Ellipsoid Method and the Amazing Oracles (I)

Wai-Shing Luk

Fudan University

October 29, 2019

Introduction

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

Sir Arthur Conan Doyle, stated by Sherlock Holmes

Common Perspective of Ellipsoid Method

- It is commonly believed that it is inefficient in practice for large-scale problems.
  - The convergent rate is slow, even with the use of deep cuts.
  - Cannot exploit sparsity.
- Since then, it was supplanted by interior-point methods.
- Only treated as a theoretical tool for proving the polynomial-time solvability of combinatorial optimization problems.

But. . .

- The ellipsoid method works very differently compared with the interior point method.
- Only require a *separtion oracle*. Can play nicely with other techniques.
- The ellipsoid method itself cannot exploit sparsity, but the oracle can.

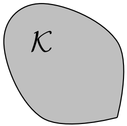
Consider Ellipsoid Method When…

- 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
- Oracle can be implemented efficiently.

Cutting-plane Method Revisited

Basic Idea

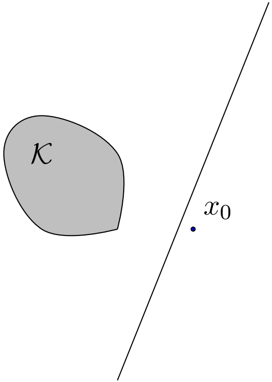
- Let $\mathcal{K} \subseteq \mathbb{R}^n$ be a convex set.
- Consider the feasibility problem:
  - Find a point $x^* \in \mathbb{R}^n$ in $\mathcal{K}$,
  - or determine that $\mathcal{K}$ is empty (i.e., no feasible solution)



Separation Oracle

- 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 $\mathcal{K}$:

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



Separation oracle (cont'd)

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

Subgradient

- $\mathcal{K}$ 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 $f(z) \geq f(x_0) + g^{\mathsf{T}}(z - x_0)$.
- 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

- Cutting plane oracle Ω
- A search space $\mathcal{S}$ initially big 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 $\mathcal{E} = \{z \mid (z - x_c)P^{-1}(z - x_c) \leq 1\}$

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:

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

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

Corresponding Python code

```python
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:  # empty cut
            break
        if tsq < options.tol:
            status = 2
            break
    return S.xc, niter+1, feasible, status
```

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 $f_0(x) \leq t$
- $f_0(x)$ could be a convex function 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:

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

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

```python
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
```

```python
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
```

Shrinking

- Another possible way is, to update the best-so-far t whenever a feasible solution x_0 is found by solving the equation $\Phi(x_0, t_{\text{new}}) = 0$.
- If the equation is difficult to solve but t is also convex w.r.t. Φ, then we may create a new varaible, $say x_{n+1}$ and let $x_{n+1} \leq t'$.

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_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 != 0:  # empty cut
            break
        if tau < options.tol:
            status = 2
            break
    return x_best, t, niter+1, feasible, status
```

Example - Profit Maximization Problem

This example is taken from [Aliabadi and Salahi, 2013].

$$\begin{array}{ll} \text{maximize} & p(Ax_1^{\alpha}x_2^{\beta}) - v_1x_1 - v_2x_2 \\ \text{subject to} & x_1 \leq k. \end{array}$$

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

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

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 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
```

```python
# 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)
```

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

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:

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

Example - Profit Maximization Problem (convex)

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

- 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):
        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)
```

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 [Liu et al., 2007].

Multi-parameter Network Problem

Parametric Network Problem

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

Consider:

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

- $h_{ij}(x)$ 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 [Cherkassky and Goldberg, 1999], in which Tarjan's algorithm [Tarjan, 1981] is one of the fastest algorithms in practice [Dasdan, 2004, Cherkassky and Goldberg, 1999].

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 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
```

Example - Optimal Matrix Scaling [Orlin and Rothblum, 1985]

- 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:

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

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

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
```

Example - clock period & yield-driven co-optimization

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

- 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:

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

Matrix Inequalities

Problems With Matrix Inequalities

Consider the following problem:

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

- $F(x)$: 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
    - $v = R^{-1}_{:p,:p} e_p$, 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 $O(p^3)$ per iteration, independent of N!

```python
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
```

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 (I)

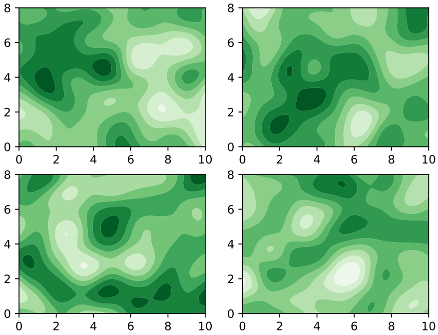


Figure 1: Data Sample (kern=0.5)

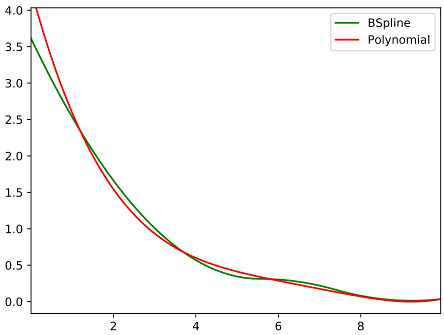


Figure 2: Least Square Result

Experimental Result (II)

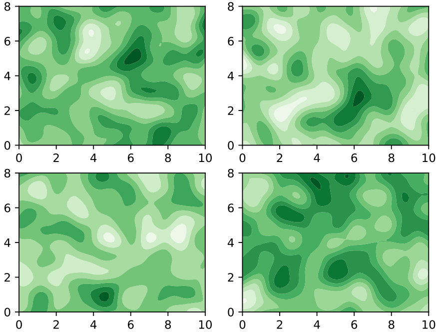


Figure 3: Data Sample (kern=1.0)

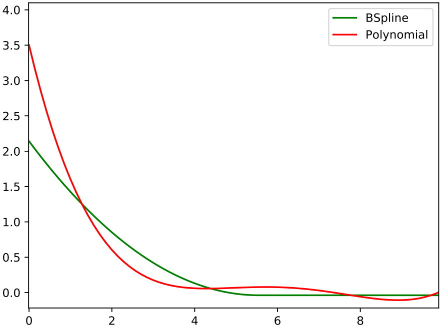


Figure 4: Least Square Result

Experimental Result (III)

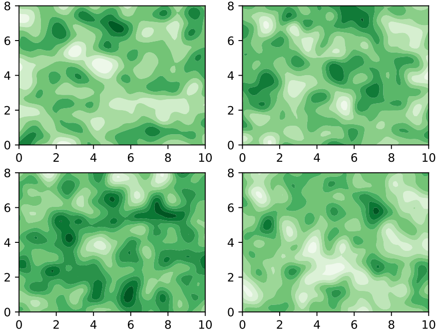


Figure 5: Data Sample (kern=2.0)

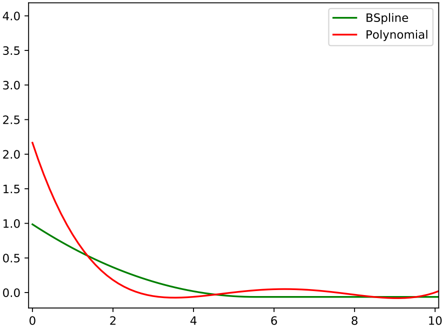


Figure 6: Least Square Result

Reference I

Hossein Aliabadi and Maziar Salahi. Robust geometric programming approach to profit maximization with interval uncertainty. *Computer Science Journal of Moldova*, 21(1):86–96, 2013.

Boris V Cherkassky and Andrew V Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85(2):277–311, 1999.

Ali Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Transactions on Design Automation of Electronic Systems*, 9(4): 385–418, 11 2004.

Xuexin Liu, Wai-Shing Luk, Yu Song, Pushan Tang, and Xuan Zeng. Robust analog circuit sizing using ellipsoid method and affine arithmetic. In *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 203–208. IEEE Computer Society, 2007.

Reference II

James B Orlin and Uriel G Rothblum. Computing optimal scalings by parametric network algorithms. *Mathematical programming*, 32(1):1–10, 1985.

R.E. Tarjan. Shortest paths. Technical report, AT&T Bell Laboratories, 1981.