Ellipsoid Method and the Amazing Oracles

Wai-Shing Luk

Abstract

The ellipsoid method is an optimization technique that offers distinct advantages over interior-point methods because it does not require evaluating all constraint functions. This makes it ideal for convex problems with many or even infinite constraints. The method utilizes an ellipsoid as a search space and employs a separation oracle to provide a cutting plane for updating it. Notably, the importance of the separation oracle is often overlooked. This article evaluates the usage of the ellipsoid method in three specific applications, namely robust convex optimization, semidefinite programming, and parametric network optimization. The effectiveness of separation oracles is analyzed for each application. We also discuss implementation issues of the ellipsoid method, such as utilizing parallel cuts to update the ellipsoid. In some instances, parallel cuts can drastically reduce computation time, as observed in FIR filter design. Discrete optimization is also investigated, illustrating how the ellipsoid method can be applied to problems that involve discrete design variables. An oracle implementation is required solely for locating the nearest discrete solutions

# Introduction

The ellipsoid method’s reputation suffers due to its perceived slowness when solving large-scale convex problems in comparison to interior-point methods. This perception is unfair. Unlike the interior-point method, the ellipsoid method does not need explicit evaluation of all constraint functions. Instead, it utilizes an ellipsoid as a search space and only requires a separation oracle that provides a *cutting plane* (§ 1). The method is well-suited for problems that involve a moderate number of design variables but have many or even infinite constraints. Some criticize the method, claiming it is unable to leverage sparsity. Nevertheless, although the ellipsoid method cannot take advantage of the sparsity of the problem, the separation oracle is capable of taking advantage of certain structural types.

Despite decades of investigation into the ellipsoid method [[1](#ref-BGT81)], the importance of the separation oracle is often overlooked. In this article, we examine three specific applications, namely robust convex optimization, network optimization, and semidefinite programming. The effectiveness of separation oracles is analyzed for each application.

Robust optimization incorporates parameter uncertainties into the optimization problem by analyzing the worst-case scenario. The goal is to find a reliable solution that performs optimally under various possible parameter values in a given set of uncertainties. A robust counterpart of a convex problem preserves its convexity, although the number of constraints grows infinitely. This makes the ellipsoid method an excellent choice for tackling such problems. This is detailed in § 2.1.

An example of network optimization where the ellipsoid method can be employed is also presented. The separation oracle involves constructing a cutting plane by identifying a negative cycle in a network graph. There are algorithms available for negative cycle finding that utilize network locality and other properties, resulting in an effective oracle implementations. This is discussed in more detail in § 2.2.

Meanwhile, § 2.3 describes concerns surrounding matrix inequalities. Recall that utilizing Cholesky or LDLT decomposition can efficiently check the positive definiteness of a symmetric matrix. If a symmetric matrix with dimensions encounters a non-positive diagonal entry during decomposition that causes the process to stop at row , cannot be positive definite. In such cases, a witness vector can be constructed to certify that . With the row-based decomposition and a lazy evaluation technique, the cutting plane can be constructed in , enabling its use for efficient oracle implementations.

The implementation of the ellipsoid method is discussed in § 3. Basically it generates a sequence of ellipsoids whose volume uniformly decreases at every step. The ellipsoid is conventionally represented as:

where is the center of the ellipsoid. The matrix is symmetric positive definite. During each iteration, the ellipsoid method updates and . While updating ellipsoids can be straightforward and has been implemented for decades, we show that the cost can be further reduced by floating point operations by splitting into and , resulting in this form:

In addition, § 3.3 discusses the use of parallel cuts. Some articles suggest that this technique does not lead to significant improvements. However, we show that in situations where certain constraints have tight upper and lower bounds, such as in FIR filter designs, the implementation of parallel cuts can significantly speed up the runtime.

In addition, we show that if the ellipsoid method is implemented carefully, any update, whether it uses a single cut or a parallel cut, results in at most one square root operation.

In many practical engineering problems, some design variables may be restricted to discrete forms. Since the cutting-plane method requires only a separation oracle, it can also be used for discrete problems…

# 1 Cutting-plane Method Revisited

## 1.1 Convex Feasibility Problem

Let be a compact and convex set. Consider the feasibility problem:

1. Find a point in , or
2. Determine that is empty (i.e., has no feasible solution).

A separation oracle, also called a cutting-plane oracle, is a technique used to describe a convex set that serves as input to a cutting-plane method. When a separation oracle is *queried* at , it produces one of the following:

1. Assert that , or
2. Return a hyperplane that separates from :

The pair of is called a *cutting-plane*, because it eliminates the half-space from our search. We have the following observations:

* If ( is on the boundary of the half-space), the cutting-plane is called *central-cut*.
* If ( lies in the interior of the half-space), the cutting-plane is called *deep-cut*.
* If ( lies in the exterior of the half-space), the cutting-plane is called *shadow-cut*.

The convex set is usually given by a set of inequalities or for , where is a convex function. The vector is called the *sub-gradient* of a convex function at if . Thus, the cut is given by . Note that if is differentiable, we can simply take .

The cutting-plane method consists of two key components: separation oracle and a search space initially sufficiently large to cover . For example,

* Polyhedron = .
* Ellipsoid = .
* Interval = (for one-dimensional problem).

Denote the center of the current as . Here is a basic outline of how the cutting-plane method works:

1. **Initialization**: Start with a search space that is guaranteed to contain a .
2. **Iteration**: In each iteration, query the separation oracle at . If , then quit.
3. **Update**: Compute the smaller that contains the half-space from step 2.
4. **Repeat**: Repeat steps 2 and 3 until is empty or it is small enough.

## 1.2 From Feasibility to Optimization

Now consider:

where can be a convex function or a quasi-convex function. We treat the above optimization problem as a feasibility problem with an additional constraint , where is called the best-so-far value of . Thus, we can reformulate the problem as:

where is the -sublevel set of when is quasi-convex. For every , is a non-increasing function of , i.e., whenever . Denote as the new constraint set.

An easy way to solve the optimization problem is to apply a binary search on and solve the corresponding feasibility problems at each . Another possibility is to update the best-so-far whenever a feasible solution is found such that .

Here is a basic outline of how the cutting-plane method (optim) works:

1. **Initialization**: Start with a search space that is guaranteed to contain a .
2. **Iteration**: In each iteration, query the separation oracle at . Compute a subgradient of the function at . This gives us a half-space that is guaranteed to contain a .
3. If , update such that .
4. **Update**: Compute the smaller that contains the half-space from step 2.
5. **Repeat**: Repeat steps 2 and 3 until is empty or it is small enough.

Generic Cutting-plane method (Optim)

* **Given** initial known to contain .
* **Repeat**
  1. Choose a point in
  2. Query the separation oracle at
  3. **If** , update such that .
  4. Update to a smaller set that covers:
  5. **If** or it is small enough, exit.

We assume that the oracle takes responsibility for that.

def cutting\_plane\_optim(omega, space, gamma, options=Options()):  
 x\_best = None  
 for niter in range(options.max\_iters):  
 cut, t1 = omega.assess\_optim(space.xc(), t)  
 if t1 is not None: # better gamma obtained  
 gamma = t1  
 x\_best = copy.copy(space.xc())  
 status = space.update\_central\_cut(cut)  
 else:  
 status = space.update\_deep\_cut(cut)  
 if status != CutStatus.Success or space.tsq() < options.tol:  
 return x\_best, gamma, niter  
 return x\_best, gamma, options.max\_iters

## 1.3 Example: Profit Maximization

This example is taken from [[2](#ref-Aliabadi2013Robust)]. Consider the following *short-run* profit maximization problem:

where is the market price per unit, is the scale of production, and are output elasticities, and are the i-th input quantity and output price, is the Cobb-Douglas production function which is a widely accepted model used to represent the relationship between inputs and outputs in production. is a constant that limits the quantity of . The above formulation is not in convex form. First, we rewrite the problem:

By the change of variables, we can obtain the following convex form of Eq. 1:

where and .

class ProfitOracle(OracleOptim):  
 def \_\_init\_\_(self, params, elasticities, price\_out):  
 unit\_price, scale, limit = params  
 self.log\_pA = math.log(unit\_price \* scale)  
 self.log\_k = math.log(limit)  
 self.price\_out = price\_out  
 self.elasticities = elasticities  
  
 def assess\_optim(self, y, t):  
 if (fj := y[0] - self.log\_k) > 0.0: # constraint  
 grad = np.array([1.0, 0.0])  
 return (grad, fj), None  
  
 log\_Cobb = self.log\_pA + self.elasticities.dot(y)  
 q = self.price\_out \* np.exp(y)  
 vx = q[0] + q[1]  
 if (fj := math.log(t + vx) - log\_Cobb) >= 0.0:  
 grad = q / (t + vx) - self.elasticities  
 return (grad, fj), None  
  
 gamma = np.exp(log\_Cobb) - vx  
 grad = q / (t + vx) - self.elasticities  
 return (grad, 0.0), t

Some readers may recognize that we can also write the problem in a geometric program by introducing one additional variable [[2](#ref-Aliabadi2013Robust)].

# 2 Amazing Oracles

* Robust convex optimization
  + oracle technique: affine arithmetic
* Parametric network potential problem
  + oracle technique: negative cycle detection
* Semidefinite programming
  + oracle technique: Cholesky decomposition

## 2.1 Robust Convex Optimization

Overall, robust optimization accounts for parameter uncertainties by formulating problems that consider worst-case scenarios. This approach allows for more reliable and robust solutions when dealing with uncertainty. The study presented in this paper addresses profit maximization using a robust geometric programming approach with interval uncertainty. The authors study the well-established Cobb-Douglas production function and introduce an approximate equivalent of the robust counterpart using piecewise convex linear approximations. This approximation takes the form of a geometric programming problem. An example is used to demonstrate the impact of uncertainties.

Interval uncertainties refer to uncertainties in model parameters that are represented as intervals. The authors of this study consider interval uncertainties in the model parameters. Upper and lower piecewise convex linear approximations of the robust counterpart are presented that are efficiently solvable using interior point methods. These approximations are used to incorporate the interval uncertainties into the model.

Consider:

where represents a set of varying parameters. We can reformulate the problem as:

### Algorithm

The oracle only needs to determine:

* If for some and , then
* the cut =
* If for some , then
* the cut =
* Otherwise, is feasible, then
* Let .
* .
* The cut =

### Example: Robust Profit Maximization {#sec:profit-rb}

Consider again the profit maximization problem in § 1.3. Uncertainties in the model parameters over a given interval. Now suppose that the parameters are subject to interval uncertainties: [[2](#ref-Aliabadi2013Robust)].

The problem formulation of the robust counterpart considering the worst-case scenario is:

In [[2](#ref-Aliabadi2013Robust)], the authors propose the use of piecewise convex linear approximations as a close approximation of the robust counterpart, leading to more solvability using interior-point algorithms. This requires a lot of programming, but the results are imprecise. However, this can be easily solved using the cutting plane method. Note that in this simple example, the worst case happens when:

* ,
* , ,
* if , , else
* if , , else

We can even reuse the original oracle to compose the robust counterpart.

class ProfitRbOracle(OracleOptim):  
 def \_\_init\_\_(self, params, elasticities, price\_out, vparams):  
 e1, e2, e3, e4, e5 = vparams  
 self.elasticities = elasticities  
 self.e = [e1, e2]  
 unit\_price, scale, limit = params  
 params\_rb = unit\_price - e3, scale, limit - e4  
 self.omega = ProfitOracle(  
 params\_rb, elasticities, price\_out + np.array([e5, e5])  
 )  
  
 def assess\_optim(self, y, t):  
 a\_rb = copy.copy(self.elasticities)  
 for i in [0, 1]:  
 a\_rb[i] += -self.e[i] if y[i] > 0.0 else self.e[i]  
 self.omega.elasticities = a\_rb  
 return self.omega.assess\_optim(y, t)

Note that the argmax may be non-convex and therefore difficult to solve. For more complex problems, one way is to use affine arithmetic for help [[3](#ref-liu2007robust)].

## 2.2 Multi-parameter Network Problems

Given a network represented by a directed graph . Consider :

where is the weight function of edge .

Assume that the network is large but the number of parameters is small. Given and , the problem has a feasible solution if and only if contains no negative cycle. Let be a set of all cycles of . We can formulate the problem as:

where is a cycle of :

The minimum cycle ratio (MCR) problem is a fundamental problem in the analysis of directed graphs. Given a directed graph, the MCR problem seeks to find the cycle with the minimum ratio of the sum of the edge weights to the number of edges in the cycle. In other words, the MCR problem tries to find the “tightest” cycle in the graph, where the tightness of a cycle is measured by the ratio of the total weight of the cycle to its length.

The MCR problem has many applications in the analysis of discrete event systems, such as digital circuits and communication networks. It is closely related to other problems in graph theory, such as the shortest path problem and the maximum flow problem. Efficient algorithms for solving the MCR problem are therefore of great practical importance.

### Negative Cycle Detection Algorithm

The negative cycle detection is the most time-consuming part of the proposed method, so it is very important to choose the proper negative cycle detection algorithm. There are lots of methods to detect negative cycles in a weighted graph [[4](#ref-cherkassky1999negative)], in which Tarjan’s algorithm [[5](#ref-Tarjan1981negcycle)] is one of the fastest algorithms in practice [[4](#ref-cherkassky1999negative),[6](#ref-alg:dasdan_mcr)].

Howard’s method is a minimum cycle ratio (MCR) algorithm that uses a policy iteration algorithm to find the minimum cycle ratio of a directed graph. The algorithm maintains a set of candidate cycles and iteratively updates the cycle with the minimum ratio until convergence.

To detect negative cycles, Howard’s method uses a cycle detection algorithm based on the Bellman-Ford algorithm. Specifically, the algorithm maintains a predecessor graph of the original graph and performs cycle detection on this graph using the Bellman-Ford algorithm. If a negative cycle is detected, the algorithm stops and returns the cycle.

The separation oracle only needs to determine:

* If there exists a negative cycle under , then
* the cut =
* If , then the cut = .
* Otherwise, is feasible, then
  + .
  + The cut =

### Example: Optimal matrix scalings under the min-max-ratio criterion

This example is taken from [[7](#ref-orlin1985computing)]. According to [[7](#ref-orlin1985computing)], optimal matrix scaling has several practical applications. One application is in linear programming, where groups of constraints and groups of variables might express the same physical commodity for which common measurement units are used. Another application is in telecommunication, where matrix scaling can be used to optimize the transmission of signals. Additionally, matrix scaling has been used in approximation theory to approximate functions of several variables by the sum of functions of fewer variables. Finally, matrix scaling has been used in Gaussian elimination, a widely used method for solving systems of linear equations, to improve the numerical stability of the algorithm.

Given a matrix . A *symmetric scaling* of is a matrix of the form where is a nonnegative diagonal matrix with the same dimension. According to the *min-max criterion*, the aim is to minimize the largest absolute value of ’s elements [[7](#ref-orlin1985computing), (Program 3)]:

The authors show that the problems of determining the best symmetric scalings under the min-max criterion can be converted into a single parameter network optimization problem, which can be solved efficiently using the parameteric network algorithms.

Another possible criterion is to minimize the ratio of largest absolute value of the element to the smallest. One motivation for using this criterion is that high ratios cause difficulties in performing the simplex method. With this *min-max-ratio* criterion, the symmetric scaling problem can be formulated as [[7](#ref-orlin1985computing), (Program 8)]:

Let denotes . By taking the logarithm of the variables, we can transform the above programming into a two-parameter network problem:

where . The authors of [[7](#ref-orlin1985computing)] claim they have devised an algorithm for solving multi-parameter problems. However, we did not uncover any further publications to support this claim. Notably, the cutting-plane method readily extends the single-parameter network algorithm to be multi-parameter.

In this application, is:

We can find fast algorithms for finding a negative cycle in [[8](#ref-dasdan1998faster),[9](#ref-dasdan2004experimental)]. More applications to clock skew scheduling can be found in [[10](#ref-zhou2015multi)].

## 2.3 Problems Involving Matrix Inequalities

Consider the following problem:

where is a matrix-valued function, denotes is positive semidefinite. Recall that a matrix is positive semidefinite if and only if for all . We can transform the problem into:

Consider is concave for all w.r.t. , then the above problem is a convex programming. Reduce to *semidefinite programming* if is linear w.r.t. , i.e., .

In convex optimization, a **Linear Matrix Inequality (LMI)** is an expression of the form:

where is a real vector, are symmetric matrices, and is a generalized inequality meaning is a positive semidefinite matrix¹.

This linear matrix inequality specifies a convex constraint on . There are efficient numerical methods to determine whether an LMI is feasible (e.g., whether there exists a vector such that ), or to solve a convex optimization problem with LMI constraints¹.

Many optimization problems in control theory, system identification, and signal processing can be formulated using LMIs¹. Also, LMIs find application in Polynomial Sum-Of-Squares¹.

### Cholesky decomposition Algorithm

The Cholesky factorization method is employed in linear algebra to decompose a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose¹. This factorization is beneficial for efficient numerical solutions, including Monte Carlo simulations¹.

The Cholesky decomposition of a Hermitian positive-definite matrix A is a unique decomposition where A = LL*, with L being a lower triangular matrix containing real and positive diagonal entries, and L* representing the conjugate transpose of L¹. Every real-valued symmetric positive-definite matrix and Hermitian positive-definite matrix have a Cholesky decomposition¹.

If A is a real matrix that is symmetric positive-definite, it can be decomposed as A = LLT. Here, L is a real lower triangular matrix that has positive diagonal entries¹.

The Cholesky and LDLT decompositions are matrix decomposition methods used in linear algebra for different purposes, with distinct properties¹².

The Cholesky decomposition involves decomposing a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose¹. It is typically faster and more numerically stable than the LDLT decomposition³. However, the input matrix must be positive-definite¹ for this to work.

The LDLT decomposition, a variant of the LU decomposition that applies to positive-definite symmetric matrices², is more versatile as it can be applied to a wider range of matrices and does not require them to be positive-definite¹. The LDLT decomposition factors a matrix into the product of a lower triangular matrix, a diagonal matrix, and the transpose of the lower triangular matrix². This decomposition is as fast as Cholesky decomposition but does not require any square roots, making it faster and more numerically stable³.

If is real, the following recursive relations apply for the entries of and :

Again, the pattern of access allows the entire computation to be performed in-place if desired.

The Cholesky or LDLT decomposition can be computed using either row-based or column-based methods.

* Column-Based: In this method, the computation proceeds by columns. The inner loops compute the current column using a matrix-vector product that accumulates the effects of previous columns.
* Row-Based: In this method, the computation proceeds by rows. The inner loops compute the current row by solving a triangular system involving previous rows.

Each outer loop index selection produces a distinct Cholesky algorithm, which is named after the portion of the matrix updated by the basic operation within the inner loops. Whether to use a row-based or column-based method depends on the specific problem requirements, as well as system properties such as memory layout and access patterns. With row-based decomposition and lazy evaluation technique, the cutting plane construction can be done with exactness in . This allows for effective oracle implementation. The following is the algorithm written in Python:

def factor(self, getA):  
 T = self.T  
 for i in range(self.n): # from 0 to n-1  
 for j in range(i+1): # from 0 to i  
 d = getA(i, j) - np.dot(T[:j, i], T[j, :j])  
 T[i, j] = d  
 if i != j:  
 T[j, i] = d / T[j, j]  
 if d <= 0.: # strictly positive  
 self.p = i  
 return  
 self.p = self.n

The Cholesky factorization provides a witness vector certifying that a matrix is not positive definite. If a matrix fails the Cholesky factorization, then it is not positive definite¹².

During the factorization process, compute the diagonal of the lower diagonal matrix by finding the square root of a value, denoted as x. If x<0, then this indicates that the matrix is not positive definite¹. This failure serves as evidence for a non-positive definite matrix.

If the Cholesky factorization fails due to a negative diagonal element, it indicates that the leading principal submatrix up to that point is not positive definite. The vector that confirms this is one of the standard basis vectors¹². This basis vector comprises a 1 in the position that corresponds to the failed diagonal element and zeros elsewhere. When you pre-multiply it by the original matrix and post-multiply it by its transpose, it will yield a negative value, thus serving as a witnessing vector¹².

The following is the algorithm written in Python:

def witness(self):  
 p = self.p  
 n = p + 1  
 v = np.zeros(n)  
 v[p] = 1  
 for i in range(p, 0, -1): # backward substitution  
 v[i-1] = -np.dot(self.T[i-1, i:n], v[i:n])  
 return v, -self.T[p, p]

The oracle only needs to:

* Perform a *row-based* Cholesky decomposition such that .
* Let denotes a submatrix .
* If Cholesky decomposition fails at row ,
  + there exists a vector , such that
    - , and
    - .
  + The cut =

### Example: Matrix Norm Minimization

Let . Problem can be reformulated as

A binary search on can be used for this problem.

### Example: Estimation of Correlation Function

## 2.4 Random Field [[11](#ref-Schabenberger05)]

*Random field*, also known as *stochastic process*, can be regarded as an indexed family of random variables denoted as {}, where is a subset of -dimensional Euclidean space . To specify a stochastic process, the joint probability distribution function of any finite subset must be given in a consistent way, which is called *distribution* of the process. For ease of analysis, a random field is often assumed to be with *Gaussian* distribution and is called Gaussian random field.

A random field has several key properties useful in practical problems. The field is *stationary* under translations, or *homogeneous*, if the distribution is unchanged when the point set is translated. The field is *isotropic* if the distribution is invariant under any rotation of the whole points in the parameter space. We study the homogeneous isotropic field in this paper.

The *covariance* and *correlation* of a stochastic process are defined by:

and

respectively for all , where denotes the expectation of . Thus a process is homogeneous if and depend only on the separation vector . Furthermore, it is isotropic if and depend upon only through its length , i.e.,

If we denote , the variance of , as , then the relationship between covariance and correlation is .

When the two components are considered, the measurement data can still be regarded as a Gaussian random field, but the correlation function will have a discontinuity at the origin. We call this phenomenon “nugget effect” [[12](#ref-Diggle07)].

Let , where ’s are the unknown coefficients to be fitted ’s are a family of basis functions. The covariance matrix can be recast as:

where .

# 3 Ellipsoid Method Revisited

Some History of the Ellipsoid Method [[1](#ref-BGT81)]. Introduced by Shor and Yudin and Nemirovskii in 1976. It 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.

## 3.1 Basic Ellipsoid Method

An ellipsoid is specified as a set

where is the center of the ellipsoid and is a positive definite matrix.

Updating the ellipsoid (deep-cut)

Calculation of minimum volume ellipsoid covering:

Let , . We can make the following observations:

1. If (shallow cut), then no smaller ellipsoid can be found.
2. If , then intersection is empty.
3. Otherwise,

* where

Even better, split into two variables . Let , , .

Reduce multiplications per iteration. Note that:

* The determinant of decreases monotonically.
* The range of is

## 3.2 Central Cut

A Special case of when . Deserve a separate implement because it is much simpler. Let , ,

## 3.3 Parallel Cuts

Oracle returns a pair of cuts instead of just one. The pair of cuts is given by and such that:

for all .

Only linear inequality constraint can produce such parallel cut:

Usually, provide faster convergence.

![Parallel cuts](data:application/pdf;base64,)

Parallel cuts

Updating the ellipsoid.

Let , .

* If , intersection is empty.
* If , no smaller ellipsoid can be found.
* If , it reduces to deep-cut with .

Otherwise,

where

### Example: FIR filter design

A typical structure of digital Finite Impulse Response (FIR) filter is shown in Fig. 1, where the coefficients must be determined to meet given specifications. Usually, they can be manually designed using windowing or frequency-sampling techniques [[13](#ref-oppenheim1989discrete)].

However, the experience and knowledge of designers are highly demanded in this kind of design methods. Moreover, there is no guarantee about the design’s quality. Therefore, the optimization-based techniques (e.g. [[14](#ref-wu1999fir)], more reference) have attracted tons of research effort. In this kind of method, facilitated with growing computing resources and efficient optimization algorithms, the solution space can be effectively explored.

![A typical structure of an FIR filter [15].](data:application/pdf;base64,)

A typical structure of an FIR filter [[15](#ref-mitra2006digital)].

In optimization algorithms, what is particularly interesting is the convex optimization. If a problem is in a convex form, it can be efficiently and optimally solved. Convex optimization techniques are also implementable in designing FIR filters, including the Parks-McClellan algorithm [[16](#ref-park1972chebyshev)], METEOR [[17](#ref-steiglitz1992meteor)], and peak-constrained least-squares (PCLS) [[18](#ref-selesnick1996constrained),[19](#ref-adams1998peak)]. In the mentioned articles, with the help of exchange algorithms (e.g. Remez exchange algorithm), certain FIR filter design problems can be formed as linear or quadratic programs. They are two simple forms of convex optimization problems, which can be optimally solved with existing algorithms, such as the interior-point method [[20](#ref-boyd2009convex)]. Tempted by the optimality, more efforts were devoted to forming the problem convex. Particularly, in [[14](#ref-wu1999fir)], via spectral decomposition [[21](#ref-goodman1997spectral)], the problem of designing an FIR filter with magnitude constraints on frequency-domain is formulated as a convex optimization problem. More examples are provided in [[22](#ref-davidson2010enriching)].

Its time response is

where is the filter coefficients. Its frequency response is

where , is the order of the filter. The design of a filter with magnitude constraints is often formulated as a constraint optimization problem as the form

where is the vector of design variables, represents the characteristics of the desirable filter and is the performance metric to be optimized. For example, the magnitude constraints on frequency domain are expressed as

where and are the lower and upper (nonnegative) bounds at frequency respectively. Note that is periodic and . Therefore, we can only consider the magnitude constraint on [[14](#ref-wu1999fir)].

Generally, the problem might be difficult to solve, since we can only obtain the global optimal solution with resource-consuming methods, such as branch-and-bound [[22](#ref-davidson2010enriching)]. However, the situation is totally different if the problem is convex, where and are convex functions. In such a case, the problem can be optimally solved with many efficient algorithms.

Attracted by the benefits, the authors of [[14](#ref-wu1999fir)] transformed (?), originally non-convex, into a convex form via spectral decomposition:

where and are the autocorrelation coefficients. Especially, can be determined by , with the following equation vice versa [[14](#ref-wu1999fir)]:

where for or .

![Result](data:application/pdf;base64,)

Result

### Example: Maximum Likelihood estimation

Consider

Note that the first term is concave, the second term is convex. However, if there are enough samples such that is a positive definite matrix, then the function is convex within . Therefore, the following problem is convex:

## 3.4 Discrete Optimization

Many engineering problems can be formulated through convex/geometric programming, such as digital circuit sizing. However, in ASIC design, there is frequently a limited number of cell types to select from in the cell library. This means that some design variables are discrete. We can map the design variables to integers to represent the discrete version as Mixed-Integer Convex programming (MICP).

What are the issues with existing methods? They are primarily based on relaxation. The more relaxed solution is used as the lower bound, and then the branch-and-bound method is applied to find the discrete optimal solution. It should be noted that the branch-and-bound method does not utilize the convexity of the issue. What if only constraints regarding discrete data could be evaluated?

Typically, a more relaxed optimal solution (convex) is obtained beforehand. After that, the optimized discrete solution is obtained using an exhaustive neighborhood search. However, tight constraints can cause a significant difference between the discrete and relaxed continuous optimal solutions. Enumerating the discrete domains can be challenging.

Consider:

where and are “convex”. Note that some design variables are discrete. The oracle looks for a nearby discrete solution of with the cutting-plane:

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

### Example: Multiplierless FIR Filter Design

However, there are still many filter design problems that are non-convex, such as multiplierless FIR filter design problems. Note that in Fig. 1, each coefficient associated with a multiplier unit makes the filter power-hungry, especially in *application specific integrated circuits* (ASIC). Fortunately, if each coefficient is quantized and represented as a sum of Singed Power-of-Two (SPT), a multiplierless filter can be implemented. Such coefficients can be uniquely represented by a Canonical Signed-Digit (CSD) code with a minimum number of non-zero digits[[23](#ref-george1960csd)]. In this case, it confines the multiplication to addition and shift operations. The coefficient 0.40625 = 13/32 can be written as . Thus, the multiplier can be replaced with three shifters and two adders at a much lower cost. However, the coefficient quantization constraint is non-convex, making the convex optimization algorithm not directly applicable. A similar case is the consideration of the finite word-length effect [[24](#ref-lim1982finite)].

Attracted by the benefits of this “multiplier-free” approach, many efforts have been devoted to its design techniques. For its general problems, integer programming (e.g. [[24](#ref-lim1982finite)–[27](#ref-lim1999signed)]) can be implemented to achieve the optimal solution. However, it requires excessive computational resources. Other heuristic techniques, such as genetic algorithm [[28](#ref-xu1995design)] and dynamic-programming-like method [[29](#ref-chen1999trellis)], also have inefficiency. If the quantization constraint is the only non-convex constraint in the design problem, a lower bound can be efficiently obtained by solving the relaxed problem [[22](#ref-davidson2010enriching)]. Then to make the solution feasible, it can be rounded to the nearest CSD code or used as a starting point of a local search algorithm to obtain a better solution [[30](#ref-kodek1981comparison)]. However, neither method guarantees the feasibility of the final solution. Besides, the local search problem remains non-convex. Therefore, the adopted algorithm may also be inefficient, such as branch-and-bound in [[30](#ref-kodek1981comparison)].

![Result](data:application/pdf;base64,)

Result

# 4 Concluding Remarks

Should be known to students. The ellipsoid method is not a competitor but a companion of interior-point methods.

TBD.

# References

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

[2] H. Aliabadi, M. Salahi, Robust geometric programming approach to profit maximization with interval uncertainty, Computer Science Journal of Moldova. 21 (2013) 86–96.

[3] X. Liu, W.-S. Luk, Y. Song, P. Tang, X. Zeng, Robust analog circuit sizing using ellipsoid method and affine arithmetic, in: Proceedings of the 2007 Asia and South Pacific Design Automation Conference, IEEE Computer Society, 2007: pp. 203–208.

[4] B.V. Cherkassky, A.V. Goldberg, Negative-cycle detection algorithms, Mathematical Programming. 85 (1999) 277–311.

[5] R.E. Tarjan, Shortest paths, AT&T Bell Laboratories, 1981.

[6] A. Dasdan, Experimental analysis of the fastest optimum cycle ratio and mean algorithms, ACM Transactions on Design Automation of Electronic Systems. 9 (2004) 385–418.

[7] J.B. Orlin, U.G. Rothblum, Computing optimal scalings by parametric network algorithms, Mathematical Programming. 32 (1985) 1–10.

[8] A. Dasdan, R.K. Gupta, Faster maximum and minimum mean cycle algorithms for system-performance analysis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 17 (1998) 889–899.

[9] A. Dasdan, Experimental analysis of the fastest optimum cycle ratio and mean algorithms, ACM Transactions on Design Automation of Electronic Systems (TODAES). 9 (2004) 385–418.

[10] X. Zhou, W.-S. Luk, H. Zhou, F. Yang, C. Yan, X. Zeng, Multi-parameter clock skew scheduling, Integration, the VLSI Journal. 48 (2015) 129–137.

[11] O. Schabenberger, C.A. Gotway, Statistical Methods for Spatial Data Analysis, Chapman & Hall/CRC, Florida, 2005.

[12] P.J. Diggle, P.J.R. Jr., Model-based Geostatistics, Springer, New York, 2007.

[13] A.V. Oppenheim, R.W. Schafer, J.R. Buck, others, Discrete-time signal processing, Prentice hall Englewood Cliffs, NJ, 1989.

[14] S.-P. Wu, S. Boyd, L. Vandenberghe, FIR filter design via spectral factorization and convex optimization, in: Applied and Computational Control, Signals, and Circuits, Springer, 1999: pp. 215–245.

[15] S.K. Mitra, Y. Kuo, Digital signal processing: A computer-based approach, McGraw-Hill New York, 2006.

[16] T.W. Parks, J.H. McClellan, Chebyshev approximation for nonrecursive digital filters with linear phase, Circuit Theory, IEEE Transactions On. CT-19 (1972) 189–194.

[17] K. Steiglitz, T.W. Parks, J.F. Kaiser, METEOR: A constraint-based FIR filter design program, Signal Processing, IEEE Transactions On. 40 (1992) 1901–1909.

[18] I.W. Selesnick, M. Lang, C.S. Burrus, Constrained least square design for FIR filters without specified transition bands, Signal Processing, IEEE Transactions On. 44 (1996) 1879–1892.

[19] J.W. Adams, J.L. Sullivan, Peak-constrained least-squares optimization, Signal Processing, IEEE Transactions On. 46 (1998) 306–321.

[20] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge university press, 2009.

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

[22] T.N. Davidson, Enriching the art of FIR filter design via convex optimization, Signal Processing Magazine, IEEE. 27 (2010) 89–101.

[23] G.W. Reitwiesner, Binary Arithmetic, Advances in Computers. 1 (1960) 231–308.

[24] Y.C. Lim, S. Parker, A. Constantinides, Finite word length FIR filter design using integer programming over a discrete coefficient space, Acoustics, Speech and Signal Processing, IEEE Transactions On. 30 (1982) 661–664.

[25] D.M. Kodek, Design of optimal finite wordlength FIR digital filters using integer programming techniques, Acoustics, Speech and Signal Processing, IEEE Transactions On. 28 (1980) 304–308.

[26] Y. Lim, S. Parker, FIR filter design over a discrete powers-of-two coefficient space, Acoustics, Speech and Signal Processing, IEEE Transactions On. 31 (1983) 583–591.

[27] Y.C. Lim, R. Yang, D. Li, J. Song, Signed power-of-two term allocation scheme for the design of digital filters, Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions On. 46 (1999) 577–584.

[28] D.J. Xu, M.L. Daley, Design of optimal digital filter using a parallel genetic algorithm, Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions On. 42 (1995) 673–675.

[29] C.-L. Chen, A.N. Willson Jr, A trellis search algorithm for the design of FIR filters with signed-powers-of-two coefficients, Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions On. 46 (1999) 29–39.

[30] D. Kodek, K. Steiglitz, Comparison of optimal and local search methods for designing finite wordlength FIR digital filters, Circuits and Systems, IEEE Transactions On. 28 (1981) 28–32.