CSCI 5254 - Convex Optimization
Homework 5

Aritra Chakrabarty

November 13, 2024

Problem 6.9

Show that the following problem is quasiconvex:

$$\text{minimize} \quad \max_{i=1,\ldots,k} \left| \frac{p(t_i)}{q(t_i)} - y_i \right|$$

where

$$p(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_m t^m, \quad q(t) = 1 + b_1 t + \cdots + b_n t^n,$$

and the domain of the objective function is defined as

$$D = \{(a, b) \in \mathbb{R}^{m+1} \times \mathbb{R}^n \mid q(t) > 0,\ \alpha \leq t \leq \beta\}.$$

In this problem we fit a rational function $\frac{p(t)}{q(t)}$ to given data, while constraining the denominator polynomial to be positive on the interval $[\alpha, \beta]$. The optimization variables are the numerator and denominator coefficients a_i, b_i. The interpolation points $t_i \in [\alpha, \beta]$, and desired function values y_i, $i = 1, \ldots, k$, are given.

Answer: A function $f(x)$ is quasiconvex when it's sublevel sets are convex. For every $\gamma \in \mathbb{R}$, the set $\{x \mid f(x) \leq \gamma\}$ is convex.
We can denote the objective function as

$$f(a, b) = \max_{i=1,\ldots,k} \left| \frac{p(t_i)}{q(t_i)} - y_i \right|$$

To prove the quasiconvexity of this function, we need to show that for $\gamma \geq 0$, the set

$$s_\gamma = \{(a, b) \in D \mid f(a, b) \leq \gamma\}$$

is convex.

$$f(a, b) \leq \gamma \implies \left| \frac{p(t_i)}{q(t_i)} - y_i \right| \leq \gamma$$

$$-\gamma \leq \frac{p(t_i)}{q(t_i)} - y_i \leq \gamma$$

$$y_i - \gamma \leq \frac{p(t_i)}{q(t_i)} \leq y_i + \gamma$$

Given that $q(t_i) > 0, \quad \forall i = 1, \dots, k$, we can multiply the inequality by $q(t_i)$ without changing the sign.

$$y_i q(t_i) - \gamma q(t_i) \leq p(t_i) \leq y_i q(t_i) + \gamma q(t_i)$$

Splitting this up into two parts, we get

$$p(t_i) - y_i q(t_i) \leq \gamma q(t_i) \quad \text{and} \quad y_i q(t_i) - p(t_i) \leq \gamma q(t_i)$$

We know that $p(t_i)$ and $q(t_i)$ are linear in the coefficients $a = (a_0, a_1, \dots, a_m)$ and $b = (b_1, \dots, b_n)$, so the above inequalities are linear in a and b.

Thus,

$$p(t_i) - y_i q(t_i) \leq \gamma q(t_i) \implies p(t_i) - y_i q(t_i) - \gamma q(t_i) \leq 0$$

which is linear in a and b.

Similarly,

$$-p(t_i) + y_i q(t_i) \leq \gamma q(t_i) \implies -p(t_i) + y_i q(t_i) - \gamma q(t_i) \leq 0$$

which is also linear in a and b.

For a fixed γ, these inequalities are linear constraints on a and b. Linear inequalities define half-spaces, and the intersection of half-spaces is a convex set. Thus, the set s_γ is convex for all $\gamma \geq 0$. Therefore, the function $f(a, b)$ is quasiconvex for $\gamma \geq 0$.

Additional Exercise 3.9

Complex least-norm problem. We consider the complex least ℓ_p-norm problem

$$\begin{array}{ll} \text{minimize} & \|x\|_p \\ \text{subject to} & Ax = b \end{array}$$

where $A \in \mathbb{C}^{m \times n}$, $b \in \mathbb{C}^m$, and the variable is $x \in \mathbb{C}^n$. Here $\|\cdot\|_p$ denotes the ℓ_p-norm on $\mathbb{C}^n$, defined as

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

for $p \geq 1$, and $\|x\|_\infty = \max_{i=1,\dots,n} |x_i|$. We assume A is full rank, and $m < n$.

Part (a)

Formulate the complex least ℓ_2-norm problem as a least ℓ_2-norm problem with real problem data and variable. *Hint: Use $z = (\Re x, \Im x) \in \mathbb{R}^{2n}$ as the variable.*

Answer: Let $z \in \mathbb{R}^{2n}$ be the represeantation $z = \begin{bmatrix} \Re x \\ \Im x \end{bmatrix}$, where $\Re(x) \in \mathbb{R}^n$ and $\Im(x) \in \mathbb{R}^n$ are the real and imaginary parts of $x \in \mathbb{C}^n$.

Let $A = \Re(A) + i\Im(A)$, where $\Re(A) \in \mathbb{R}^{m \times n}$ and $\Im(A) \in \mathbb{R}^{m \times n}$ are the real and imaginary parts of $A \in \mathbb{C}^{m \times n}$. Similarly, let $b = \Re(b) + i\Im(b)$, where $\Re(b) \in \mathbb{R}^m$ and $\Im(b) \in \mathbb{R}^m$ are the real and imaginary parts of $b \in \mathbb{C}^m$.

The complex equation $Ax = b$ can be written as

$$\Re(A)\Re(x) - \Im(A)\Im(x) = \Re(b)$$

$$\Re(A)\Im(x) + \Im(A)\Re(x) = \Im(b)$$

In matrix form this is

$$\begin{bmatrix} \Re(A) & -\Im(A) \\ \Im(A) & \Re(A) \end{bmatrix} \begin{bmatrix} \Re(x) \\ \Im(x) \end{bmatrix} = \begin{bmatrix} \Re(b) \\ \Im(b) \end{bmatrix}$$

Formulating this as a real least ℓ_2-norm problem, we have

$$\begin{array}{ll} \text{minimize} & \|z\|_2 \\ \text{subject to} & \begin{bmatrix} \Re(A) & -\Im(A) \\ \Im(A) & \Re(A) \end{bmatrix} z = \begin{bmatrix} \Re(b) \\ \Im(b) \end{bmatrix} \end{array}$$

where $z \in \mathbb{R}^{2n}$ is the variable.

Part (b)

Formulate the complex least ℓ_∞-norm problem as an SOCP.

Answer: First of all, the ℓ_∞-norm is defined as $\|x\|_\infty = \max_{i=1,\dots,n} |x_i|$, wherre $|x_i| = \sqrt{\Re(x_i)^2 + \Im(x_i)^2}$.

Let $z = \begin{bmatrix} \Re x \\ \Im x \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \vdots \\ z_n \\ z_{n+1} \\ z_{n+2} \\ \vdots \\ z_{2n} \end{bmatrix}$ where $z_i = \Re(x_i)$ for $i = 1, \dots, n$ and $z_{n+i} = \Im(x_i)$ for $i = 1, \dots, n$.

Similar to the previous part, we can write the complex equation $Ax = b$ as

$$\begin{bmatrix} \Re(A) & -\Im(A) \\ \Im(A) & \Re(A) \end{bmatrix} z = \begin{bmatrix} \Re(b) \\ \Im(b) \end{bmatrix}$$

We do need an upper bound on the ℓ_∞-norm of x. Let t be the upper bound on the ℓ_∞-norm of x. Then, we have

$$\sqrt{{z_i}^2 + {z_{n+i}}^2} \le t \quad \text{for } i = 1, \dots, n$$

Putting this all together, we have the SOCP

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \begin{bmatrix} \Re(A) & -\Im(A) \\ \Im(A) & \Re(A) \end{bmatrix} z = \begin{bmatrix} \Re(b) \\ \Im(b) \end{bmatrix}, \\ & \sqrt{{z_i}^2 + {z_{n+i}}^2} \le t \quad \text{for } i = 1, \dots, n \end{array}$$

Part (c)

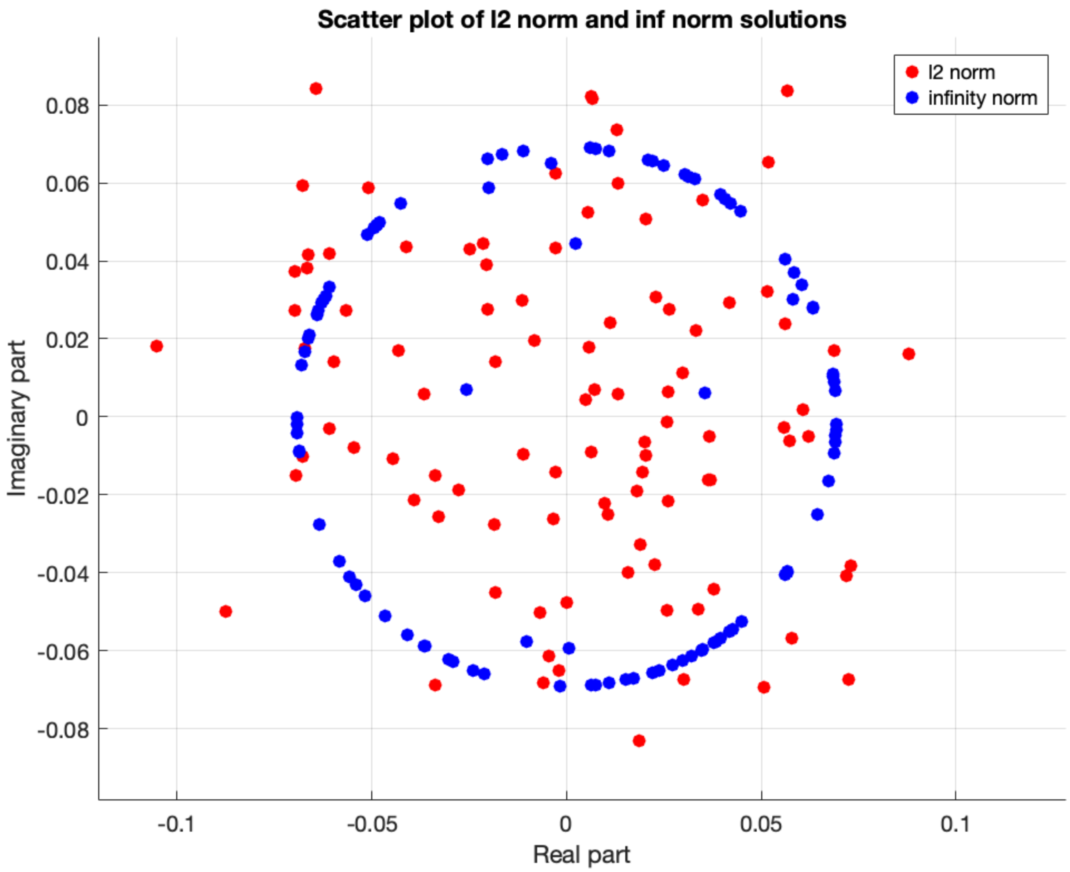
Solve a random instance of both problems with $m = 30$ and $n = 100$. To generate the matrix A, you can use the Matlab command `A = randn(m,n) + i*randn(m,n)`. Similarly, use `b = randn(m,1) + i*randn(m,1)` to generate the vector b. Use the Matlab command `scatter` to plot the optimal solutions of the two problems on the complex plane, and comment (briefly) on what you observe. You can solve the problems using the CVX functions `norm(x,2)` and `norm(x,inf)`, which are overloaded to handle complex arguments. To utilize this feature, you will need to declare variables to be complex in the `variable` statement. (In particular, you do not have to manually form or solve the SOCP from part (b).)

Answer: We can solve this question with the following Matlab code:

```matlab
% Dimensions
m = 30;
n = 100;

% Random complex matrix A and vector b
A = randn(m,n) + 1i*randn(m,n);
b = randn(m,1) + 1i*randn(m,1);

% Solve least l2-norm problem
cvx_begin
    variable x_l2(n) complex
    minimize( norm(x_l2,2))
    subject to
        A*x_l2 == b;
cvx_end

% Solve least l_inf-norm problem
cvx_begin
    variable x_linf(n) complex
    minimize( norm(x_linf,inf))
    subject to
        A*x_linf == b;
cvx_end

% Obtain real and imaginary parts of x_l2 and x_linf
x_l2_real = real(x_l2);
x_l2_imag = imag(x_l2);
x_linf_real = real(x_linf);
x_linf_imag = imag(x_linf);

% Scatter Plot
figure;
hold on;
scatter(x_l2_real, x_l2_imag, 'r', 'filled');
scatter(x_linf_real, x_linf_imag, 'b', 'filled');
xlabel('Real part');
ylabel('Imaginary part');
title('Scatter plot of l2 norm and inf norm solutions');
legend('l2 norm', 'infinity norm');
```

```
axis equal;
legend;
grid on;
hold off;
```

The image we obtain is:



Interestingly, the ℓ_∞-norm solution mostly forms a circle, while the ℓ_2-norm solution is more spread out. This is because the ℓ_∞-norm is the maximum of the absolute values of the real and imaginary parts of x, and hence the solution is a circle, whereas the ℓ_2-norm is the square root of the sum of the squares, so the solution is allowed to be more spread out.

Additional Exercise 4.1

Numerical perturbation analysis example. Consider the quadratic program

$$\begin{array}{ll} \text{minimize} & x_1^2 + 2x_2^2 - x_1x_2 - x_1 \\ \text{subject to} & x_1 + 2x_2 \le u_1, \\ & x_1 - 4x_2 \le u_2, \\ & 5x_1 + 76x_2 \le 1 \end{array}$$

with variables x_1, x_2, and parameters u_1, u_2.

Part (a)

Solve this QP, for parameter values $u_1 = -2$, $u_2 = -3$, to find optimal primal variable values $x_1^\star$ and $x_2^\star$, and optimal dual variable values $\lambda_1^\star$, $\lambda_2^\star$ and $\lambda_3^\star$. Let $p^\star$ denote the optimal objective value. Verify that the KKT conditions hold for the optimal primal and dual variables you found (within reasonable numerical accuracy). *Matlab hint:* See §3.7 of the CVX users' guide to find out how to retrieve optimal dual variables. To specify the quadratic objective, use `quad_form()`.

Answer: The given objective function $f(x) = x_1^2 + 2x_2^2 - x_1x_2 - x_1$ can be written in the quadratic form $f(x) = \frac{1}{2}x^T P x + q^T x + r$, where

$$P = \begin{bmatrix} 2 & -1 \\ -1 & 4 \end{bmatrix}, \quad q = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad r = 0$$

However, due to the way CVX works, the quadratic form actually used doesn't include the factor of $\frac{1}{2}$, so we will use the following quadratic form in CVX:

$$f(x) = x^T P x + q^T x + r$$

This makes $P = \begin{bmatrix} 1 & \frac{-1}{2} \\ \frac{-1}{2} & 2 \end{bmatrix}$, $q = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$, and $r = 0$.

The constraints can be written as $Ax \le b$, where

$$A = \begin{bmatrix} 1 & 2 \\ 1 & -4 \\ 5 & 76 \end{bmatrix}, \quad b = \begin{bmatrix} -2 \\ -3 \\ 1 \end{bmatrix}$$

We can solve this QP using CVX in Matlab as follows:

```
% Defining objective function and constraints in Matrix form
Q = [1 -1/2; -1/2 2];
%f = [-1; 0]; % cvx needs a rowvector
f = [-1 0];
A = [1 2; 1 -4; 5 76];
b = [-2; -3; 1];

% Solving the problem using cvx quad_form
```

```
cvx_begin
    variable x(2)
    dual variable lambda
    minimize(quad_form(x, Q) + f * x)
    subject to
        lambda: A*x <= b
cvx_end

p_star = cvx_optval;

% Displaying the results
disp('Optimal value of x:')
disp(x)

disp('Optimal value of the objective function:')
disp(cvx_optval)

disp('Optimal value of the dual variable:')
disp(lambda)

% Verifying KKT conditions
% Primal Feasibility A*x <= b
primal_feasibility = A*x;
if all(primal_feasibility <= b + 1e-6)
    disp('Primal Feasibility: Satisfied')
else
    disp('Primal Feasibility: Not Satisfied')
    disp(primal_feasibility)
end

% Dual Feasibility lambda >= 0
dual_feasibility = lambda;
if all(dual_feasibility >= 0 - 1e-6)
    disp('Dual Feasibility: Satisfied')
else
    disp('Dual Feasibility: Not Satisfied')
    disp(dual_feasibility)
end

% Complementary Slackness lambda_i * (A*x - b)_i = 0
slack = A * x - b; % this needs to be leq 0
complementary_slackness = lambda .* slack;
if all(abs(complementary_slackness) <= 1e-6)
    disp('Complementary Slackness: Satisfied')
else
    disp('Complementary Slackness: Not Satisfied')
    disp(complementary_slackness)
end

% Stationarity Q*x + f + A'*lambda = 0
% 2 is needed because of the way cvx handles the problem
```

```
stationarity = 2*Q*x + f' + A'*lambda;
if all(abs(stationarity) <= 1e-4)
    disp('Stationarity: Satisfied')
else
    disp('Stationarity: Not Satisfied')
    disp(stationarity)
end
```

The output we obtain is:

```
------------------------------------------------------------
Status: Solved
Optimal value (cvx_optval): +8.22222

Optimal value of x:
   -2.3333
    0.1667

Optimal value of the objective function:
    8.2222

Optimal value of the dual variable:
    1.8994
    3.4684
    0.0931

Primal Feasibility: Satisfied
Dual Feasibility: Satisfied
Complementary Slackness: Satisfied
Stationarity: Satisfied
```

The optimal primal variable values are $x_1^\star = -2.3333$ and $x_2^\star = 0.1667$, and the optimal dual variable values are $\lambda_1^\star = 1.8994$, $\lambda_2^\star = 3.4684$, and $\lambda_3^\star = 0.0931$. The optimal objective value is $p^\star = 8.2222$.

We also see that all the KKT conditions are satsified via the code.

- Primal Feasibility: $Ax \leq b$ is satisfied.
- Dual Feasibility: $\lambda \geq 0$ is satisfied.
- Complementary Slackness: $\lambda_i(A_i x - b_i) = 0$ is satisfied.
- Stationarity: $2Qx + f^\top + A^\top \lambda = 0$ is satisfied.

Part (b)

We will now solve some perturbed versions of the QP, with

$$u_1 = -2 + \delta_1, \quad u_2 = -3 + \delta_2,$$

where δ_1 and δ_2 each take values from $\{-0.1, 0, 0.1\}$. (There are a total of nine such combinations, including the original problem with $\delta_1 = \delta_2 = 0$.) For each combination of δ_1 and δ_2, make a prediction $p^\star_{\text{pred}}$ of the optimal value of the perturbed QP, and compare it to $p^\star_{\text{exact}}$, the exact optimal value of the perturbed QP (obtained by solving the perturbed QP). Put your results in the two righthand columns in a table with the form shown below. Check that the inequality $p^\star_{\text{pred}} \le p^\star_{\text{exact}}$ holds.

δ_1	δ_2	$p^\star_{\text{pred}}$	$p^\star_{\text{exact}}$
0	0		
0	-0.1		
0	0.1		
-0.1	0		
-0.1	-0.1		
-0.1	0.1		
0.1	0		
0.1	-0.1		
0.1	0.1		

Answer: We can solve the perturbed QP using CVX in Matlab as follows:

```
% -------------------- Part b --------------------%
delta_values = [-0.1, 0, 0.1];
num_cases = length(delta_values)^2;
results = zeros(num_cases, 4); % 4 columns: delta1, delta2, p_pred, p_exact
count = 1;

for delta1 = delta_values
    for delta2 = delta_values
        % Perturbed b
        delta = [delta1; delta2; 0];
        b_perturbed = b + delta;

        % Find predicted optimal value
        % lambda is subtracted because increasing b will decrease the optimal value
        p_pred = p_star - lambda(1:2)' * delta(1:2);

        % Solve the problem with perturbed b
        cvx_begin quiet
            variable x(2)
            minimize(quad_form(x, Q) + f * x)
            subject to
                A*x <= b_perturbed
        cvx_end
        p_exact = cvx_optval;

        % Store the results
        results(count, :) = [delta1, delta2, p_pred, p_exact];
        count = count + 1;
    end
```

```
end

% Display results in a table
Tab = array2table(results, 'VariableNames', {'delta1', 'delta2', 'p_pred',
    'p_exact'});
disp(Tab)
```

The output we obtain is:

```
        delta1    delta2    p_pred    p_exact
        ------    ------    ------    -------

         -0.1      -0.1      8.759    8.8156
         -0.1         0     8.4122     8.565
         -0.1       0.1     8.0653    8.3189
            0      -0.1     8.5691    8.7064
            0         0     8.2222    8.2222
            0       0.1     7.8754      7.98
          0.1      -0.1     8.3791    8.7064
          0.1         0     8.0323    8.2222
          0.1       0.1     7.6854    7.7515
```

Putting this into the tabular format requested, we get:

δ_1	δ_2	$p^\star_{\text{pred}}$	$p^\star_{\text{exact}}$
0	0	8.2222	8.2222
0	-0.1	8.5691	8.7064
0	0.1	7.8754	7.98
-0.1	0	8.4122	8.565
-0.1	-0.1	8.759	8.8156
-0.1	0.1	8.0653	8.3189
0.1	0	8.0323	8.2222
0.1	-0.1	8.3791	8.7064
0.1	0.1	7.6854	7.7515

At a quick glance, we can see that $p^\star_{\text{pred}} \leq p^\star_{\text{exact}}$ for all the cases.

Additional Exercise 5.2

Minimax rational fit to the exponential. (See exercise 6.9 of *Convex Optimization.*) We consider the specific problem instance with data

$$t_i = -3 + 6(i-1)/(k-1), \quad y_i = e^{t_i}, \quad i = 1, \ldots, k,$$

where $k = 201$. (In other words, the data are obtained by uniformly sampling the exponential function over the interval $[-3, 3]$.) Find a function of the form

$$f(t) = \frac{a_0 + a_1 t + a_2 t^2}{1 + b_1 t + b_2 t^2}$$

that minimizes $\max_{i=1,\ldots,k} |f(t_i) - y_i|$. (We require that $1 + b_1 t_i + b_2 t_i^2 > 0$ for $i = 1, \ldots, k$.)

Find optimal values of a_0, a_1, a_2, b_1, b_2, and give the optimal objective value, computed to an accuracy of 0.001. Plot the data and the optimal rational function fit on the same plot. On a different plot, give the fitting error, i.e., $f(t_i) - y_i$.

Hint. You can use `strcmp(cvx_status, 'Solved')`, after `cvx_end`, to check if a feasibility problem is feasible.

Answer: The goal of this problem is to minimise the maximum error between $f(t_i)$ and $y_i = e^{t_i}$ over $k = 201$ data points t_i in the interval $[-3, 3]$.

For each data point t_i, we have $|f(t_i) - y_i| \leq E$ where E is the error. Similar to exercise 6.9, we end up with

$$-E \leq \frac{p(t_i)}{q(t_i)} \leq E$$

where $p(t_i) = a_0 + a_1 t_i + a_2 t_i^2$ and $q(t_i) = 1 + b_1 t_i + b_2 t_i^2$.

We can rewrite this as two inequalities:

$$p(t_i) - y_i q(t_i) - E q(t_i) \leq 0 \quad \text{and} \quad -p(t_i) + y_i q(t_i) - E q(t_i) \leq 0$$

To find the minimal E such that the above inequalities hold $\forall i$, notice that for a fixed E, the constraints become linear and thus convex in the variables a_0, a_1, a_2, b_1, b_2 (because we don't multiply a variable E with a bunch of variables in $q(t_i)$). This quasiconvex structure allows us to employ the bisection method to efficiently search for the smallest feasible E. Therefore, we convert this to a feasibility problem.

We let $p_i = p(t_i)$ and $q_i = q(t_i)$ for simplicity.

The feasibility problem, put altogether, is:

$$\begin{array}{rl} \text{find} & a_0, a_1, a_2, b_1, b_2 \\ \text{subject to} & p_i = a_0 + a_1 t_i + a_2 t_i^2 \\ & q_i = 1 + b_1 t_i + b_2 t_i^2 \\ & q_i > 0 \\ & p_i - y_i q_i - E q_i \leq 0 \\ & -p_i + y_i q_i - E q_i \leq 0 \end{array}$$

We can use MATLAB to solve this problem. The code is as follows:

```
% Generate data
k = 201;
t = linspace(-3, 3, k)';
y = exp(t);

% Set bisection parameters and initial bounds
E_lower = 0;
E_upper = max(y); % the error cannot be larger than the maximum value of y
tol = 1e-3;

```

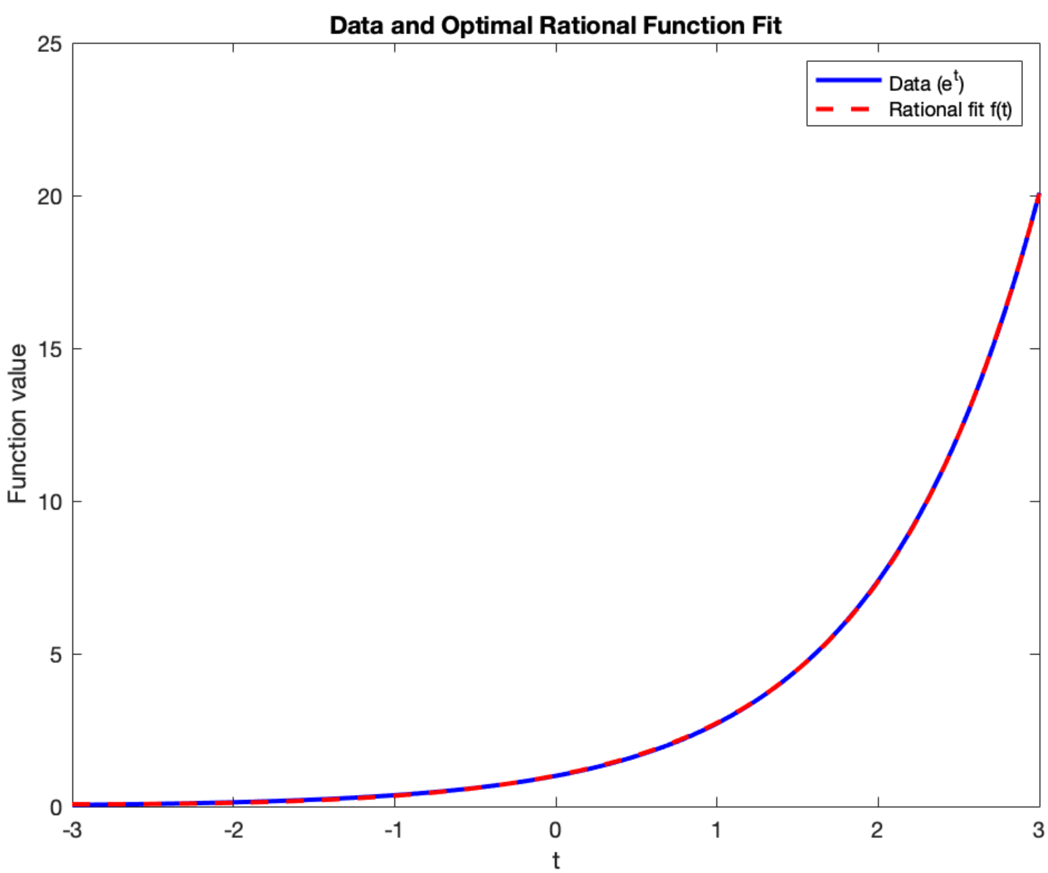
```
% Variables to store optimal solution
opt_a = [];
opt_b = [];
E_opt = E_upper; % Initialize with upper bound as we work our way down

% Bisection loop
while (E_upper - E_lower) > tol
    E = (E_upper + E_lower) / 2; % Bisection to obtain midpoint

    cvx_begin quiet
        variables a0 a1 a2 b1 b2 p(k) q(k)
        epsilon = 1e-6;
        q >= epsilon;
        p == a0 + a1 * t + a2 * t.^2;
        q == 1 + b1 * t + b2 * t.^2;
        % Constraints
        p - y .* q - E * q <= 0;
        -p + y .* q - E * q <= 0;
    cvx_end

    if strcmp(cvx_status, 'Solved')
        % Update upper bound for this feasible solution to error
        E_upper = E; % if problem is feasible at E, this means we have the coefficients that achieve a maximum error less than or equal to previous E_upper
        E_opt = E;
        opt_a = [a0; a1; a2];
        opt_b = [b1; b2];
    else
        % If not possible, update lower bound
        E_lower = E; % if problem is infeasible at E, this means no coefficents can achieve a maximum error less than or equal to E
    end
end

% Display optimal solution
disp('Optimal a coefficients:');
disp(opt_a);
disp('Optimal b coefficients:');
disp(opt_b);
disp('Optimal error:');
disp(E_opt);

% Compute fitted values
f_t = (opt_a(1) + opt_a(2) * t + opt_a(3) * t.^2) ./ (1 + opt_b(1) * t + opt_b(2) * t.^2);

% Plotting
figure;
plot(t, y, 'b', 'LineWidth', 2); hold on;
```

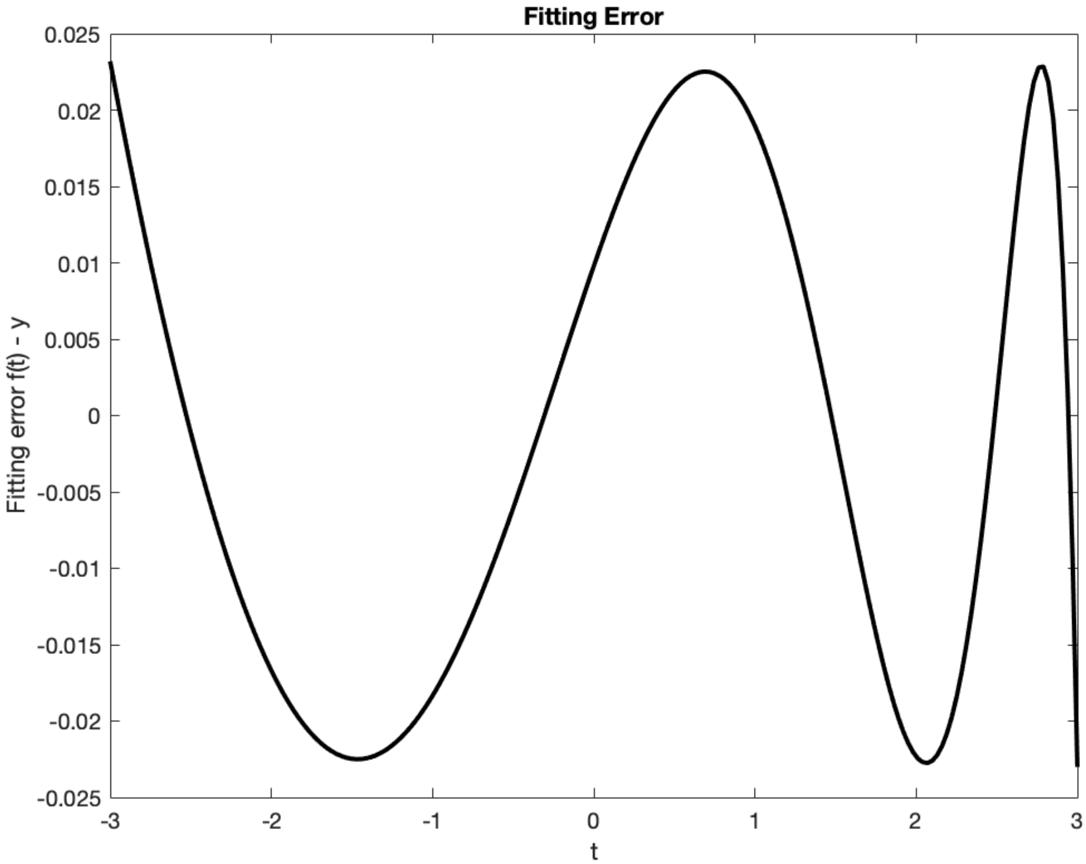
```
plot(t, f_t, 'r--', 'LineWidth', 2);
xlabel('t');
ylabel('Function value');
legend('Data (e^{t})', 'Rational fit f(t)');
title('Data and Optimal Rational Function Fit');

figure;
plot(t, f_t - y, 'k', 'LineWidth', 2);
xlabel('t');
ylabel('Fitting error f(t) - y');
title('Fitting Error');
```

The optimal values of a_0, a_1, a_2, b_1, b_2 and E are:

$$a_0 = 1.0098, \quad a_1 = 0.6119, \quad a_2 = 0.1135, \quad b_1 = -0.4146, \quad b_2 = 0.0485, \quad E = 0.0233$$

The figures generated by the code are shown below.





Additional Exercise 5.6

Total variation image interpolation. A grayscale image is represented as an $m \times n$ matrix of intensities U^{orig}. You are given the values U_{ij}^{orig} for $(i, j) \in \mathcal{K}$, where $\mathcal{K} \subset \{1, \ldots, m\} \times \{1, \ldots, n\}$. Your job is to *interpolate* the image, by guessing the missing values. The reconstructed image will be represented by $U \in \mathbb{R}^{m \times n}$, where U satisfies the interpolation conditions $U_{ij} = U_{ij}^{\text{orig}}$ for $(i, j) \in \mathcal{K}$.

The reconstruction is found by minimizing a roughness measure subject to the interpolation conditions. One common roughness measure is the ℓ_2 variation (squared),

$$\sum_{i=2}^{m} \sum_{j=1}^{n} (U_{ij} - U_{i-1,j})^2 + \sum_{i=1}^{m} \sum_{j=2}^{n} (U_{ij} - U_{i,j-1})^2$$

Another method minimizes instead the *total variation*,

$$\sum_{i=2}^{m} \sum_{j=1}^{n} |U_{ij} - U_{i-1,j}| + \sum_{i=1}^{m} \sum_{j=2}^{n} |U_{ij} - U_{i,j-1}|$$

Evidently both methods lead to convex optimization problems.

Carry out ℓ_2 and total variation interpolation on the problem instance with data given in `tv_img_interp.m`.

This will define m, n, and matrices `Uorig` and `Known`. The matrix `Known` is $m \times n$, with (i,j) entry one if $(i,j) \in \mathcal{K}$, and zero otherwise. The mfile also has skeleton plotting code. (We give you the entire original image so you can compare your reconstruction to the original; obviously your solution cannot access U_{ij}^{orig} for $(i,j) \notin \mathcal{K}$.)

Answer: This problem mostly involves programming, but we can set up a quick example of an optimization problem using ℓ_2 variation.

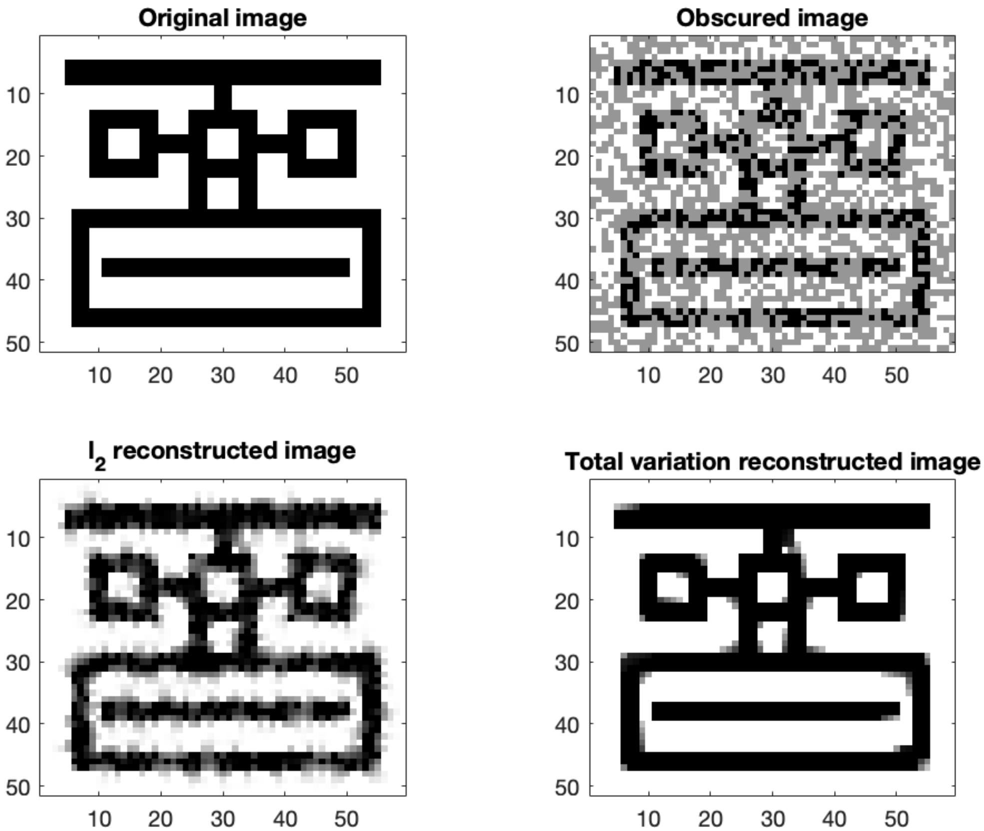
$$\begin{array}{ll} \underset{U}{\text{minimize}} & \sum_{i=2}^{m}\sum_{j=1}^{n}(U_{ij} - U_{i-1,j})^2 + \sum_{i=1}^{m}\sum_{j=2}^{n}(U_{ij} - U_{i,j-1})^2 \\ \text{subject to} & U_{ij} = U_{ij}^{\mathrm{orig}} \quad \forall (i,j) \in \mathcal{K} \end{array}$$

The MATLAB code using the skeleton code provided is as follows:

```
% tv_img_interp.m
% Total variation image interpolation.
% Defines m, n, Uorig, Known.

% Load original image.
Uorig = double(imread('tv_img_interp.png'));

[m, n] = size(Uorig);

% Create 50% mask of known pixels.
rand('state', 1029);
Known = rand(m,n) > 0.5;

%%%%% Put your solution code here

% Calculate and define Ul2 and Utv

% L2 variation interpolation
cvx_begin
    variable Ul2(m, n)
    minimize( sum(sum((Ul2(2:m, :) - Ul2(1:m-1, :)).^2)) + sum(sum((Ul2(:,
        2:n) - Ul2(:, 1:n-1)).^2)) )
    subject to
        Ul2(Known) == Uorig(Known);
cvx_end

% Total variation interpolation
cvx_begin
    variable Utv(m, n)
    minimize( sum(sum(abs(Utv(2:m, :) - Utv(1:m-1, :)))) + sum(sum(abs(Utv
        (:, 2:n) - Utv(:, 1:n-1)))) )
    subject to
        Utv(Known) == Uorig(Known);
cvx_end

```

```
%%%%%


% Placeholder:
%Ul2 = ones(m, n);
%Utv = ones(m, n);

%%%%%

% Graph everything.
figure(1); cla;
colormap gray;

subplot(221);
imagesc(Uorig)
title('Original image');
axis image;

subplot(222);
imagesc(Known.*Uorig + 256-150*Known);
title('Obscured image');
axis image;

subplot(223);
imagesc(Ul2);
title('l_2 reconstructed image');
axis image;

subplot(224);
imagesc(Utv);
title('Total variation reconstructed image');
axis image;
```

The figure generated by the code is shown below.



Additional Exercise 5.13

Fitting with censored data. In some experiments there are two kinds of measurements or data available: The usual ones, in which you get a number (say), and *censored data*, in which you don't get the specific number, but are told something about it, such as a lower bound. A classic example is a study of lifetimes of a set of subjects (say, laboratory mice). For those who have died by the end of data collection, we get the lifetime. For those who have not died by the end of data collection, we do not have the lifetime, but we do have a lower bound, i.e., the length of the study. These are the censored data values.

We wish to fit a set of data points,

$$(x^{(1)}, y^{(1)}), \dots, (x^{(K)}, y^{(K)})$$

with $x^{(k)} \in \mathbb{R}^n$ and $y^{(k)} \in \mathbb{R}$, with a linear model of the form $y \approx c^\top x$. The vector $c \in \mathbb{R}^n$ is the model parameter, which we want to choose. We will use a least-squares criterion, i.e., choose c to minimize

$$J = \sum_{k=1}^{K} \left(y^{(k)} - c^\top x^{(k)} \right)^2$$

Here is the tricky part: some of the values of $y^{(k)}$ are censored; for these entries, we have only a (given) lower bound. We will re-order the data so that $y^{(1)}, \dots, y^{(M)}$ are given (i.e., uncensored), while $y^{(M+1)}, \dots, y^{(K)}$ are all censored, i.e., unknown, but larger than D, a given number. All the values of $x^{(k)}$ are known.

Part (a)

Explain how to find c (the model parameter) and $y^{(M+1)}, \dots, y^{(K)}$ (the censored data values) that minimize J.

Answer: Since our goal here is to fit a linear model $y \approx c^\top x$ to the data, we can write the least-squares criterion as

$$J = \sum_{k=1}^{K} \left(y^{(k)} - c^\top x^{(k)} \right)^2$$

However, since we have censored data points beyond M, we can modify the objective function to account for this. Separating out the censored data points, we cna then add a constraint that they should be at least D.

Let's denote a letter, say $z \in \mathbb{R}^{K-M}$, to represent the censored data points. Then, we can write the objective function as

$$J = \sum_{k=1}^{M} \left(y^{(k)} - c^\top x^{(k)} \right)^2 + \sum_{k=M+1}^{K} \left(z^{(k-M)} - c^\top x^{(k)} \right)^2$$

where the second part handles the censored points with the constraint $z^{(k-M)} \geq D$.

Combining this, we get the quadratic program

$$\begin{array}{ll} \underset{c,\, z}{\text{minimize}} & \sum_{k=1}^{M} \left(y^{(k)} - c^\top x^{(k)} \right)^2 + \sum_{k=M+1}^{K} \left(z^{(k-M)} - c^\top x^{(k)} \right)^2 \\ \text{subject to} & z^{(k-M)} \geq D \quad \forall k = M+1, \dots, K \end{array}$$

Part (b)

Carry out the method of part (a) on the data values in `cens_fit_data.m`. Report $\hat{c}$, the value of c found using this method.

Also find $\hat{c}_{\text{ls}}$, the least-squares estimate of c obtained by simply ignoring the censored data samples, i.e., the least-squares estimate based on the data

$$(x^{(1)}, y^{(1)}), \dots, (x^{(M)}, y^{(M)}).$$

The data file contains c_{true}, the true value of c, in the vector `c_true`. Use this to give the two relative errors

$$\frac{\|c_{\text{true}} - \hat{c}\|_2}{\|c_{\text{true}}\|_2}, \quad \frac{\|c_{\text{true}} - \hat{c}_{\text{ls}}\|_2}{\|c_{\text{true}}\|_2}.$$

Answer: The MATLAB code using the skeleton code provided is as follows:

```
% data for censored fitting problem.
randn('state',0);

n = 20; % dimension of x's
```

```
M = 25; % number of non-censored data points
K = 100; % total number of points
c_true = randn(n,1);
X = randn(n,K);
y = X'*c_true + 0.1*(sqrt(n))*randn(K,1);

% Reorder measurements, then censor
[y, sort_ind] = sort(y);
X = X(:,sort_ind);
D = (y(M)+y(M+1))/2;
y = y(1:M);

% ---------------------------------------------
% Solution
% ---------------------------------------------

% Separate uncensored and censored data
X_uncensored = X(:, 1:M);                % (n x M)
X_censored = X(:, M+1:K);                % (n x (K - M))

% Number of censored data points
num_censored = K - M;

cvx_begin
    variables c(n) z(num_censored) % minimize over c and z because we don't know censored data
    minimize( sum_square(y - X_uncensored' * c) + sum_square(z - X_censored' * c) )
    subject to
        z >= D
cvx_end
% Assign the estimated c to c_hat
c_hat = c;

% Least squares method
cvx_begin
    variable c(n) % we only work with the uncensored data
    minimize( sum_square(y - X_uncensored' * c) )
cvx_end
% Assign the estimated c to c_ls
c_ls_hat = c;

% Display the results
disp('True, Estimated, and Least-Squares Estimates of c:');
disp([c_true c_hat c_ls_hat]);

% Compute relative errors
c_hat_relerr = norm(c_hat - c_true) / norm(c_true);
c_ls_relerr = norm(c_ls_hat - c_true) / norm(c_true);

% Display the relative errors
```

```
fprintf('Relative Error for c_hat: %.4f\n', c_hat_relerr);
fprintf('Relative Error for c_ls_hat: %.4f\n', c_ls_relerr);
```

The output obtained from running the script is as follows:

```
        True, Estimated, and Least-Squares Estimates of c:
        -0.4326   -0.2946   -0.3476
        -1.6656   -1.7541   -1.7955
         0.1253    0.2589    0.2000
         0.2877    0.2241    0.1672
        -1.1465   -0.9917   -0.8357
         1.1909    1.3017    1.3005
         1.1892    1.4262    1.8276
        -0.0376   -0.1554   -0.5612
         0.3273    0.3785    0.3686
         0.1746    0.2261   -0.0454
        -0.1867   -0.0826   -0.1096
         0.7258    1.0427    1.5265
        -0.5883   -0.4648   -0.4980
         2.1832    2.1942    2.4164
        -0.1364   -0.3586   -0.5563
         0.1139   -0.1973   -0.3701
         1.0668    1.0194    0.9900
         0.0593   -0.1186   -0.2539
        -0.0956   -0.1211   -0.1762
        -0.8323   -0.7523   -0.4349


     Relative Error for c_hat: 0.1784
     Relative Error for c_ls_hat: 0.3907
```

The relative error for $\hat{c}$ is 0.1784 and for $\hat{c}_{\text{ls}}$ is 0.3907. The output also has a table showing the c_{true}, $\hat{c}$, and $\hat{c}_{\text{ls}}$ values.

Additional Exercise 5.15

Learning a quadratic pseudo-metric from distance measurements. We are given a set of N pairs of points in $\mathbb{R}^n$, $x_1, \dots, x_N$, and $y_1, \dots, y_N$, together with a set of distances $d_1, \dots, d_N > 0$.

The goal is to find (or estimate or learn) a quadratic pseudo-metric d,

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

with $P \in \mathbb{S}^n_+$, 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 $P \in \mathbb{S}^n_+$ that minimizes the mean squared error objective

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

Part (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.

Answer: We cna show that this is a convex problem in P given $P \in \mathbb{S}^n_+$.

Let's take the $(d_i - \sqrt{(x_i - y_i)^\top P(x_i - y_i)})^2$ term and expand it out:

$$(d_i - \sqrt{(x_i - y_i)^\top P(x_i - y_i)})^2 = d_i^2 - 2d_i\sqrt{(x_i - y_i)^\top P(x_i - y_i)} + (x_i - y_i)^\top P(x_i - y_i)$$

With respect to P, d_i^2 is a constant term.

The term $(x_i - y_i)^\top P(x_i - y_i)$ is linear because if we were to take the trace of the expression, we would get $\mathrm{tr}((x_i - y_i)P(x_i - y_i)^\top)$ which is linear in P. (This is $(x_i - y_i)^\top(x_i - y_i)$ is a constant matrix and P is linear in the trace operator.)

Finally, the term $-2 \cdot d_i\sqrt{(x_i - y_i)^\top P(x_i - y_i)}$ is convex. This is because we have a linear or affine function inside a square root which is a concave function. The total term $d_i\sqrt{(x_i - y_i)^\top P(x_i - y_i)}$ is concave because an affine function does not change the concavity of a function. Finally, the negative sign flips the concavity of the function, making it convex.

Overall then, we have shown that a constant, a linear term, and a convex term are all present in the objective function. This means that the objective function is convex in P.

So, we can set up the optimization problem as

$$\begin{array}{ll} \underset{P}{\text{minimize}} & \frac{1}{N}\sum_{i=1}^{N}(d_i - \sqrt{(x_i - y_i)^\top P(x_i - y_i)})^2 \\ \text{subject to} & P \geq 0 \end{array}$$

One thing to note is that since P is constrained to be a symmetric positive semidefinite matrix, the problem is a semidefinite program.

Part (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).

Answer: This problem can be set up in MATLAB as stated above. The code is as follows:

```
%% data for learning a quadratic metric
% provides X, Y, d, X_test, Y_test, d_test
cvx_clear;
rand('seed',0);
randn('seed',0);
n = 5; % dimension
N = 100; % number of distance samples
N_test = 10;

X = randn(n,N);
Y = randn(n,N);
X_test = randn(n,N_test);
Y_test = randn(n,N_test);

P =randn(n,n);
P = P*P'+eye(n);
sqrtP = sqrtm(P);

d = norms(sqrtP*(X-Y)); % exact distances
d = pos(d+randn(1,N)); % add noise and make nonnegative
d_test = norms(sqrtP*(X_test-Y_test));
d_test = pos(d_test+randn(1,N_test));

clear P sqrtP;

% Compute difference vectors for training and test data
Diff = X - Y; % Training data
Diff_test = X_test - Y_test; % Test data

% Solve optimization problem
cvx_begin SDP
    variable P(n, n) symmetric
    expression f
    f = 0;
    for i = 1:N
        f = f + (d(i)^2) - 2 * d(i) * sqrt(quad_form(Diff(:, i), P)) +
            quad_form(Diff(:, i), P);
    end
    minimize (f / N)
    subject to
        P >= 0; % Enforce P is positive semidefinite
cvx_end

% Mean squared distance error on training data
% Compute (P * Diff) which results in an n x N matrix
% Element-wise multiply by Diff to get element-wise products
```

```
% Sum over rows to get a 1 x N vector of quadratic forms
% Take square roots to get the estimated distances
d_hat_train = sqrt(sum((P * Diff) .* Diff, 1))';

% Mean squared distance error on test data
d_hat_test = sqrt(sum((P * Diff_test) .* Diff_test, 1))';

% Compute Mean Squared Errors
MSE_train = mean((d' - d_hat_train).^2);
MSE_test = mean((d_test' - d_hat_test).^2);

% Report
disp('MSE on training data:');
disp(MSE_train);
disp('MSE on test data:');
disp(MSE_test);
```

The output of the code is as follows:

```
MSE on training data:
0.886688

MSE on test data:
0.826620
```

The mean squared error on the training data is 0.886688 and the mean squared error on the test data is 0.826620.