Ellipsoid Method and the Amazing Oracles (II)

Wai-Shing Luk

Fudan University

October 30, 2021

Revisiting the ellipsoid method

Parallel Cuts

Discrete Optimization

Revisiting the ellipsoid method

Some history of the ellipsoid Method [Bland et al., 1981]

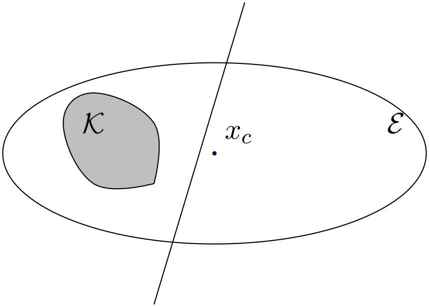
- Proposed by Shor and Yudin and Nemirovskii in 1976.
- used to prove that linear programming (LP) is polynomial time solvable (Kachiyan 1979), settling the long-standing problem of determining the theoretical complexity of LP.
- However, in practice, the simplex method runs much faster, despite its exponential worst-case complexity.

The 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

```python
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):...
```

Updating the ellipsoid (deep-cut)

Compte the minimum volume ellipsoid covering:

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

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

Otherwise,

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

where

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

Updating the ellipsoid (cont'd)

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

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

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

Python code (updating)

```python
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
```

Python code (deep cut)

```python
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)
```

Central Cut

- It is a special case of deep cut where $\beta = 0$
- It is worth implementing it separately, as it is much simpler.
- Let $\tilde{g} = Q\,g$, $\tau = \sqrt{\kappa \cdot \omega}$,

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

Python code (central cut)

```python
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)
```

Parallel Cuts

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:

  $$\begin{array}{l} g^{\mathsf{T}}(x - x_c) + \beta_1 \leq 0, \\ g^{\mathsf{T}}(x - x_c) + \beta_2 \geq 0, \end{array}$$

  for all $x \in \mathcal{K}$.
- Only linear inequality constraints can produce such a parallel cut:

  $$l \leq a^{\mathsf{T}} x + b \leq u, \qquad L \preceq F_0 + x_1 F_1 + \cdots + x_n F_n \preceq U.$$

- usually provides faster convergence.

Parallel Cuts

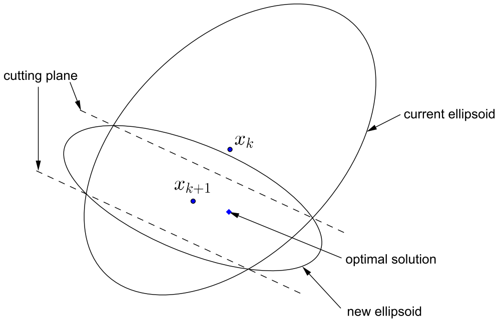


Figure 1: Parallel Cut

Updating the ellipsoid

- Let $\tilde{g} = Q\,g$ and $\tau^2 = \kappa \cdot \omega$.
- If $\beta_1 > \beta_2$, the 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 a deep-cut with $\alpha = \alpha_1$.
- Otherwise,

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

where

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

Python code (parallel cut)

```python
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)
```

Example: FIR filter design

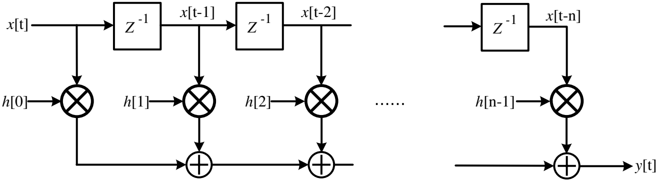


Figure 2: A typical structure of an FIR filter Mitra and Kuo [2006].

- 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 constraint on frequency domain is 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 (non-negative) bounds at the frequency ω, respectively.

- The constraint is not convex in general.

Example: FIR filter design (II)

- However, via *spectral factorization* [Goodman et al., 1997], it can be transformed into a convex one [Wu et al., 1999]:

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

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.

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:

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

Experiment

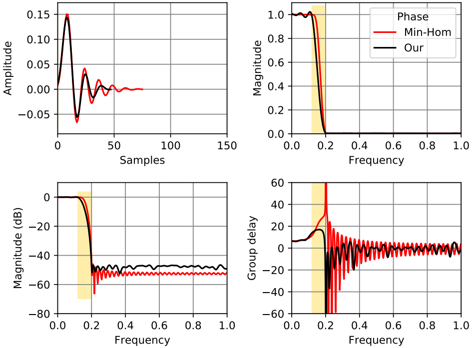


Figure 3: Result

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

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

Note that the 1st term is concave and the 2nd term is convex

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

Example: Maximum Likelihood Estimation (cont'd)

- Thus, the following problem is convex:

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

Discrete Optimization

Why discrete convex programming

- Many engineering problems can be formulated as a convex/geometric programming, such as digital circuit sizing
- Yet in ASIC design, there is usually only a limited set of choices of cell in the library. In other words, some design variables are discrete.
- The discrete version can be formulated as mixed-integer convex programming (MICP), which maps design variables to integers.

What's wrong with the existing approach?

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

Mixed integer convex programming

Consider:

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

where

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

Oracle Requirement

- Oracle looks for a 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 various cuts as possible in each iteration (e.g. round-robin the evaluation of the constraints)

Example: Multiplier-less FIR filter design

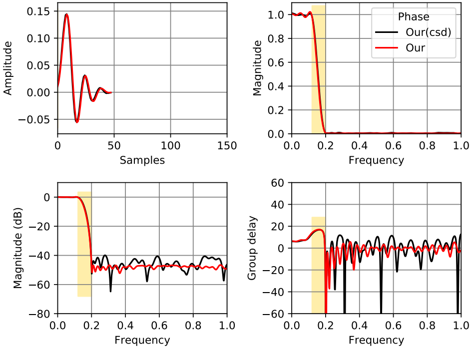


Figure 4: Result

Reference I

R. G. Bland, D. Goldfarb, and M. J. Todd. The ellipsoid method: A survey. *Operations Research*, 29(6):1039–1091, 1981.

T. N. T. Goodman, C. A. Micchelli, G. Rodriguez, , and S. Seatzu. Spectral factorization of laurent polynomials. *Advances Comput. Math.*, 7(4):429–454, 1997.

Sanjit Kumar Mitra and Yonghong Kuo. *Digital signal processing: a computer-based approach*, volume 2. McGraw-Hill New York, 2006.

Shao-Po Wu, Stephen Boyd, and Lieven Vandenberghe. FIR filter design via spectral factorization and convex optimization. In *Applied and Computational Control, Signals, and Circuits*, pages 215–245. Springer, 1999.