Ellipsoid Method and Its Amazing Oracles*

Wai-Shing Luk†

April 10, 2019

Abstract

Ellipsoid method is revisited. Besides that, three separation oracles are investigated for applications. They are robust optimization, semidefinite programming, and network optimization. Discuss stability issue. Finally, the parallel cut is described.

1 Introduction

The bad reputation of the ellipsoid method is not good. And that is unfair. It is commonly believed that the method is inefficient in practice for large-scale convex problems. The convergent rate is slow. It cannot exploits sparsity. It was supplanted by the interior-point methods. It can be treated as a theoretical tool for proving the polynomial-time solvability of combinatorial optimization problems.

However, the ellipsoid method works very differently compared with the interior point method. it only requires a separation oracle. Thus, it can play nicely with other techniques. Consider 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. Whenever separation oracle can be implemented efficiently.

2 Cutting-plane Method Revisited

2.1 Convex Feasibility Problem

Let $\mathcal{K} \subseteq \mathbb{R}^n$ be a convex set. Consider the feasibility problem:

- Find a point $x^* \in \mathbb{R}^n$ in $\mathcal{K}$,

*This work was supported by the Society for Industrial and Applied Mathematics
†Fudan University

- or determine that $\mathcal{K}$ is empty (i.e., no feasible solution)

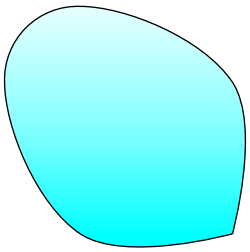


Figure 1: Feasibility region

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 $\mathcal{K}$:

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

{#eq:cut}

The pair (g, h) is called a *cutting-plane*, or cut, since it eliminates the halfspace $\{x \mid g^\top (x - x_0) + h > 0\}$ from our search. If $h = 0$ (x_0 is on the boundary of halfspace that is cut), cutting-plane is called *neutral cut*. If $h > 0$ (x_0 lies in the interior of halfspace that is cut), cutting-plane is called *deep cut*.

The $\mathcal{K}$ is usually given by a set of inequalities $f_j(x) \leq 0$ 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 $f(z) \geq f(x_0) + g^{\mathrm{T}}(z - x_0)$. Hence, the cut (g, h) is given by $(\partial f(x_0), f(x_0))$

Note that if $f(x)$ is differentiable, we can simply take $\partial f(x_0) = \nabla f(x_0)$

Cutting-plane method consists of two key components: separation oracle Ω and a search space $\mathcal{S}$ initially big enough to cover $\mathcal{K}$. For example,

- Polyhedron $\mathcal{P} = \{z \mid Cz \preceq d\}$
- Interval $\mathcal{I} = [l, u]$ (for one-dimensional problem)
- Ellipsoid $\mathcal{E} = \{z \mid (z - x_c)P^{-1}(z - x_c) \leq 1\}$

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:

$$\mathcal{S}^+ = \mathcal{S} \cap \{z \mid g^\top (z - x_0) + h \leq 0\}$$

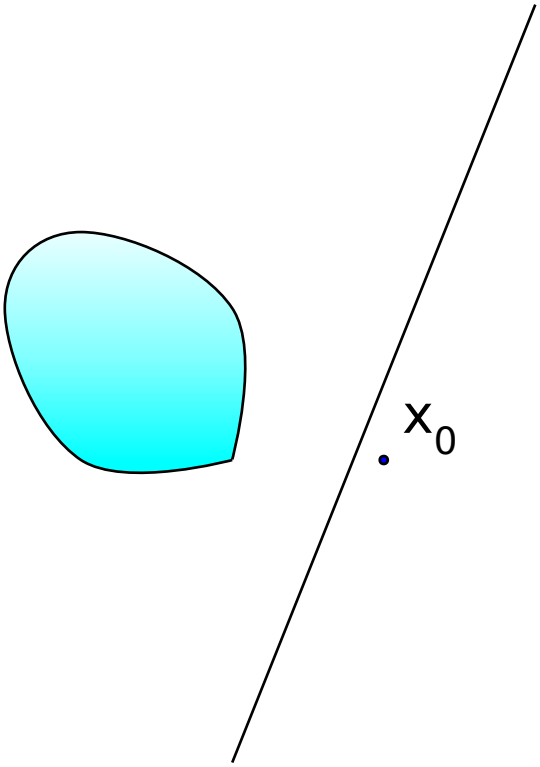


Figure 2: Cut

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

Listing: Feasibility Code

```
def cutting_plane_feas(evaluate, S, options=Options()):
    feasible = False
    status = 0
    for niter in range(options.max_it):
        cut, feasible = evaluate(S.xc)
        if feasible:  # feasible sol'n obtained
            break
        status, tsq = S.update(cut)
        if status != 0:
            break
        if tsq < options.tol:
            status = 2
            break
    return S.xc, niter+1, feasible, status
```

2.2 Convex Optimization Problem

Consider:

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

{#eq:convex_optimization}

The optimization problem is treated as a feasibility problem with an additional constraint $f_0(x) < t$. Here, $f_0(x)$ could be a convex function or a quasiconvex function. t is the best-so-far value of $f_0(x)$. The problem can be reformulated as:

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

{#eq:cvx_in_feasibility_form}

where $\Phi(x,t) < 0$ is the t-sublevel set of $f_0(x)$. Note that $\mathcal{K}_t \subseteq \mathcal{K}_u$ if and only if $t \leq u$ (monotonicity). One easy way to solve the optimization problem is to apply the binary search on t.

Listing: Binary search

```
def bsearch(evaluate, I, options=Options()):
    feasible = False
    l, u = I
```

```
    t = l + (u - l)/2
    for niter in range(options.max_it):
        if evaluate(t):  # feasible sol'n obtained
            feasible = True
            u = t
        else:
            l = t
        tau = (u - l)/2
        t = l + tau
        if tau < options.tol:
            break
    return u, niter+1, feasible

class bsearch_adaptor:
    def __init__(self, P, E, options=Options()):
        self.P = P
        self.E = E
        self.options = options

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

    def __call__(self, t):
        E = self.E.copy()
        self.P.update(t)
        x, _, feasible, _ = cutting_plane_feas(
            self.P, E, self.options)
        if feasible:
            self.E._xc = x.copy()
            return True
        return False
```

2.3 Shrinking

- Another possible way is, to update the best-so-far t whenever a feasible solution x_0 is found such that $\Phi(x_0, t) = 0$.
- We assume that the oracle takes the responsibility for that.

2.4 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:

$$\mathcal{S}^+ = \mathcal{S} \cap \{z \mid g^\top (z - x_0) + h \leq 0\}$$

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

2.5 Corresponding Python code

```
def cutting_plane_dc(evaluate, S, t, options=Options()):
    feasible = False  # no sol'n
    x_best = S.xc
    for niter in range(options.max_it):
        cut, t1 = evaluate(S.xc, t)
        if t != t1:  # best t obtained
            feasible = True
            t = t1
            x_best = S.xc
        status, tau = S.update(cut)
        if status == 1:
            break
        if tau < options.tol:
            status = 2
            break
    return x_best, t, niter+1, feasible, status
```

2.6 Example: Profit Maximization Problem

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

- $p(Ax_1^\alpha x_2^\beta)$: 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

2.7 Example: Profit maximization (cont'd)

- The formulation is not in the convex form.

- Rewrite the problem in the following form:

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

2.8 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:

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

2.9 Python code (Profit oracle)

```
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 + np.dot(self.a, y)
        x = np.exp(y)
        vx = np.dot(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
```

2.10 Python code (Main program)

```
import numpy as np
from profit_oracle import *
from cutting_plane import *
from ell import *

p, A, k = 20.0, 40.0, 30.5
params = p, A, k
a = np.array([0.1, 0.4])
v = np.array([10.0, 35.0])
```

```
y0 = np.array([0., 0.])  # initial x0
E = ell(200, y0)
P = profit_oracle(params, a, v)
yb1, fb, niter, feasible, status = \
    cutting_plane_dc(P, E, 0.0)
print(fb, niter, feasible, status)
```

2.11 Area of Applications

- Robust convex optimization
  - oracle technique: affine arithmetic
- Parametric network potential problem
  - oracle technique: negative cycle detection
- Semidefinite programming
  - oracle technique: Cholesky factorization

3 Robust Convex Optimization

3.1 Robust Optimization Formulation

- Consider:

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

  where q represents a set of varying parameters.

- The problem can be reformulated as:

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

3.2 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 $(g, h) = (\partial f_j(x_0, q_0), f_j(x_0, q_0))$
  - If $f_0(x_0, q) \ge t$ for some $q = q_0$, then
    - the cut $(g, h) = (\partial f_0(x_0, q_0), f_0(x_0, q_0) - t)$
  - 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 $(g, h) = (\partial f_0(x_0, q_{\max}), 0)$
- Random sampling trick

3.3 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 $\pm e_3$.

3.4 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 $\beta = \bar{\beta} + e_2$

Remark: for more complicated problems, affine arithmetic could be used.

3.5 `profit_rb_oracle`

```
class profit_rb_oracle:
    def __init__(self, params, a, v, vparams):
        ui, e1, e2, e3 = vparams
        self.uie = [ui * e1, ui * e2]
        self.a = a
        p, A, k = params
        p -= ui * e3
        k -= ui * e3
        v_rb = v.copy()
        v_rb += ui * e3
        self.P = profit_oracle((p, A, k), a, v_rb)

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

4 Parametric Network Potential Problem

4.1 Parametric Network Potential Problem

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

Consider:

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

- $h_{ij}(x,t)$ is the weight function of edge (i,j),
- Assume: network is large but the number of parameters is small.

4.2 Network Potential Problem (cont'd)

Given x and t, 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.

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & W_k(x,t) \geq 0, \forall C_k \in C, \\ \text{variables} & x \end{array}$$

- C_k is a cycle of G
- $W_k(x,t) = \sum_{(i,j) \in C_k} h_{ij}(x,t)$.

4.3 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 $(g, h) = (-\partial W_k(x_0), -W_k(x_0))$
  - If $f_0(x_0) \geq t$, then
    - the cut $(g, h) = (\partial f_0(x_0), f_0(x_0) - t)$
  - Otherwise, x_0 is feasible, then
    - $t := f_0(x_0)$.
    - The cut $(g, h) = (\partial f_0(x_0), 0)$

4.4 Python Code

```
class network_oracle:
    def __init__(self, G, f, p):
        self.G = G
        self.f = f
        self.p = p  # partial derivative of f w.r.t x
        self.S = negCycleFinder(G)

    def __call__(self, x):
        def get_weight(G, e):
            return self.f(G, e, x)

        self.S.get_weight = get_weight
        C = self.S.find_neg_cycle()
        if C is None:
            return None, 1
        f = -sum(self.f(self.G, e, x) for e in C)
        g = -sum(self.p(self.G, e, x) for e in C)
        return (g, f), 0
```

4.5 Example: Optimal Matrix Scaling

- 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 $U = \text{diag}([u_1, u_2, \ldots, u_N])$. Under the min-max-ratio criterion, the problem can be formulated as:

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

4.6 Optimal Matrix Scaling (cont'd)

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

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

where k' denotes $\log(|k|)$ and $x = (\pi', \psi')^\top$.

4.7 Corresponding Python Code

```
def constr(G, e, x):
    u, v = e
    i_u = G.node_idx[u]
    i_v = G.node_idx[v]
    cost = G[u][v]['cost']
    return x[0] - cost if i_u <= i_v else cost - x[1]

def pconstr(G, e, x):
    u, v = e
    i_u = G.node_idx[u]
    i_v = G.node_idx[v]
    return np.array([1.,  0.] if i_u <= i_v else [0., -1.])

class optscaling_oracle:
    def __init__(self, G):
        self.network = network_oracle(G, constr, pconstr)

    def __call__(self, x, t):
        cut, feasible = self.network(x)
        if not feasible: return cut, t
        s = x[0] - x[1]
        fj = s - t
        if fj < 0.:
            t = s
            fj = 0.
        return (np.array([1., -1.]), fj), t
```

4.8 Example: clock period & yield-driven co-optimization

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

- 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 $\beta \ge 0.8$, the problem becomes convex.

4.9 Inverse CDF

5 Matrix Inequalities

5.1 Problems With Matrix Inequalities

Consider the following problem:

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

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

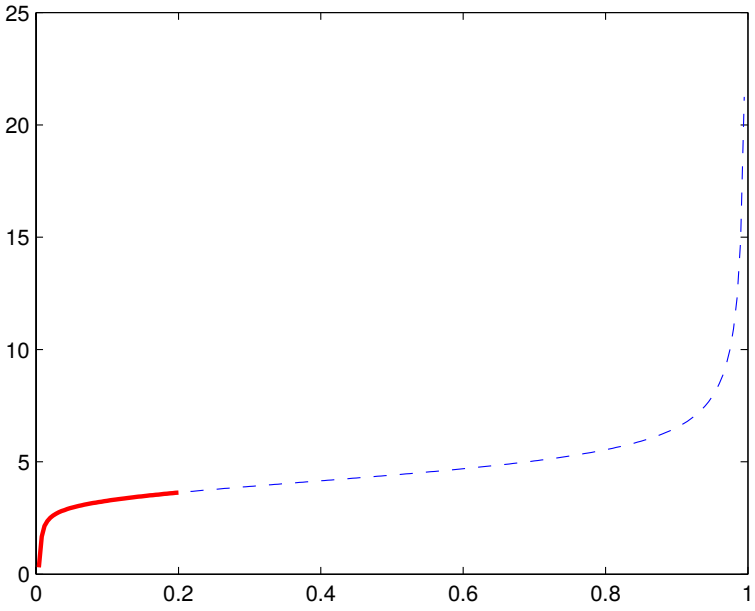


Figure 3: img

5.2 Problems With Matrix Inequalities

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

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

- Consider $v^\top F(x,t)v$ 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 $F(x,t)$ is linear w.r.t. x, i.e., $F(x) = F_0 + x_1F_1 + \cdots + x_nF_n$

5.3 Oracle in Matrix Inequalities

The oracle only needs to:

- Perform a *row-based* Cholesky factorization such that $F(x_0,t) = R^\top R$.
- Let $A_{:p,:p}$ denotes a submatrix $A(1:p, 1:p) \in \mathbb{R}^{p\times p}$.
- If Cholesky factorization fails at row p,
  - there exists a vector $e_p = (0,0,\cdots,0,1)^\top \in \mathbb{R}^p$, such that
    - $v = R^{-1}_{:p,:p}e_p$, and
    - $v^\top F_{:p,:p}(x_0)v < 0$.
  - The cut $(g,h) = (-v^\top \partial F_{:p,:p}(x_0)v, -v^\top F_{:p,:p}(x_0)v)$

5.4 Corresponding Python Code

```
class lmi_oracle:
    ''' Oracle for LMI constraint F*x <= B '''

    def __init__(self, F, B):
        self.F = F
        self.F0 = B
        self.Q = chol_ext(len(self.F0))

    def __call__(self, x):
        n = len(x)

        def getA(i, j):
            return self.F0[i, j] - sum(
                self.F[k][i, j] * x[k] for k in range(n))

        self.Q.factor(getA)
        if self.Q.is_spd():
            return None, True
        v, ep = self.Q.witness()
        g = np.array([self.Q.sym_quad(v, self.F[i])
                      for i in range(n)])
        return (g, ep), False
```

5.5 Example: Matrix Norm Minimization

- Let $A(x) = A_0 + x_1A_1 + \cdots + x_nA_n$
- Problem $\min_x \|A(x)\|$ can be reformulated as

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

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

5.6 Python Code

```
class qmi_oracle:
    t = None
    count = 0

    def __init__(self, F, F0):
        self.F = F
        self.F0 = F0
        self.Fx = np.zeros(F0.shape)
        self.Q = chol_ext(len(F0))

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

    def __call__(self, x):
        self.count = 0; nx = len(x)

        def getA(i, j):
            if self.count < i + 1:
                self.count = i + 1
                self.Fx[i] = self.F0[i]
                self.Fx[i] -= sum(self.F[k][i] * x[k]
                                  for k in range(nx))
            a = -self.Fx[i].dot(self.Fx[j])
            if i == j: a += self.t
            return a

        self.Q.factor(getA)
        if self.Q.is_spd(): return None, True
        v, ep = self.Q.witness()
        p = len(v)
        Av = v.dot(self.Fx[:p])
        g = np.array([-2*v.dot(self.F[k][:p]).dot(Av)
                      for k in range(nx)])
        return (g, ep), False
```

5.7 Example: Estimation of Correlation Function

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

- 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 $\Omega(p)$ can be recast as:

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

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

5.8 Experimental Result

6 Ellipsoid Method Revisited

6.1 Some History of Ellipsoid Method

- Introduced by Shor and Yudin and Nemirovskii in 1976

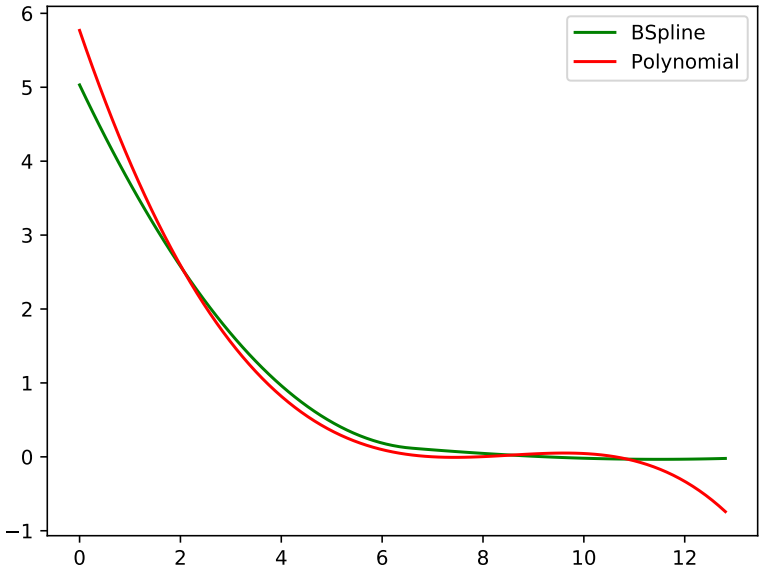


Figure 4: BSpline vs. Polynomail

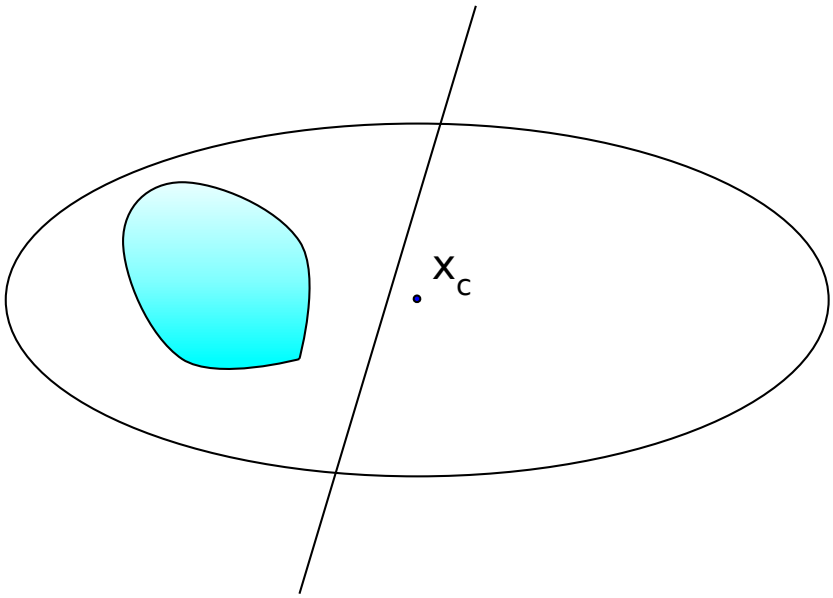
- Used to show that linear programming (LP) is polynomial-time solvable (Kachiyan 1979), settled the long-standing problem of determining the theoretical complexity of LP.
- In practice, however, the simplex method runs much faster than the method, although its worst-case complexity is exponential.

6.2 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.



6.3 Python code

```
import numpy as np

class ell:
    def __init__(self, val, x):
        '''ell = { x | (x - xc)' * P^-1 * (x - xc) <= 1 }'''
        n = len(x)
        if np.isscalar(val):
            self.P = val * np.identity(n)
        else:
            self.P = np.diag(val)
        self.xc = np.array(x)
        self.c1 = float(n*n)/(n*n-1.)

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

6.4 Updating the ellipsoid (deep-cut)

- Calculation of minimum volume ellipsoid covering:

$$\mathcal{E} \cap \{z \mid g^\top (z - x_c) + h \leq 0\}$$

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

$$x_c^+ = x_c - \frac{\rho}{\tau^2}\tilde{g}, \qquad P^+ = \delta \cdot \left(P - \frac{\sigma}{\tau^2}\tilde{g}\tilde{g}^\top\right)$$

where

$$\rho = \frac{\tau + nh}{n+1}, \qquad \sigma = \frac{2\rho}{\tau + h}, \qquad \delta = \frac{n^2(\tau^2 - h^2)}{(n^2-1)\tau^2}$$

6.5 Updating the ellipsoid (cont'd)

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

$$x_c^+ = x_c - \frac{\rho}{\omega}\tilde{g}, \qquad Q^+ = Q - \frac{\sigma}{\omega}\tilde{g}\tilde{g}^\top, \qquad \kappa^+ = \delta \cdot \kappa$$

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

6.6 Python code (updating)

```
def update_core(self, calc_ell, cut):
    g, beta = cut
    Qg = self.Q.dot(g)
    omega = g.dot(Qg)
    tsq = self.kappa * omega
    if tsq <= 0.:
        return 4, 0.
    status, params = calc_ell(beta, tsq)
    if status != 0:
        return status, tsq
    rho, sigma, delta = params
    self._xc -= (rho / omega) * Qg
    self.Q -= (sigma / omega) * np.outer(Qg, Qg)
    self.kappa *= delta
    return status, tsq
```

6.7 Python code (deep cut)

```
def calc_dc(self, beta, tsq):
    '''deep cut'''
    tau = math.sqrt(tsq)
    if beta > tau:
        return 1, None     # no sol'n
    if beta == 0.:
        return self.calc_cc(tau)
    n = self._n
    gamma = tau + n*beta
    if gamma < 0.:
        return 3, None  # no effect
```

```
    rho = gamma/(n + 1)
    sigma = 2.*rho/(tau + beta)
    delta = self.c1*(tsq - beta**2)/tsq
    return 0, (rho, sigma, delta)
```

6.8 Central Cut

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

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

6.9 Python code (deep cut)

```
def calc_cc(self, tau):
    '''central cut'''
    np1 = self._n + 1
    sigma = 2. / np1
    rho = tau / np1
    delta = self.c1
    return 0, (rho, sigma, delta)
```

7 Parallel Cuts

7.1 Parallel Cuts

- Oracle returns a pair of cuts instead of just one.
- The pair of cuts is given by g and (β_1, β_2) such that:

$$g^\top (x - x_c) + \beta_1 \leq 0,$$
$$g^\top (x - x_c) + \beta_2 \geq 0,$$

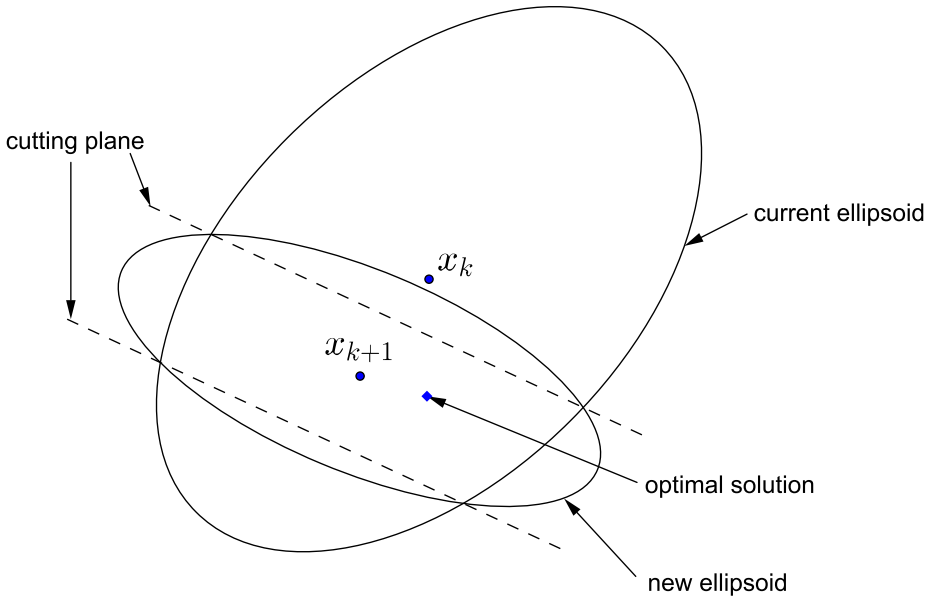
  for all $x \in \mathcal{K}$.

- Only linear inequality constraint can produce such parallel cut:

$$l \leq a^\top x + b \leq u, \qquad L \preceq F(x) \preceq U$$

- Usually provide faster convergence.

7.2 Parallel Cuts



7.3 Updating the ellipsoid

- Let $\tilde{g} = Q\,g$, $\tau^2 = \kappa \cdot \omega$.
- If $\beta_1 > \beta_2$, intersection is empty.
- If $\beta_1\beta_2 < -\tau^2/n$, no smaller ellipsoid can be found.
- If $\beta_2^2 > \tau^2$, it reduces to deep-cut with $\alpha = \alpha_1$.
- Otherwise,

$$x_c^+ = x_c - \frac{\rho}{\omega}\tilde{g}, \qquad Q^+ = Q - \frac{\sigma}{\omega}\tilde{g}\tilde{g}^\top, \qquad \kappa^+ = \delta\kappa$$

where

$$\begin{aligned}
\bar{\beta} &= (\beta_1 + \beta_2)/2 \\
\xi^2 &= (\tau^2 - \beta_1^2)(\tau^2 - \beta_2^2) + (n(\beta_2 - \beta_1)\bar{\beta})^2, \\
\sigma &= (n + (\tau^2 - \beta_1\beta_2 - \xi)/(2\bar{\beta}^2))/(n+1), \\
\rho &= \bar{\beta} \cdot \sigma, \\
\delta &= (n^2/(n^2-1))(\tau^2 - (\beta_1^2 + \beta_2^2)/2 + \xi/n)/\tau^2
\end{aligned}$$

7.4 Python code (parallel cut)

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

7.5 Example: FIR filter design

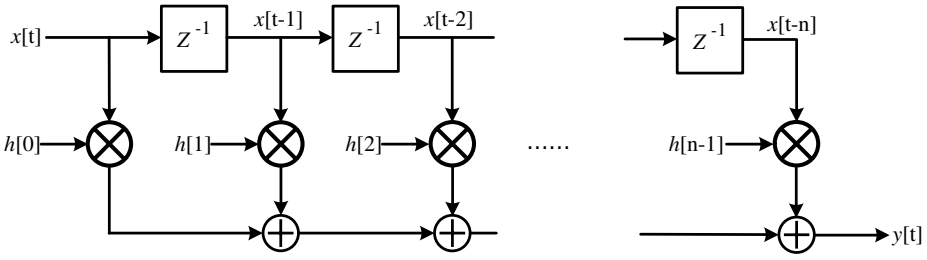


Figure 5: img

- The time response is:

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

7.6 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

$$L(\omega) \;\leq\; |H(\omega)| \;\leq\; U(\omega),\; \forall\, \omega \in (-\infty, +\infty)$$

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

- The constraint is non-convex in general.

7.7 Example: FIR filter design (cont'd)

- However, via *spectral factorization*, it can transform into a convex one:

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

  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.

7.8 Example: FIR filter design (cont'd)

- $\mathbf{r}$ 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:

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

7.9 Experiment

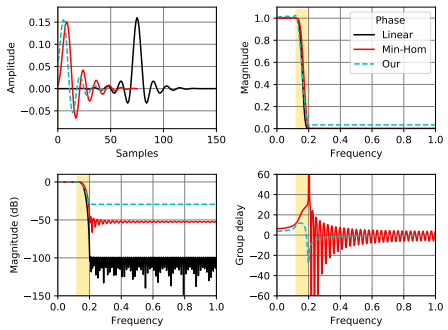


Figure 6: Result

7.10 Example: Maximum Likelihood estimation

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

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 $[0, 2Y]$

7.11 Example: Maximum Likelihood estimation (cont'd)

- Therefore, the following problem is convex:

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

8 Discrete Optimization

8.1 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.

8.2 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?

8.3 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.

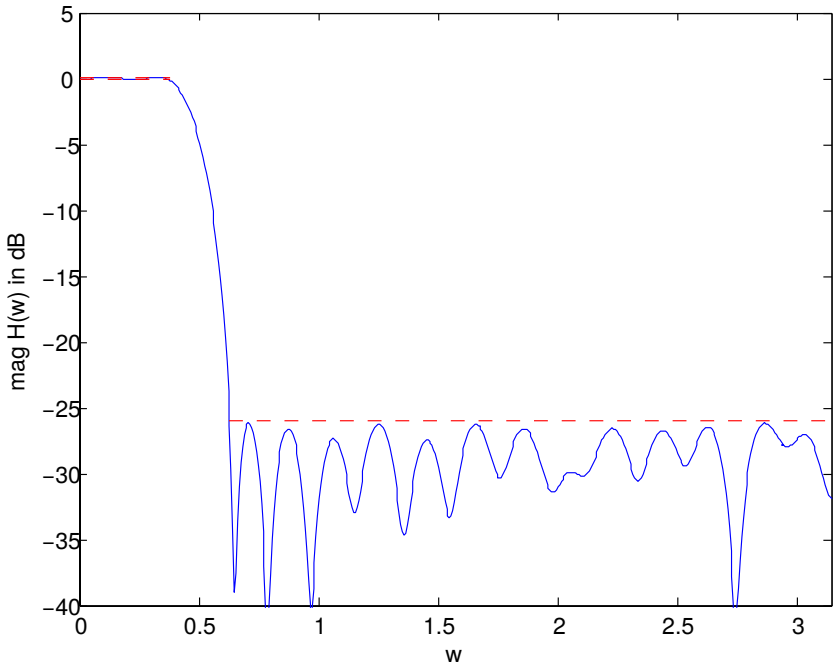
8.4 Oracle Requirement

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

$$g^\top (x - x_d) + \beta \leq 0, \beta \geq 0, g \neq 0$$

- 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)

8.5 Example: FIR filter design



References