(cvx.Minimize(c.T*g_rel),
                        [A[:k,:]*P == -g_rel*np.ones((1,m)),
                         A[k:,:]*P == d.T*np.ones((1,m)),
                         cvx.diag(P) == 0,
                         cvx.abs(P) <= Pmax.T*np.ones((1,m)),
                         g_rel <= Gmax,
                         g_rel >= 0.]).solve()

# show nominal optimal and reliable optimal cost
print "nom_cost", nom_cost
print "rel_cost", rel_cost

# show nominal optimal and reliable optimal generator powers
print "g_nom:", g_nom.value.A1
print "g_rel:", g_rel.value.A1
```

The following Julia code solves parts (a) and (b).

```julia
#include("rel_pwr_flow_data.jl")

using Convex, SCS

# nominal case
p = Variable(m);
g_nom = Variable(k);
constraints = [];
constraints += A*p == [-g_nom; d];
constraints += abs(p) <= Pmax;
constraints += g_nom <= Gmax;
constraints += g_nom >= 0;
problem = minimize(c'*g_nom, constraints);
solve!(problem, SCSSolver(max_iters=20000));
nom_cost = problem.optval;
println("nominal cost: ", nom_cost)
println("nominal generator powers: ")
println(g_nom.value)

# N-1 case
P = Variable(m,m);
g_rel = Variable(k);
constraints = [];
constraints += A*P == [-g_rel; d]*ones(1,m);
constraints += abs(P) <= Pmax * ones(1,m);
constraints += diag(P) == 0;
constraints += g_rel <= Gmax;
constraints += g_rel >= 0;
problem = minimize(c'*g_rel, constraints);
solve!(problem, SCSSolver(max_iters=20000));
rel_cost = problem.optval;
println("reliable cost: ", rel_cost)
println("reliable generator powers: ")
println(g_rel.value)