

By Spigget [CC BY-SA 3.0], via [Wikimedia Commons](#) (edited)

Generating Reflectance Curves from sRGB Triplets

by Scott Allen Burns, Urbana, IL

Published April 29, 2015

Note: This is a PDF version of the web page <http://scottburns.us/reflectance-curves-from-srgb/>

Overview

I present several algorithms for generating a reflectance curve from a specified sRGB triplet. Although there are an infinite number of reflectance curves that can give rise to the specific color sensation associated with an sRGB triplet, the algorithms presented here are designed to generate reflectance curves that are similar to those found with naturally occurring colored objects. My hypothesis is that the reflectance curve with the least sum of slope squared (or in the continuous case, the integral of the squared first derivative) will do this. After presenting the algorithms, I examine the quality of the computed reflectance curves compared to thousands of documented reflectance curves measured from paints and pigments available commercially or in nature. Being able to generate reflectance curves from three-dimensional color information is useful in computer graphics, particularly when modeling color transformations that are wavelength specific.

Introduction

There are many different 3D color space models, such as XYZ, RGB, HSV, $L^*a^*b^*$, etc., and one thing they all have in common is that they require only three quantities to describe a unique color sensation. This reflects the “trichromatic” nature of human color perception. The space of *color stimuli*, however, is not three dimensional. To specify a unique color stimulus that enters the eye, the power level at every wavelength over the visible range (e.g., 380 nm to 730 nm) must be specified. Numerically, this is accomplished by discretizing the spectrum into narrow wavelength bands (e.g., 10 nm bands), and specifying the total power in each band. In the case of 10 nm bands between 380 and 730 nm, the space of color stimuli is 36 dimensional. As a result, there are many different color stimuli that give rise to the same color sensation (infinitely many, in fact).

For most color-related applications, the three-dimensional representation of color is efficient and appropriate. But it is sometimes necessary to have the full wavelength-based description of a color, for example, when modeling color transformations that are wavelength specific, such as dispersion or scattering of light, or the subtractive mixture of colors, for example, when mixing paints or illuminating colored objects with various illumi-

nants. In fact, this web page was developed in support of another page on this site, concerning [how to compute the RGB color produced by subtractive mixture of two RGB colors](#).

I present several algorithms for converting a three-dimensional color specifier (sRGB) into a wavelength-based color specifier, expressed in the form of a reflectance curve. When quantifying object colors, the reflectance curve describes the fraction of light that is reflected from the object by wavelength, across the visible spectrum. This provides a convenient, dimensionless color specification, a curve that varies between zero and one (although fluorescent objects can have reflectance values >1). The motivating idea behind these algorithms is that the one reflectance curve that has the least sum of slope squared (integral of the first derivative squared, in the continuous case) seems to match reasonably well the reflectance curves measured from real paints and pigments available commercially and in nature. After presenting the algorithms, I compare the computed reflectance curves to thousands of documented reflectance measurements of paints and pigments to demonstrate the quality of the match.

sRGB Triplet from a Reflectance Curve

The reverse process of computing an sRGB triplet from a reflectance curve is straightforward. It requires two main components: (1) a mathematical model of the “standard observer,” which is an empirical mathematical relationship between color stimuli and color sensation (tristimulus values), and (2) a definition of the sRGB color space that specifies the reference illuminant and the mathematical transformation between tristimulus values and sRGB values.

The linear transformation relating a color stimulus, S , to its corresponding tristimulus values, XYZ , is

$$XYZ_{3 \times 1} = A_{3 \times n} S_{n \times 1}.$$

The matrix A has three rows (called the three “color matching functions”) and n columns, where n is the number of discretized wavelength bands. In this study, all computations are performed with 36 wavelength bands of width 10 nm, running over the range 380 nm to 730 nm. The stimulus vector also has n components, representing the total power of all wavelengths within each band. The specific color matching functions I use in this work are the [CIE 1931 color matching functions](#).

The stimulus vector can be constructed as the product of an $n \times n$ diagonal illuminant matrix, $\text{diag}(I)$, and an $n \times 1$ reflectance vector, ρ . Matrix A is usually normalized so the Y tristimulus value is 1 when ρ is a perfect reflector (contains all 1s). The normalizing factor, w , is thus the inner product of the second row of A and the illuminant vector, I , yielding the alternate form of the tristimulus value equation

$$XYZ = A \text{diag}(I) \rho / w.$$

The transformation from tristimulus values to sRGB is a two-step process. First, a 3×3 linear transformation is applied to convert XYZ to rgb , which is a triplet of “linear RGB” values:

$$rgb = M A \text{diag}(I) \rho / w.$$

The second step is to apply a “gamma correction” to the linear rgb values, also known as “companding” or applying the “color component transfer function.” This is how it is done: for each r , g , and b component of rgb , let’s generically call it v , if $v \leq 0.0031308$, use $12.92v$, otherwise use $1.055v^{1/2.4} - 0.055$. This gives sRGB values in the range of 0 to 1. To convert them to the alternate integer range of 0 to 255 (used in most 24-bit color devices), we multiply each by 255 and round to the nearest integer.

The expression relating rgb and ρ above can be simplified by combining the three matrices and the normalizing factor into a single matrix,

$$T = M A \text{diag}(I)/w$$

so that

$$rgb = T \rho.$$

The [formal definition](#) of the sRGB color space uses an illuminant similar to daylight, called D65, as its "reference" illuminant. Here are the specific values

for the M [matrix](#), the A [matrix](#), the [D65 I vector](#), and the T [matrix](#). The normalizing factor, w , has a value of 10.5677. Most of the theory I present here I learned from [Bruce Lindbloom's highly informative website](#).

sRGB to rgb

The inverse operation of converting sRGB to rgb , expressed in code, is:

```
% convert 0-255 range to 0-1 range
sRGB=sRGB/255;
for i=1:3
    if sRGB(i)<0.04045
        rgb(i)=sRGB(i)/12.92;
    else
        rgb(i)=((sRGB(i)+0.055)/1.055)^2.4;
    end
end
```

Now that we have a simple expression for computing sRGB from a reflectance curve, we can use that as the basis of doing the opposite, computing a reflectance curve from an sRGB triplet. In the sections that follow, I will present five different algorithms for doing this. Each has its strengths and weaknesses. Once they are presented, I will then compare them to each other and to reflectance curves found in nature.

Linear Least Squares (LLS) Method

Since there are so many more columns of T than rows, the linear system is under-determined and gives rise to a 33-dimensional subspace of reflectance curves for a single sRGB triplet. There are well-established techniques for solving under-determined linear systems. The most common method goes by various names: the linear least squares method, the pseudo-inverse, the least-squares inverse, the Moore-Penrose inverse, and even the "[Fundamental Color Space](#)" transformation.

Suppose we pose this optimization problem:

$$\begin{aligned} &\text{minimize } \rho^T \rho \\ &\text{s.t. } T\rho = rgb \end{aligned}$$

This linearly constrained minimization can be solved easily by forming the Lagrangian function

$$L(\rho, \lambda) = \rho^T \rho + \lambda^T (T\rho - rgb).$$

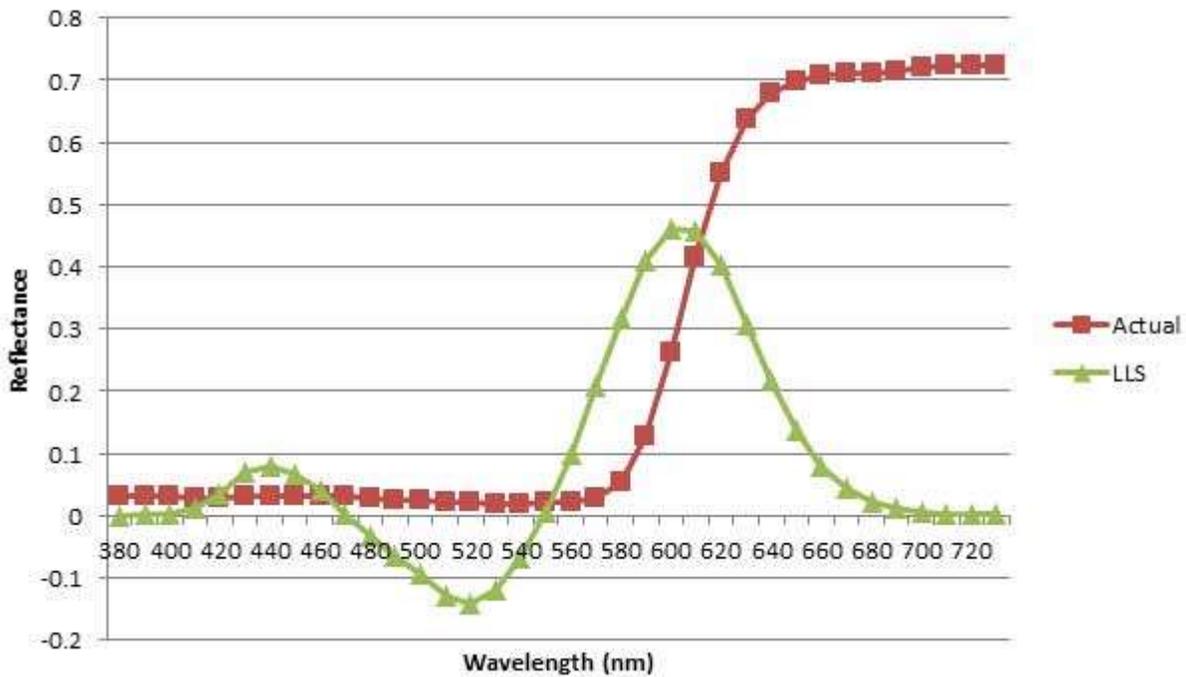
The solution can be found by finding a stationary point of L , i.e., setting partial derivatives with respect to ρ and λ equal to zero:

$$\begin{aligned} \partial L / \partial \rho &= 2\rho + T^T \lambda = 0 \\ \partial L / \partial \lambda &= T\rho - rgb = 0. \end{aligned}$$

Solving this system by eliminating λ gives the LLS solution

$$\rho = T^T (T T^T)^{-1} rgb.$$

Thus, a reflectance curve can be found from a sRGB triplet by simply converting it to linear rgb and solving a 3×3 linear system of equations. Unfortunately, the resulting solution is sometimes not very useful. Consider its application to this reflectance curve, which represents a bright red object color:



Reflectance curve for Munsell 5R 4/14 color sample and linear least-squares reconstruction.

The LLS solution contains negative reflectance values, which don't have physical meaning and limit its usefulness in realistic applications. Computationally, this is a very efficient method. The matrix $T^T(T T^T)^{-1}$ can be computed in advance, [as shown here](#), and each new sRGB value needs only be multiplied by it to get a reflectance curve.

Least Slope Squared (LSS) Method

Note that the standard LLS method minimizes $\rho^T \rho$, that is, it finds the solution that is nearest to the origin, or the reflectance curve with the least sum of squares of all its entries. For purposes of computing reflectance curves, I can't think of a compelling reason why this should be a useful objective.

It dawned on me that it might be better to try a different objective function. The reflectance curves of most natural colored objects don't tend to oscillate up and down very much. I came up with the idea to minimize the square of the slope of the reflectance curve, summed over the entire curve. In the continuous case, this would be equivalent to

$$\min \int_{\text{visible } \lambda} (d\rho/d\lambda)^2 d\lambda.$$

The square is used because it equally penalizes upward and downward movement of ρ . This objective will favor flatter reflectance curves and avoid curves that have a lot of up and down movement.

Other researchers have developed methods for reconstructing reflectance curves from tristimulus values. In order to reduce the oscillations, they typically introduce basis functions that oscillate very little, such as segments of low-frequency sinusoids, or they "frequency limit" or "band limit" the solution by constraining portions of the Fourier transform of the reflectance curve. To my mind, these approaches seem to ignore that fact that realistic reflectance curves can sometimes exhibit relatively sharp wavelength cutoffs, which would have relatively large high frequency Fourier components. These methods would not be able to create such reflectance curves.

My proposed method would be able to create sharp cutoffs, but only as a last resort when flatter magnitude curves are not able to match the target tristimulus values. The other advantage of the minimum slope squared approach is that it can be expressed as a quadratic objective function subject to linear constraints, which is solvable by standard least-square strategies.

Consider this optimization formulation:

$$\begin{aligned} & \text{minimize} \quad \sum_{i=1}^{n-1} (\rho_{i+1} - \rho_i)^2 \\ & \text{s.t.} \quad T\rho = rgb, \end{aligned}$$

where n is the number of discrete wavelength bands (36 in this study). This optimization can be solved by solving the system of linear equations arising from the Lagrangian stationary conditions

$$\begin{bmatrix} D & T^T \\ T & 0 \end{bmatrix} \begin{Bmatrix} \rho \\ \lambda \end{Bmatrix} = \begin{Bmatrix} 0 \\ rgb \end{Bmatrix},$$

where D is a 36x36 tridiagonal matrix

$$D = \begin{bmatrix} 2 & -2 & & & & \\ -2 & 4 & -2 & & & \\ & -2 & 4 & -2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -2 & 4 & -2 \\ & & & & -2 & 2 \end{bmatrix}.$$

Since D and T do not depend on rgb , the matrix can be inverted ahead of time, instead of each time an sRGB value is processed. Defining

$$B = \begin{bmatrix} D & T^T \\ T & 0 \end{bmatrix}^{-1},$$

we have

$$\begin{Bmatrix} \rho \\ \lambda \end{Bmatrix} = B \begin{Bmatrix} 0 \\ rgb \end{Bmatrix}$$

or

$$\rho = B_{12} rgb,$$

where B_{12} is the upper-right 36×3 portion of the B matrix, [as shown here](#). Thus, the LSS method is just as computationally efficient as the LLS method, and tends to give much better reflectance curves, as I'll demonstrate later.

[Here is a Matlab program](#) for the LSS (Least Slope Squared) method.

Least Log Slope Squared (LLSS) Method

It is generally not a good idea to allow reflectance curves with negative values. Not only is this physically meaningless, but it also can cause problems down the road when the reflectance curves are used in other computations. For example, when [modeling subtractive color mixture](#), it may be necessary to require the reflectance curves to be strictly positive.

One way to modify the LSS method to keep reflectances positive is to operate the algorithm in the space of the logarithm of reflectance values, $z = \ln(\rho)$. I call this the **Least Log Slope Squared (LLSS) method**:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{35} (z_{i+1} - z_i)^2 \\ \text{s.t.} \quad & T e^z = rgb. \end{aligned}$$

This new optimization is not as easy to solve as the previous one. Nevertheless, the Lagrangian formulation can still be used, giving rise to a system of 39 *nonlinear* equations and 39 unknowns:

$$F = \begin{Bmatrix} Dz - \text{diag}(e^z) T^T \lambda \\ -T e^z + rgb \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix},$$

where D is the same 36×36 tridiagonal matrix presented earlier.

Newton's method solves this system of equations with ease, typically in just a few iterations. Forming the Jacobian matrix,

$$J = \left[\begin{array}{c|c} D - \text{diag}(\text{diag}(e^z) T^T \lambda) & -\text{diag}(e^z) T^T \\ \hline -T \text{diag}(e^z) & 0 \end{array} \right],$$

the change in the variables with each Newton iteration is found by solving the linear system

$$J \begin{Bmatrix} \Delta z \\ \Delta \lambda \end{Bmatrix} = -F.$$

[Here is a Matlab program](#) for the LLSS (Least Log Slope Squared) method. I added a check for the special case of sRGB = (0,0,0), which simply returns $\rho = (0.0001, 0.0001, \dots, 0.0001)$. This is necessary since the log formulation is not able to create a reflectance of exactly zero (nor is that desirable in some applications). It can come very close to zero as z approaches $-\infty$, but it is numerically better to handle this one case specially. I chose the value of 0.0001 because it is the largest power of ten that translates back to an integer sRGB triplet of (0,0,0).

The LLSS method requires substantially more computational effort than the previous two methods. Each iteration of Newton's method requires the solution of 39 linear equations in 39 unknowns.

Iterative Least Log Slope Squared (ILLSS) Method

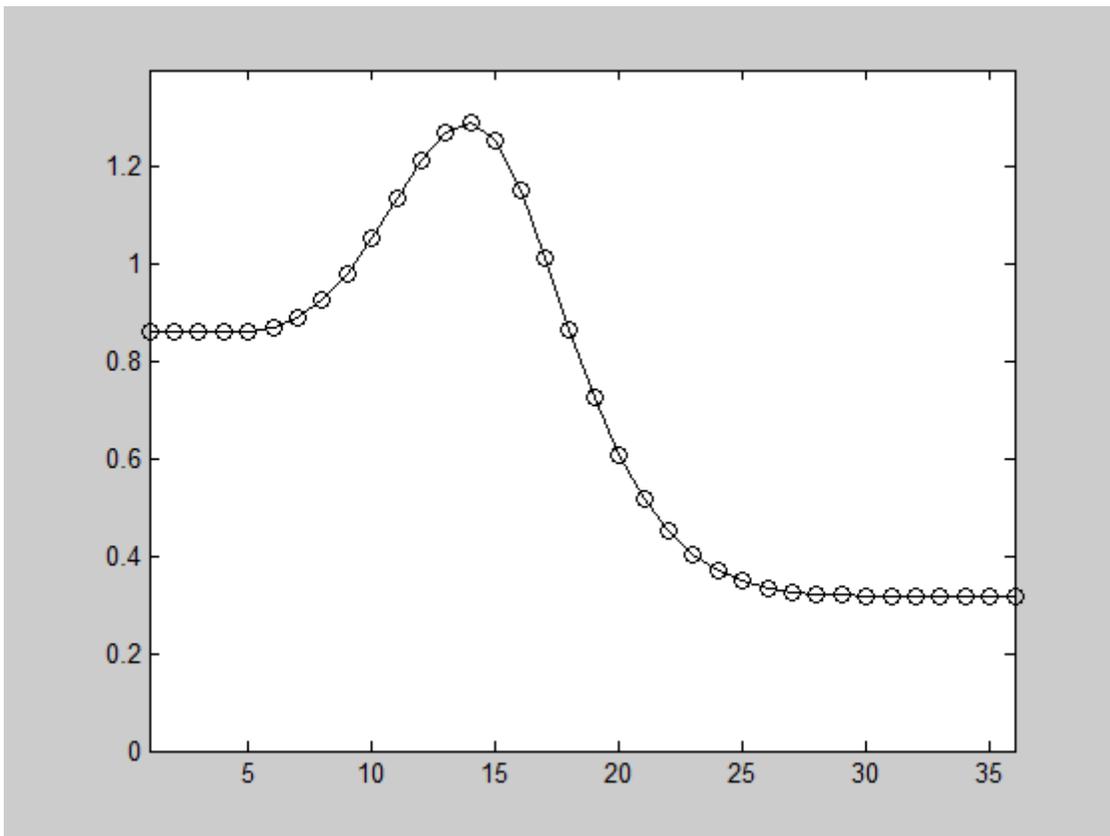
The LLSS method above can return reflectance curves with values >1 . Although this is physically meaningful phenomenon (fluorescent objects can exhibit this), it may be desirable in some applications to have the entire reflectance curve between 0 and 1. It dawned on me that I might be able to modify the Lagrangian formulation to cap the reflectance values at 1. The main obstacle to doing this is that the use of inequality constraints in the

Lagrangian approach greatly complicates the solution process, requiring the solution of the "KKT conditions," and in particular, the myriad "complementary slackness" conditions. If only there were some way to know which reflectance values need to be constrained at 1, then these could be treated by a set of equality constraints and no KKT solution would be necessary.

That led me to investigate the nature of the LLSS reflectance curves with values >1 . I ran the LLSS routine on every value of sRGB by intervals of five, that is, sRGB = (0,0,0), (0,0,5), (0,0,10), ..., (255,255,250), (255,255,255). In every one of those 140,608 cases, the algorithm found a solution in less than a dozen or so iterations (usually just a handful), and 38,445 (27.3%) of them had reflectance values >1 .

Of the 38,445 solutions with values >1 , 36,032 of them had a single contiguous region of reflectance values >1 . The remaining 2,413 had two regions, always located at both ends of the visible spectrum. Since the distribution of values >1 is so well defined, I started thinking of an algorithm that would iteratively force the reflectance to 1. It would start by running the LLSS method. If any of the reflectance values ended up >1 , I would add a single equality constraint forcing the reflectance at the maximum of each contiguous >1 region to equal 1, solve that optimization, then force the adjacent values that were still >1 to 1, optimize again, and repeat until all values were ≤ 1 .

That was getting to be an algorithmic headache to implement, so I tried a simpler approach, as follows. First, run the LLSS method. If any reflectance values end up >1 , constrain ALL of them to equal 1, and re-solve the optimization. This will usually cause some more values adjacent to the old contiguous regions to become >1 , so constrain them in addition to the previous ones. Re-solve the optimization. Repeat the last two steps until all values are ≤ 1 . Here is an GIF animation of this process, which I call the **Iterative Least Log Slope Squared (ILLSS)** process, applied to sRGB = (75, 255, 255):



Animation of the ILLSS process (click image link to animate).

To express the ILLSS algorithm mathematically, let's begin with the LLSS optimization statement and add the additional equality constraints:

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^{35} (z_{i+1} - z_i)^2 \\ & \text{s.t.} && T e^z = rgb \\ & && z_k = 0, \quad k \in \text{FixedSet}. \end{aligned}$$

"FixedSet" is the set of reflectance indices that are constrained to equal 1, or equivalently, the set of z indices constrained to equal zero (since $z = \ln(\rho)$). Initially, FixedSet is set to be the empty set. Each time the optimization is repeated, the z values ≥ 0 have their indices added to this set.

We can define a matrix that summarizes the fixed set, for example:

$$K = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}_{2 \times 36}.$$

This example indicates that there are two reflectance values being constrained (because K has two rows), and the third and fifth reflectance values are the ones being constrained.

The Lagrangian formulation now has additional Lagrange multipliers, called μ , one for each of the constrained reflectances. The system of nonlinear equations produced by finding a stationary point of the Lagrangian (setting partial derivatives of the Lagrangian with respect to each set of variables (z , λ , and μ) equal to zero) is

$$F = \begin{Bmatrix} Dz - \text{diag}(e^z) T^T \lambda + K^T \mu \\ -T e^z + rgb \\ Kz \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix},$$

where D is the same 36×36 tridiagonal matrix presented earlier. As before, we solve this nonlinear system with Newton's method. Forming the Jacobian matrix,

$$J = \left[\begin{array}{c|c|c} D - \text{diag}(\text{diag}(e^z) T^T \lambda) & -\text{diag}(e^z) T^T & K^T \\ \hline -T \text{diag}(e^z) & 0 & 0 \\ \hline K & 0 & 0 \end{array} \right],$$

the change in the variables with each Newton iteration is found by solving the linear system

$$J \begin{Bmatrix} \Delta z \\ \Delta \lambda \\ \Delta \mu \end{Bmatrix} = -F.$$

[Here is a Matlab program](#) that performs the ILLSS (Iterative Least Log Slope Squared) optimization. I included a check for the two special cases of $rgb = (0,0,0)$ or $(255,255,255)$, which simply return $\rho = (0.0001, 0.0001, \dots, 0.0001)$ or $(1, 1, \dots, 1)$. The additional special case of $(255,255,255)$ is needed because numerical issues arise if the K matrix grows to 36×36 , as it would in that second special case.

Iterative Least Slope Squared (ILSS) Method

For completeness, I thought it would be a good idea to add one more algorithm. Recall the ILLSS method modifies the LLSS method to cap reflectances >1 . Similarly, the ILSS method will modify the LSS method to cap values both >1 and <0 . The ILSS may reduce computational effort in comparison to the ILLSS method since the inner loop of the ILLSS method requires an iterative Newton's method solution, whereas there would be no inner loop needed with the ILSS method; it is simply the solution of a linear system of equations. Here is the ILSS formulation:

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^{35} (\rho_{i+1} - \rho_i)^2 \\ & \text{s.t.} && T\rho = rgb \\ & && \rho_{k_1} = 1, \quad k_1 \in \text{FixedSet}_1 \\ & && \rho_{k_0} = \rho_{min}, \quad k_0 \in \text{FixedSet}_0. \end{aligned}$$

FixedSet_1 is the set of reflectance indices that are constrained to equal 1, and FixedSet_0 is the set of indices that are constrained to equal ρ_{min} (the smallest allowable reflectance, typically 0.00001). Initially, both fixed sets are the empty set. Each time the optimization is repeated, the ρ values ≥ 1 have their indices added to FixedSet_1 and those $\leq \rho_{min}$ have their indices added to FixedSet_0 .

We define two matrices that summarize the fixed sets, for example:

$$K_1 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}_{2 \times 36}$$

$$K_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 \end{bmatrix}_{3 \times 36}.$$

This example indicates that there are two reflectance values being constrained to equal 1 (because K_1 has two rows), and the third and fifth reflectance values are the particular ones being constrained. There are three reflectance values being constrained to ρ_{min} (because K_0 has three rows), and the first, sixth, and fourth are the particular ones being constrained. The order in which the rows appear in these matrices is not important.

At each iteration of the ILSS method, this linear system is solved:

$$\begin{bmatrix} D & T^\top & K_1^\top & K_0^\top \\ T & 0 & 0 & 0 \\ K_1 & 0 & 0 & 0 \\ K_0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \rho \\ \lambda \\ \mu \\ \nu \end{Bmatrix} = \begin{Bmatrix} 0 \\ rgb \\ 1 \\ \rho_{min} \end{Bmatrix},$$

where μ and ν are Lagrange multipliers associated with the K_1 and K_0 sets, respectively.

This system can be solved for ρ , yielding the expression

$$\rho = R - B_{11}K^\top \left[KB_{11}K^\top \right]^{-1} \left(KR - \begin{Bmatrix} 1 \\ \rho_{min} \end{Bmatrix} \right),$$

where

$$B = \begin{bmatrix} D & T^T \\ T & 0 \end{bmatrix}^{-1}, \quad R = B_{12} \text{ rgb}, \quad \text{and} \quad K = \begin{bmatrix} K_1 \\ K_0 \end{bmatrix}.$$

B_{11} and B_{12} are the upper-left 36×36 and upper-right 36×3 parts of B , respectively. They can be computed ahead of time. Note that only an $m \times m$ matrix needs to be inverted at each iteration, where m is the number of ρ values being held fixed, typically zero or a small number. When m is zero, the ILSS method simplifies to the LSS method.

[Here is a Matlab program](#) that performs the ILSS (Iterative Least Slope Squared) method.

Comparison of Methods

I ran each of the five algorithms with every sRGB value (by fives) and have summarized the results in the table below. The total number of runs for each solution was 140,608.

Name	Execution Time (sec)	Max ρ	Min ρ	Num $\rho > 1$	Num $\rho < 0$	Max Iter.	Mean Iter.	Computational Effort
LLS, Linear Least Squares	2.7	1.36	-0.28	26,317	50,337	n/a	n/a	Matrix mult only
LSS, Least Slope Squared	2.7	1.17	-0.17	9,316	48,164	n/a	n/a	Matrix mult only
ILSS, Iterative Least Slope Squared	25.7	1	0	0	0	5	1.49	Mult soln of m linear eqns**
LLSS, Least Log Slope Squared	322.	3.09	0	38,445	0	16	6.77	Mult soln of 39 linear eqns
ILLSS, Iterative Least Log Slope Squared	525.	1	0	0	0	5*	1.41*	Mult soln of (39+n) linear eqns**

* This is outer loop iteration count. The inner loop has iteration count similar to previous line.

** The quantity m is the number of reflectance values fixed at either 1 or 0.

Explanation of Columns

- Execution Time: Real-time duration to compute 140,608 reflectance curves of all sRGB values (in intervals of five), on a relatively slow Thinkpad X61 tablet. Relative times are more important than absolute times.
- Max ρ : The maximum reflectance value of all computed curves.
- Min ρ : The minimum reflectance value of all computed curves. Zero is used to represent some small specified lower bound, typically 0.0001.
- Num $\rho > 1$: The number of reflectance curves with maxima above 1.
- Num $\rho < 0$: The number of reflectance curves with minima below 0.
- Max Iter.: The largest number of iterations required for any reflectance curve (for iterative methods).
- Mean Iter.: The mean value of all iteration counts (for iterative methods).
- Computational Effort: Comments on the type of computation required for the method.

Clearly, the methods that don't need to solve linear systems of equations are much faster. The reflectance curves they create, however, may not be very realistic.

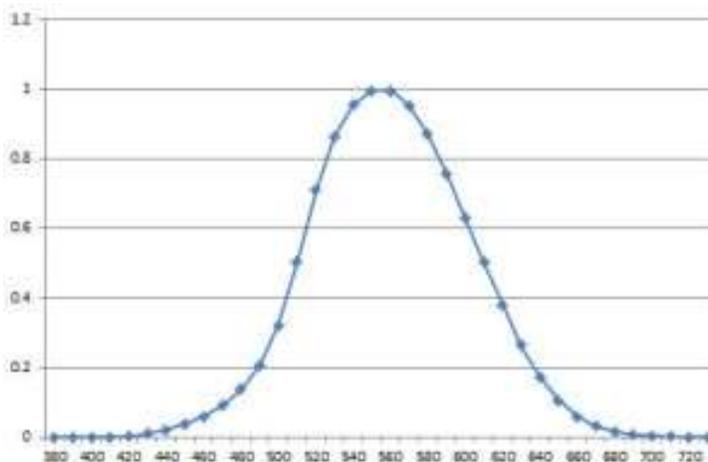
Comparison of Reflectance Curves

In this section, I compare the computed reflectance curves against two large sets of measured reflectance data. My goal is to assess how "realistic" the computed reflectance curves are in comparison to reflectances measured from real colored objects.

Munsell Color Samples

The first set comes from the [Munsell Book of Colors](#), specifically the 2007 glossy version. The Munsell system organizes colors by hue, chroma (like saturation), and value (like lightness). In 2012, Paul Centore measured 1485 different Munsell samples and [published the results here](#). I computed the sRGB values of each sample, and used those quantities to compute reflectance curves for each of the five methods described above. Of the 1485 samples, 189 of them are outside the sRGB gamut (have values <0 or >255) and were not examined in this study, leaving 1296 in-gamut samples.

When comparing reflectance curves, some parts of the curve are more important than others. As we approach both ends of the visible spectrum, adjacent to the UV and IR ranges, the human eye's sensitivity to these wavelengths rapidly decreases. This phenomenon is described by the "luminous efficiency" curve shown in the adjacent figure. As I was computing the reflectance curves, I noticed that there are often large discrepancies between the computed and measured reflectance curves at the ends of the visible spectrum. These large differences have relatively little impact on color perception, and would also have little consequence on operations performed on the computed reflectance curves, such as when modeling subtractive color mixture. Consequently, I decided to downplay these end differences when comparing the curves.



The luminous efficiency curve, which describes the relative sensitivity of the eye to spectral light across the visible spectrum.

To assess how well each computed reflectance matches the measured reflectance curve, I first subtracted one from the other and took absolute values of the difference. Then I multiplied the differences by the luminous efficiency curve. Finally, I summed all of the values to give a single measure of how well the reflectance curves match, or a "reflectance match measure" (RMM):

$$RMM = \sum_{i=380}^{730} lum_i |(\rho_i^{measured} - \rho_i^{computed})|.$$

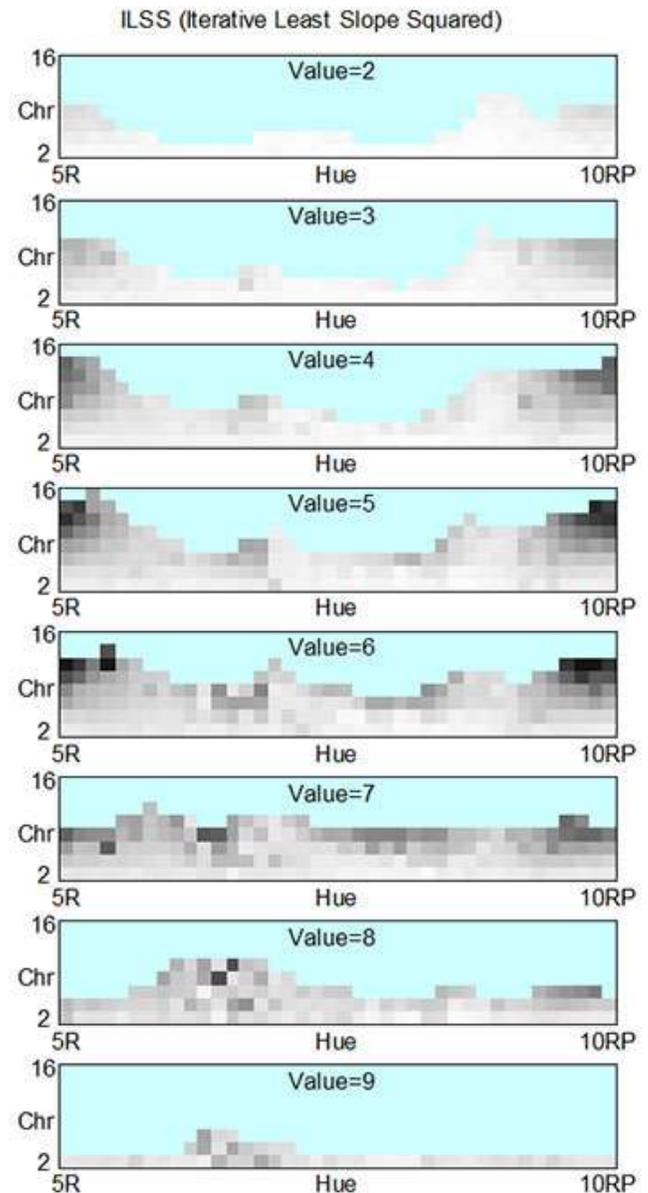
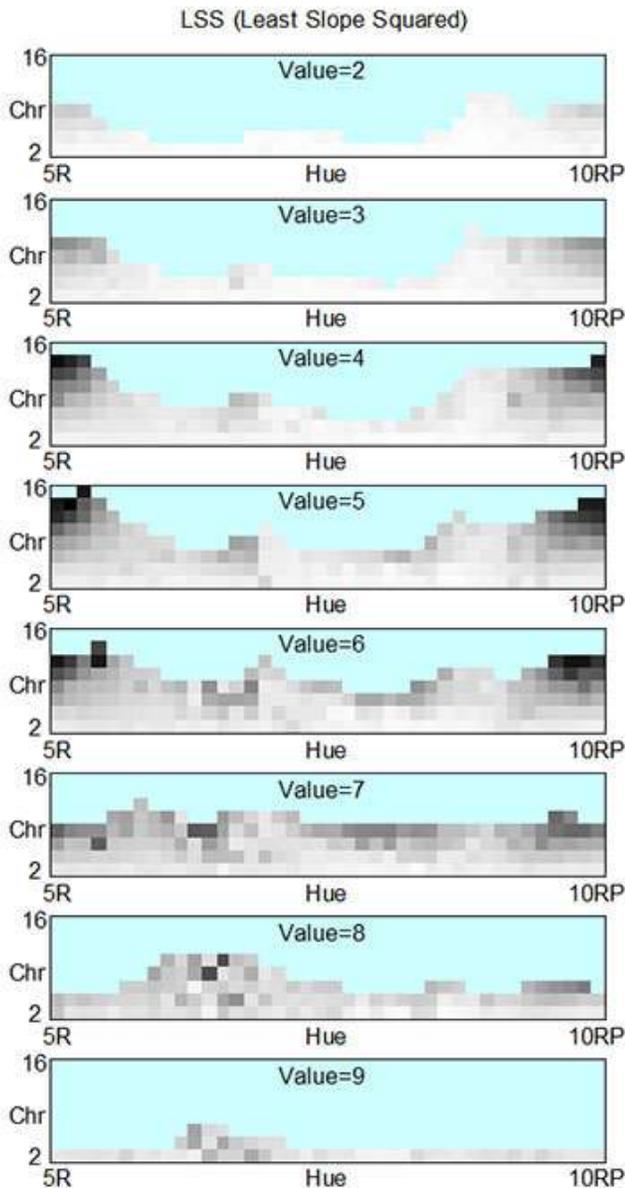
Lower values of RMM represent a better match, with zero being a perfect match of the two reflectance curves. Keep in mind that regardless of the value of RMM , the sRGB triplets of the computed and measured curves always match exactly, as would the perceived color evoked by the two reflectance curves.

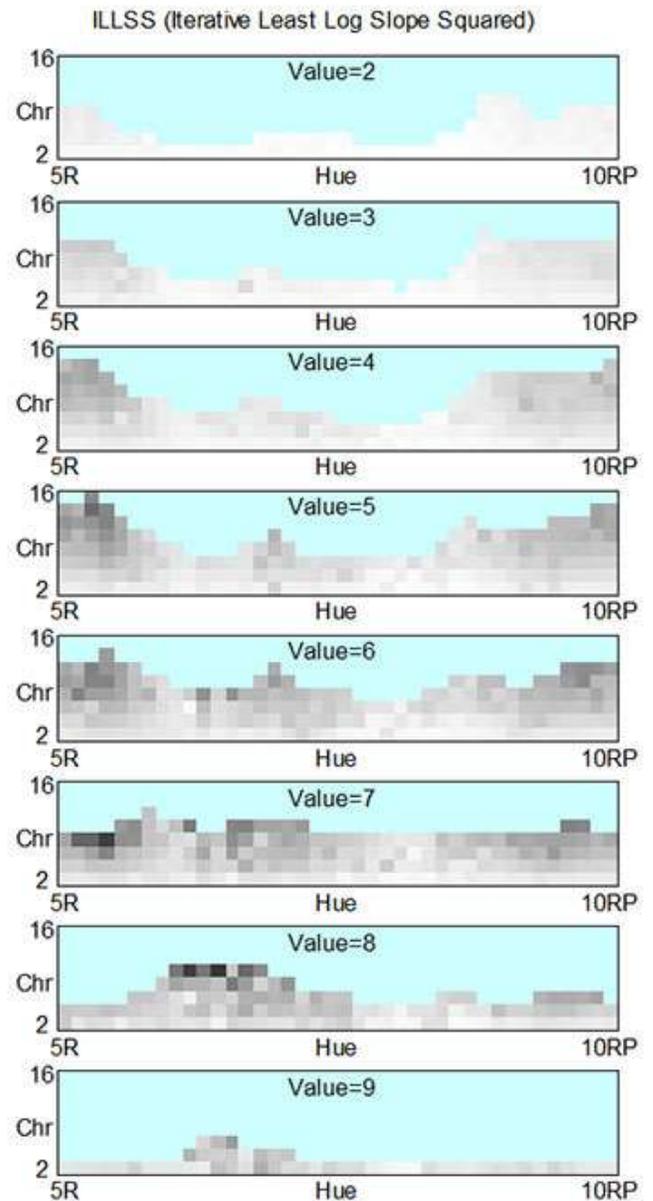
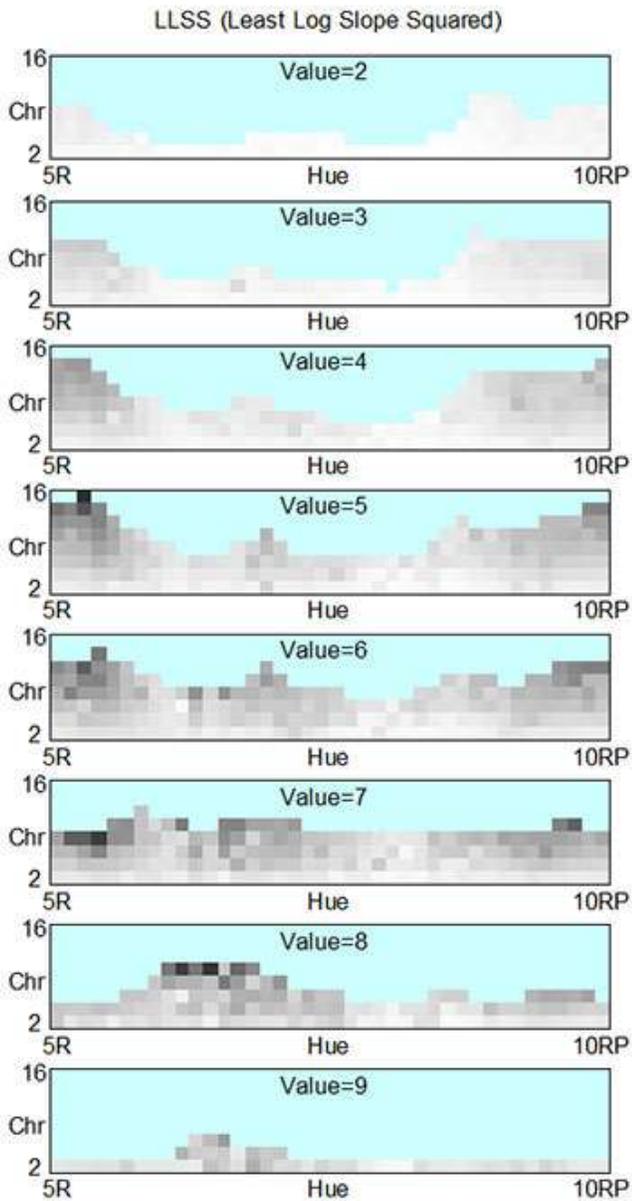
I examined the maximum and mean values of RMM for each of the five methods when applied to all 1296 Munsell samples:

Name	Max <i>RMM</i>	Mean <i>RMM</i>
LLS, Linear Least Squares	2.46	0.88
LSS, Least Slope Squared	1.11	0.17
ILSS, Iterative Least Slope Squared	1.04	0.16
LLSS, Least Log Slope Squared	0.92	0.15
ILLSS, Iterative Least Log Slope Squared	0.86	0.15

The Linear Least Squares method is clearly much worse than the others. Considering that the Least Slope Squared method requires the same computational effort as LLS, but with much better results, I decided to present the comparison figures for just the last four methods.

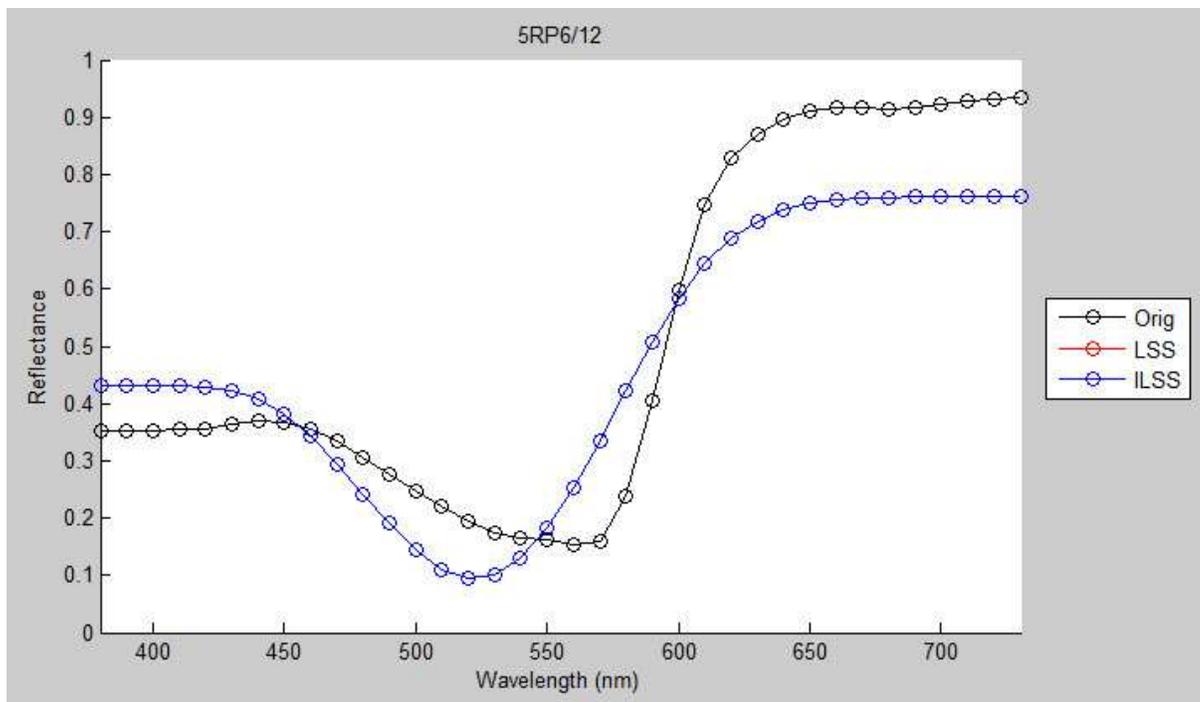
The following figures compare the four methods. Each gray square represents one Munsell sample, and the shade of gray indicates the corresponding *RMM* value. The shade of gray is linearly mapped so that it is white for $RMM = 0$ and black for $RMM = 1.11$, the maximum value *RMM* for all four methods.





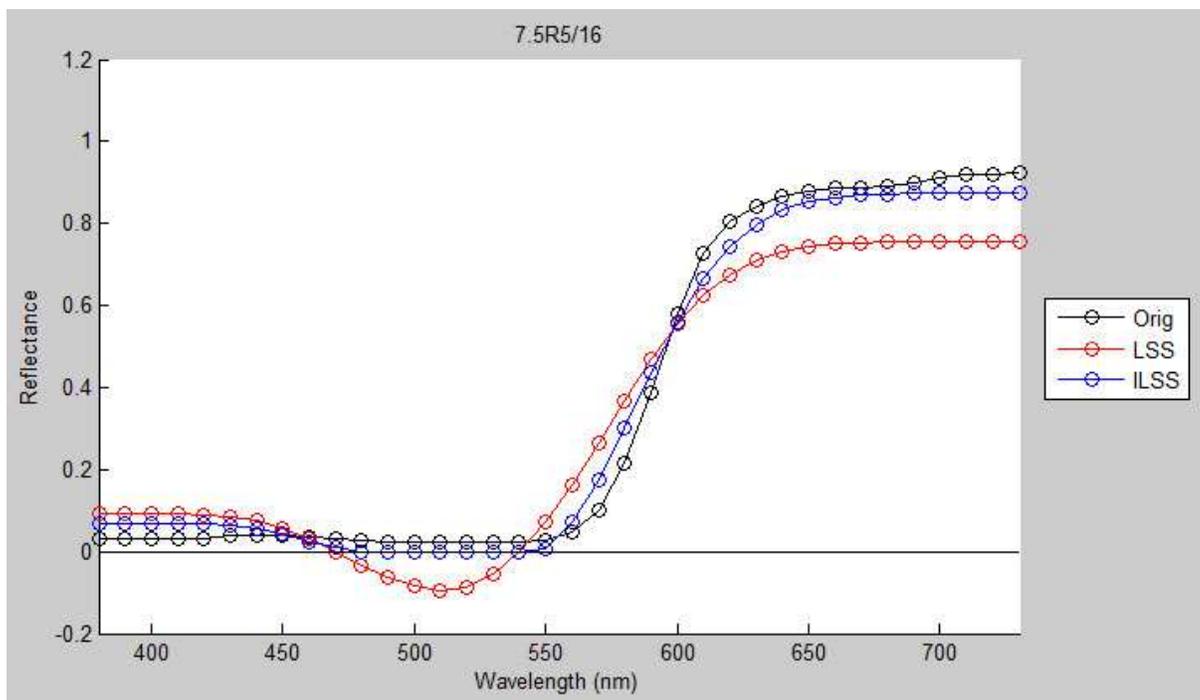
Overall, the log version of the algorithms do a little better. The worst matches with non-log-based LSS/ILSS take place with the highly chromatic reds and purples, and sometimes with the bright yellows. With log-based LLSS/ILLSS, the worst matches tend to be in the bright yellows, and sometimes in the chromatic reds and oranges. In both pairs, the iterative version clips the reflectance curves between 0 and 1, and usually reduces the mismatch somewhat, but not by very much.

Here are some examples of LSS and ILSS with high RMM s:

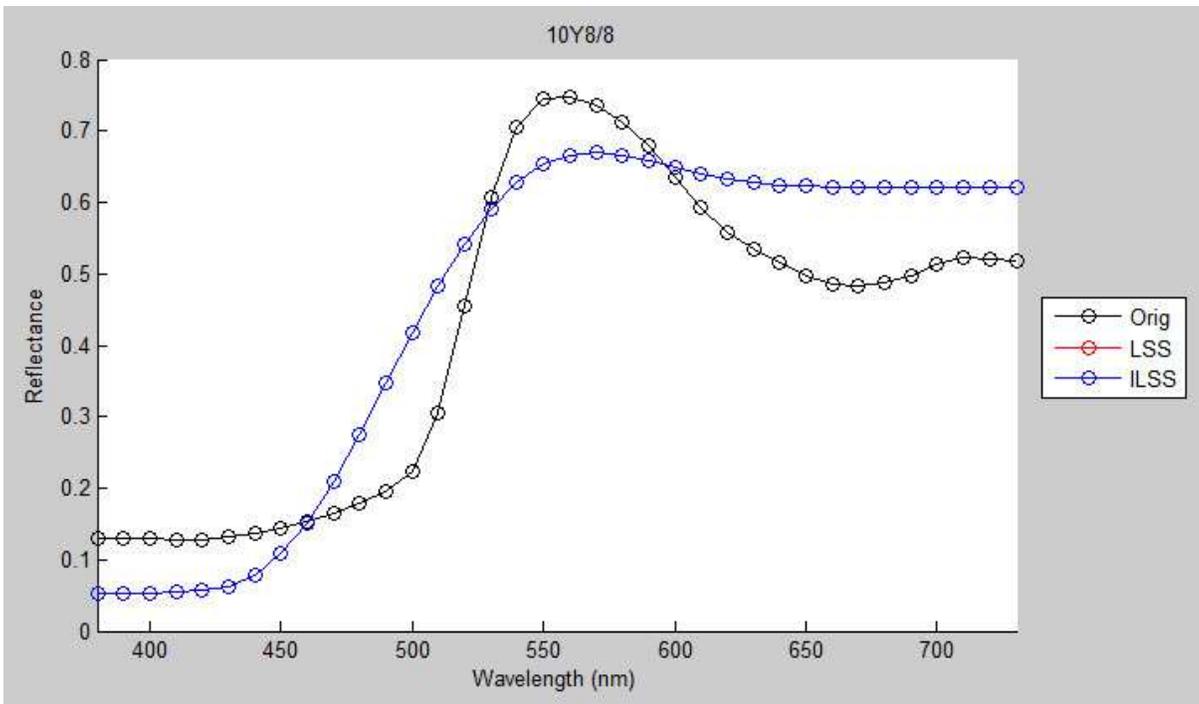


This is typical of the mismatch with chromatic reds and purples. This one (Hue=5RP, Value=6, Chroma=12) has an RMM of 1.04. The curve for LSS is not visible because it is directly behind the ILSS curve. There was no clipping needed, so the two curves are the same.

One weakness of the LSS method is that it tends to drop into the negative region for chromatic red colors. In these cases, the clipping provided by the ILSS method greatly improves the match. The following example has an LSS RMM of 1.01 and an improved ILSS RMM of 0.40:

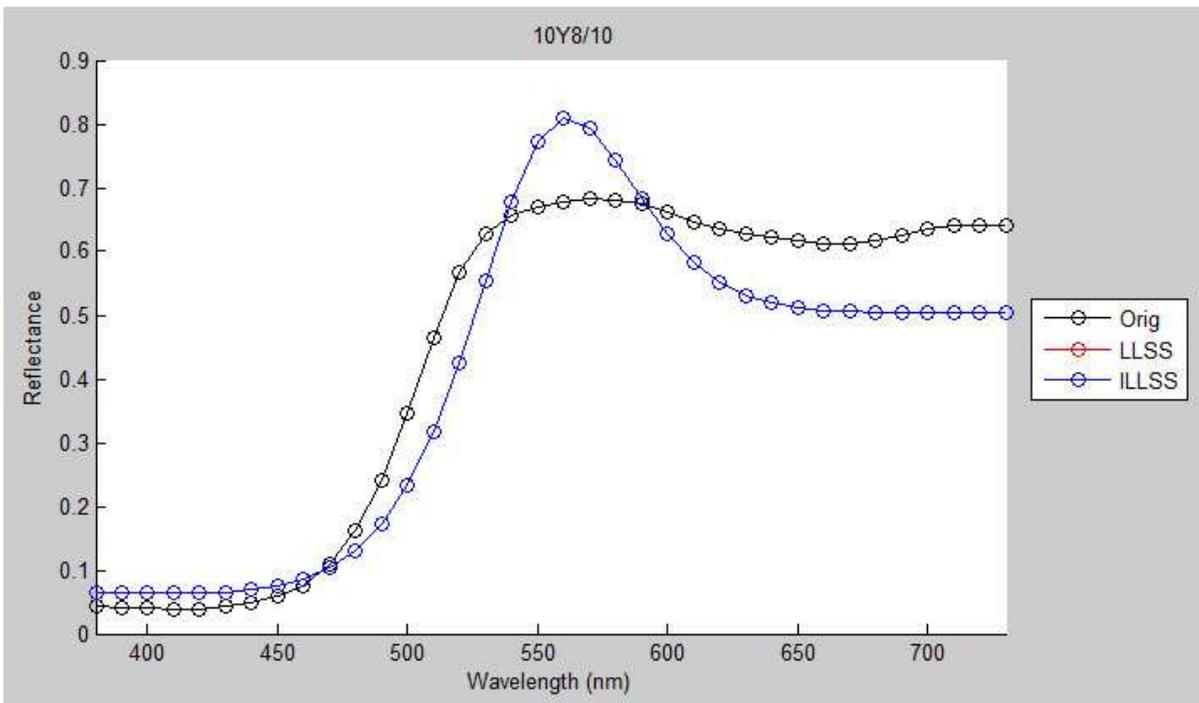


The following is a typical mismatch for LSS/ILSS in the yellow region:



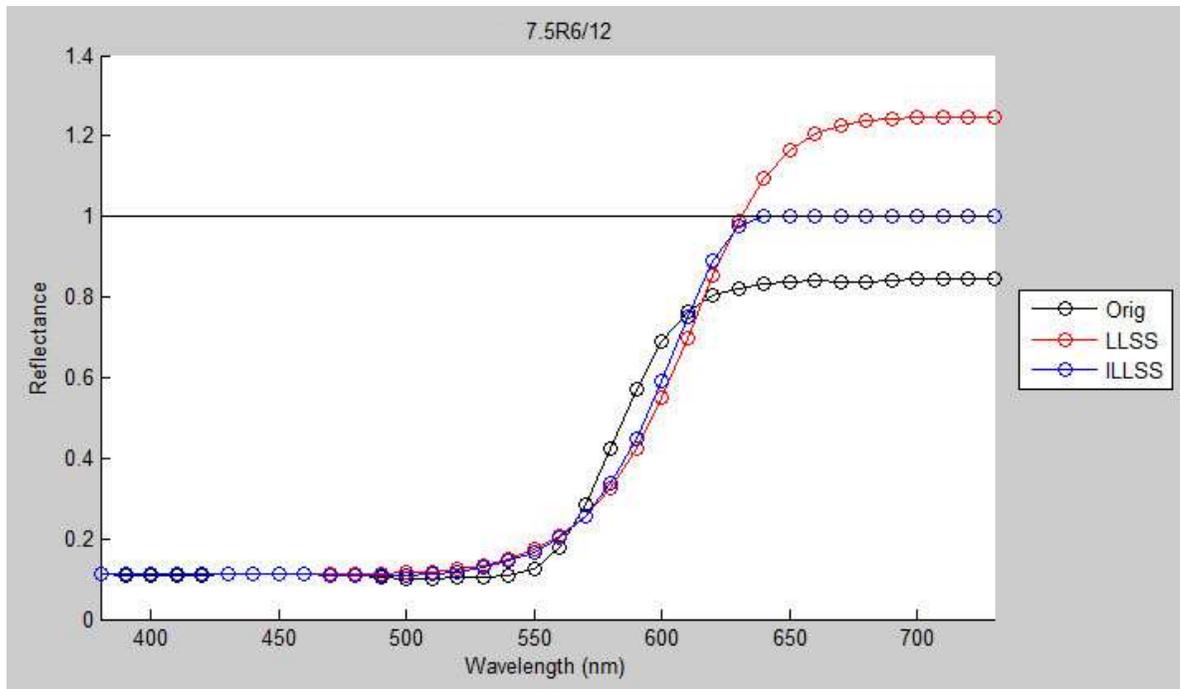
Generally, when LSS/ILSS has a bad match, it is because it oscillates too little.

Here are some examples of log-based LLSS/ILSS with high *RMM*s:



This example has an *RMM* of 0.88, mainly because it takes advantage of the wavelengths of light that give rise to a yellow sensation, in contrast to how the pigments in this sample get the same yellow sensation from a broader mix of wavelengths.

On bright red and orange Munsell samples, the LLSS/ILLSS curves tend to shoot high on the red side:



Even though there is considerable clipping with the ILLSS version, most of it takes place in the long wavelengths, which does not help the *RMM* score as much. This is evident in how little ILLSS needs to adjust the mid-wavelength range to make up for the huge difference in the long wavelengths, while maintaining the same sRGB values.

In summary, the non-log versions (LSS/ILSS) tend to undershoot peaks and the log versions (LLSS/ILLSS) tend to overshoot them. Overall, the log versions give a somewhat better match, but at the expense of considerably more computation.

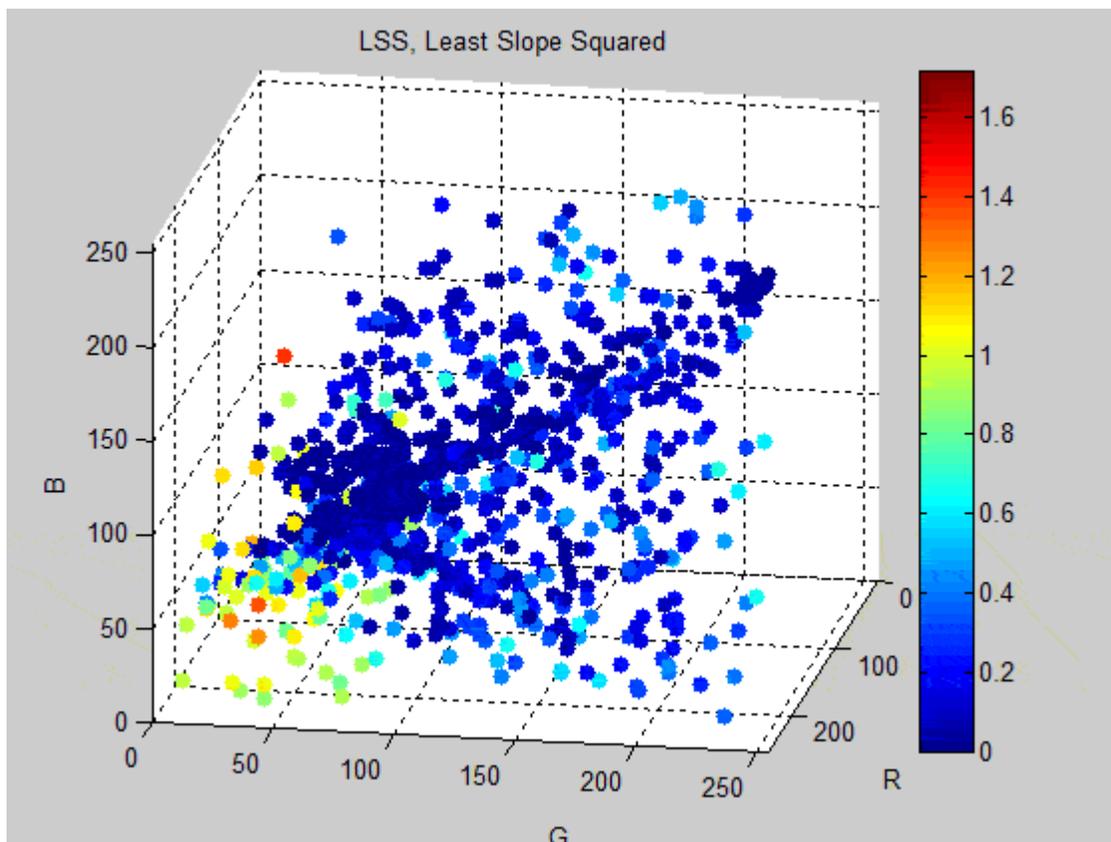
Commercial Paints and Pigments

The second large dataset of reflectance curves comes from [Zsolt Kovacs-Vajna's RS2color webpage](#). He has a database of reflectance curves for many sets of commercial paints and pigments, which can be obtained by emailing a request to him. They are grouped into the following families:

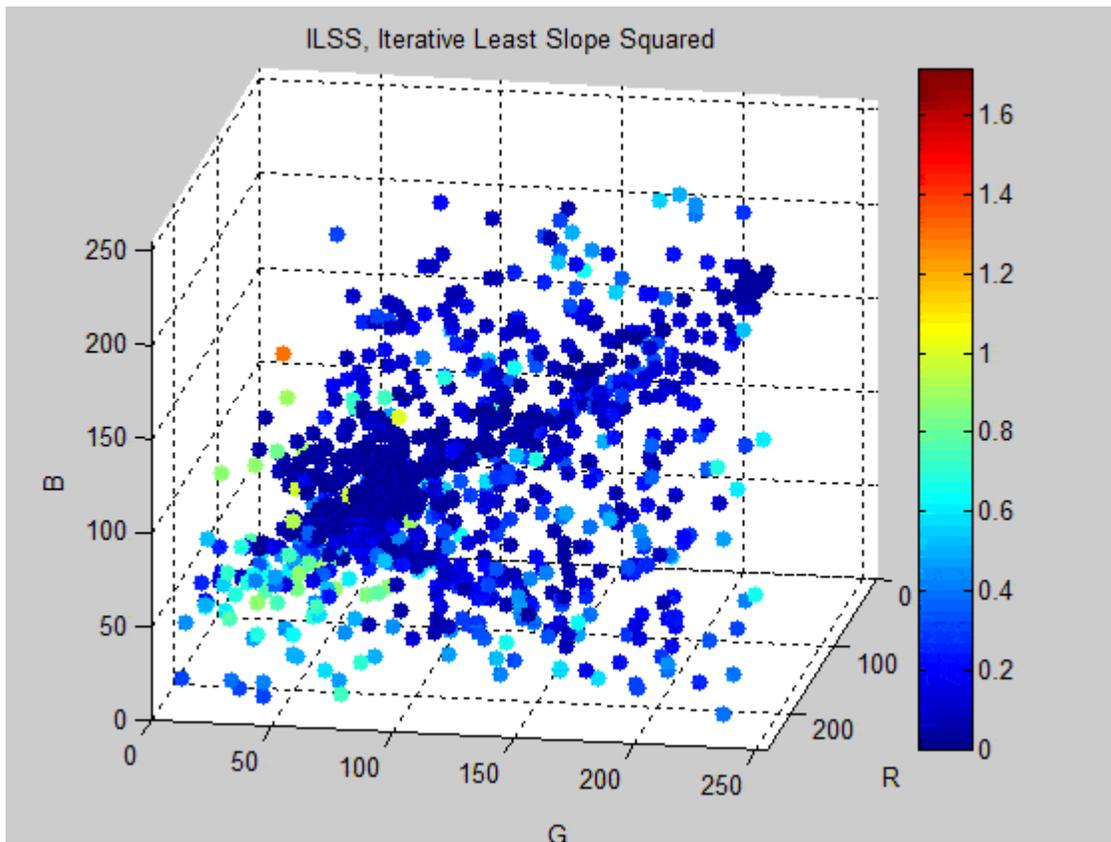
1. apaFerrario_PenColor
2. Chroma_AtelierInteractive
3. ETAC_EFX500b
4. ETAC_EFX500t
5. GamblinConservationColors
6. Golden_HB
7. Golden_OpenAcrylics
8. GretagMacbethMini
9. Holbein_Aeroflash
10. Holbein_DuoP
11. KremerHistorical
12. Liquitex_HB
13. Maimeri_Acqua
14. Maimeri_Brera
15. Maimeri_Polycolor

16. MunsellGlossy5B
17. MunsellGlossy5G
18. MunsellGlossy5P
19. MunsellGlossy5R
20. MunsellGlossy5Y
21. MunsellGlossyN
22. pigments
23. supports
24. Talens_Ecoline
25. Talens_Rembrandt
26. Talens_VanGoghH2Oil
27. whites
28. WinsorNewton_ArtAcryl
29. WinsorNewton_Artisan
30. WinsorNewton_Finity
31. WinsorNewtonHandbook

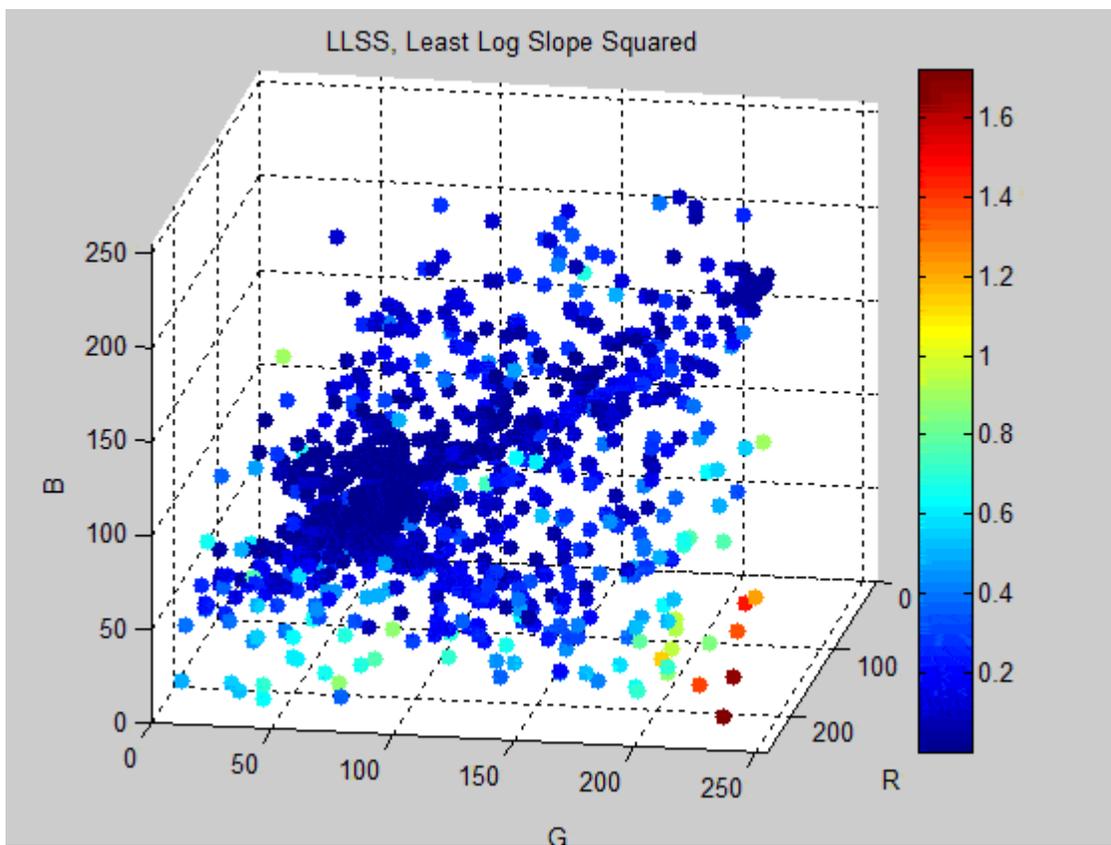
In total, there are 1493 different samples in these 31 groups. I discarded the 275 of them that were out of the sRGB gamut, and computed reflectance curves from the sRGB values of the remaining 1218 samples. I again computed *RMM* values to compare the computed curves to the actual measured curves. The following animated GIFs show the *RMM* values (color coded according to the colormap on the right) plotted in sRGB space.



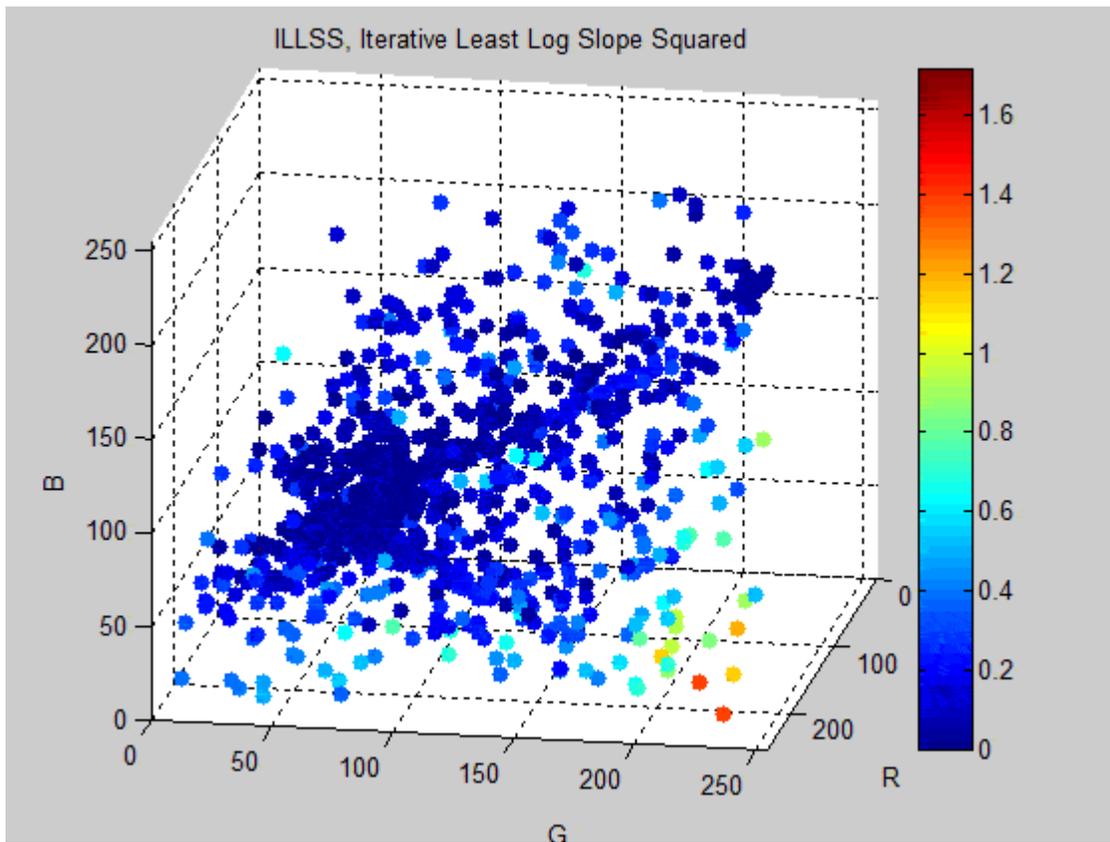
[Follow image link to view animated GIF](#)



[Follow image link to view animated GIF](#)

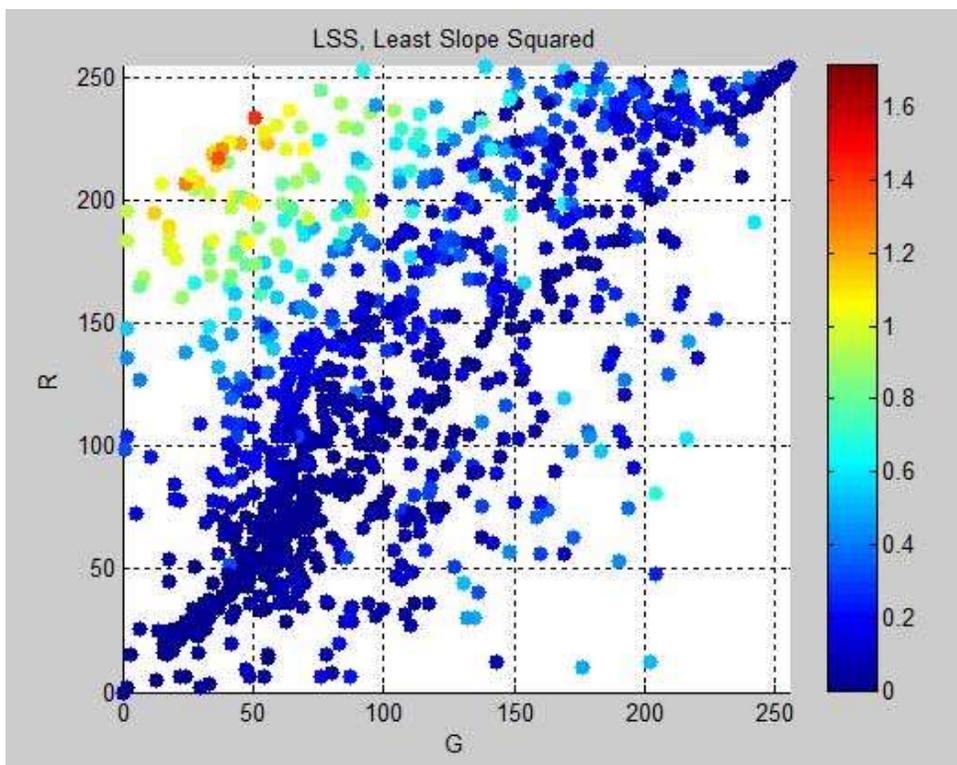


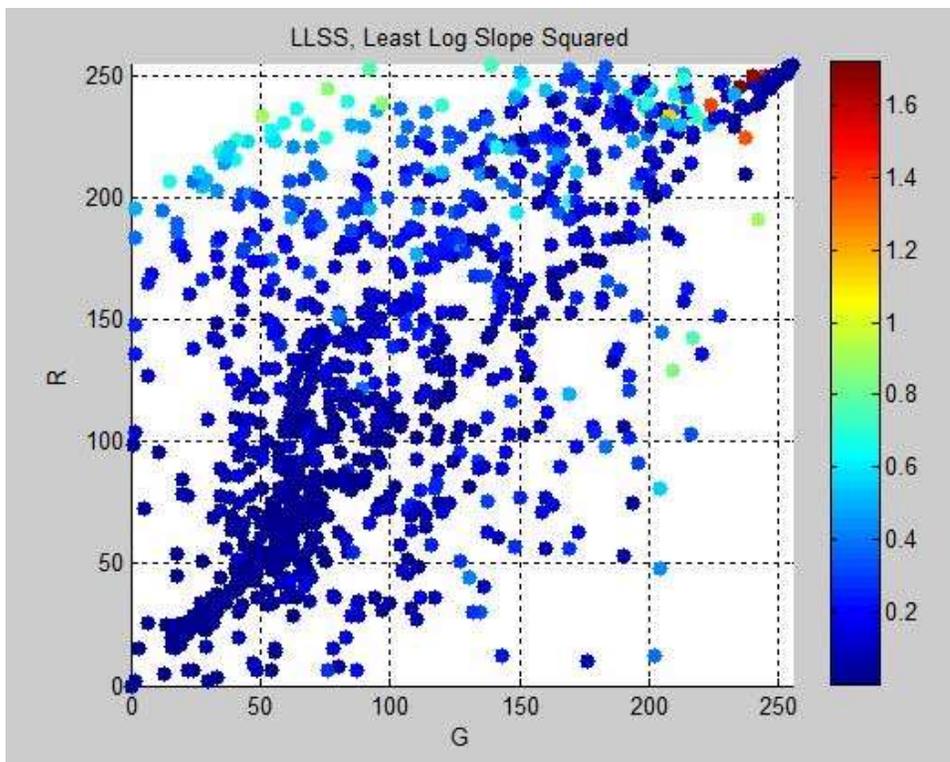
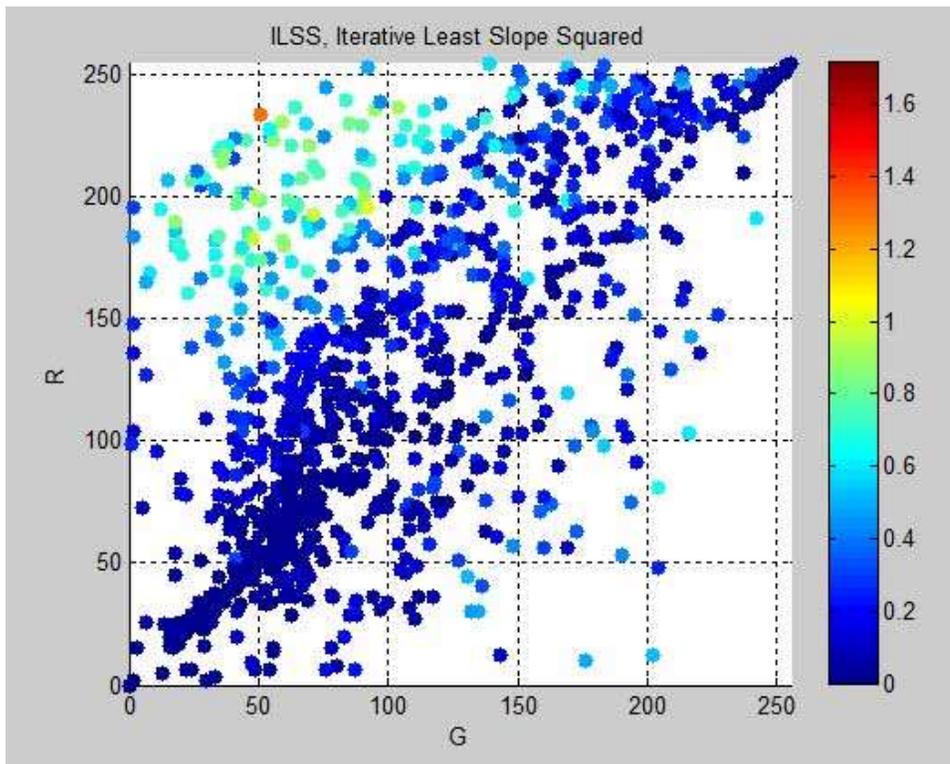
[Follow image link to view animated GIF](#)

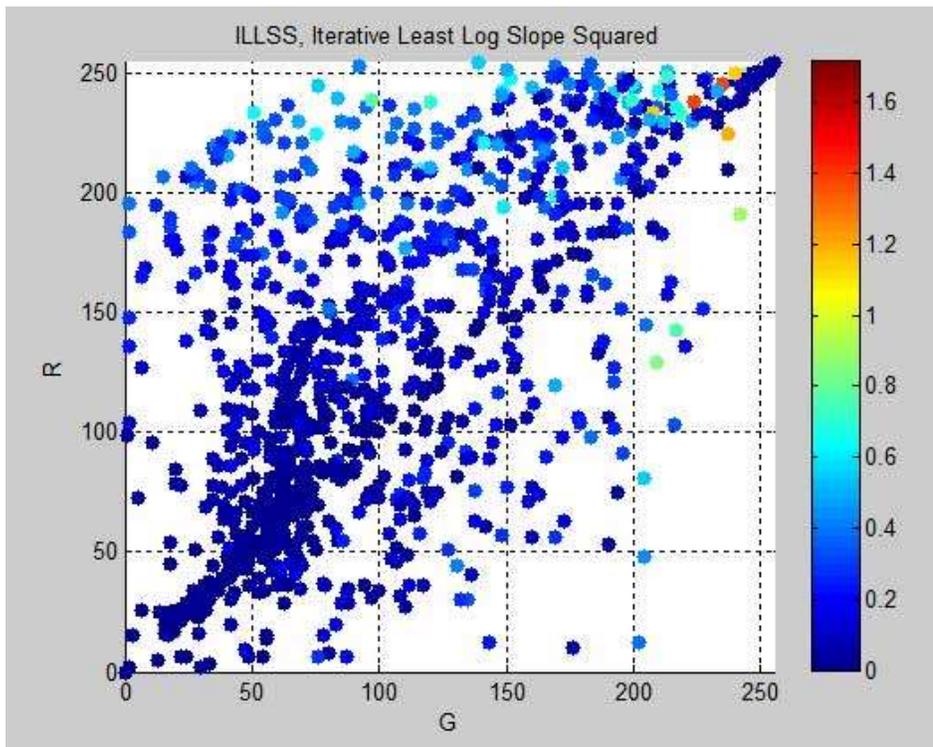


Follow image link to view animated GIF

It is apparent from these GIFs that the non-log-based methods have a fairly large region of mismatch in the red region (large R, small G and B). The log-based versions have a much smaller region of mismatch in the yellow region (large R and G, small B). The iterative version of both methods improve the matches in both cases. The relative sizes of the mismatch regions can be seen more clearly in a projection onto the R-G plane:







Conclusions

In summary, the method that best matches paint and pigment colors found commercially and in nature is the ILLSS (Iterative Least Log Slope Squared) method. It suffers, however, from very large computational requirements. If efficiency is more important, then the ILSS (Iterative Least Slope Squared) method is the preferred one. A third alternative that is midway between the match quality and computing effort is LLSS (Least Log Slope Squared), but be aware that some reflectance curves can end up with values >1 . I would not recommend the LSS (Least Slope Squared) method, despite its spectacular computational efficiency, because it can give reflectance curves with physically meaningless negative values. The following table summarizes the three suggested methods:

Algorithm Name	Computational Effort	Comments	Link to Matlab Code
ILSS (Iterative Least Slope Squared)	Relatively little.	Very fast, but tends to undershoot reflectance curve peaks, especially for bright red and purple colors. Always returns reflectance values in the range 0-1.	link
LLSS (Least Log Slope Squared)	About 12 times that of ILSS.	Better quality matches overall, but tends to overshoot peaks in the yellow region. Some reflectance values can be >1 , especially for bright red colors.	link
ILLSS (Iterative Least Log Slope Squared)	About 20 times that of ILSS.	Best quality matches. Tends to overshoot peaks in the yellow region. Always returns reflectance values in the range 0-1.	link

Acknowledgments

This work has been a tremendous learning experience for me, and I want to thank several people who graciously posted high quality material, upon which most of my work has been based: [Bruce Lindbloom](#), [Bruce MacEvoy](#), [Zsolt Kovacs-Vajna](#), [David Briggs](#), and [Paul Centore](#). If you are interested in learning more about color theory, these five links are excellent places to start!



Generating Reflectance Curves from sRGB Triplets by [Scott Allen Burns](#) is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#).