Lecture 04b - Robust Geometric Programming

.pull-left[

@luk036

2022-10-12

] .pull-right[

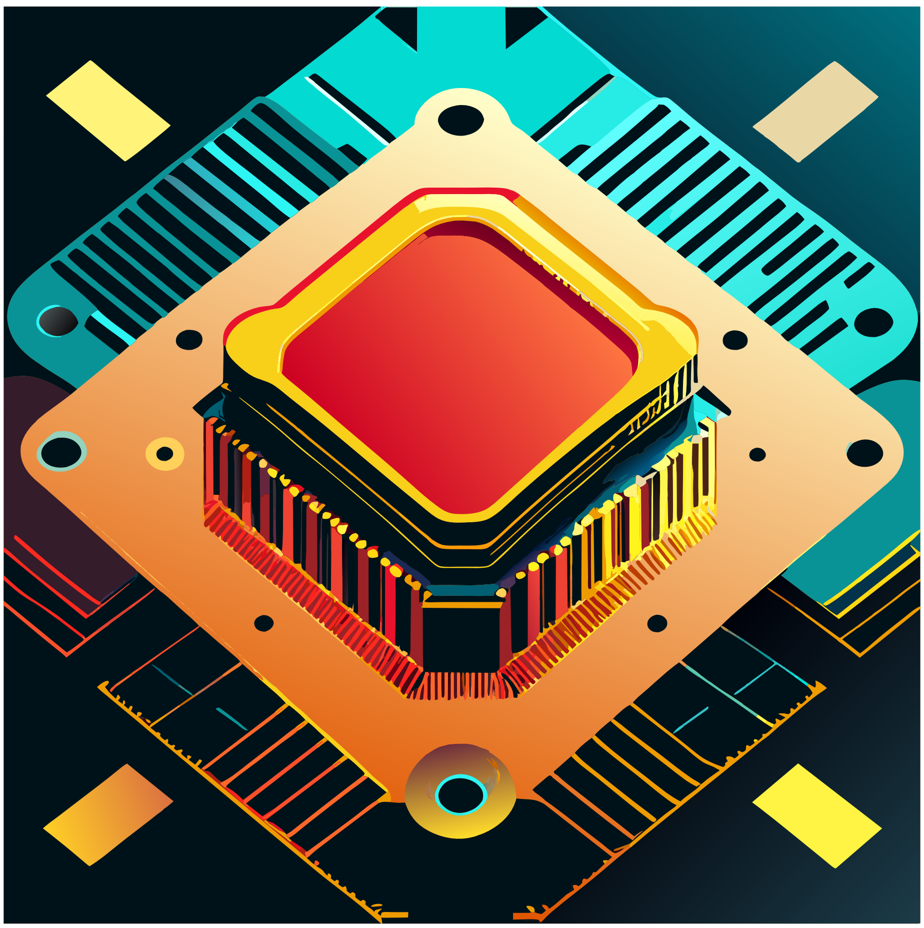

Figure 1: image

]

Outline

- Problem Definition for Robust Analog Circuit Sizing
- Robust Geometric Programming
- Affine Arithmetic
- Example: CMOS Two-stage Op-Amp
- Numerical Result
- Conclusions

Robust Analog Circuit Sizing Problem

- Given a circuit topology and a set of specification requirements:

.font-sm.mb-xs[

Constraint	Spec.	Units
Device Width	≥ 2.0	μm
Device Length	≥ 1.0	μm
Estimated Area	minimize	μm^2
⋮	⋮	⋮
CMRR	≥ 75	dB
Neg. PSRR	≥ 80	dB
Power	≤ 3	mW

]

- Find the worst-case design variable values that meet the specification requirements and optimize circuit performance.

Robust Optimization Formulation

- Consider

$$
\begin{array}{ll}
\text{minimize} & \sup_{q \in \mathbb{Q}} f_0(x, q), \\
\text{subject to} & f_j(x, q) \leq 0 \\
 & \forall q \in \mathbb{Q} \text{ and } j = 1, 2, \cdots, m,
\end{array}
$$

  where
  - $x \in \mathbb{R}^n$ represents a set of design variables (such as L, W),
  - q represents a set of varying parameters (such as T_{OX})
  - $f_j \leq 0$ represents the jth specification requirement (such as phase margin $\geq 60^{\circ}$).

Geometric Programming in Standard Form

- We further assume that $f_i(x, q)$'s are convex for all $q \in \mathbb{Q}$.
- Geometric programming is an optimization problem that takes the following standard form:

$$\begin{array}{lll} \text{minimize} & p_0(y) & \\ \text{subject to} & p_i(y) \leq 1, & i = 1, \ldots, l \\ & g_j(y) = 1, & j = 1, \ldots, m \\ & y_k > 0, & k = 1, \ldots, n, \end{array}$$

  where
  - p_i's are posynomial functions and g_j's are monomial functions.

Posynomial and Monomial Functions

- A monomial function is simply:

$$g(y_1, \ldots, y_n) = c y_1^{\alpha_1} y_2^{\alpha_2} \cdots y_n^{\alpha_n}, \quad y_k > 0.$$

  where
  - c is non-negative and $\alpha_k \in \mathbb{R}$.
- A posynomial function is a sum of monomial functions:

$$p(y_1, \ldots, y_n) = \sum_{s=1}^{T} c_s y_1^{\alpha_{1,s}} y_2^{\alpha_{2,s}} \cdots y_n^{\alpha_{n,s}}, \quad y_k > 0,$$

- A monomial can also be viewed as a special case of posynomial where there is only one term of the sum.

Geometric Programming in Convex Form

- Many engineering problems can be formulated as a GP.
- On Boyd's website there is a Matlab package "GGPLAB" and an excellent tutorial material.
- GP can be converted into a convex form by changing the variables $x_k = \log(y_k)$ and replacing p_i with $\log p_i$:

$$\begin{array}{lll} \text{minimize} & \log p_0(\exp(x)) & \\ \text{subject to} & \log p_i(\exp(x)) \leq 0, & i = 1, \ldots, l \\ & a_j^\mathsf{T} x + b_j = 0, & j = 1, \ldots, m \end{array}$$

  where
  - $\exp(x) = (e^{x_1}, e^{x_2}, \cdots, e^{x_n})$
  - $a_j = (\alpha_{1,j}, \cdots, \alpha_{n,j})$
  - $b_j = \log(c_j)$

Robust GP

- GP in the convex form can be solved efficiently by interior-point methods.
- In robust version, coefficients c_s are functions of q.
- The robust problem is still convex. Moreover, there is an infinite number of constraints.
- Alternative approach: Ellipsoid Method.

Example - Profit Maximization Problem

This example is taken from [@Aliabadi2013Robust].

$$\begin{array}{ll} \text{maximize} & p(Ax_1^\alpha x_2^\beta) - v_1 x_1 - v_2 x_2 \\ \text{subject to} & x_1 \leq k. \end{array}$$

- $p(Ax_1^\alpha x_2^\beta)$: Cobb-Douglas production function
- p: the market price per unit
- A: the scale of production
- α, β: the output elasticities
- x: input quantity
- v: output price
- k: a given constant that restricts the quantity of x_1

Example - Profit maximization (cont'd)

- The formulation is not in the convex form.
- Rewrite the problem in the following form:

$$\begin{array}{ll} \text{maximize} & t \\ \text{subject to} & t + v_1 x_1 + v_2 x_2 \leq pAx_1^\alpha x_2^\beta \\ & x_1 \leq k. \end{array}$$

Profit maximization in Convex Form

- By taking the logarithm of each variable:
  - $y_1 = \log x_1$, $y_2 = \log x_2$.
- We have the problem in a convex form:

$$\begin{array}{ll} \max & t \\ \text{s.t.} & \log(t + v_1 e^{y_1} + v_2 e^{y_2}) - (\alpha y_1 + \beta y_2) \leq \log(pA) \\ & y_1 \leq \log k. \end{array}$$

.font-sm.mb-xs[

```python
class profit_oracle:
    def __init__(self, params, a, v):
        p, A, k = params
        self.log_pA = np.log(p * A)
        self.log_k = np.log(k)
        self.v = v
        self.a = a

    def __call__(self, y, t):
        fj = y[0] - self.log_k  # constraint
        if fj > 0.:
            g = np.array([1., 0.])
            return (g, fj), t
        log_Cobb = self.log_pA + self.a @ y
        x = np.exp(y)
        vx = self.v @ x
        te = t + vx
        fj = np.log(te) - log_Cobb
        if fj < 0.:
            te = np.exp(log_Cobb)
            t = te - vx
            fj = 0.
        g = (self.v * x) / te - self.a
        return (g, fj), t
```

]

.font-sm.mb-xs[

```python
# Main program

import numpy as np
from ellpy.cutting_plane import cutting_plane_dc
from ellpy.ell import ell
from .profit_oracle import profit_oracle

p, A, k = 20., 40., 30.5
params = p, A, k
alpha, beta = 0.1, 0.4
v1, v2 = 10., 35.
a = np.array([alpha, beta])
v = np.array([v1, v2])
y0 = np.array([0., 0.])  # initial x0
r = np.array([100., 100.])  # initial ellipsoid (sphere)
```

```
E = ell(r, y0)
P = profit_oracle(params, a, v)
yb1, ell_info = cutting_plane_dc(P, E, 0.)
print(ell_info.value, ell_info.feasible)
```

]

Example - Profit Maximization Problem (convex)

$$
\begin{array}{ll}
\max & t \\
\text{s.t.} & \log(t + \hat{v}_1 e^{y_1} + \hat{v}_2 e^{y_2}) - (\hat{\alpha} y_1 + \hat{\beta} y_2) \leq \log(\hat{p}\, A) \\
 & y_1 \leq \log \hat{k},
\end{array}
$$

- Now assume that:
  - $\hat{\alpha}$ and $\hat{\beta}$ vary $\bar{\alpha} \pm e_1$ and $\bar{\beta} \pm e_2$ respectively.
  - $\hat{p}$, $\hat{k}$, $\hat{v}_1$, and $\hat{v}_2$ all vary $\pm e_3$.

Example - Profit Maximization Problem (oracle)

By detail analysis, the worst case happens when:

- $p = \bar{p} - e_3$, $k = \bar{k} - e_3$
- $v_1 = \bar{v}_1 + e_3$, $v_2 = \bar{v}_2 + e_3$,
- if $y_1 > 0$, $\alpha = \bar{\alpha} - e_1$, else $\alpha = \bar{\alpha} + e_1$
- if $y_2 > 0$, $\beta = \bar{\beta} - e_2$, else $\beta = \bar{\beta} + e_2$

```python
class profit_rb_oracle:
    def __init__(self, params, a, v, vparams):
        e1, e2, e3, e4, e5 = vparams
        self.a = a
        self.e = [e1, e2]
        p, A, k = params
        params_rb = p - e3, A, k - e4
        self.P = profit_oracle(params_rb, a, v + e5)

    def __call__(self, y, t):
        a_rb = self.a.copy()
        for i in [0, 1]:
            a_rb[i] += self.e[i] if y[i] <= 0 else -self.e[i]
        self.P.a = a_rb
        return self.P(y, t)
```

Oracle in Robust Optimization Formulation

- The oracle only needs to determine:
  - If $f_j(x_0, q) > 0$ for some j and $q = q_0$, then
    - the cut $(g, \beta) = (\partial f_j(x_0, q_0), f_j(x_0, q_0))$
  - If $f_0(x_0, q) \geq t$ for some $q = q_0$, then
    - the cut $(g, \beta) = (\partial f_0(x_0, q_0), f_0(x_0, q_0) - t)$
  - Otherwise, x_0 is feasible, then
    - Let $q_{\max} = \text{argmax}_{q \in \mathbb{Q}} f_0(x_0, q)$.
    - $t := f_0(x_0, q_{\max})$.
    - The cut $(g, \beta) = (\partial f_0(x_0, q_{\max}), 0)$

Remark:

- for more complicated problems, affine arithmetic could be used [@liu2007robust].