Enforcing Kubelka-Munk Constraints for Opaque Paints

Paul Centore

Abstract

The Kubelka-Munk model relates the colours of paint mixtures to the absorption and scattering coefficients (K and S) of the constituent paints, and to their concentrations (C) in the mixtures. All K's and S's are non-negative, and C's are physically constrained to be between 0 and 1. Standard estimation procedures cast the Kubelka-Munk relationships as an overdetermined linear system, and apply ordinary least squares (OLS). OLS, however, sometimes produces coefficients or concentrations that are less than 0 or greater than 1. These physically impossible solutions occur because OLS projects a target vector (such as a desired reflectance spectrum) onto a vector subspace, while in fact the set of physically realizable paint combinations is a convex polytope, which is a subset of that subspace. This paper reformulates Kubelka-Munk estimation problems geometrically, as the problem of finding the point on that polytope that is closest to a target vector. The solutions to the reformulated problem are always physically realizable. If feasible, a worker could solve the reformulated problem with a ready-made commercial solver. Otherwise, the Gilbert-Johnson-Keerthi (GJK) algorithm is recommended as especially suitable for Kubelka-Munk estimation; this algorithm has been tested on some simple cases and released as open-source code.

1 Introduction

The Kubelka-Munk model[1] predicts the reflectance spectrum of a mixture of opaque paints from the absorption coefficients (K), scattering coefficients (S), and concentration (C), for each constituent paint. Physical limits constrain K, S, and C. A paint cannot absorb or scatter a negative fraction of incoming light, so K and S must be non-negative at each wavelength. Similarly, a paint's concentration in a mixture is between 0 and 100 percent, so each C is also between 0 and 1. Furthermore, all the C's must sum to 1.

Two typical Kubelka-Munk problems are to estimate the K's and S's from a measured set of mixtures of known concentrations, and to estimate the concentrations of constituent paints, whose K's and S's are known, in a target paint. Standard approaches to such problems, such as the algorithms of Walowit, McCarthy, and Berns,[2,3] recast the Kubelka-Munk relationships into an overdetermined linear system of the form

$$Mx = b, \tag{1}$$

where M is a matrix with more rows than columns, x is a column vector to be solved for, and b is a target column vector. Depending on the situation, x might be a vector of K's and

S's, or a vector of concentrations. Equation (1) likely cannot be solved exactly, so instead ordinary least squares (OLS) is used to find an x that minimizes the Euclidean distance between Mx and b.

The OLS solution, however, can have difficulties: some entries in x can be negative or exceed 1, violating the physical constraints on K, S, and C. The current paper suggests a reformulation of Equation (1), that insures that a solution satisfies physical constraints.

The left side of Equation (1) can be interpreted as a subset of the vector space $\mathbb{R}^r$, where r is the number of rows in M. Mx is seen as a set of linear combinations of the columns of M, with the restriction that any combination's coefficients appear in the vector x. For ordinary least squares, which places no restrictions on x, Mx is the column space of M, denoted $\text{col}(M)$, and is a vector subspace of $\mathbb{R}^r$. Assuming a Euclidean metric, the target vector b can be projected onto the point $b_{\text{col}(M)}$ in $\text{col}(M)$ that is nearest to b. Since $b_{\text{col}(M)}$ is in $\text{col}(M)$, an exact (though possibly not unique) solution to $Mx = b_{\text{col}(M)}$ can be found; in fact, the OLS method finds such a solution, and also minimizes $Mx - b$.

The main geometric idea in this paper is to incorporate the constraints on x into the set Mx. The target vector b will then be projected to the closest point b_P on the constrained Mx. As a result, there will exist a solution which not only satisfies $Mx = b_P$ exactly, and $Mx = b$ as closely as possible, but which also satisfies all the constraints. The constrained Mx is a subset of $\text{col}(M)$, and the subset relationship explains why OLS sometimes gives unrealistic results. While $b_{\text{col}(M)}$ is in $\text{col}(M)$ by design, it might not belong to the constrained Mx. If it does not, then the OLS solution will necessarily violate some physical constraint. If b_P *is* in the constrained Mx, on the other hand, then a physically valid solution will exist.

The approach presented in this paper, and its two Kubelka-Munk applications, depend on the fact that the constrained Mx is a convex polytope P. While constraints on x will not always make Mx a convex polytope, many sets of constraints will, including those encountered in Kubelka-Munk problems. A polytope is the convex hull of a finite set of generating points in $\mathbb{R}^r$. As such, it is bounded, so each entry of x must have a lower and upper bound. The entries for Kubelka-Munk cases will be K's, S's, and C's, all of which can be chosen to be between 0 and 1 for opaque paints. Convexity insures that the projection point b_P is unique.[4] In this context, projection does not mean orthogonal projection; rather b_P is the point on P whose distance to b is a minimum.

Various algorithms can be used to calculate b_P. Many workers would be satisfied with a commercial solver and never inquire about the algorithm; other workers, however, would prefer a more hands-on approach. For the latter group, the Gilbert-Johnson-Keerthi (GJK) algorithm[5,6] is recommended as particularly suitable for calculating b_P for Kubelka-Munk estimation problems. The GJK algorithm is a fast, iterative method that finds the minimum distance between two convex polytopes. In this paper, the first polytope is the set Mx, and the second polytope is the point b. The GJK algorithm expresses b_P as a linear combination of generators, which, in the Kubelka-Munk cases to be considered, are columns of M. The coefficients in that expression can be taken as entries in x : they not only solve $Mx = b_P$, but also satisfy the constraints. Since b_P minimizes the distance from Mx to b, this solution is in fact a least squares solution to Equation (1). The GJK algorithm has been tested successfully on a few Kubelka-Munk problems that arose in practice, and an open-source implementation is available on the author's website.

This paper applies the geometric reformulation to two Kubelka-Munk problems involving opaque paints. The first problem estimates the concentration C_i of the i^{th} base paint in a mixture, when all K_i's and S_i's are known. Concentrations are between 0 and 1, and sum to 1:

$$0 \leq C_i \leq 1 \text{ for all } i, \tag{2}$$

$$\sum_{i=1}^{n} C_i = 1. \tag{3}$$

The second problem estimates the K's and S's (at one wavelength) for a set of n base paints from spectral measurements of mixtures of those base paints. All paint films are assumed to be masstones, that is, they are applied thickly enough to be opaque for practical purposes. K is the instantaneous rate at which incoming light is absorbed by the paint film, and S is the rate for scattering. Both K and S must be non-negative, so the following constraints hold for every i from 1 to n:

$$0 \leq K_i, \tag{4}$$

$$0 \leq S_i. \tag{5}$$

We will see that the absolute magnitudes of the K's and S's are not important for opaque paints, but that the ratios of any K or S to any other K or S are. Without loss of generality, then, we could multiply each K and S by a positive constant, so that they satisfy the constraint

$$\sum_{i=1}^{n}(K_i + S_i) = 1. \tag{6}$$

This paper is organized as follows. First, the geometric reformulation will be presented, along with a brief discussion of implementation. Then, the Kubelka-Munk model will be used to derive expressions of the form $Mx = b$, where $x = [C_1, C_2, \ldots, C_n]$ in the first case, and $x = [K_1, K_2, \ldots, K_n, S_1, S_2, \ldots S_n]$ in the second case. In both cases, there will be constraints on x, and the set Mx will be shown to be a convex polytope, whose generators are just the columns of M. This information is sufficient for calculating b_P, and the expression for b_P as a convex combination of the generators will lead to solutions for x. Next, this approach will be compared to other Kubelka-Munk approaches. Finally, an appendix will describe the GJK algorithm, and its suitability in the Kubelka-Munk context.

2 Reformulating Constrained Least Squares Problems

While often treated as ordinary least squares (OLS) problems, many Kubelka-Munk calculations in fact lead to constrained least squares (CLS) problems. This section outlines the OLS and CLS problems, and shows geometrically why the methods used for OLS problems sometimes do not satisfy the desired constraints. A geometric interpretation of the CLS problem, as the projection of a target vector onto a convex polytope, motivates non-OLS algorithms whose results do satisfy the constraints. Practical implementations of such algorithms are briefly discussed.

2.1 OLS and CLS Problems

A least squares problem arises in the context of a linear system

$$Mx = b, \tag{7}$$

where M is a matrix, and x and b are column vectors. M and b are assumed to be known, and x is to be solved for. In many practical cases, because of measurement or model error, no x satisfies $Mx = b$ exactly. Instead, one finds an $\hat{x}$, called a least squares (LS) solution, that satisfies $Mx = b$ as closely as possible. Closeness is measured by the coordinate-wise sum of squared residuals (SSR) between $M\hat{x}$ and b:

$$\mathrm{S}SR = \sum_i \left((M\hat{x})_i - b_i\right)^2, \tag{8}$$

where the subscript i denotes the i^{th} coordinate. Minimizing Equation (8), when there are no restrictions on $\hat{x}$, is the ordinary (or unconstrained) least squares (OLS) problem.

The constrained least squares (CLS) problem is to minimize (8) when there are restrictions, or constraints, on the coordinates of x. For example, the i^{th} coordinate x_i might have to satisfy $a_i \leq x_i \leq b_i$. In general, many constraints are possible. We will see a Kubelka-Munk example in which not only must each coordinate be between 0 and 1, but also the sum of the coordinates must be 1. While the OLS problem has a simple closed-form solution,[7] CLS solutions usually do not, and tend to be much more demanding computationally. A main result of this paper is a geometric formulation for some CLS problems that arise when using the Kubelka-Munk model. The geometric formulation, in which Mx is viewed as a convex polytope, can use non-OLS algorithms.

2.1.1 Constructing a Convex Polytope

While the system $Mx = b$ is written algebraically, it can also be interpreted geometrically. The geometric interpretation will transform the system into a point and a convex polytope, between which the shortest distance is sought. The polytope will be naturally expressed as the convex hull of a set of vectors, called generators. This section describes the convex polytope, and shows how the generator form arises naturally.

Geometrically, the matrix M, which has r rows and c columns, can be viewed as a linear transformation from the vector space $\mathbb{R}^c$ to the vector space $\mathbb{R}^r$. The vector b on the right side of Equation (7) is a target vector in $\mathbb{R}^r$. The left side, Mx, can be seen as a set of linear combinations of the columns of M. In fact, each column of M can be seen as a vector in $\mathbb{R}^r$. The i^{th} column, M_i, is the image under M of the vector x which is zero in all coordinates except the i^{th}, where it takes on the value 1. If the variables in x are unconstrained, then Mx is just the column space (or equivalently the range) of M, which is the subspace of $\mathbb{R}^r$ generated by the column vectors. Denote this subspace by $\mathrm{col}(M)$. While there is considerable structure in a subspace, for now it is best just to view it as a subset S of $\mathbb{R}^r$. The form of this subset S will change if various constraints are placed on x.

If Mx is seen as a set S, then Equation (7) is a geometric problem involving distances: what point b_S on S is closest to b? Once we have found such a b_S, the equation $Mx = b_S$

will have an exact (though possibly not unique) solution, because b_S is in the range of M. Furthermore, that solution can be expressed so as to satisfy constraints on x, because those constraints were incorporated into Mx. In fact, this method is used implicitly in OLS. The set Mx is a subspace, and b is projected orthogonally onto that subspace. Closest points are simple to calculate when S is a subspace, but can also be found whenever S is a convex polytope.

A simple Kubelka-Munk case that produces a convex polytope occurs when x gives the concentrations of a set of n base paints in a mixture. Concentrations are always between 0 and 1, and always sum to 1, so we have the constraints

$$0 \leq x_i \leq 1, \tag{9}$$

$$\sum_{i=1}^{n} x_i = 1. \tag{10}$$

In Mx, the components x_i are viewed as coefficients of linear combinations of the columns of M. Placing the constraints (9) and (10) on the coefficients, however, causes Mx to be the convex hull of the columns of M. This convex hull exists in $\mathbb{R}^r$, and is generated by the columns of M, where each column of M is considered as a vector in $\mathbb{R}^r$. The convex hull of a finite set of points is a convex polytope, whose i^{th} generator is the i^{th} column of M.

To sum up, the general approach to constrained least squares problems is to express the set Mx as a convex polytope, and then to find the point b_S in Mx that is closest to b. At least one solution to $Mx = b_S$ is then guaranteed to exist and to satisfy the constraints. There are still issues of solution uniqueness (and conceivably some solutions to $Mx = b_S$ satisfy the constraints while other solutions do not), but in the problems investigated, algorithms exist that quickly produce a valid solution.

2.1.2 Practical Implementations

The previous section reformulated CLS problems geometrically, and later sections will show that this new formulation is suitable for some Kubelka-Munk problems. This reformulation has been largely theoretical, but workers in the paint industry need practical implementations. Broadly speaking, a worker can either use a commercial solver, or implement an algorithm himself. This section discusses the advantages and disadvantages of each approach. For workers who desire their own implementation, the GJK algorithm is recommended as an intuitive, easily-implemented algorithm that is tailored to Kubelka-Munk problems.

Commercial solvers have the obvious advantage of not requiring any programming effort, but solvers can be expensive, and not all of them can handle constrained optimization problems. One solver that seems to is Matlab's Optimization Toolbox, whose routine fmincon.m minimizes a function f in the presence of constraints. Since the author has not actually tested this routine, he cannot unqualifiedly recommend the Toolbox, but it appears to be an option worth investigating; likely there are many other options, too.

Since the squared distance is a quadratic function, a quadratic programming (QP) solver seems like a natural tool for finding the nearest point on a convex polytope. In fact, however, QP likely would not work well in the Kubelka-Munk context. The reason is that the Kubelka-Munk polytope will be expressed as the convex hull of a finite set of generators, while QP

problems express a polytope as the intersection of a set of half-spaces. Converting between the two kinds of expression can be computationally demanding, and would require writing or purchasing additional code. Similarly, once the nearest point is found, it must be expressed as a convex combination of the polytope's vertices, since the coefficients of that combination are the desired K's, S's, or C's. Again, more code is required, thus undercutting the value of a ready-made QP solver.

Another resource for CLS algorithms is Chapters 20 to 23 of Lawson and Hanson's standard text, *Solving Least Squares Problems*.[8] Those chapters suggest multiple approaches, including a heavily weighted constraint row, and interior methods that iterate to a point that satisfies the Kuhn-Tucker conditions. While the algorithms presented draw on much rigorous theory and practical experience, programming them is a difficult task that is left to the reader. Possibly, however, implementations have now been included in some commercially available solvers.

In case a satisfactory solver cannot be found, the appendix describes the GJK algorithm, which is especially suited for Kubelka-Munk problems. Its only inputs are a target vector and a set of generating vectors for the convex polytope; later sections will show how these are easily calculated for the Kubelka-Munk problems of interest. The quantity to be minimized has the special form of the distance (with regard to appropriate coordinate weightings) from the target to the polytope. The author has tested the GJK algorithm for some simple Kubelka-Munk cases, and released an open-source implementation on his website.

3 Kubelka-Munk Derivations

3.1 Linear Kubelka-Munk Relationships

The Kubelka-Munk model[1] characterizes a paint by using an absorption coefficient, K, and a scattering coefficient, S. Both coefficients are functions of wavelength, and both are non-negative. If the paint is applied as a masstone, the bulk reflectance R can be related to the ratio of K and S:

$$\frac{K}{S} = \frac{(1-R)^2}{2R}. \tag{11}$$

For convenience, the right-hand side will be denoted as a function, $f(R)$. A set of n paints can be mixed together, in concentrations $C_1, C_2, ...C_n$, to produce a new paint, of a different colour. If the i^{th} paint has Kubelka-Munk coefficients K_i and S_i, then a premise of the Kubelka-Munk model is that the coefficients for the mixture are given by

$$K_{\text{mix}} = C_1K_1 + C_2K_2 + ... + C_nK_n, \tag{12}$$
$$S_{\text{mix}} = C_1S_1 + C_2S_2 + ... + C_nS_n. \tag{13}$$

Equations (12) and (13) can be substituted into Equation (11) to give relationships for the paint mixture as a whole:

$$\left(\frac{K}{S}\right)_{\text{mix}} = \frac{C_1K_1 + C_2K_2 + ... + C_nK_n}{C_1S_1 + C_2S_2 + ... + C_nS_n} = \frac{(1-R_{\text{mix}})^2}{2R_{\text{mix}}}. \tag{14}$$

For each mixture, rearranging Equation (14) gives a linear relationship:

$$-C_1K_1 - C_2K_2 - ... - C_nK_n + \left(\frac{K}{S}\right)_{\text{mix}} (C_1S_1 + C_2S_2 + ... + C_nS_n) = 0. \tag{15}$$

Substituting $f(R)$ for $(K/S)_{\text{mix}}$ gives

$$-C_1K_1 - C_2K_2 - ... - C_nK_n + f(R)\,(C_1S_1 + C_2S_2 + ... + C_nS_n) = 0. \tag{16}$$

Indicate the dependencies explicitly, using λ to indicate a wavelength, and μ as an index to the set of mixtures. Then Equation (16) becomes

$$\begin{aligned} &-C_1(\mu)K_1(\lambda) - C_2(\mu)K_2(\lambda) - ... - C_n(\mu)K_n(\lambda) + \dots \\ &\cdots + f(R(\lambda,\mu))\,(C_1(\mu)S_1(\lambda) + C_2(\mu)S_2(\lambda) + ... + C_n(\mu)S_n(\lambda)) = 0. \end{aligned} \tag{17}$$

Once dependencies are included, it is seen that Equation (16) is actually a set of equations, one for each mixture-wavelength pair. The variables C depend only on the mixture, while K and S depend only on the wavelength. The term $f(R)$ connects the other terms, because it depends on both the mixture and the wavelength. Typically, $f(R)$ is calculated, via Equation (11), from reflectance measurements made with a spectrophotometer.

The set of equations represented by (16) is basic to many Kubelka-Munk applications. It can be partitioned into subsets by wavelength or mixture, depending on the problem of interest. This paper will be concerned with two problems:

1. **Estimating concentrations:** If the K and S coefficients are known for each base paint at each wavelength, and the reflectance spectrum of a target mixture has been measured, then estimate the concentrations of the base paints in the target mixture.
2. **Estimating K's and S's:** If the concentrations of a set of mixtures are known, and the mixtures' reflectance spectra have been measured, then estimate K and S for each base paint, at each wavelength.

The following subsections will treat these problems in greater detail.

3.2 Estimating Concentrations

When estimating concentrations, all the equations corresponding to one mixture are grouped into one subset; this subset contains information from all wavelengths. Each subset gives a linear system to be solved. If we let m denote the total number of mixtures, then there will be m systems. For convenience, we will assume that reflectance measurements have been made at 10 nm increments from 400 nm to 700 nm, so each system will consist of 31 equations.

The equation for a fixed wavelength λ and mixture μ can be written in vector form:

$$[f(R(\lambda,\mu))S_1(\lambda) - K_1(\lambda), f(R(\lambda,\mu))S_2(\lambda) - K_2(\lambda), ..., f(R(\lambda,\mu))S_n(\lambda) - K_n(\lambda)] \begin{bmatrix} C_1(\mu) \\ C_2(\mu) \\ ... \\ C_n(\mu) \end{bmatrix} = 0. \tag{18}$$

To solve for the concentrations C_i in mixture μ, use equations from all the wavelengths for mixture μ. When considering all wavelengths simultaneously, one should give greater weight to wavelengths which humans are more sensitive to, or which affect perceived colour differences more. Assign such a weight $w(\lambda)$ to a relationship in the system by multiplying both sides of that relationship by $w(\lambda)$. Then Equation (18) becomes

$$\left[w(\lambda)\left(f(R(\lambda,\mu))S_1(\lambda)-K_1(\lambda)\right),...,w(\lambda)\left(f(R(\lambda,\mu))S_n(\lambda)-K_n(\lambda)\right)\right]\begin{bmatrix} C_1(\mu) \\ ... \\ C_n(\mu)\end{bmatrix}=0. \quad (19)$$

It is not quite clear how $w(\lambda)$ should be chosen, and good cases can be made for different choices; ideally $w(\lambda)$ should correspond to perceived colour differences, which depend not only on a reflectance spectrum, but also on an illuminant. This question will be left aside for now, and the development will use the photopic luminous efficiency function[9] to provide weights. This function, denoted $\bar{y}(\lambda)$, is a colour-matching function in standard CIE colorimetry,[10] and quantifies human visual response to stimuli at different wavelengths.

Combine the weighted equations into a single matrix expression:

$$\begin{bmatrix} \bar{y}(400)\left(f(R(400,\mu))S_1(400)-K_1(400)\right) & ... & \bar{y}(400)\left(f(R(400,\mu))S_n(400)-K_n(400)\right) \\ \bar{y}(410)\left(f(R(410,\mu))S_1(410)-K_1(410)\right) & ... & \bar{y}(410)\left(f(R(410,\mu))S_n(410)-K_n(410)\right) \\ ... & ... & ... \\ \bar{y}(700)\left(f(R(700,\mu))S_1(700)-K_1(700)\right) & ... & \bar{y}(700)\left(f(R(700,\mu))S_n(700)-K_n(700)\right)\end{bmatrix}\begin{bmatrix} C_1(\mu) \\ ... \\ C_n(\mu)\end{bmatrix}=\begin{bmatrix} 0 \\ ... \\ 0\end{bmatrix}. \quad (20)$$

The matrix has 31 rows and n columns. The equation as a whole has the form $Mx = b$.

An obvious mathematical solution to Equation (20) would be $C_i = 0$ for every i. This "solution" is rejected on physical grounds, because the concentrations must sum to 1. In addition, each concentration must be between 0 and 1. These two constraints can be expressed formally:

$$0 \leq C_i \leq 1, \quad (21)$$

$$\sum_{i=1}^{n} C_i = 1. \quad (22)$$

Any solution set of C_i's must satisfy these constraints.

Section 2 has already shown how Equation (20) can be seen as a CLS problem. As a set, the left side of Equation (20) is the convex polytope that is generated by the columns of M. The target point is the origin, on the right side of Equation (20). The target point and generators are inputs into an algorithm which produces the point b_P on the polytope that is nearest the origin. This "point" is actually a vector in $\mathbb{R}^{31}$, and defines a reflectance spectrum which differs from the target reflectance spectrum by as little as possible. Since b_P is unique, the reflectance spectrum is unique.

Note that even though the reflectance spectrum is unique, the concentrations that produce it might not be: possibly, two disjoint subsets of base colorants could each combine to produce the same reflectance spectrum. Most algorithms find only one set of concentrations that produce the optimal reflectance spectrum. To find any others, substitute the optimal reflectance spectrum for the zero vector on the right side of Equation (20). The result will be a linear system, $Mx = b_P$, with one known solution. Other solutions can be produced by adding vectors (if any exist) in the kernel of M to that solution, being careful, of course, that the sums do not violate the constraints.

3.3 Estimating K's and S's

3.3.1 Least Squares Formulation

When estimating K's and S's, all instances of Equation (16) that correspond to one wavelength are grouped into one subset, which contains information from all the mixtures. Each subset gives a linear system to be solved. Since we are assuming 31 wavelengths, there will be 31 systems in total. If there are m mixtures in total, then each of the 31 systems will contain m equations.

The equation for a fixed wavelength λ and mixture μ can be written in vector form:

$$[-C_1(\mu), -C_2(\mu), ..., -C_n(\mu), f(R(\lambda,\mu))C_1(\mu), f(R(\lambda,\mu))C_2(\mu), ..., f(R(\lambda,\mu))C_n(\mu)] \begin{bmatrix} K_1(\lambda) \\ K_2(\lambda) \\ ... \\ K_n(\lambda) \\ S_1(\lambda) \\ S_2(\lambda) \\ ... \\ S_n(\lambda) \end{bmatrix} = 0. \quad (23)$$

To solve for K and S at the wavelength λ, combine equations from all m mixtures into a single matrix expression of the form $Mx = b$:

$$\begin{bmatrix} -C_1(1) & ... & -C_n(1) & f(R(\lambda,1))C_1(1) & ... & f(R(\lambda,1))C_n(1) \\ -C_1(2) & ... & -C_n(2) & f(R(\lambda,2))C_1(2) & ... & f(R(\lambda,2))C_n(2) \\ ... & -... & ... & ... & ... & ... \\ -C_1(m) & ... & -C_n(m) & f(R(\lambda,m))C_1(m) & ... & f(R(\lambda,m))C_n(m) \end{bmatrix} \begin{bmatrix} K_1(\lambda) \\ ... \\ K_n(\lambda) \\ S_1(\lambda) \\ ... \\ S_n(\lambda) \end{bmatrix} = \begin{bmatrix} 0 \\ ... \\ 0 \\ 0 \\ ... \\ 0 \end{bmatrix}. \quad (24)$$

The matrix has m rows and $2n$ columns. (As before, weights should be considered. Ref. 11 suggests one set of weights.)

3.3.2 The CLS Approach for K and S

In addition to Equation (24), the K's and S's are subject to some constraints. Some difficulty arises in determining these constraints because there are competing interpretations of K and S. In the first interpretation, K_i is the fraction of incoming light that is absorbed by a unit thickness of the i^{th} paint film, and S_i is the fraction that is scattered. The remaining fraction, T_i, is transmitted perfectly. Under this interpretation, we must have

$$K_i + S_i + T_i = 1, \quad (25)$$

and each K_i, S_i, and T_i must be between 0 and 1. Furthermore, since T_i is not used in the Kubelka-Munk analysis of opaque paint films, we must also have

$$0 \leq K_i + S_i \leq 1 \quad (26)$$

for every i.

In the second interpretation, however, K and S are not fractions of incoming light. Rather, they are coefficients in the equations that describe how light interacts with a paint

film. If $I(t)$ is the power of the downward light at a depth t in the paint film, and $J(t)$ is the power of the upward light, then the Kubelka-Munk model[1] posits two differential equations:

$$-\frac{dI}{dt}(t) = -(K+S)I(t) + SJ(t), \tag{27}$$

$$\frac{dJ}{dt}(t) = -(K+S)J(t) + SI(t). \tag{28}$$

If there were no scattering, then S would be 0, $J(t)$ would be always 0, and Equation (27) would simplify to

$$\frac{dI}{-dt}(t) = -KI(t), \tag{29}$$

whose solution is

$$I(t) = e^{Kt}. \tag{30}$$

This example makes it clear that K represents the instantaneous *rate* at which the paint film absorbs light, rather than a fraction of light which is absorbed. The second interpretation is therefore significantly different from the first.

The first interpretation would apply when films are applied in discrete layers. For example, a printing process might lay down a film of red ink or paint, and apply a second film, of blue, on top of it. Each film would then absorb some fraction K, between 0 and 1, of the light which reached it, transmit another fraction T, and scatter the remaining fraction S. The three fractions would sum to 1, as posited by Equation (25). Rather than layering the films, however, one could mix the two inks or paints directly and apply a single film of the mixture. Equations (27) and (28) were derived under this assumption of mixing (and laying down an opaque layer). In that case, the second interpretation applies. Since we are using the Kubelka-Munk derivations for mixing, we will also use the second interpretation.

The signs in Equations (27) and (28) were chosen to make K and S non-negative, so for every i we have the constraints

$$0 \leq K_i, \tag{31}$$

$$0 \leq S_i. \tag{32}$$

The basic Kubelka-Munk relationship given by Equation (11), furthermore, shows that a paint mixture's colour depends not on the magnitudes of the constituent paints' K's and S's, but rather on their ratios. Suppose that every K and S in the vector $x = [K_1(\lambda), ..., K_n(\lambda), S_1(\lambda), ...S_n(\lambda)]$ were multiplied by a positive constant ϵ. As long as x satisfies the constraints (31) and (32), the reduced vector ϵx also satisfies those constraints. Equations (12) and (13) show that a mixture's colour depends on the cross-paint ratios, between, for example, K_1 and K_2, or S_1 and S_2. Multiplying every K and S by the same constant ϵ preserves all such ratios, and thus all Kubelka-Munk quantities. Equation (24) can be satisfied as nearly as desired, simply by choosing ϵ small enough, but multiplying by ϵ does not really produce a new solution. We will impose an additional constraint to avoid spurious distinctions between solutions that are constant multiples of each other:

$$\sum_{i=1}^{n}(K_i + S_i) = 1. \tag{33}$$

Any set of K's and S's would satisfy this constraint if multiplied by an appropriate ϵ, so there is no loss of generality in this imposition.

The vector x consists of the K_i's for each base paint, followed by the S_i's for each base paint. Constraint (33) therefore says that the $2n$ entries of x sum to 1. Constraints (31) and (32) say that each entry of x is non-negative. Since the non-negative entries sum to 1, each entry itself is less than or equal to 1. Constraints (31) through (33) could therefore be rewritten as

$$0 \leq x_i \leq 1, \tag{34}$$

$$\sum_{i=1}^{2n} x_i = 1. \tag{35}$$

Surprisingly, these constraints are formally equivalent to the constraints (21) and (22) that occurred when estimating concentrations. This equivalence allows the CLS approach for estimating K and S to follow the same path. The columns of the matrix in Equation (24) become the generators of a convex polytope P. A CLS algorithm is applied to P to find a convex combination of generators which is as close as possible to the zero vector (which appears on the right-hand side of Equation (24)). The coefficients of this convex combination are the x_i's, each of which is a K or S for some base paint. The CLS algorithm guarantees that each K or S will satisfy the non-negativity constraint, so we can estimate physically realistic K's and S's, as desired.

4 Comparison to Previous Work

Historically, it is unclear exactly when the least squares nature of Kubelka-Munk problems was first recognized. As early as 1973, Gall[12] presented the linear relationship that appears in Equation (15), and, in 1987, Walowit, McCarthy, and Berns[2,3] formulated Kubelka-Munk estimation problems as a linear system. The derivation of Equations (20) and (24) follows their approach. To avoid an all-zero "solution," the three latter authors added a constraint, requiring entries in x to sum to 1. Formally, a row of all 1's was appended to M, and an additional 1 was appended to b. The augmented linear system was then solved with ordinary least squares.

Ordinary least squares can sometimes give a physically invalid solution, even when there is a simple, valid solution. For example, suppose we are trying to solve

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 9 \\ 12 \end{bmatrix}. \tag{36}$$

An obvious solution is $[C_1, C_2, C_3] = [0, 0, 1]$. Ordinary least squares, however, as implemented in the Octave/MATLAB routine ols.m, returns $[C_1, C_2, C_3] = [-1/6, 1/3, 5/6]$. The OLS solution would presumably be treated as invalid because it contains a negative number.

The possibility of physically invalid solutions was recognized early on. On p. 361 of Ref. 3, which deals with calculating concentrations, the authors, following McGinnis,[13] recommend

discarding paints with negative concentrations, and rerunning the method with the remaining paints. Solutions with negatives were simply discarded in some recent work of Koirala et al,[14] also involving concentration estimation. As the above example shows, this approach might overlook a desirable solution.

Such situations in fact motivated the current paper. Physical reasoning implies that a valid solution exists. Suppose one is looking for the concentrations in a mixture of base paints that most closely matches a target paint. The mixture of any set of concentrations will differ from the target by some amount. That difference could be measured by a difference expression (DE), a sum of squared reflectance differences, or some other metric. Since any set of concentrations gives a difference, at least one set of concentrations gives the minimum difference. If an algorithm cannot find such a set, then the problem is with the algorithm, and not with the physical situation. The current paper incorporates physical constraints explicitly, so that algorithms can avoid invalid solutions.

In a series of papers,[15-17] Cogno et al. presented another approach that also incorporates constraints explicitly, but that only finds an approximate solution. Their methods apply to concentration estimation problems; it is not immediately clear how to generalize them to K and S estimation problems. The concentrations are used as the basis of a vector space, and the constraints in (2) and (3) then define a convex polytope (in fact, a simplex) in that vector space—this polytope is *not* the polytope constructed in the current paper. The colour of the mixture defined by a point on the simplex differs from the target colour by some amount, so that difference is a function on the simplex. The best mixture occurs when that function is minimized over the simplex. If that function were linear (or quadratic) in the concentrations, then linear (or quadratic) programming algorithms could be used. Though the difference function is neither linear nor quadratic, Cogno et al. suggest some approximations that make it nearly linear or quadratic, at least locally, so that algorithms involving programming techniques give useful results. The current paper differs from Cogno's approach by only using exact expressions: the distance we are minimizing really is the least-squares difference function between the target colour and a paint mixture.

5 Summary

To sum up, the present paper modifies some OLS Kubelka-Munk algorithms to produce only physically valid solutions. Physical constraints are incorporated directly, by restricting the domain of the OLS problem. In two problems of interest (estimating concentrations from reflectance spectra and K's and S's, and estimating K's and S's from spectra and concentrations), this domain is a convex polytope. The problems then reduce to finding the point on that polytope that is closest to a target point. An optimization algorithm, such as GJK, finds this closest point, and expresses it in terms of some generating vectors for the polytope. From this expression, one can work backwards to Kubelka-Munk quantities of interest, such as K's, S's, and C's. Now that the general approach has been outlined, it is hoped that it can be applied to further estimation problems.

6 Appendix: The GJK Algorithm

6.1 Algorithm Setting

The Gilbert-Johnson-Keerthi (GJK) algorithm[5] was introduced in 1988. The algorithm finds the shortest distance between two convex polytopes in a Euclidean space of arbitrary finite dimension.

A convex polytope in $\mathbb{R}^r$ is most easily defined as the convex hull of a finite set of points. Formally, suppose $V = \{v_1, v_2, ..., v_k\}$ is a set of points in $\mathbb{R}^r$. Then co(V), the convex hull of V, is given by

$$\text{co}(V) = \left\{ \sum_{i=1}^{k} \alpha_i v_i \,\middle|\, \sum_{i=1}^{k} \alpha_i = 1 \text{ and } 0 \leq \alpha_i \leq 1 \forall i \right\}. \tag{37}$$

Intuitively, convex polytopes generalize convex polygons and polyhedra to an arbitrary number of dimensions. Polytopes in three dimensions can be thought of as bodies with corners, sharp edges, and flat faces; in higher dimensions, a polytope's boundary is similarly a union of flat objects of lower dimension. (See Chap. 8 of Ref. 4 for proofs and further details.) The requirement that a polytope be generated by a finite set of points means that curved surfaces such as spheres cannot be polytopes. Polytopes are not required to be fully dimensional. For example, a line segment in $\mathbb{R}^3$ is a polytope, as is a triangle. Convex polytopes are always closed and bounded. The set of generating points for a convex polytope is not unique. A cube, for example, is a polytope, one of whose generating sets is its eight vertices; the union of the cube's vertices and the cube's center is again a generating set.

Though the GJK algorithm can take two convex polytopes as input, the Kubelka-Munk applications studied in this paper need only a special case: one of the polytopes is a target point b. The GJK algorithm then determines the minimum distance from the target point to the polytope, the point b_P on the polytope which is closest to the target point, and an expression for b_P as a convex linear combination of the generators, as in Equation (37). In some cases, the coefficients of the convex combination are actually the entries of the vector x, in a matrix equation $Mx = b$. In these cases, which include the Kubelka-Munk problems presented here, the GJK algorithm actually provides a least squares solution to $Mx = b$. While that solution might not be unique, it will at least be a valid solution which satisfies the given constraints. In other cases, x can be written as a function of the coefficients, still allowing a solution.

6.2 Algorithm Description

The GJK algorithm will be explained with a two-dimensional example, which extrapolates easily to higher dimensions. Suppose that a convex polytope P in $\mathbb{R}^2$ is generated from a finite set V of points $v_i, i = 1, ..., k$, as shown in Figure 1. While all the extreme points, or corners, of P are points in V, there can also be points of V, such as v_4 and v_6, in the interior of P. The interior points are superfluous but harmless: the algorithm will reach the same result whether or not they are present. A target point b is also shown. The GJK algorithm finds the point b_P on P which is closest to b. The convexity of P guarantees that such a

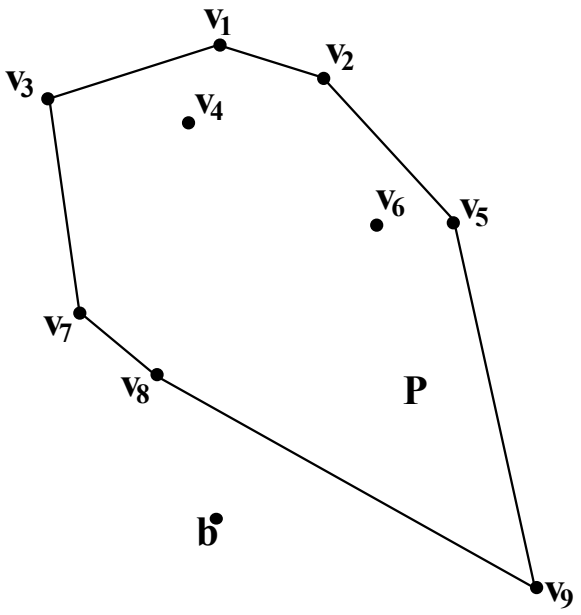

Figure 1: Two-Dimensional Example for GJK Algorithm

point exists and is unique (see Sect. 19.2 of Ref. 4). While b is often outside P, b can also be inside P, or on its boundary; in such cases the distance to P is 0, and the closest point b_P is just b itself.

An important concept for the GJK algorithm is the support function. The support function $\rho_{d,u}$ uses a unit direction vector d, which originates at a point u, as shown in Figure 2. Given any point v in the space, the support of v, relative to d and u, is defined as

$$\rho_{d,u}(v) = (v - u) \cdot d. \tag{38}$$

Geometrically, the vector d can be extended to a line, with the point u indexed by 0. Other points on this line are indexed by their distance from u, with positive points in the direction of d and negative points in the direction of $-d$. The point v can be projected perpendicularly onto this line. The index of the projection point is then the support of v, relative to d and u. The GJK algorithm uses the support function to move across P in certain directions, approximately towards b. When the algorithm has converged, the generating points that make a non-zero contribution to b_P will all have minimum support, and no further reduction is possible.

The GJK algorithm begins by choosing a subset W of V. W can contain up to $n + 1$ points, where n is the dimension of the space. The points should be affinely independent so that they generate a simplex. Apart from these restrictions, the choice of W is arbitrary. In two dimensions, W will be either a single point, two points which define a line segment, or three points which define a triangle. For simplicity, W can be chosen to be a single point. For purposes of illustration, we will start with the points v_3, v_6, and v_8, shown in Figure 3, which generate a triangle.

Once W has been chosen, the convex hull $\text{co}(W)$ of its elements will form a simplex, such as the triangle in Figure 3. At this point, Johnson's sub-distance algorithm[6] can be used to determine the point b_W on $\text{co}(W)$ that is closest to b, as well as the minimal subsimplex

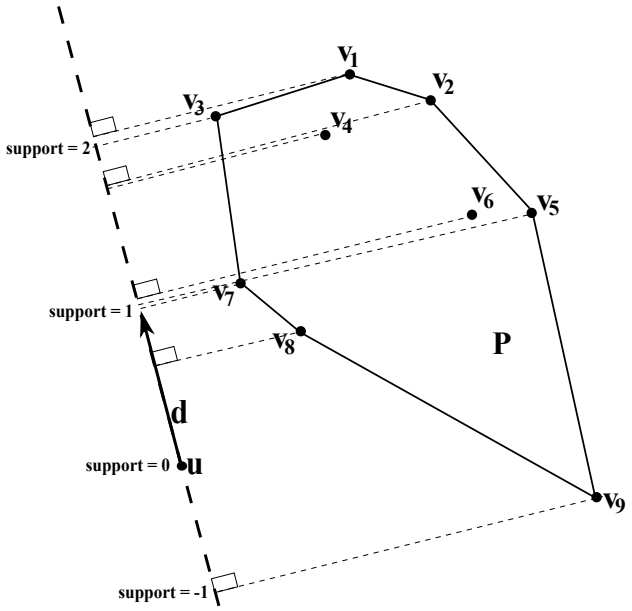

Figure 2: The Support Function

of co(W) that contains b_W. In Figure 3, for example, the point b_W is on the line segment generated by v_8 and v_6. That line segment is the minimal subsimplex. Since v_3 is not needed to generate this subsimplex, discard v_3 from W. Johnson's algorithm returns not just the minimal subsimplex, but also an expression for b_W as a convex combination of the polytope's generators. In the figure, for example, $b_W = 0.85v_8 + 0.15v_6$.

Information about W is useful, but there are also many points of P that are outside W that should also be considered. For the next step, Figure 4 shows how the support function expands the minimal W into a new simplex that contains points of P that are closer to b. Construct the unit vector d that originates at b and points toward b_w. Calculate the support for each point in V, relative to d and b. Choose a point v_i of minimal support, and add v_i to the set W. It is possible for the support to be negative, and a point with negative support should be chosen over a point with positive support, so that none of the set P is neglected. In Figure 4, the point v_9 has minimal support, so it was added to the new W that is shown there. This step has some subtleties. It is possible that the "simplex" created from the union of W and the new point has a smaller dimension than a simplex with that many vertices would. In that case, one must delete further points from W, try a different new point, or adjust one of the points very slightly.

Now iterate over the previous two steps:

1. Find the nearest point b_W on the simplex generated by W, and extract the minimal containing subsimplex.
2. Find a point whose support, with respect to the vector from b towards b_W, is as small as possible, and that still makes a simplex when added to W. Add that point to W.

These two steps will produce a series of points b_W, that terminate in the desired b_P. The algorithm terminates when all points of minimal support are already contained in the current W. Figure 5 shows the termination for the example, which occurs after the second set W

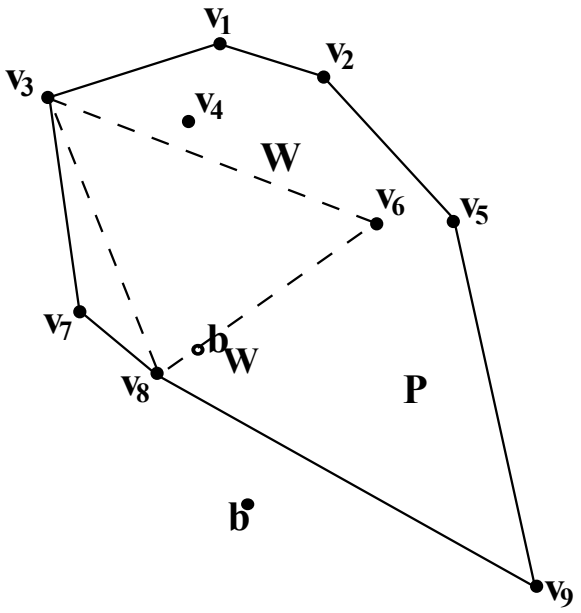

Figure 3: Choosing a Simplex in the Polytope P

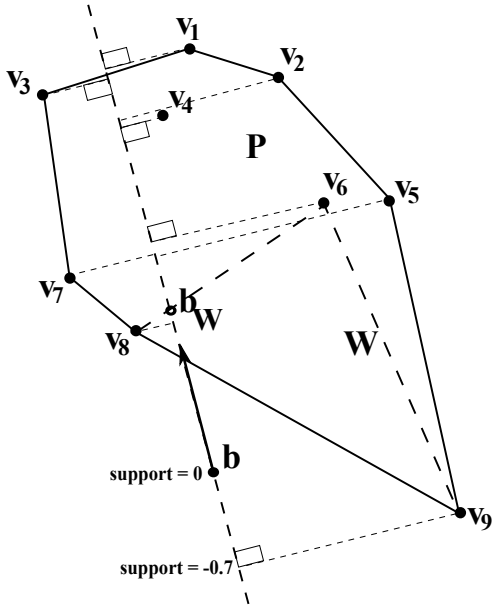

Figure 4: Choosing a New Simplex in the Polytope P

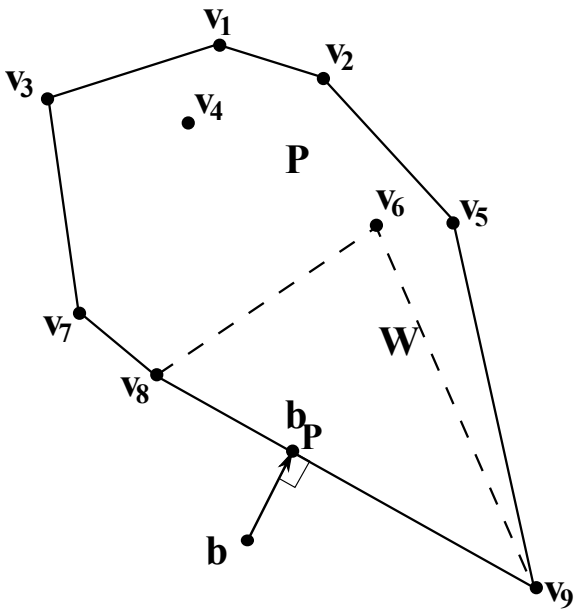

Figure 5: Termination of the GJK Algorithm

has been constructed. The point b_P is contained in the minimal subsimplex generated by v_8 and v_9. The support function of $b_P - b$ takes on a minimum only at v_8 and v_9, which are already in the latest W. Geometrically, the vector from b to b_P is perpendicular to an edge of P, so both vertices for that edge will have the same support. In three dimensions, the vector from b to b_P could be perpendicular to a bounding face, all of whose vertices have the same minimal support. Analogues hold for higher dimensions. Again, the Johnson sub-algorithm provides an expression for b_P. In this case, $b_P = 0.6v_8 + 0.4v_9$.

The GJK algorithm is very fast. The reason for its speed is that generally only a small number of simplices are needed, and the simplices themselves require only a few vertices, often less than half a dozen. (The slowest part of the algorithm is evaluating the support function.) This speed, however, requires that the polytope of interest be expressed by a set of generating vectors. Although the polytope is implicitly a convex hull of its generators, a surprising GJK feature is that there is no need to calculate the convex hull directly. This features saves time, too, because finding convex hulls can be slow, especially in high dimensions. Fortuitously, the Kubelka-Munk model produces polytopes with just these kinds of generators. Another convenient feature for Kubelka-Munk problems is that the GJK algorithm expresses the closest point as a convex combination of the generators. The derivations show that the coefficients in this combination are in fact the desired K's, S's, and C's. Other algorithms or solvers might find b_P, but not express it as a convex combination, so additional effort might be needed to extract the desired quantities. Overall, the GJK algorithm seems tailor-made for the Kubelka-Munk problems that motivated this paper.

References

1. Eugene Allen, "Colorant Formulation and Shading," Chapter 7 of *Optical Radiation Measurements, Volume 2: Color Measurement*, eds. Franc Grum & C. James Bartleson, Academic Press, 1980.
2. Eric Walowit, Cornelius J. McCarthy, & Roy S. Berns, "An Algorithm for the Optimization of Kubelka-Munk Absorption and Scattering Coefficients," COLOR Research and Application, Vol. 12, Number 6, December 1987, pp. 340-343.
3. Eric Walowit, Cornelius J. McCarthy, & Roy S. Berns, "Spectrophotometric Color Matching Based on Two-Constant Kubelka-Munk Theory," COLOR Research and Application, Vol. 13, Number 6, December 1988, pp. 358-362.
4. S. R. Lay, *Convex Sets and Their Applications*, Dover Publications, 2007.
5. Elmer G. Gilbert, Daniel W. Johnson, & Sathiya Keerthi, "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," IEEE Journal of Robotics and Automation, Vol. 4, No. 2, April 1988, pp. 193-203.
6. Rich Rabbitz, "Fast Collision Detection of Moving Convex Polyhedra," in Section I.8 of *Graphics Gems IV (IBM version)*, ed. Paul Heckbert, Academic Press, 1994.
7. William H. Press, Saul A. Teukolsky, William T. Vetterling, & Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing, 3rd ed.*, Cambridge University Press, 2007.
8. Charles L. Lawson & Richard J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.
9. G. Wyszecki and W. S. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae, 2nd ed.* (New York: John Wiley & Sons, 1982).
10. CIE, *Colorimetry, 3rd ed.*, CIE Publication No. 15:2004, Vienna, 2004.
11. Paul Centore, "Perceptual Reflectance Weighting for Estimating Kubelka-Munk Coefficients," 2014, available at www.99main.com/∼centore.
12. L. Gall, "Computer Colour Matching," pp. 153-178 in *Colour 73*, Adam Hilger, London, 1973.
13. Paul H. McGinnis, "Spectrophotometric Color Matching With the Least Squares Technique," Color Engineering, Vol. 5, No. 6, pp. 22-27, November-December 1967.
14. Pesal Koirala, Markku Hauta-Kasari, Birgitta Martinkauppi, & Jouni Hiltunen "Color mixing and color separation of pigments with concentration prediction," Color Research & Application, Vol. 33, No. 6, pp. 461-469, December 2008.
15. J. A. Cogno, D. Jungman, & J. C. Conno, "Linear and Quadratic Optimization Algorithms for Computer Color Matching," Color Research & Application, Vol. 13, No. 1, pp. 40-42, February 1988.
16. J. A. Cogno, "Recursive Quadratic Programming Algorithm for Color Matching," Color Research & Application, Vol. 13, No. 2, pp. 124-126, April 1988.
17. J. A. Cogno, "Mixed-Integer Programming Algorithm for Computer Color Matching," Color Research & Application, Vol. 13, No. 1, pp. 43-45, February 1988.

