Cutting-plane Method and Its Amazing Oracles

.pull-left[

@luk036

2022-11-03

] .pull-right[

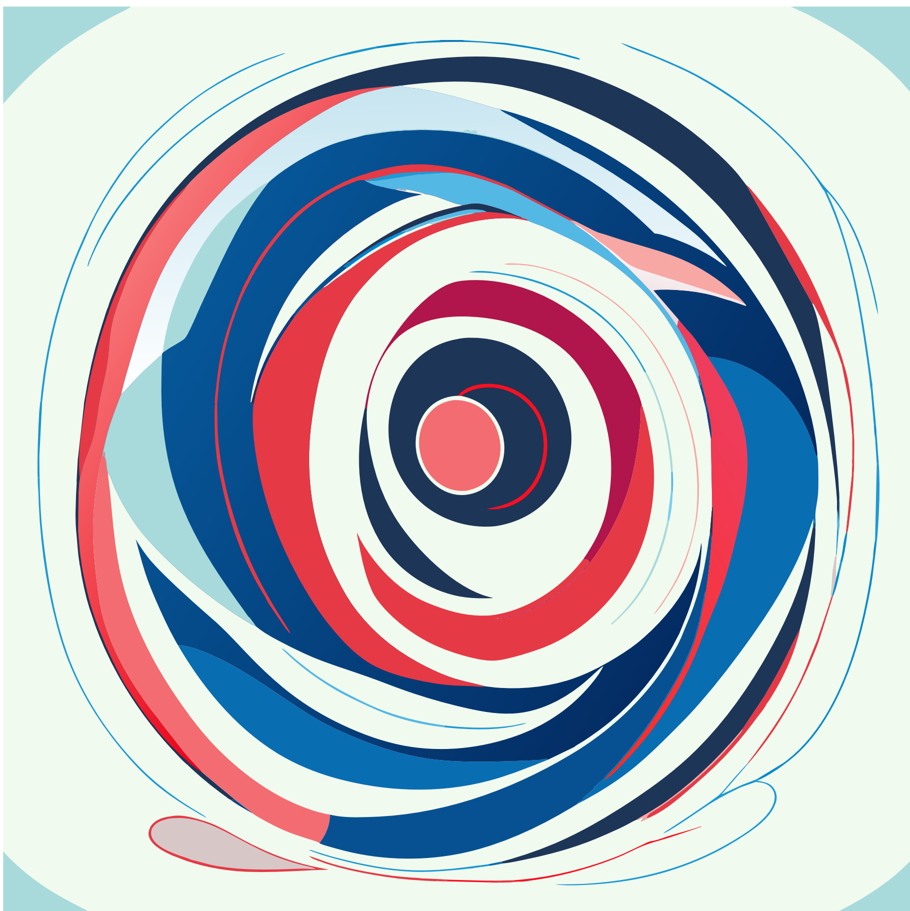


Figure 1: image

]

---

class: middle, right

> When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

Sir Arthur Conan Doyle, stated by Sherlock Holmes

---

class: middle, center

Introduction

Common Perspective of Ellipsoid Method

- It is widely believed to be inefficient in practice for large-scale problems.
  - Convergent rate is slow, even when using deep cuts.
  - Cannot exploit sparsity.
- It has since then supplanted by the interior-point methods.
- Used only as a theoretical tool to prove polynomial-time solvability of some combinatorial optimization problems.

But...

- The ellipsoid method works very differently compared with the interior point methods.
- Only require a *separation oracle*. Can play nicely with other techniques.
- While the ellipsoid method itself cannot take advantage of sparsity, the oracle can.

Consider the ellipsoid method when...

- The number of optimization variables is moderate, e.g. ECO flow, analog circuit sizing, parametric problems
- The number of constraints is large, or even infinite
- Oracle can be implemented effectively.

---

class: middle, center

Cutting-plane Method Revisited

Convex Set

.pull-left70[

- Let $\mathcal{K} \subseteq \mathbb{R}^n$ be a convex set.
- Consider the feasibility problem:
  - Find a point $x^* \in \mathbb{R}^n$ in $\mathcal{K}$,
  - or determine that $\mathcal{K}$ is empty (i.e., there is no feasible solution)

] .pull-right30[

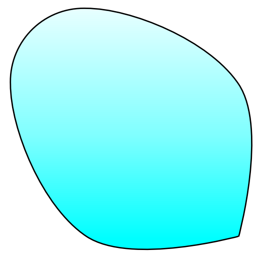


Figure 2: image

]

Separation Oracle

.pull-left70[

- When a separation oracle Ω is *queried* at x_0, it either
  - asserts that $x_0 \in \mathcal{K}$, or
  - returns a separating hyperplane between x_0 and $\mathcal{K}$:

$$g^\mathsf{T}(x - x_0) + \beta \leq 0, \beta \geq 0, g \neq 0, \ \forall x \in \mathcal{K}$$

] .pull-right30[

]

Separation Oracle (cont'd)

- (g, β) is called a *cutting-plane*, or cut, because it eliminates the half-space $\{x \mid g^\mathsf{T}(x - x_0) + \beta > 0\}$ from our search.

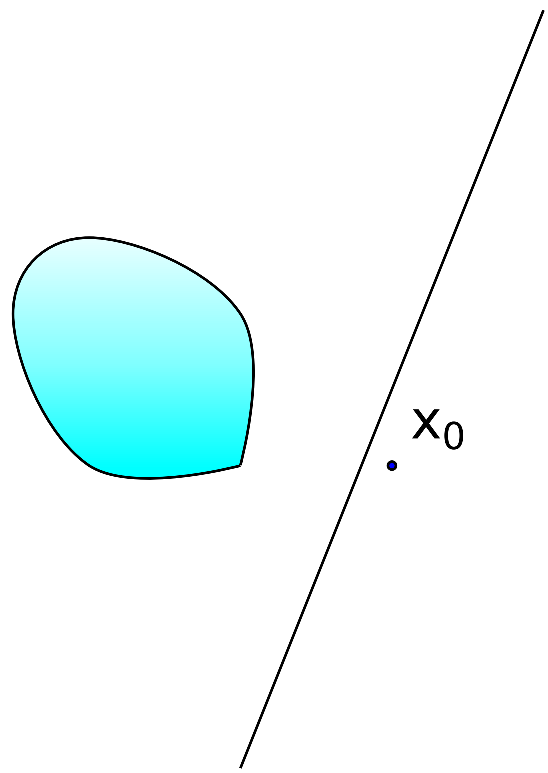


Figure 3: image

- If $\beta = 0$ (x_0 is on the boundary of halfspace that is cut), the cutting-plane is called *neutral cut*.
- If $\beta > 0$ (x_0 lies in the interior of halfspace that is cut), the cutting-plane is called *deep cut*.
- If $\beta < 0$ (x_0 lies in the exterior of halfspace that is cut), the cutting-plane is called *shallow cut*.

Subgradient

- $\mathcal{K}$ is usually given by a set of inequalities $f_j(x) \le 0$ or $f_j(x) < 0$ for $j = 1 \cdots m$, where $f_j(x)$ is a convex function.
- A vector $g \equiv \partial f(x_0)$ is called a subgradient of a convex function f at x_0 if $f(z) \geq f(x_0) + g^\mathsf{T}(z - x_0)$.
- Hence, the cut (g, β) is given by $(\partial f(x_0), f(x_0))$

Remarks:

- If $f(x)$ is differentiable, we can simply take $\partial f(x_0) = \nabla f(x_0)$

Key components of Cutting-plane method

- A cutting plane oracle Ω
- A search space $\mathcal{S}$ initially large enough to cover $\mathcal{K}$, e.g.
  - Polyhedron $\mathcal{P} = \{z \mid Cz \preceq d\}$
  - Interval $\mathcal{I} = [l, u]$ (for one-dimensional problem)
  - Ellipsoid $\mathcal{E} = \{z \mid (z - x_c)P^{-1}(z - x_c) \le 1\}$

Generic Cutting-plane method

- **Given** initial $\mathcal{S}$ known to contain $\mathcal{K}$.
- **Repeat**
  1. Choose a point x_0 in $\mathcal{S}$
  2. Query the cutting-plane oracle at x_0
  3. **If** $x_0 \in \mathcal{K}$, quit
  4. **Else**, update $\mathcal{S}$ to a smaller set that covers:

$$\mathcal{S}^+ = \mathcal{S} \cap \{z \mid g^\mathsf{T}(z - x_0) + \beta \le 0\}$$

  5. **If** $\mathcal{S}^+ = \emptyset$ or it is small enough, quit.

Corresponding Python code

```
def cutting_plane_feas(omega, S, options=Options()):
    for niter in range(options.max_iter):
        cut = omega.assess_feas(S.xc)  # query the oracle at S.xc
        if cut is None:  # feasible sol'n obtained
            return True, niter, CutStatus.Success
        cutstatus, tsq = S.update(cut)  # update S
        if cutstatus != CutStatus.Success:
            return False, niter, cutstatus
        if tsq < options.tol:
            return False, niter, CutStatus.SmallEnough
    return False, options.max_iter, CutStatus.NoSoln
```

From Feasibility to Optimization

$$\begin{array}{ll} \text{minimize} & f_0(x), \\ \text{subject to} & x \in \mathcal{K} \end{array}$$

- The optimization problem is treated as a feasibility problem with an additional constraint $f_0(x) \le t$.
- $f_0(x)$ could be a convex or a *quasiconvex function*.
- t is also called the *best-so-far* value of $f_0(x)$.

Convex Optimization Problem

- Consider the following general form:

$$\begin{array}{ll} \text{minimize} & t, \\ \text{subject to} & \Phi(x,t) \le 0, \\ & x \in \mathcal{K}, \end{array}$$

  where $\mathcal{K}'_t = \{x \mid \Phi(x,t) \le 0\}$ is the t-sublevel set of $\{x \mid f_0(x) \le t\}$.
- Note: $\mathcal{K}'_t \subseteq \mathcal{K}'_u$ if and only if $t \le u$ (monotonicity)
- One easy way to solve the optimization problem is to apply the binary search on t.

```
def bsearch(omega, intrvl, options=Options()):
    # assume monotone
    lower, upper = intrvl
    T = type(upper)  # T could be `int`
```

```python
    for niter in range(options.max_iter):
        tau = (upper - lower) / 2
        if tau < options.tol:
            return upper, niter, CutStatus.SmallEnough
        t = T(lower + tau)
        if omega.assess_bs(t):  # feasible sol'n obtained
            upper = t
        else:
            lower = t
    return upper, options.max_iter, CutStatus.Unknown
```

```python
class bsearch_adaptor:
    def __init__(self, P, S, options=Options()):
        self.P = P
        self.S = S
        self.options = options

    @property
    def x_best(self):
        return self.S.xc

    def assess_bs(self, t):
        S = self.S.copy()
        self.P.update(t)
        ell_info = cutting_plane_feas(self.P, S, self.options)
        if ell_info.feasible:
            self.S.xc = S.xc
        return ell_info.feasible
```

Shrinking

- Another possible way is, to update the best-so-far t whenever a feasible solution x' is found by solving the equation:

$$\Phi(x', t_{\text{new}}) = 0\,.$$

- If the equation is difficuit to solve but t is also convex w.r.t. Φ, then we may create a new varaible, say z and let $z \leq t$.

Generic Cutting-plane method (Optim)

- **Given** initial $\mathcal{S}$ known to contain $\mathcal{K}_t$.

- **Repeat**
  1. Choose a point x_0 in $\mathcal{S}$
  2. Query the separation oracle at x_0
  3. **If** $x_0 \in \mathcal{K}_t$, update t such that $\Phi(x_0, t) = 0$.
  4. Update $\mathcal{S}$ to a smaller set that covers:

$$\mathcal{S}^+ = \mathcal{S} \cap \{z \mid g^\mathsf{T}(z - x_0) + \beta \leq 0\}$$

  5. **If** $\mathcal{S}^+ = \emptyset$ or it is small enough, quit.

```python
def cutting_plane_optim(omega, S, t, options=Options()):
    x_best = None
    for niter in range(options.max_iter):
        cut, t1 = omega.assess_optim(S.xc, t)
        if t1 is not None:  # better t obtained
            t = t1
            x_best = S.xc.copy()
        status, tsq = S.update(cut)
        if status != CutStatus.Success:
            return x_best, t, niter, status
        if tsq < options.tol:
            return x_best, t, niter, CutStatus.SmallEnough
    return x_best, t, options.max_iter, CutStatus.Success
```

Example - Profit Maximization Problem

This example is taken from [@Aliabadi2013Robust].

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

- $p(A x_1^{\alpha} x_2^{\beta})$: 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:

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

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:

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

```python
class ProfitOracle:
    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 assess_optim(self, y, t):
        if (fj := y[0] - self.log_k) > 0.0:  # constraint
            g = np.array([1.0, 0.0])
            return (g, fj), None

        log_Cobb = self.log_pA + self.a @ y
        q = self.v * np.exp(y)
        vx = q[0] + q[1]
        if (fj := np.log(t + vx) - log_Cobb) >= 0.0:
            g = q / (t + vx) - self.a
            return (g, fj), None

        t = np.exp(log_Cobb) - vx
        g = q / (t + vx) - self.a
        return (g, 0.0), t
```

```python
# Main program
```

```python
import numpy as np
from ellalgo.cutting_plane import cutting_plane_optim
from ellalgo.ell import Ell
from ellalgo.oracles.profit_oracle import ProfitOracle

p, A, k = 20.0, 40.0, 30.5
params = p, A, k
alpha, beta = 0.1, 0.4
v1, v2 = 10.0, 35.0
a = np.array([alpha, beta])
v = np.array([v1, v2])
r = np.array([100.0, 100.0])  # initial ellipsoid (sphere)

E = Ell(r, np.array([0.0, 0.0]))
P = ProfitOracle(params, a, v)
x, f, num_iters, status = cutting_plane_optim(P, E, 0.0)
assert x is not None
```

Area of Applications

- Robust convex optimization
  - oracle technique: affine arithmetic
- Parametric network potential problem
  - oracle technique: negative cycle detection
- Semidefinite programming
  - oracle technique: Cholesky or LDL^{T} factorization

---

class: middle, center

Robust Convex Optimization

Robust Optimization Formulation

- Consider:

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

  where q represents a set of varying parameters.

- The problem can be reformulated as:

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

Example - Profit Maximization Problem (convex)

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

- 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 ProfitRbOracle:
    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 = ProfitOracle(params_rb, a, v + e5)

    def assess_optim(self, y, t):
        a_rb = self.a.copy()
        for i in [0, 1]:
            a_rb[i] += -self.e[i] if y[i] > 0.0 else self.e[i]
        self.P.a = a_rb
        return self.P.assess_optim(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].

---

class: middle, center

Multi-parameter Network Problem

Parametric Network Problem

Given a network represented by a directed graph $G = (V, E)$.

Consider:

$$\begin{array}{ll}
\text{find} & x, {\color{red} u} \\
\text{subject to} & {\color{red} u_j} - {\color{red} u_i} \leq h_{ij}(x), \; \forall (i, j) \in E,
\end{array}$$

- $h_{ij}(x)$ is the concave function of edge (i, j),

- Assume: network is large, but the number of parameters is small.

Network Potential Problem (cont'd)

Given x, the problem has a feasible solution if and only if G contains no negative cycle. Let $\mathcal{C}$ be a set of all cycles of G.

$$\begin{array}{ll}
\text{find} & x \\
\text{subject to} & w_k(x) \geq 0, \forall C_k \in \mathcal{C},
\end{array}$$

- C_k is a cycle of G

- $w_k(x) = \sum_{(i,j)\in C_k} h_{ij}(x)$.

Negative Cycle Finding

There are lots of methods to detect negative cycles in a weighted graph [@cherkassky1999negative], in which Tarjan's algorithm [@Tarjan1981negcycle] is one of the fastest algorithms in practice [@alg:dasdan_mcr; @cherkassky1999negative].

Oracle in Network Potential Problem

- The oracle only needs to determine:
  - If there exists a negative cycle C_k under x_0, then
    - the cut $(g, \beta) = (-\partial w_k(x_0), -w_k(x_0))$
  - Otherwise, the shortest path solution gives the value of u.

Python Code

```python
class NetworkOracle:
    def __init__(self, G, u, h):
        self._G = G
        self._u = u
        self._h = h
        self._S = NegCycleFinder(G)

    def update(self, t):
        self._h.update(t)

    def assess_feas(self, x) -> Optional[Cut]:
        def get_weight(e):
            return self._h.eval(e, x)

        for Ci in self._S.find_neg_cycle(self._u, get_weight):
            f = -sum(self._h.eval(e, x) for e in Ci)
            g = -sum(self._h.grad(e, x) for e in Ci)
            return g, f  # use the first Ci only
        return None
```

Example - Optimal Matrix Scaling [@orlin1985computing]

- Given a sparse matrix $A = [a_{ij}] \in \mathbb{R}^{N\times N}$.
- Find another matrix $B = UAU^{-1}$ where U is a nonnegative diagonal matrix, such that the ratio of any two elements of B in absolute value is as close to 1 as possible.
- Let $U = \mathrm{diag}([u_1, u_2, \ldots, u_N])$. Under the min-max-ratio criterion, the problem can be formulated as:

$$\begin{array}{ll}
\text{minimize} & \pi/\psi \\
\text{subject to} & \psi \leq u_i |a_{ij}| u_j^{-1} \leq \pi, \;\; \forall a_{ij} \neq 0, \\
& \pi, \psi, u, \text{positive} \\
\text{variables} & \pi, \psi, u \, .
\end{array}$$

Optimal Matrix Scaling (cont'd)

By taking the logarithms of variables, the above problem can be transformed into:

$$\begin{array}{ll}
\text{minimize} & t \\
\text{subject to} & {\color{blue}\pi'} - {\color{blue}\psi'} \leq t \\
& {\color{red}u_i'} - {\color{red}u_j'} \leq {\color{blue}\pi'} - a_{ij}', \;\; \forall a_{ij} \neq 0 \, , \\
& {\color{red}u_j'} - {\color{red}u_i'} \leq a_{ij}' - {\color{blue}\psi'}, \;\; \forall a_{ij} \neq 0 \, , \\
\text{variables} & {\color{blue}\pi'}, {\color{blue}\psi'}, {\color{red}u'} \, .
\end{array}$$

where k' denotes $\log(|k|)$ and $x = ({\color{blue}\pi'}, {\color{blue}\psi'})^{\mathsf{T}}$.

```python
class OptScalingOracle:
    class Ratio:
        def __init__(self, G, get_cost):
            self._G = G
            self._get_cost = get_cost

        def eval(self, e, x: Arr) -> float:
            u, v = e
            cost = self._get_cost(e)
            return x[0] - cost if u < v else cost - x[1]

        def grad(self, e, x: Arr) -> Arr:
            u, v = e
            return np.array([1.0, 0.0] if u < v else [0.0, -1.0])
```

```python
def __init__(self, G, u, get_cost):
    self._network = NetworkOracle(G, u, self.Ratio(G, get_cost))

def assess_optim(self, x: Arr, t: float):
    s = x[0] - x[1]
    g = np.array([1.0, -1.0])
    if (fj := s - t) >= 0.0:
        return (g, fj), None
    if (cut := self._network.assess_feas(x)):
        return cut, None
    return (g, 0.0), s
```

Example - clock period & yield-driven co-optimization

$$
\begin{array}{ll}
\text{minimize} & T_{\text{CP}} / \beta \\
\text{subject to} & u_i - u_j \le T_{\text{CP}} - F_{ij}^{-1}(\beta), \quad \forall (i,j) \in E_s\,, \\
 & u_j - u_i \le F_{ij}^{-1}(1 - \beta), \quad \forall (j,i) \in E_h\,, \\
 & T_{\text{CP}} \ge 0, \, 0 \le \beta \le 1\,, \\
\text{variables} & T_{\text{CP}}, \beta, u.
\end{array}
$$

- Note that $F_{ij}^{-1}(x)$ is not concave in general in $[0, 1]$.
- Fortunately, we are most likely interested in optimizing circuits for high yield rather than the low one in practice.
- Therefore, by imposing an additional constraint to β, say $\beta \geq 0.8$, the problem becomes convex.

Example - clock period & yield-driven co-optimization

The problem can be reformulated as:

$$
\begin{array}{ll}
\text{minimize} & t \\
\text{subject to} & T_{\text{CP}} - \beta t \le 0 \\
 & u_i - u_j \le T_{\text{CP}} - F_{ij}^{-1}(\beta), \quad \forall (i,j) \in E_s\,, \\
 & u_j - u_i \le F_{ij}^{-1}(1 - \beta), \quad \forall (j,i) \in E_h\,, \\
 & T_{\text{CP}} \ge 0, \, 0 \le \beta \le 1\,, \\
\text{variables} & T_{\text{CP}}, \beta, u.
\end{array}
$$

---

class: middle, center

Matrix Inequalities

Problems With Matrix Inequalities

Consider the following problem:

$$
\begin{array}{ll}
\text{find} & x, \\
\text{subject to} & F(x) \succeq 0,
\end{array}
$$

- $F(x)$: a matrix-valued function
- $A \succeq 0$ denotes A is positive semidefinite.

Problems With Matrix Inequalities

- Recall that a matrix A is positive semidefinite if and only if $v^{\mathsf{T}} A v \geq 0$ for all $v \in \mathbb{R}^N$.
- The problem can be transformed into:

$$
\begin{array}{ll}
\text{find} & x, \\
\text{subject to} & v^{\mathsf{T}} F(x) v \geq 0, \ \forall v \in \mathbb{R}^N
\end{array}
$$

- Consider $v^{\mathsf{T}} F(x) v$ is concave for all $v \in \mathbb{R}^N$ w. r. t. x, then the above problem is a convex programming.
- Reduce to *semidefinite programming* if $F(x)$ is linear w.r.t. x, i.e., $F(x) = F_0 + x_1 F_1 + \cdots + x_n F_n$

Oracle in Matrix Inequalities

The oracle only needs to:

- Perform a *row-based* LDLT factorization such that $F(x_0) = LDL^{\mathsf{T}}$.
- Let $A_{p,p}$ denotes a submatrix $A(1:p, 1:p) \in \mathbb{R}^{p \times p}$.
- If the process fails at row p,
  - there exists a vector $e_p = (0, 0, \cdots, 0, 1)^{\mathsf{T}} \in \mathbb{R}^p$, such that
    - $v = R_{p,p}^{-1} e_p$, and
    - $v^{\mathsf{T}} F_{p,p}(x_0) v < 0$.
  - The cut $(g, \beta) = (-v^{\mathsf{T}} \partial F_{p,p}(x_0) v, -v^{\mathsf{T}} F_{p,p}(x_0) v)$

Lazy evaluation

- Don't construct the full matrix at each iteration!
- Only $\mathrm{O}(p^3)$ per iteration, independent of N!

```python
class LMIOracle:
    def __init__(self, F, B):
        self.F = F
        self.F0 = B
        self.Q = LDLTMgr(len(B))

    def assess_feas(self, x: Arr) -> Optional[Cut]:
        def get_elem(i, j):
            return self.F0[i, j] - sum(
                Fk[i, j] * xk for Fk, xk in zip(self.F, x))

        if self.Q.factor(get_elem):
            return None
        ep = self.Q.witness()
        g = np.array([self.Q.sym_quad(Fk) for Fk in self.F])
        return g, ep
```

Google Benchmark Comparison

```
2: -------------------------------------------------------------
2: Benchmark                   Time             CPU   Iterations
2: -------------------------------------------------------------
2: BM_LMI_Lazy           131235 ns       131245 ns         4447
2: BM_LMI_old            196694 ns       196708 ns         3548
2/4 Test #2: Bench_BM_lmi .....................   Passed    2.57 sec
```

Example - Matrix Norm Minimization

- Let $A(x) = A_0 + x_1 A_1 + \cdots + x_n A_n$
- Problem $\min_x \|A(x)\|$ can be reformulated as

$$\begin{array}{ll} \text{minimize} & t, \\ \text{subject to} & \begin{pmatrix} t\,I & A(x) \\ A^\mathsf{T}(x) & t\,I \end{pmatrix} \succeq 0, \end{array}$$

- Binary search on t can be used for this problem.

Example - Estimation of Correlation Function

$$\begin{array}{ll} \min_{\kappa,p} & \|\Sigma(p) + \kappa I - Y\| \\ \text{s. t.} & \Sigma(p) \succcurlyeq 0, \kappa \geq 0\ . \end{array}$$

- Let $\rho(h) = \sum_i^n p_i \Psi_i(h)$, where

- p_i's are the unknown coefficients to be fitted
- Ψ_i's are a family of basis functions.

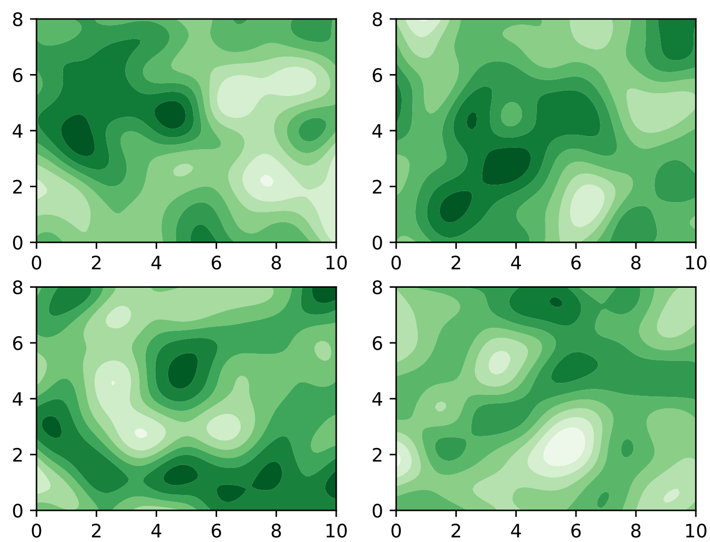
- The covariance matrix $\Sigma(p)$ can be recast as:

$$\Sigma(p) = p_1 F_1 + \cdots + p_n F_n$$

where $\{F_k\}_{i,j} = \Psi_k(\|s_j - s_i\|_2)$

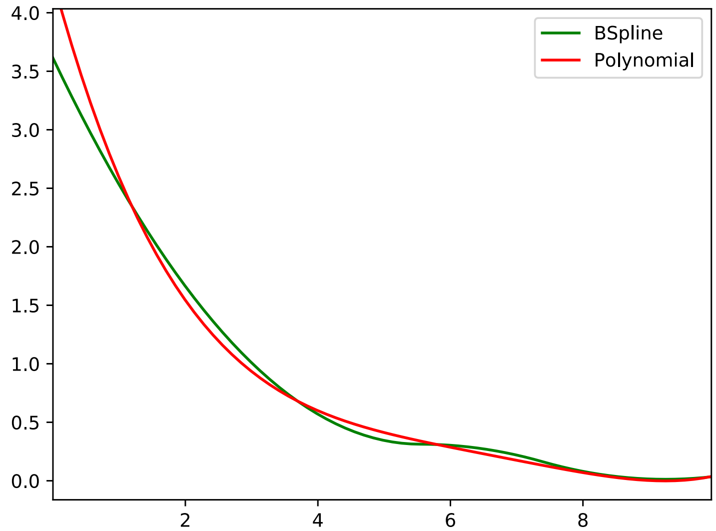
Experimental Result

.pull-left[



Data Sample (kern=0.5)

] .pull-right[

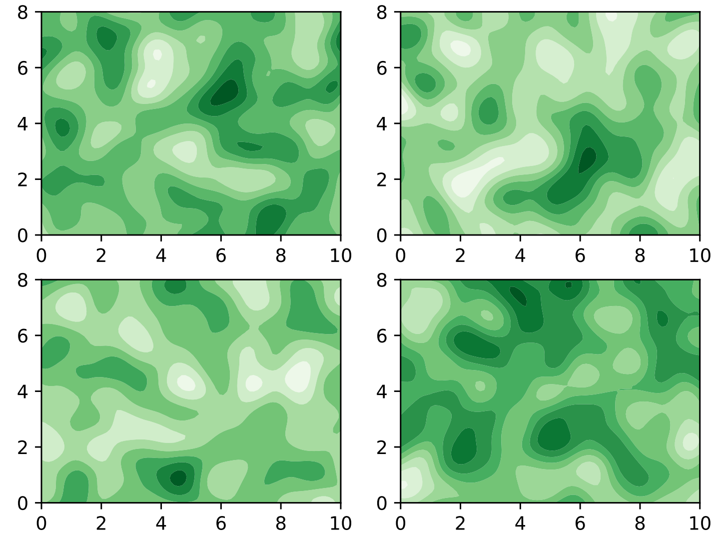


Least Square Result

]

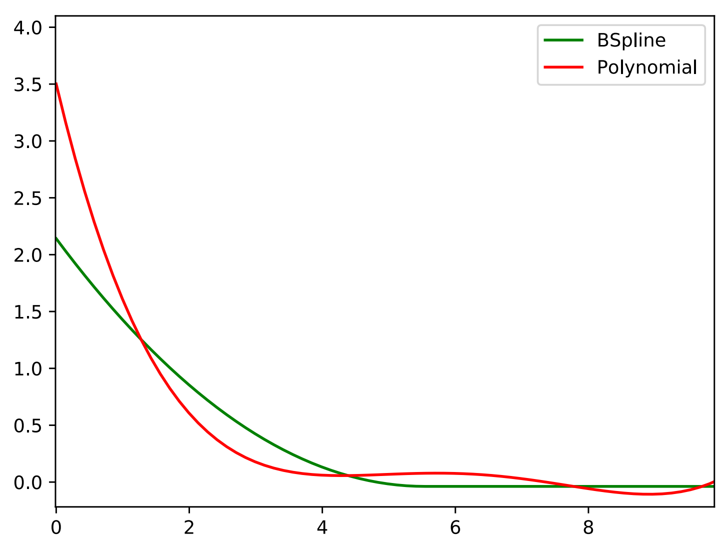
Experimental Result II

.pull-left[



Data Sample (kern=1.0)

] .pull-right[

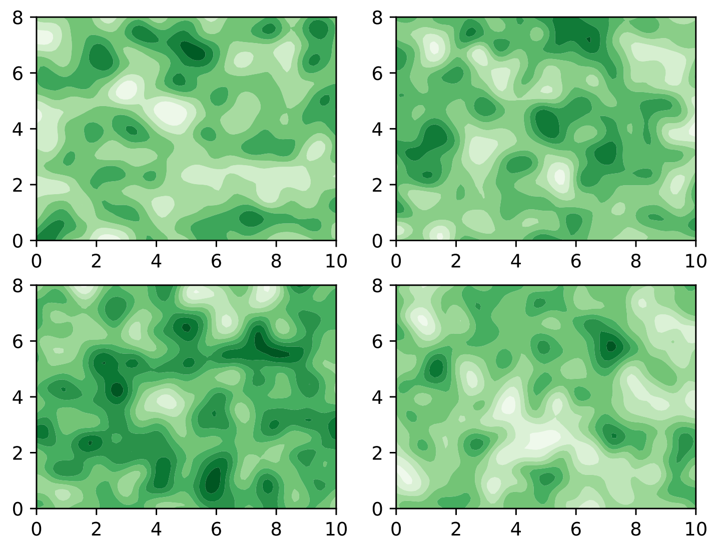


Least Square Result

]

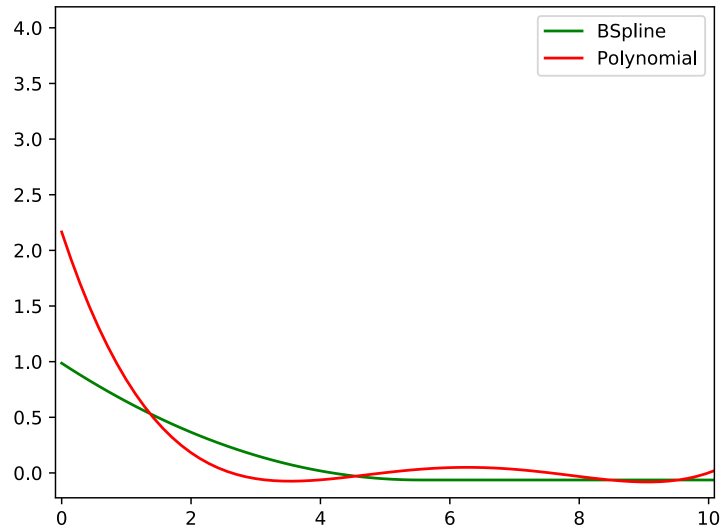
Experimental Result III

.pull-left[



Data Sample (kern=2.0)

] .pull-right[



Least Square Result

]

Ellipsoid Method Revisited

.pull-left[

@luk036

2022-11-03

] .pull-right[

]

Some History of Ellipsoid Method [@BGT81]

- Introduced by Shor and Yudin and Nemirovskii in 1976
- Used to show that linear programming (LP) is polynomial-time solvable (Kachiyan 1979), settled the long-standing problem of determining the theoretical complexity of LP.

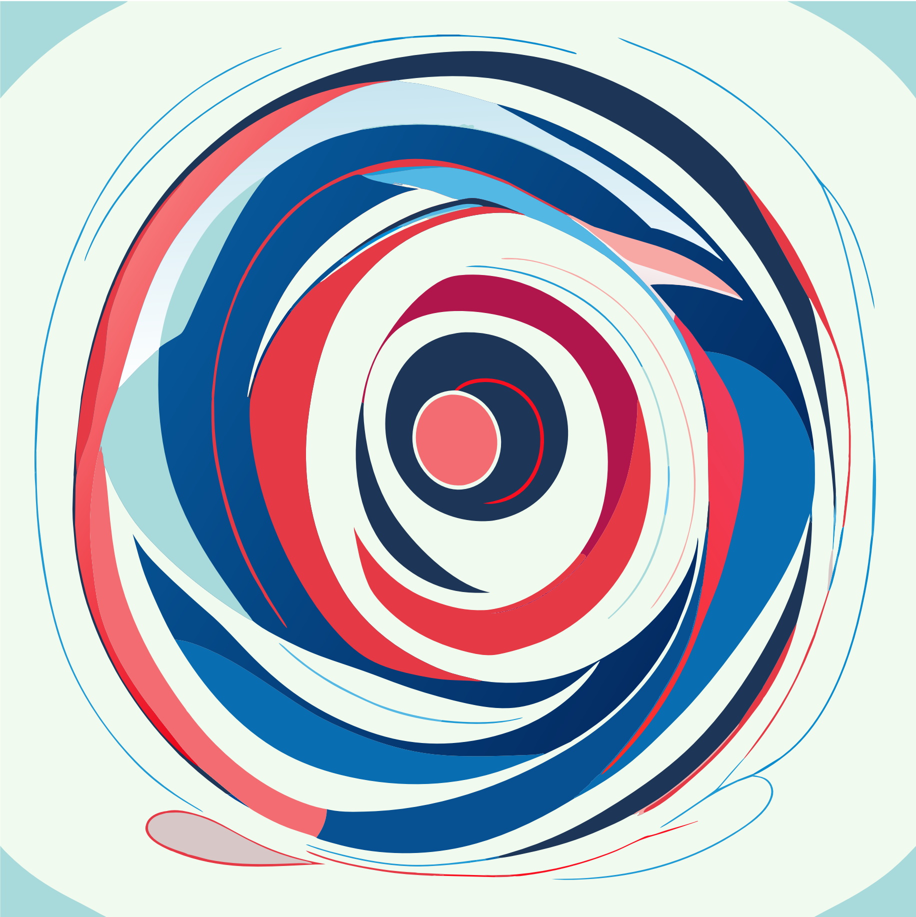


Figure 4: image

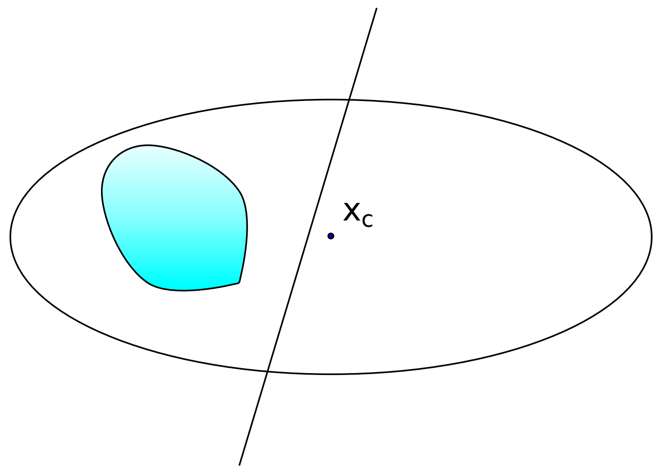
- In practice, however, the simplex method runs much faster than the method, although its worst-case complexity is exponential.

Basic Ellipsoid Method

- An ellipsoid $\mathcal{E}(x_c, P)$ is specified as a set

$$x \mid (x - x_c)P^{-1}(x - x_c) \leq 1,$$

where x_c is the center of the ellipsoid.



Python code

```
class Ell:
    """Ellipsoid Search Space
        Ell = {x | (x - xc)' P^-1 (x - xc)   1}
    """
    def __init__(self, val, x):
        self._n = n = len(x)
        self.c1 = float(n * n) / (n * n - 1)
        self._xc = x.copy()
        if np.isscalar(val):
            self.P = val * np.eye(n)
        else:
            self.P = np.diag(val)

    def update_core(self, calc_ell, cut): ...
    def calc_cc(self, ...): ...
```

```python
def calc_dc(self, ...): ...
def calc_ll(self, ...): ...
```

Updating the ellipsoid (deep-cut)

Calculation of minimum volume ellipsoid $\mathcal{E}^+$ covering:

$$\mathcal{E} \cap z \mid g^{\mathsf{T}}(z - x_c) + \beta \leq 0.$$

- Let $\tilde{g} = P\,g$, $\tau^2 = g^{\mathsf{T}}Pg$.
- If $n \cdot \beta < -\tau$ (shallow cut), no smaller ellipsoid can be found.
- If $\beta > \tau$, intersection is empty.

Otherwise,

$$x_c^+ = x_c - \frac{\rho}{\tau^2}\tilde{g}, \quad P^+ = \delta \cdot \left(P - \frac{\sigma}{\tau^2}\tilde{g}\tilde{g}^{\mathsf{T}}\right), \quad (P')^{-1} = \delta^{-1} \cdot \left(P^{-1} + \frac{\mu}{\tau^2}gg^{\mathsf{T}}\right).$$

where

$$\rho = \frac{\tau + n \cdot \beta}{n+1}, \quad \sigma = \frac{2\rho}{\tau + \beta}, \quad \delta = \frac{n^2(\tau+\beta)(\tau-\beta)}{(n^2-1)\tau^2}, \quad \mu = \frac{2(\tau + n \cdot \beta)}{(n-1)(\tau-\beta)}$$

Deep cut

Updating the ellipsoid (cont'd)

- Even better, split P into two variables $\kappa \cdot Q$
- Let $\tilde{g} = Q \cdot g$, $\omega = g^{\mathsf{T}}\tilde{g}$, $\tau = \sqrt{\kappa \cdot \omega}$.

$$x_c^+ = x_c - \frac{\rho}{\omega}\tilde{g}, \quad Q' = Q - \frac{\sigma}{\omega}\tilde{g}\tilde{g}^{\mathsf{T}}, \quad (Q')^{-1} = Q^{-1} + \frac{\mu}{\omega}gg^{\mathsf{T}}, \quad \kappa^+ = \delta \cdot \kappa.$$

- Reduce n^2 multiplications per iteration.
- Note:
  - The determinant of Q decreases monotonically.
  - The range of δ is $(0, \frac{n^2}{n^2-1})$.

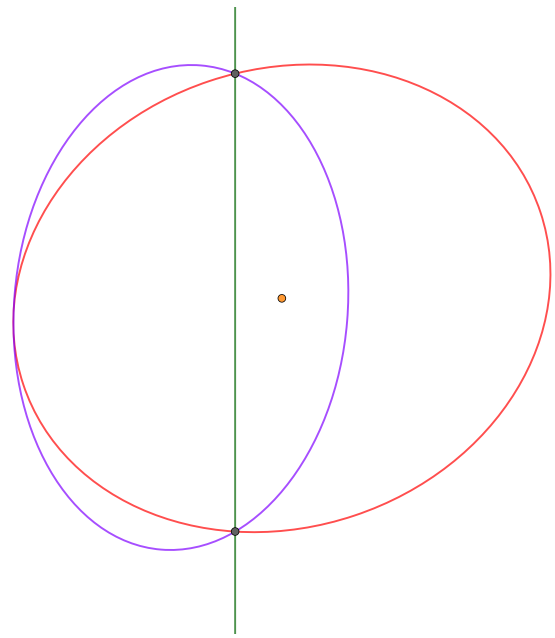


Figure 5: Deep-cut

Python code (updating)

```
def update_core(self, calc_ell, cut):
    grad, beta = cut
    grad_t = self._Q @ grad
    omega = grad @ grad_t
    self._tsq = self._kappa * omega
    status = calc_ell(beta)
    if status != CutStatus.Success:
        return status, self._tsq

    self._xc -= (self._rho / omega) * grad_t
    self._Q -= (self._sigma / omega) \
        * np.outer(grad_t, grad_t)
    self._kappa *= self._delta
    return status, self._tsq
```

Python code (deep cut)

```
def _calc_dc(self, beta: float) -> CutStatus:
    """Calculate new ellipsoid under Deep Cut """
```

```
tau = math.sqrt(self._tsq)
if tau < beta:
    return CutStatus.NoSoln  # no sol'n
if beta == 0.0:
    self._calc_cc(tau)
    return CutStatus.Success
n = self._n
gamma = tau + n * beta
if gamma < 0.0:
    return CutStatus.NoEffect  # no effect, unlikely

self._rho = gamma / self._nPlus1
self._sigma = 2.0 * self._rho / (tau + beta)
self._delta = self._c1 * (1.0 - beta * (beta / self._tsq))
return CutStatus.Success
```

Central Cut

- A Special case of deep cut when $\beta = 0$
- Deserve a separate implement because it is much simplier.
- Let $\tilde{g} = Q\,g$, $\tau = \sqrt{\kappa \cdot \omega}$,

$$\rho = \frac{\tau}{n+1}, \quad \sigma = \frac{2}{n+1}, \quad \delta = \frac{n^2}{n^2 - 1}, \quad \mu = \frac{2}{n-1}.$$

Central Cut

---

class: middle, center

Parallel Cuts

Parallel Cuts

- Oracle returns a pair of cuts instead of just one.
- The pair of cuts is given by g and (β_0, β_1) such that:

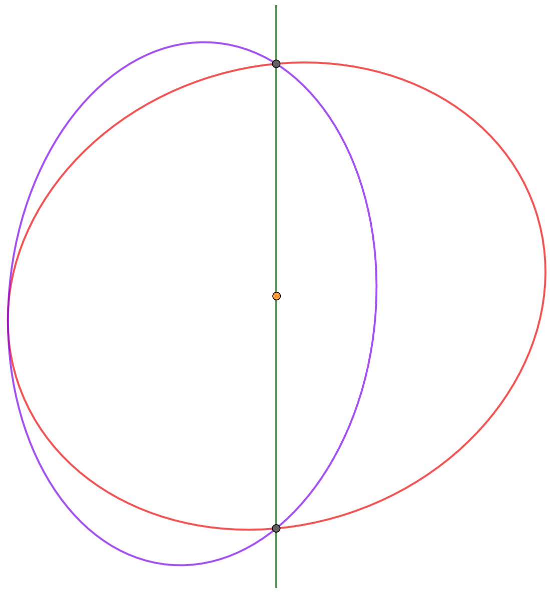


Figure 6: Central-cut

$$g^\mathsf{T}(x - x_c) + \beta_0 \le 0,$$

$$g^\mathsf{T}(x - x_c) + \beta_1 \ge 0,$$

for all $x \in \mathcal{K}$. $$

- Only linear inequality constraint can produce such parallel cut:

$$l \le a^\mathsf{T} x + b \le u, \quad L \preceq F(x) \preceq U.$$

- Usually provide faster convergence.

Parallel Cuts

Updating the ellipsoid

- Let $\tilde{g} = Q\,g$, $\tau^2 = \kappa \cdot \omega$.
- If $\beta_0 > \beta_1$, intersection is empty.
- If $\beta_0 \beta_1 < -\tau^2/n$, no smaller ellipsoid can be found.
- If $\beta_1^2 > \tau^2$, it reduces to deep-cut with $\alpha = \alpha_1$

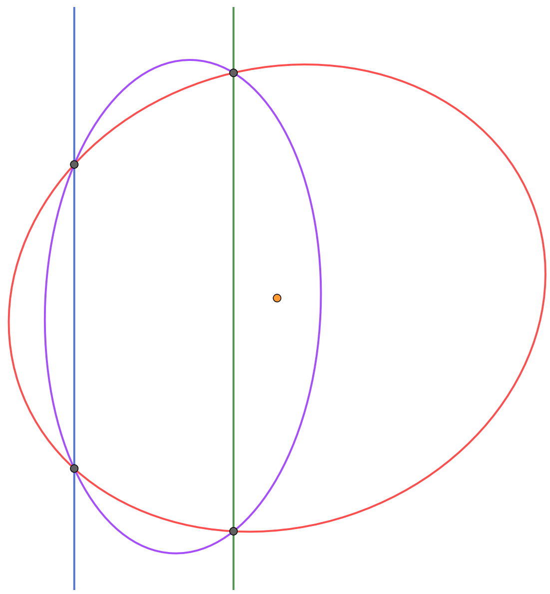


Figure 7: Parallel Cut

- Otherwise,

$$x_c' = x_c - \frac{\rho}{\omega}\tilde{g}, \quad Q' = Q - \frac{\sigma}{\omega}\tilde{g}\tilde{g}^{\mathsf{T}}, \quad (Q')^{-1} = Q^{-1} + \frac{\mu}{\omega}gg^{\mathsf{T}}, \quad \kappa^{+} = \delta\kappa.$$

where

$$\begin{aligned}
\bar{\beta} &= (\beta_0 + \beta_1)/2, \\
\xi^2 &= (\tau^2 - \beta_0^2)(\tau^2 - \beta_1^2) + (n(\beta_1 - \beta_0)\bar{\beta})^2, \\
\sigma &= (n + (\tau^2 + \beta_0\beta_1 - \xi)/(2\bar{\beta}^2))/(n+1), \\
\rho &= \bar{\beta} \cdot \sigma, \\
\mu &= \sigma/(1 - \sigma), \\
\delta &= (n^2/(n^2 - 1))(\tau^2 - (\beta_0^2 + \beta_1^2)/2 + \xi/n)/\tau^2.
\end{aligned}$$

Python code (parallel cut)

```
def calc_ll_core(self, b0, b1, tsq):
    if b1 < b0:
        return 1, None  # no sol'n
```

```python
n = self._n
b0b1 = b0*b1
if n*b0b1 < -tsq:
    return 3, None  # no effect
b1sq = b1**2
if b1sq > tsq or not self.use_parallel:
    return self.calc_dc(b0, tsq)
if b0 == 0:
    return self.calc_ll_cc(b1, b1sq, tsq)
# parallel cut
t0 = tsq - b0**2
t1 = tsq - b1sq
bav = (b0 + b1)/2
xi = math.sqrt( t0*t1 + (n*bav*(b1 - b0))**2 )
sigma = (n + (tsq - b0b1 - xi)/(2 * bav**2)) / (n + 1)
rho = sigma * bav
delta = self.c1 * ((t0 + t1)/2 + xi/n) / tsq
return 0, (rho, sigma, delta)
```

Example - FIR filter design

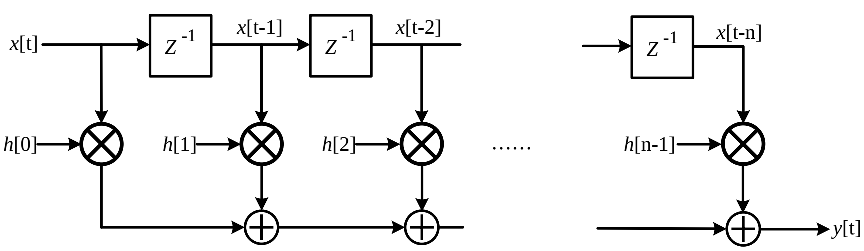


Figure 8: A typical structure of an FIR filter @mitra2006digital.

- The time response is:

$$y[t] = \sum_{k=0}^{n-1} h[k]u[t-k].$$

Example - FIR filter design (cont'd)

- The frequency response:

$$H(\omega) \;=\; \sum_{m=0}^{n-1} h(m)e^{-jm\omega}.$$

- The magnitude constraints on frequency domain are expressed as

$$L(\omega) \ \le \ |H(\omega)| \ \le \ U(\omega), \ \forall \ \omega \in (-\infty, +\infty).$$

where $L(\omega)$ and $U(\omega)$ are the lower and upper (nonnegative) bounds at frequency ω respectively.

- The constraint is non-convex in general.

Example - FIR filter design (II)

- However, via *spectral factorization* [@goodman1997spectral], it can transform into a convex one [@wu1999fir]:

$$L^2(\omega) \ \le \ R(\omega) \ \le \ U^2(\omega), \ \forall \ \omega \in (0, \pi),$$

where

- $R(\omega) = \sum_{i=-1+n}^{n-1} r(t)e^{-j\omega t} = |H(\omega)|^2$
- $\mathbf{r} = (r(-n+1), r(-n+2), ..., r(n-1))$ are the autocorrelation coefficients.

Example - FIR filter design (III)

- $\mathbf{r}$ can be determined by $\mathbf{h}$:

$$r(t) \ = \ \sum_{i=-n+1}^{n-1} h(i)h(i+t), \ t \in \mathbf{Z},$$

where $h(t) = 0$ for $t < 0$ or $t > n-1$.

- The whole problem can be formulated as:

$$\begin{aligned} \min \quad & \gamma \\ \text{s.t.} \quad & L^2(\omega) \le R(\omega) \le U^2(\omega), \ \forall \omega \in [0, \pi] \\ & R(\omega) > 0, \forall \omega \in [0, \pi] \end{aligned}$$

Experiment

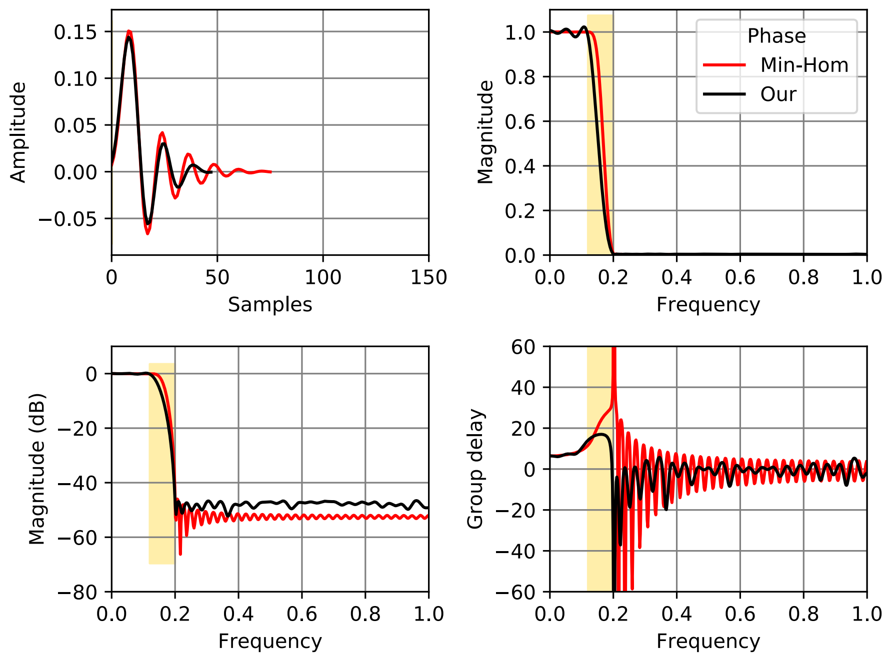


Figure 9: Result

Google Benchmark Result

```
3: ------------------------------------------------------------------
3: Benchmark                          Time             CPU   Iterations
3: ------------------------------------------------------------------
3: BM_Lowpass_single_cut     627743505 ns    621639313 ns            1
3: BM_Lowpass_parallel_cut    30497546 ns     30469134 ns           24
3/4 Test #3: Bench_BM_lowpass .................   Passed    1.72 sec
```

Example - Maximum Likelihood estimation

$$
\begin{array}{ll}
\min_{\kappa, p} & \log\det(\Omega(p) + \kappa \cdot I) + \mathrm{Tr}((\Omega(p) + \kappa \cdot I)^{-1} Y) \\
\text{s.t.} & \Omega(p) \succeq 0, \kappa \geq 0
\end{array}
$$

Note: the 1st term is concave, the 2nd term is convex

- However, if there are enough samples such that Y is a positive definite matrix, then the function is convex within $[0, 2Y]$

Example - Maximum Likelihood estimation (cont'd)

- Therefore, the following problem is convex:

$$
\begin{array}{ll}
\min_{\kappa, p} & \log\det V(p) + \mathrm{Tr}(V(p)^{-1} Y) \\
\text{s.t.} & \Omega(p) + \kappa \cdot I = V(p) \\
& 0 \preceq V(p) \preceq 2Y, \kappa > 0
\end{array}
$$

---

class: middle, center

Discrete Optimization

Why Discrete Convex Programming

- Many engineering problems can be formulated as a convex/geometric programming, e.g. digital circuit sizing
- Yet in an ASIC design, often there is only a limited set of choices from the cell library. In other words, some design variables are discrete.

- The discrete version can be formulated as a *Mixed-Integer Convex programming* (MICP) by mapping the design variables to integers.

What's Wrong w/ Existing Methods?

- Mostly based on relaxation.
- Then use the relaxed solution as a lower bound and use the branch–and–bound method for the discrete optimal solution.
  - Note: the branch-and-bound method does not utilize the convexity of the problem.
- What if I can only evaluate constraints on discrete data? Workaround: convex fitting?

Mixed-Integer Convex Programming

Consider:

$$\begin{array}{ll} \text{minimize} & f_0(x), \\ \text{subject to} & f_j(x) \leq 0, \ \forall j = 1, 2, \ldots \\ & x \in \mathbb{D} \end{array}$$

where

- $f_0(x)$ and $f_j(x)$ are "convex"
- Some design variables are discrete.

Oracle Requirement

- The oracle looks for the nearby discrete solution x_d of x_c with the cutting-plane:

$$g^\mathsf{T}(x - x_d) + \beta \leq 0, \beta \geq 0, g \neq 0$$

- Note: the cut may be a shallow cut.
- Suggestion: use different cuts as possible for each iteration (e.g. round-robin the evaluation of constraints)

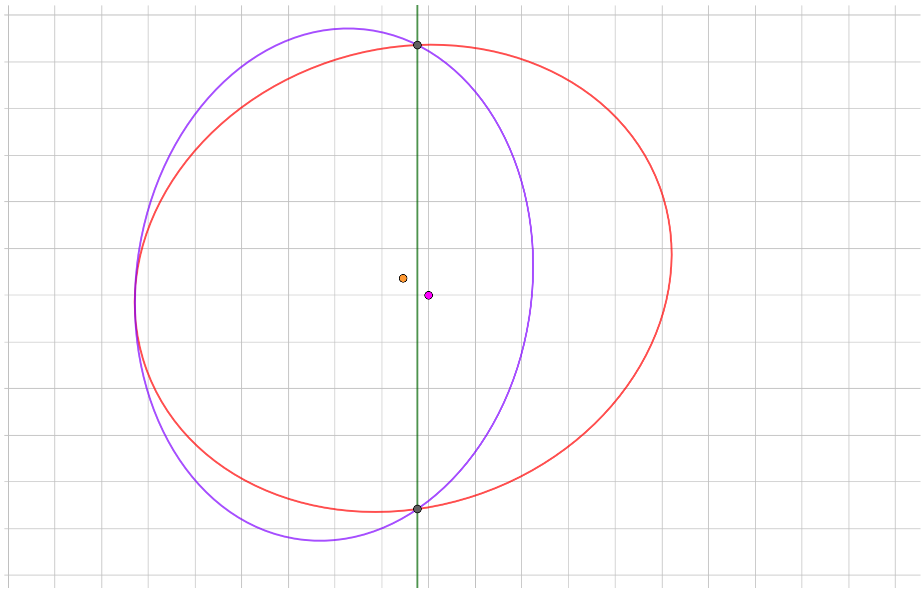


Figure 10: Discrete Cut

Discrete Cut

Example - Multiplier-less FIR filter design (nnz=3)

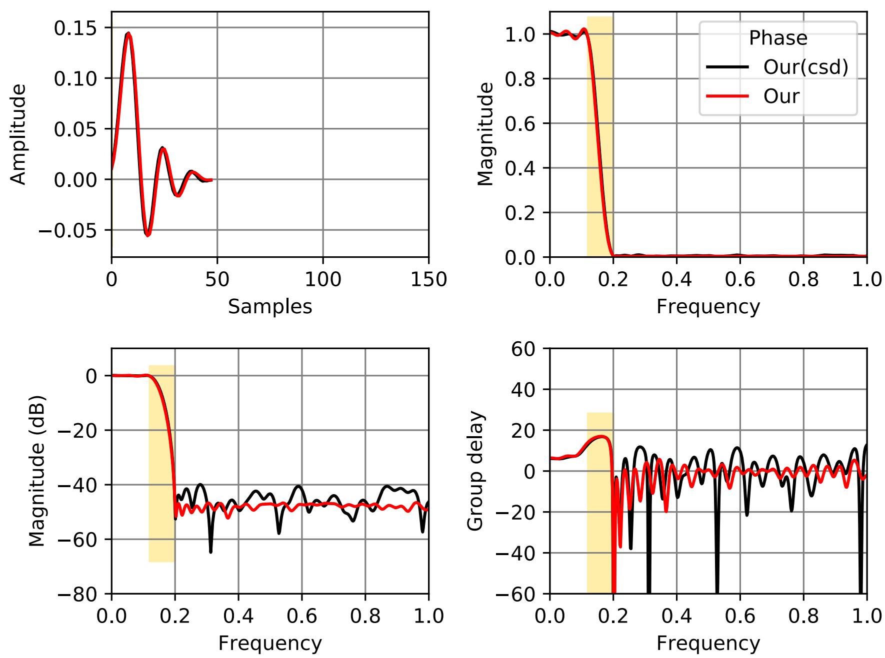


Figure 11: Lowpass