# Cutting-plane Method and the Amazing Oracles (I)

Wai-Shing Luk

Fudan University

April 12, 2019

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

*Sir Arthur Conan Doyle, stated by Sherlock Holmes*

# Introduction

# Some History of Ellipsoid Method

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

# 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 cutting-plane oracle. Can play nicely with other techniques.
- The oracle can exploit sparsity.

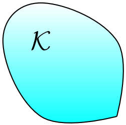
# Consider Ellipsoid Method When...

- 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

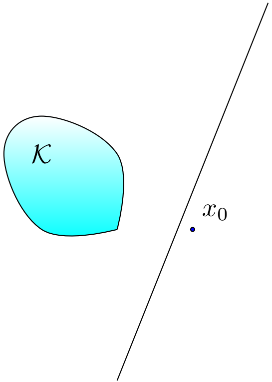
- Let $\mathcal{K} \subseteq \mathbb{R}^n$ be a convex set.
- Consider the feasibility problem:
  - Find a point $x^* \in \mathbb{R}^n$ in $\mathcal{K}$,
  - or determine that $\mathcal{K}$ is empty (i.e., no feasible solution)



# Separation Oracle

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

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



# Separation oracle (cont'd)

- $(g, h)$ 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*.

# 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 $f(z) \geq f(x_0) + g^{\mathrm{T}}(z - x_0)$.
- Hence, the cut $(g, h)$ 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 $\Omega$
- 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:

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

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

# Convex Optimization Problem (I)

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

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

# Convex Optimization Problem (II)

- Problem can be reformulated as:

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

  where $\Phi(x, t) < 0$ is the $t$-sublevel set of $f_0(x)$.

- 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 $t$.

# Corresponding Python code

```
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 such that $\Phi(x_0, t) = 0$.
- We assume that the oracle takes the responsibility for that.

# 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.

# Corresponding Python code

```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 == 1:
            break
        if tau < options.tol:
            status = 2
            break
    return x_best, t, niter+1, feasible, status
```

# Example: Profit Maximization Problem

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

- $p(Ax_1^{\alpha}x_2^{\beta})$ : Cobb-Douglas production function
- $p$: the market price per unit
- $A$: the scale of production
- $\alpha, \beta$: 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:

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

# Profit maximization in Convex Form

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

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

# Python code (Profit oracle) I

```
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 code (Main program) I

```python
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 factorization

# Robust Convex Optimization

# Robust Optimization Formulation

- Consider:

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

  where $q$ represents a set of varying parameters.

- The problem can be reformulated as:

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

# 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 $f_0(x_0, q) \geq t$ 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

# Example: Profit Maximization Problem (convex)

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

- 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 $\beta = \bar{\beta} + e_2$
  ***Remark***: *for more complicated problems, affine arithmetic could be used.*

# profit_rb_oracle

```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)
```

# Parametric Network Potential Problem

# 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.

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

# 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 $(g, h) = (\partial f_0(x_0), 0)$

# 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

- 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:

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

# 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 $x = (\pi', \psi')^\top$.

# Corresponding Python Code

```python
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
```

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

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

- 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 $\beta$, say $\beta \geq 0.8$, the problem becomes convex.

# Inverse CDF

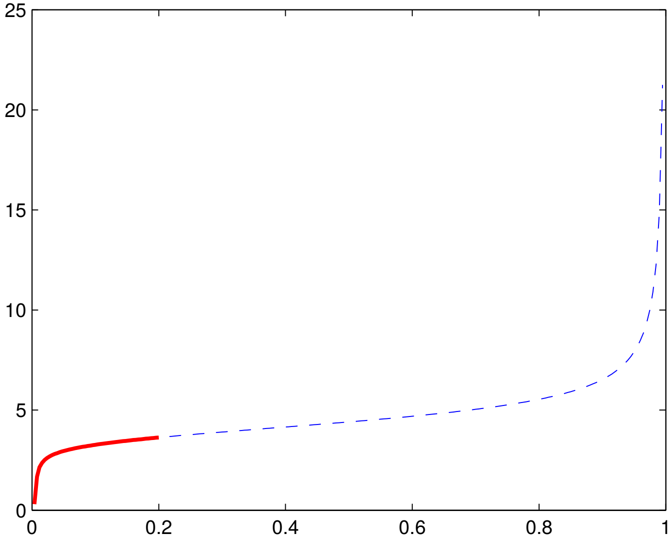



Figure 1: img

# Matrix Inequalities

# 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.

# Problems With Matrix Inequalities

- Recall that a matrix $A$ is positive semidefinite if and only if $v^\top A v \geq 0$ 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., $F(x) = F_0 + x_1 F_1 + \cdots + x_n F_n$

# 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
    - $v = R_{:p,:p}^{-1} e_p$, and
    - $v^\top F_{:p,:p}(x_0) v < 0$.
  - The cut $(g, h) = (-v^\top \partial F_{:p,:p}(x_0) v, -v^\top F_{:p,:p}(x_0) v)$

# Corresponding Python Code

```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
```

# 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

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

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

# 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)
```

# Example: Estimation of Correlation Function

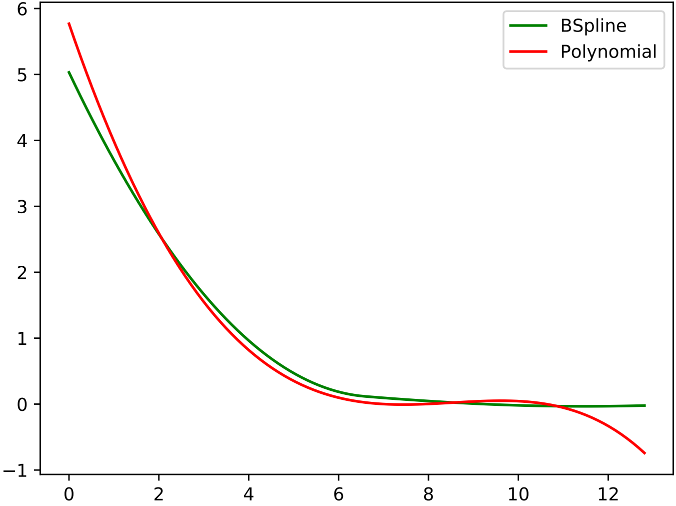
$$\begin{array}{ll} \min_{\kappa,p} & \|\Omega(p) + \kappa I - Y\| \\ \text{s. t.} & \Omega(p) \succcurlyeq 0, \kappa \geq 0 \, . \end{array}$$

- Let $\rho(h) = \sum_i^n p_i \Psi_i(h)$, where
  - $p_i$'s are the unknown coefficients to be fitted
  - $\Psi_i$'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 $\{F_k\}_{i,j} = \Psi_k(\|s_j - s_i\|_2)$

# Experimental Result



Q & A