Final Report

ECE 508
Convex Optimization

Ataher Sams

UIN: 662464270

University of Illinois Chicago

Fall 2024

Contents

1 15.2

Problem 2:
15.2 → SINR maximization

Solution
We have to maximize the minimum SINR among all the receivers. Considering our transmit powers as $p = [p_1, p_2, p_3, p_4, p_5]^T$, we have the following constraints given-

$$0 \leq p_j \leq 3, \quad \text{for } j = 1, \dots, 5$$

$$p_1 + p_2 \leq 4, \quad p_3 + p_4 + p_5 \leq 6$$

$$\sum_{j=1}^{5} G_{ij} p_j \leq 5 \quad \forall \text{receiver } i$$

$$\text{SINR}_i = \frac{G_{ii} p_i}{\sigma + \sum_{j \neq i} G_{ij} p_j} \geq \gamma$$

however, we can rewrite the last constraint as $G_{ii} p_i - \gamma \sum_{j \neq i} G_{ij} p_j \geq \gamma \sigma^2$, and this is non-convex.

As per hints, now we will use bi-section method to solve it. At each iteration, we try to find p for a fixed γ subject to satisfying all power constraints, total received power constraints, and linearized SINR constraint. We will set `gamma_min = 0`, `gamma_max = 500`, `tolerance = 0.05`. Then we compute their midpoint by, `gamma = (gamma_min + gamma_max) / 2`. after that, we check feasibility through the convex optimization problem.
If feasible, we set `gamma_min = gamma`, and store the solution value p as the probable optimal power; otherwise set `gamma_max = gamma`.
We stop either when `gamma_max - gamma_min  > tolerance` or `iteration  < max_iterations`

Output:
Maximum SINR value: 1.6785
Optimal transmitter powers:
$\mathrm{p}_1 = 2.1085$
$p_2 = 1.8739$
$p_3 = 1.6380$
$p_4 = 2.3736$
$p_5 = 1.8057$

SINR values for each receiver:
$\mathrm{SINR}_1 = 1.6836$
$SINR_2 = 1.6852$
$SINR_3 = 1.6843$
$SINR_4 = 1.6877$
$SINR_5 = 1.6955$

Code

```
import numpy as np
import cvxpy as cp

n = 5
group1 = [0, 1]
group2 = [2, 3, 4]
p_max = 3
group1_power_limit = 4
group2_power_limit = 6
sigma = 0.5
received_power_limit = 5

G = np.array([
    [1.0, 0.1, 0.2, 0.1, 0.0],
    [0.1, 1.0, 0.1, 0.1, 0.0],
    [0.2, 0.1, 2.0, 0.2, 0.2],
    [0.1, 0.1, 0.2, 1.0, 0.1],
    [0.0, 0.0, 0.2, 0.1, 1.0]
])

# Initialize gamma bounds
gamma_min = 0
gamma_max = 500

# Bisection parameters
tolerance = 0.05
max_iterations = 50

# Bisection algorithm
iteration = 0
```

```python
p_optimal = None  # To store the optimal power allocation
gamma_optimal = None  # To store the optimal gamma value

while gamma_max - gamma_min > tolerance and iteration < max_iterations:
    iteration += 1
    gamma = (gamma_min + gamma_max) / 2

    # Define variables
    p = cp.Variable(n)
    constraints = []

    # Individual power constraints
    constraints += [p >= 0, p <= p_max]

    # Group power constraints
    constraints += [cp.sum(p[group1]) <= group1_power_limit]
    constraints += [cp.sum(p[group2]) <= group2_power_limit]

    # Total received power constraints for each receiver
    for i in range(n):
        constraints += [G[i, :] @ p <= received_power_limit]

    # SINR constraints for each receiver
    for i in range(n):
        interference = G[i, :] @ p - G[i, i] * p[i]
        sinr_constraint = G[i, i] * p[i] - gamma * interference >= gamma
    * sigma
        constraints += [sinr_constraint]

    # Solve the feasibility problem
    prob = cp.Problem(cp.Minimize(0), constraints)
    result = prob.solve(solver=cp.SCS, verbose=False)

    if prob.status == cp.OPTIMAL or prob.status == cp.OPTIMAL_INACCURATE:
        # Feasible solution found
        gamma_min = gamma  # Update lower bound
        p_optimal = p.value
        gamma_optimal = gamma
    else:
        # Infeasible, update upper bound
        gamma_max = gamma

# Round final gamma to four decimal places for reporting
gamma_optimal = round(gamma_min, 4)

# Print the results
print(f"Maximum SINR value: {gamma_optimal}")
```

```python
print("Optimal transmitter powers:")
for j in range(n):
    print(f"p_{j+1} = {p_optimal[j]:.4f}")

# Compute and display the SINR values for each receiver
print("\nSINR values for each receiver:")
for i in range(n):
    interference = G[i, :] @ p_optimal - G[i, i] * p_optimal[i]
    sinr_i = (G[i, i] * p_optimal[i]) / (sigma + interference)
    print(f"SINR_{i+1} = {sinr_i:.4f}")
```

2 15.14

Problem 2:
15.14 → Wireless communication power optimization

Solution

`15.14 (a)`

To compute the trade-off, we can minimize the total power and introduce SINR threshold for the data rate R.
As $R_i = \alpha \log(1 + s_i)$, we can define the threshold γ as-

$$\gamma = \exp\left(\frac{R}{\alpha}\right) - 1$$

So, our constraint on the SINR will be -

$$\frac{G_{ii}p_i}{\sigma_i^2 + \sum_{\substack{j=1 \\ j\neq i}}^n G_{ij}p_j} \geq \gamma, \quad \forall i = 1, \dots, n$$

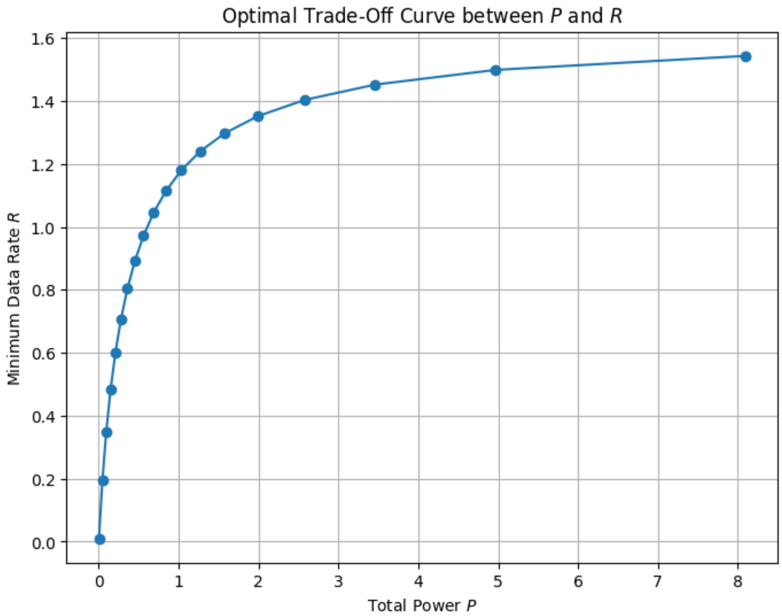
which can be written as a linear constraint,

$$G_{ii}p_i - \gamma \sum_{\substack{j=1 \\ j\neq i}}^n G_{ij}p_j \geq \gamma\sigma_i^2, \quad \forall i = 1, \dots, n$$

So, the formulation for the convex optimization problem will be-

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^n p_i \\ \text{subject to} & \frac{G_{ii}p_i}{\sigma_i^2 + \sum_{\substack{j=1 \\ j\neq i}}^n G_{ij}p_j} \geq \gamma, \quad \forall i = 1, \dots, n \\ & p^{\min} \leq p_i \leq p^{\max}, \quad \forall i = 1, \dots, n \end{array}$$

We can vary the γ over a feasible range and solve the optimization problem to obtain optimal P for the trade-off curve.



15.14 (b)

```
import numpy as np
import cvxpy as cp
import matplotlib.pyplot as plt

# Problem Data
n = 10
alpha = 1
sigma2 = np.array([0.82, 0.91, 0.16, 0.18, 0.28, 0.2, 0.04, 0.07, 0.36,
    0.97])
pmin = 0
pmax = 1.5
G = np.array([
    [13.0, 0.47, 0.84, 0.95, 0.2, 0.05, 0.8, 0.09, 0.48, 0.71],
    [0.54, 29.0, 0.91, 0.8, 0.87, 0.48, 0.51, 0.95, 0.06, 0.44],
    [0.26, 0.35, 27.0, 0.12, 0.65, 0.67, 0.39, 0.78, 0.47, 0.28],
    [0.91, 0.77, 0.72, 11.0, 0.59, 0.84, 0.48, 0.18, 0.38, 0.34],
    [0.17, 0.04, 0.12, 0.08, 21.0, 0.17, 0.07, 0.08, 0.98, 0.24],
    [0.66, 0.27, 0.77, 0.78, 0.74, 29.0, 0.48, 0.68, 0.22, 0.07],
    [0.58, 0.07, 0.8, 0.1, 0.55, 0.78, 18.0, 0.93, 0.86, 0.11],
    [0.87, 0.16, 0.04, 0.84, 0.48, 0.51, 0.45, 11.0, 0.07, 0.89],
    [0.97, 0.61, 0.48, 0.18, 0.6, 0.62, 0.64, 0.65, 29.0, 0.51],
    [0.77, 0.14, 0.9, 0.31, 0.79, 0.31, 0.67, 0.6, 0.85, 23.0]
])

# Range of gamma values
gamma_min = 0.01
gamma_max = 10  # Initial guess for maximum gamma
```

```
gamma_values = np.linspace(gamma_min, gamma_max, 50)

# Storage for results
P_values = []
R_values = []
gamma_feasible = []

for gamma in gamma_values:
    # Define variables
    p = cp.Variable(n)

    # Constraints
    constraints = []
    for i in range(n):
        interference = cp.sum([G[i, j] * p[j] for j in range(n) if j != i
])
        sinr_numerator = G[i, i] * p[i]
        sinr_denominator = sigma2[i] + interference
        constraints.append(sinr_numerator - gamma * sinr_denominator >=
0)
        constraints.append(pmin <= p[i])
        constraints.append(p[i] <= pmax)

    # Objective
    objective = cp.Minimize(cp.sum(p))

    # Problem definition
    prob = cp.Problem(objective, constraints)

    prob.solve()


    # Check feasibility
    if prob.status in [cp.OPTIMAL, cp.OPTIMAL_INACCURATE]:
        total_power = sum(p.value)
        R = alpha * np.log(1 + gamma)
        P_values.append(total_power)
        R_values.append(R)
        gamma_feasible.append(gamma)
    else:
        # Infeasible for this gamma, adjust gamma_max
        gamma_max = gamma
        break  # No need to try larger gamma values

# Convert lists to numpy arrays
P_values = np.array(P_values)
R_values = np.array(R_values)
```

```python

# Plotting the trade-off curve
plt.figure(figsize=(8, 6))
plt.plot(P_values, R_values, marker='o', linestyle='-')
plt.xlabel('Total Power $P$')
plt.ylabel('Minimum Data Rate $R$')
plt.title('Optimal Trade-Off Curve between $P$ and $R$')
plt.grid(True)
plt.show()
```

3 17.4

Problem 3:
17.4 → Bounding portfolio risk with incomplete covariance information

Solution

We want to find out the worst-case variance Σ^2_{wc} of the portfolio return, which can be defined as-

$$\sigma^2_{\text{wc}} = \max_{\Sigma} x^\top \Sigma x$$

subject to:
→ The known values and signs of Σ.
$[\Sigma_{12} \geq 0, \Sigma_{13} \geq 0, \Sigma_{14}, \Sigma_{23} \leq 0, \Sigma_{24} \leq 0, \Sigma_{34} \geq 0\ ]$
→ The positive semidefiniteness of $\Sigma(\Sigma \succeq 0)$.
Here, the objective function is in a quardratic form.
For the diagonal variance, we know,

When Σ is diagonal, all off-diagonal elements are zero $\sigma^2_{\text{diag}} = \sum_{i=1}^{n} x_i^2 \Sigma_{ii}$

So, not considering the affect of covariance matrix can lead to significant under or overestimation of portfolio risk (For our case, it was underestimating)

Output:
Worst-case variance: 0.015166
Diagonal variance: 0.007750
Ratio: 1.956929
Optimal Sigma:

$$\begin{bmatrix} 2.00000000e-01 & 8.82242409e-02 & -1.10125774e-06 & 9.32336197e-02 \\ 8.82242409e-02 & 1.00000000e-01 & -1.20196773e-01 & -1.03476482e-07 \\ -1.10125774e-06 & -1.20196773e-01 & 3.00000000e-01 & 3.81381837e-02 \\ 9.32336197e-02 & -1.03476482e-07 & 3.81381837e-02 & 1.00000000e-01 \end{bmatrix}$$

Code

```
import cvxpy as cp
import numpy as np

# Portfolio weights and known diag elements of variacne matrix
x = np.array([0.1, 0.2, -0.05, 0.1])
diag_elements = np.array([0.2, 0.1, 0.3, 0.1])

```

```
# Variables for off-diagonal elements
sigma_12 = cp.Variable()
sigma_13 = cp.Variable()
sigma_14 = cp.Variable()
sigma_23 = cp.Variable()
sigma_24 = cp.Variable()
sigma_34 = cp.Variable()

# Construct Sigma matrix
Sigma = cp.bmat([
    [diag_elements[0], sigma_12, sigma_13, sigma_14],
    [sigma_12, diag_elements[1], sigma_23, sigma_24],
    [sigma_13, sigma_23, diag_elements[2], sigma_34],
    [sigma_14, sigma_24, sigma_34, diag_elements[3]]
])

# Sign constraints
constraints = [
    sigma_12 >= 0,  # Sigma[1,2] >= 0
    sigma_13 >= 0,  # Sigma[1,3] >= 0
    sigma_23 <= 0,  # Sigma[2,3] <= 0
    sigma_24 <= 0,  # Sigma[2,4] <= 0
    sigma_34 >= 0   # Sigma[3,4] >= 0
]

# Solve optimization problem
prob = cp.Problem(
    cp.Maximize(cp.quad_form(x, Sigma)),
    constraints + [Sigma >> 0]
)
prob.solve()

# Extract results
Sigma_wc = np.array([
    [diag_elements[0], sigma_12.value, sigma_13.value, sigma_14.value],
    [sigma_12.value, diag_elements[1], sigma_23.value, sigma_24.value],
    [sigma_13.value, sigma_23.value, diag_elements[2], sigma_34.value],
    [sigma_14.value, sigma_24.value, sigma_34.value, diag_elements[3]]
])
diag_var = np.sum(x**2 * diag_elements)

print(f"Worst-case variance: {prob.value:.6f}")
print(f"Diagonal variance: {diag_var:.6f}")
print("\nOptimal Sigma:\n", Sigma_wc)
```

4 17.16

Problem 4:

17.16 → Option price bounds

Solution

Our objective is to find out minimum and maximum possible prices for the collar option with absence of arbitrage.

For Call option, payoff will be `max(0, S-K)`, for Put option, payoff will be `max(K-S, 0)`. Considering p_i is defined as risk free probabilites, the constraint should follow probability constraints-

$$\sum_{i=1}^{200} p_i = 1, \quad p_i \geq 0 \quad \forall i$$

Now, the present value of the expected future price of underlying asset must equal its current price. So,

$$\frac{1}{r}\sum_{i=1}^{200} p_i S^{(i)} = S_0$$

This is true for each traded option as well,

$$\frac{1}{r}\sum_{i-1}^{200} p_i \times \max\left(S^{(i)} - K, 0\right) = c, \quad \frac{1}{r}\sum_{i-1}^{200} p_i \times \max\left(K - S^{(i)}, 0\right) = p$$

given c and p are current market price for call and put option.

Our objective function will be

$$\text{Minimize/Maximize} \quad \frac{1}{r}\sum_{i=1}^{200} p_i \times \text{ Payoff }_{\text{collar}}^{(i)}$$

where,

$$\text{Payoff collar } = \begin{cases} C - S_0 & \text{if } S^{(i)} > C \\ S^{(i)} - S_0 & \text{if } F \leq S^{(i)} \leq C \\ F - S_0 & \text{if } S^{(i)} < F \end{cases}$$

Output

Minimum collar price: 0.032619
Maximum collar price: 0.064950

Code

```
import cvxpy as cp
import numpy as np

# Market parameters
r, S0 = 1.05, 1.0
F, C = 0.9, 1.15

# Generate scenarios and traded options data
S = np.linspace(0.5, 2.0, 200)
options = [
    {'type': 'call', 'strike': 1.1, 'price': 0.06},
    {'type': 'call', 'strike': 1.2, 'price': 0.03},
    {'type': 'put',  'strike': 0.8, 'price': 0.02},
    {'type': 'put',  'strike': 0.7, 'price': 0.01}
]

# Calculate option payoffs
def get_option_payoffs(S, options):
    payoffs = []
    for opt in options:
        payoff = np.maximum(S - opt['strike'], 0) if opt['type'] == 'call' \
                 else np.maximum(opt['strike'] - S, 0)
        payoffs.append(payoff)
    return np.array(payoffs)

# Calculate collar payoff
def get_collar_payoff(S, S0, F, C):
    payoff = np.minimum(np.maximum(S - S0, F - S0), C - S0)
    return payoff

# All payoffs
payoff_options = get_option_payoffs(S, options)
payoff_collar = get_collar_payoff(S, S0, F, C)

# Optimization problem
p = cp.Variable(len(S), nonneg=True)

# Build constraints list
constraints = [cp.sum(p) == 1]                    # Probability sum
constraints.append((1/r) * (p @ S) == S0)         # expected future price of underlying asset must equal its current price

# Add option pricing constraints
for payoff, opt in zip(payoff_options, options):
```

```
    constraints.append((1/r) * (p @ payoff) == opt['price']) # expected
    future price of traded options must equal its current price

# Solve for min and max collar prices
collar_price = (1/r) * (p @ payoff_collar)
results = {}
for objective in [cp.Minimize, cp.Maximize]:
    prob = cp.Problem(objective(collar_price), constraints)
    prob.solve()
    results[objective.__name__] = prob.value

print(f"Minimum collar price: {results['Minimize']:.6f}")
print(f"Maximum collar price: {results['Maximize']:.6f}")
```

5 16.14

Problem 5:
16.14 → Dual of an optimal control problem

Solution

`16.14 (a)`

$$L(x,u,\nu)=\sum_{t=1}^{T}\frac{1}{2}\left|u_t\right|^2 \quad +\nu_0^T\left(x_1-x_{\text{init}}\right)+\nu_{T+1}^T\left(x_{T+1}-x_{\text{term}}\right) \ +\sum_{t=1}^{T}\nu_t^T\left(x_{t+1}-Ax_t-Bu_t\right).$$

`16.14 (b)`

The dual function $g(\nu)=\inf_{x,u}L(x,u,\nu)$ we need to minimize the Lagrangian $L(x,u,\nu)$ over the primal variables x and u The terms in L with u_t,

$$L_u\left(u_t\right)=\frac{1}{2}\left\|u_t\right\|^2-\nu_t^\top Bu_t$$

As per hints and reference[1]

$$\begin{aligned}\inf_{u_t}L_u\left(u_t\right)&=-\sup_{u_t}\left(\nu_t^\top Bu_t-\frac{1}{2}\left\|u_t\right\|^2\right)\\&=-\frac{1}{2}\left\|B^\top\nu_t\right\|_*^2\end{aligned}$$

Now, the terms involve x_t,

$$L_x=\sum_{t=1}^{T+1}\nu_{t-1}^\top x_t-\sum_{t=1}^{T}\nu_t^\top Ax_t$$

We will now take derivatives with respect to each x_t and set them equal to zero.

$$\text{For } t=1,\ldots,T: \quad \frac{\partial L_x}{\partial x_t}=\nu_{t-1}-A^\top\nu_t=0 \quad\rightarrow\quad \nu_{t-1}=A^\top\nu_t$$

$$\text{For } t=T+1: \quad \frac{\partial L_x}{\partial x_{T+1}}=\nu_T=0 \quad\rightarrow\quad \nu_T+\nu_{T+1}=0$$

So, the dual function becomes-

$$g(\nu)=-\nu_0^\top x_{\text{init}} \ -\nu_{T+1}^\top x_{\text{term}} \ -\sum_{t=1}^{T}\frac{1}{2}\left\|B^\top\nu_t\right\|_*^2 \tag{1}$$

[1]https://scoop.iwr.uni-heidelberg.de/teaching/2024ss/seminar-ausgewaehlte-kapitel-der-optimierung/conjugate$_functions.pdf$

where,

$$\begin{cases} \nu_{t-1} = A^\top \nu_t, \quad t = 1, \dots, T, \\ \nu_T + \nu_{T+1} = 0. \end{cases}$$

16.14 (c)

We can write $\nu_t = -(A^T)^{T-t+1}\nu_{T+1}$ for $t = 0, 1, ..., T$.

Substituting it into equation 1, we get:

$$g(\nu_{T+1}) = \nu_{T+1}^T(A^{T+1}x_{\text{init}} - x_{\text{term}}) - \frac{1}{2}\sum_{k=1}^{T} \|B^T(A^T)^k \nu_{T+1}\|_*^2$$

So, the dual problem becomes:

$$\underset{\nu_{T+1}}{\text{maximize}} \quad \nu_{T+1}^T(A^{T+1}x_{\text{init}} - x_{\text{term}}) - \frac{1}{2}\sum_{k=1}^{T} \|B^T(A^T)^k \nu_{T+1}\|_*^2$$

where $\nu_{T+1} \in \mathbb{R}^n$.

6 18.4

Problem 6:
18.4 → A structural optimization problem

Solution

Let's define $t = \sqrt{w^2 + h^2}, u = R^2 - r^2$ So, objective function will become : $2\pi ut$

Now, we need to transform the constraints to posynomials set less than or equal to 1, to fit GP standard form. The first and second constraint can be easily written as posynomials through-

$$\frac{F_1 t}{2h\sigma\pi u} \leq 1, \quad \frac{F_2 t}{2w\sigma\pi u} \leq 1$$

For the dimensions, we can define the bounds as -

$$\frac{w_{\min}}{w} \leq 1, \frac{w}{w_{\max}} \leq 1, \quad \frac{h_{\min}}{h} \leq 1, \frac{h}{h_{\max}} \leq 1$$

From $1.1r \leq R$, we derive $R^2 - r^2 \geq (1.1^2 - 1)\, r^2 = 0.21r^2$, so $\frac{0.21r^2}{u} \leq 1$. Since $R \leq R_{\max}$, and $R^2 = u + r^2$, we have $\frac{u+r^2}{R_{\max}^2} \leq 1$. The equality $t = \sqrt{w^2 + h^2}$ is approximated with $\frac{w^2+h^2}{t^2} \leq 1$ So, final optimization problem in GP form,

$$\begin{aligned}
\text{minimize} \quad & 2\pi ut \\
\text{subject to} \quad & \frac{F_1 t}{2h\sigma\pi u} \leq 1 \\
& \frac{F_2 t}{2w\sigma\pi u} \leq 1 \\
& \frac{w_{\min}}{w} \leq 1, \quad \frac{w}{w_{\max}} \leq 1 \\
& \frac{h_{\min}}{h} \leq 1, \quad \frac{h}{h_{\max}} \leq 1 \\
& \frac{0.21r^2}{u} \leq 1 \\
& \frac{u + r^2}{R_{\max}^2} \leq 1 \\
& \frac{w^2 + h^2}{t^2} \leq 1 \\
& R > 0, \quad r > 0, \quad w > 0, \quad h > 0, \quad t > 0, \quad u > 0
\end{aligned}$$

7 19.3

Problem 7:

19.3 → Utility versus latency trade-off in a network

Solution

First let us recap the variables-

Link traffic $t_i = \sum_{j=1}^{n} R_{ij} f_j$, Network utility $U(f) = \sum_{j=1}^{n} \log f_j$,

Link delay $d_i = \frac{1}{c_i - t_i}$, Flow latency $l_j = \sum_{i=1}^{m} R_{ij} d_i$

Maximum Flow Latency $L = \max\{l_1, l_2, \ldots, l_n\}$

`19.3 (a)`

If flow latency is not an issue, the only constraint for network utility is the link traffic and non-negative flow rate. So the constraint can be written as -

$$Rf \preceq c \quad \text{and} \quad f_j \geq 0 \quad \forall j$$

So the optimization problem will be to -

$$\begin{array}{ll} \text{maximize} & U(f) \\ \text{subject to} & \quad Rf \leq c, \\ & f_j > 0 \quad \forall j. \end{array}$$

`19.3 (b)`

Now we need to find flow rates to minimize L which is the maximum flow latency. If all flow rates are zero then link traffic ($t_i = 0$) will be zero, which will mean flow latency will be equal to $l_j = \sum_{i=1}^{m} R_{ij} d_i = \sum_{i=1}^{m} R_{ij} \left(\frac{1}{c_i}\right)$ The maximum flow latency $L^{\min}$ is the maximum l_j among all flows. So,

$$L^{\min} = \max_j \sum_{i=1}^{m} R_{ij} \left(\frac{1}{c_i}\right)$$

`19.3 (c)`

To find the trade-off we can model this as CVX problem that maximizes utility and put the latency as constraint. It will be similar to part (a) of this question with additional constraint based on the latency. The latency $l_j = \sum_{i=1}^{m} R_{ij} d_i$, we can replace d_i using the formula of link traffic. So, we will finally get the following convex optimization problem -

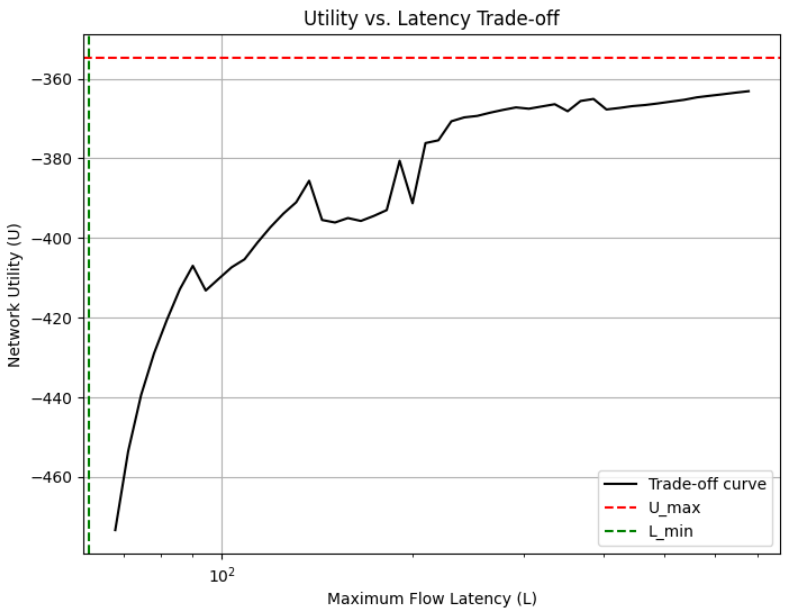
$$\begin{array}{ll} \text{maximize} & U(f) \\ \text{subject to} & \sum_{i=1}^{m} R_{ij}\left(\frac{1}{c_i - R_i f}\right) \leq L \\ & Rf \leq c, \\ & f_j > 0 \quad \forall j. \end{array}$$

19.3 (d)

Output

Maximum Utility U_{max}: -354.7965

Minimum Latency L_{min}: 61.7363



Code

```
import numpy as np
import cvxpy as cp
import scipy.io
import matplotlib.pyplot as plt

# Load data
data = scipy.io.loadmat('net_util_data.mat')
R, c = data['R'], data['c'].flatten()
m, n = R.shape

# 19.3 (a)
f = cp.Variable(n, pos=True)
```

```
prob = cp.Problem(cp.Maximize(cp.geo_mean(f)), [R @ f <= c])
prob.solve(solver=cp.SCS, eps=1e-4)
U_max = n * np.log(prob.value)

# 19.3 (b)
L_min = np.max(R.T @ (1 / c))

# 19.3 (d)
N = 50
ds = 1.10 * L_min * np.logspace(0, 1, N)
U_values = []

for d in ds:
    f = cp.Variable(n, pos=True)
    prob = cp.Problem(
        cp.Maximize(cp.geo_mean(f)),
        [R.T @ cp.inv_pos(c - R @ f) <= d]
    )

    prob.solve(solver=cp.SCS, eps=1e-4)
    U_values.append(n * np.log(prob.value) if prob.value else np.nan)


# Plot results
plt.figure(figsize=(8, 6))
plt.semilogx(ds, U_values, 'k-', label='Trade-off curve')
plt.axhline(y=U_max, color='r', linestyle='--', label='U_max')
plt.axvline(x=L_min, color='g', linestyle='--', label='L_min')
plt.xlabel('Maximum Flow Latency (L)')
plt.ylabel('Network Utility (U)')
plt.title('Utility vs. Latency Trade-off')
plt.grid(True)
plt.legend()
plt.show()

print(f"Maximum Utility U_max: {U_max:.4f}")
print(f"Minimum Latency L_min: {L_min:.4f}")
```

8 19.5

Problem 8:

19.5 → Network sizing

Solution

`19.5 (a)`

We need to find the convexity of the function $f(y, b)$, which represents the optimal value of a network flow problem. The original optimization problem:

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^{n} y_k \phi_k(|x_k|/y_k) \\ \text{subject to} & Ax = b \end{array}$$

The function $f(y, b)$ is convex jointly in y and b. $y_k \phi_k(|x_k|/y_k)$, is the perspective of the convex function $\phi_k(u)$, where $u = \frac{|x_k|}{y_k}$. The perspective of a convex function is jointly convex in (x_k, y_k) for $y_k > 0$. Since ϕ_k is convex and nondecreasing on $\mathbb{R}_+$, and $y_k > 0$, $f_k(x_k, y_k)$ is convex in (x_k, y_k). Also, the network constraint $Ax = b$ is affine in both x and b. Therefore, $f(y, b)$ is convex jointly in y and b.

`19.5 (b)`

We have to check the convexity of minimizing $g(y) + \mathbf{E}f(y, b)$. The formula for expected cost is given as:

$$\mathbf{E}f(y, b) = \sum_{j=1}^{m} \pi_j f(y, b^{(j)})$$

This minimization problem is convex for several reasons. First, each term $f(y, b^{(j)})$ is convex in y as shown is part (a). The expected value $\mathbf{E}f(y, b)$ is a positive weighted sum (with weights π_j) of these convex functions, preserving convexity. Adding this to the $g(y)$ yields a convex objective function. Since the constraint $y \succ 0$ defines a convex feasible set, the optimization problem will be convex.

9 20.8

Problem 9:

20.8 → Utility/power trade-off in a wireless network

Solution

First lets recap the variables,
Total utility $U(f) = \sum_{j=1}^{n} U_j(f_j)$, total transmit power $P = \mathbf{1}^T p = \sum_{i=1}^{m} p_i$,
Total traffic $t = Rf \preceq c$, link capacity $c_i = \alpha_i \log(1 + \beta_i p_i)$, transmit power limit $p \preceq p^{\max}$.

`20.8 (a)`

In order to find the optimal trade-off curve, we will introduce a variable λ in the range of $[0, \inf)$ where it controls the trade-off between utility and power.

So, our optimization problem will become -

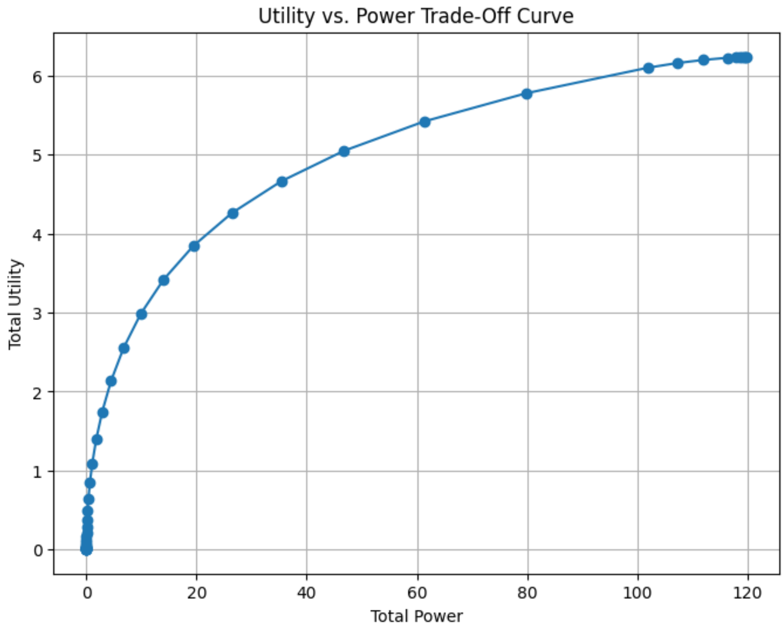
$$
\begin{array}{ll}
\max & \sum_{j=1}^{n} U_j(f_j) - \lambda \sum_{i=1}^{m} p_i \\
\text{s.t.} & Rf \preceq c \\
& c_i = \alpha_i \log(1 + \beta_i p_i), \quad \forall i \\
& 0 \le p_i \le p_i^{\max}, \quad \forall i \\
& f_j \ge 0, \quad \forall j
\end{array}
$$

As c_i is a function of p_i, we can reformulate and write -

$$
p_i = \frac{1}{\beta_i}\left(e^{c_i/\alpha_i} - 1\right)
$$

as p_i is a convex function of c_i (due to being exponential), our objective function now becomes,

$$
\begin{array}{ll}
\max & \sum_{j=1}^{n} U_j(f_j) - \lambda \sum_{i=1}^{m} \frac{1}{\beta_i}\left(e^{c_i/\alpha_i} - 1\right) \\
\text{s.t.} & Rf \preceq c \\
& 0 \le c_i \le c_i^{\max}, \quad \forall i \\
& c_i^{\max} = \alpha_i \log(1 + \beta_i p_i^{\max}), \quad \forall i \\
& f_j \ge 0, \quad \forall j
\end{array}
$$



20.8 (b)

Code

```
import numpy as np
import cvxpy as cp
import matplotlib.pyplot as plt


m = 20
n = 10

# Generate routing matrix R
np.random.seed(3)
R = np.round(np.random.rand(m, n))

# Parameters
p_max = 10  # Maximum transmit power for each link
alpha = np.ones(m)
beta = np.ones(m)
c_max = np.log(1 + beta * p_max)  # c_i^max = ln(1 + p_i^max)
lambda_values = np.logspace(-3, 3, num=50)  # Varying lambda from 1e-3 to
    1e3

# Storage for results
utility_values = []
power_values = []

for lam in lambda_values:
```

```
    # Variables
    f = cp.Variable(n, nonneg=True)
    c = cp.Variable(m)

    # Compute p_i as function of c_i
    p = cp.exp(c) - 1

    # Objective
    objective = cp.Maximize(cp.sum(cp.sqrt(f)) - lam * cp.sum(p))

    # Constraints
    constraints = [
        R @ f <= c,
        c >= 0,
        c <= c_max,
        p <= p_max,
    ]

    # Problem definition
    prob = cp.Problem(objective, constraints)

    prob.solve()

    prob.status == cp.OPTIMAL
    total_utility = sum(np.sqrt(f.value))
    total_power = sum(p.value)
    utility_values.append(total_utility)
    power_values.append(total_power)



# Plot the trade-off curve
plt.figure(figsize=(8, 6))
plt.plot(power_values, utility_values, marker='o')
plt.xlabel('Total Power')
plt.ylabel('Total Utility')
plt.title('Utility vs. Power Trade-Off Curve')
plt.grid(True)
plt.show()
```

10 20.6

Problem 10:

20.6 → AC power flow analysis via convex optimization

`20.8 (a)` The objective function is $\sum_{j=1}^{m} \psi_j \left(p_j\right)$, where

$$\psi_j(u) = \int_0^u \sin^{-1}\left(v/\kappa_j\right) dv$$

with the domain $\operatorname{dom} \psi_j = \left(-\kappa_j, \kappa_j\right)$.

The first derivative is $\psi_j'(u) = \sin^{-1}\left(\frac{u}{\kappa_j}\right)$, and the second derivative is

$$\psi_j''(u) = \frac{1}{\kappa_j \sqrt{1-\left(\frac{u}{\kappa_j}\right)^2}} \geq 0 \quad \text{for} \quad |u| < \kappa_j$$

We can see $\psi_j''(u)$ is non-negative within the domain $\left(-\kappa_j, \kappa_j\right)$ and $\psi_j'(u)$ is increasing function, so ψ_j is convex. Since the sum of convex functions is convex, the objective function itself is convex.

`20.8 (b)` The Lagrangian $\mathcal{L}(p, \nu)$ for the optimization problem is:

$$\mathcal{L}(p, \nu) = \sum_{j=1}^{m} \psi_j \left(p_j\right) + \nu^T (s - Ap)$$

At optimality, the solution must satisfy the primal constraints:

$$Ap^\star = s$$

$\frac{\partial \mathcal{L}}{\partial p_j} = \psi_j'\left(p_j^\star\right) - a_j^T \nu^\star = 0$ for all j where a_j is j-th column of A

As $\psi_j'\left(p_j\right) = \sin^{-1}\left(\frac{p_j}{\kappa_j}\right)$, we find,

$$p_j^\star = \kappa_j \sin\left(\phi_j\right)$$

So, the optimal solution $p^\star$ and $\phi = \nu^\star$ satisfy the DC power flow equations:

$$Ap^\star = s, \quad p^\star = \operatorname{diag}(\kappa) \sin\left(A^T \phi\right)$$

where, $\operatorname{diag}(\kappa)$ is a diagonal matrix with κ_j on its diagonal.