Optimizing Traffic Signal Timing via Simplex Method

MTH382: Linear Programming & Optimization Project Report

Haziq Khalid @100601

November 23, 2025

Abstract

This report presents a method for optimizing traffic signal timing by minimizing vehicle delay at an intersection involving a North-South and East-West road. While traffic delay is typically modeled using the non-linear Webster formula, this project utilizes The Simplex Method by approximating the non-linear delay surface with piecewise linear tangent planes. The results demonstrate that this linearization technique successfully identifies the optimal solution with high precision, achieving a solution that matches the true non-linear minimum.

1 Background and History

We will begin this report by providing some background information on the simplex method. Linear programming and the simplex method were born out of wartime logistics. In 1947, George B. Dantzig of the U.S. Army Air Forces devised the simplex method while working on resource allocation problems [2]. Dantzig coined the term "programming" for scheduling, thus optimizing under linear constraints became known as *linear programming* [3]. The development of the simplex method took roughly a year [2]. Dantzig's insight was that many operational "ground rules" could be captured as linear inequalities plus an objective function to maximize [2]. His method transformed previously intractable problems with hundreds of variables into computations solvable by computers [2]. For these and other accomplishments in operations research, Dantzig has been called the "father of linear programming" [2]. Notably, he received the John von Neumann Theory Prize in 1975 and the National Medal of Science the same year [2], recognizing that his simplex method had founded a major field of applied mathematics.

Early work by others (e.g. Fulkerson, Balinski) generalized the simplex to network problems, but Dantzig's algorithm was the key breakthrough. Prior to this, Leonid Kantorovich had studied similar optimization in the Soviet Union during World War II, and Wassily Leontief's input-output models hinted at linear constraints [3]. However, it was Dantzig who formalized the general algorithm for LP. By the 1950s and 60s, simplex became central to operations research and management science, underpinning airline crew scheduling, manufacturing, military planning, and many other optimization tasks [3] [1].

1.1 The Simplex Method

The Simplex Method is an iterative procedure designed to solve Linear Programming (LP) problems. It operates on the principle that if an optimal solution exists, it will lie at an extreme point (corner) of the feasible region defined by the constraints.

1.1.1 Standard Form and Tableau Initialization

The first step is to convert the LP into **Standard Form** by ensuring all constraints are equalities and all variables are non-negative ($x_i \geq 0$). This involves introducing auxiliary variables:

- **Slack Variables (s_i):** Added to $\leq$ constraints.
- **Surplus Variables (u_i):** Subtracted from $\geq$ constraints.
- **Artificial Variables (a_i):** Added to $\geq$ constraints to create an initial Basic Feasible Solution (BFS). These are typically penalized in the objective function using the Two-Phase or Big M method.

The problem, once in Standard Form, is represented in the **Simplex Tableau** where the constraints form an identity matrix corresponding to the set of basic variables (B). The objective row (bottom row), holds the coefficients (reduced costs) $c_j - z_j$ for the non-basic variables.

1.1.2 The Iterative Pivot Operation

Each iteration, or pivot, replaces one non-basic variable (the entering variable) with one basic variable (the leaving variable) to find a better adjacent vertex. The rules for selecting the pivot element (a_{ij}) are as follows:

1. **Entering Variable Selection (Pivot Column):** Identify the non-basic variable whose entry in the objective row is the most negative. This column j is the pivot column, as it offers the greatest potential rate of improvement to the objective function per unit increase of the variable.

2. **Leaving Variable Selection (Pivot Row):** The minimum ratio test determines which basic variable must leave to maintain feasibility. For each row i where the pivot column entry $a_{ij} > 0$ and $b_i > 0$, calculate the ratio:
$$\text{Ratio} = \frac{b_i}{a_{ij}}$$
The row i corresponding to the minimum non-negative ratio is the pivot row. This selects the constraint that first limits the increase of the entering variable x_j.

3. **Pivot Operation:** Elementary row operations are performed to transform the pivot element a_{ij} into a 1 and all other entries in the pivot column into 0's. This is an algebraic transformation that results in a new, improved BFS where the objective value is strictly better (assuming non-cyclic).

1.1.3 Optimality and Termination Criterion

The Simplex method continues to pivot until the **Optimality Condition** is met. The procedure terminates when:

- **Optimality:** There are no more negative entries in the objective row. At this point, increasing any non-basic variable would only worsen the objective value (or keep it the same), meaning the current BFS is optimal.

- **Unboundedness:** If, in a pivot column selected by the most negative reduced cost, all entries a_{ij} are non-positive (≤ 0), the problem is unbounded. This means the entering variable can be increased indefinitely without violating constraints, leading to an infinitely improved objective value.

This method guarantees that the algorithm will find the optimal solution in a finite number of steps (especially when rules like Bland's rule are used to prevent cycling in cyclic cases).

2 Traffic Signal Optimization

Traffic signal optimization is a critical problem in urban planning. The objective is to allocate "green time" to conflicting traffic streams in a way that minimizes the total time drivers spend waiting at the intersection.

We consider a standard four-way intersection with two phases:

- **Phase 1:** North-South (NS) traffic moves, East-West (EW) is stopped.

- **Phase 2:** East-West (EW) traffic moves; North-South (NS) is stopped.

The challenge lies in the nature of traffic delay. The standard model for estimating delay, Webster's Formula, is **non-linear** and convex. Traditional Linear Programming (LP) solvers, such as the Simplex method, cannot directly optimize curved functions. To overcome this, we will implement an approach that approximates the curved delay function using a set of linear constraints (tangent planes), allowing us to use the Simplex method to solve a complex non-linear problem.

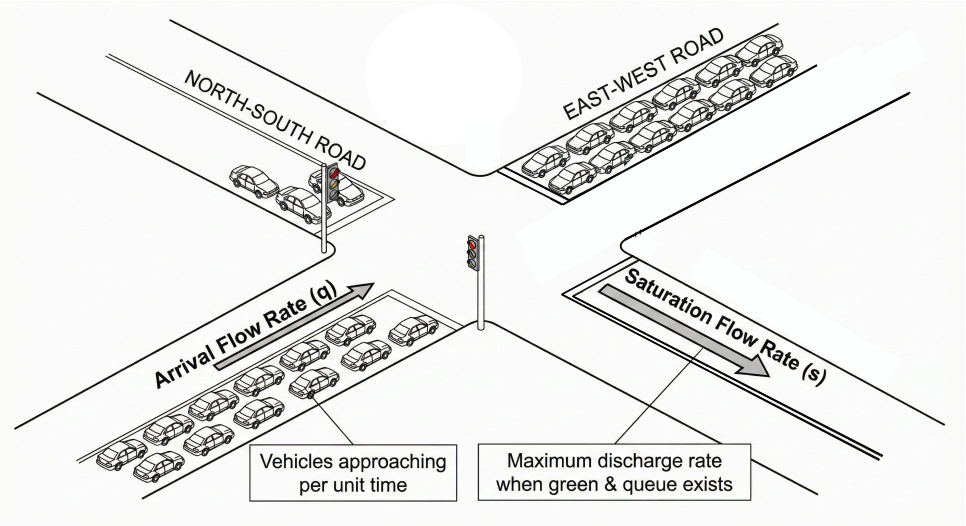


Figure 1: Visualization of the intersection we wish to optimize traffic signaling on

2.1 Webster's Delay Formula

To quantify the efficiency of the intersection, we use the classical Webster delay formula. This formula estimates the average delay per vehicle (d) based on the cycle length and the effective green time. The formula is given by:

$$d(g, C) = \frac{C(1 - g/C)^2}{2(1 - q/s)} \tag{1}$$

Where the variables are:

- C: Cycle length (total time for one complete sequence of signals).
- g: Effective green time allocated to the specific direction.
- q: Arrival flow rate (vehicles arriving per hour).
- s: Saturation flow rate (maximum capacity of the lane).

We aim to optimize C, g_{NS}, and g_{EW} while keeping the arrival flow (q) and saturation flow (s) fixed for both directions. For our problem, we assign the following values for arrival flow (q) and saturation flow (s):

- $q_{NS} = 600$
- $s_{NS} = 1800$
- $q_{EW} = 500$
- $s_{EW} = 1800$
- $g_{min} = 10$
- $C_{min} = 10$

Minimum times for g and C are included for safety purposes.

2.2 The Objective Function

Our goal is to minimize the **Total Delay** for the entire intersection. This is calculated as the sum of the delay for both directions, weighted by the number of cars arriving in each direction.

Let q_{NS}, z_{NS} and q_{EW}, z_{EW} be variables representing the number of vehicles and the delay per vehicle for North-South and East-West directions respectively. Our Linear Programming objective function is:

$$\text{Minimize } J = q_{NS} \cdot z_{NS} + q_{EW} \cdot z_{EW} \tag{2}$$

3 Linearization Methodology

Since the Webster delay formula is convex (bowl-shaped), it cannot be directly optimized using standard Linear Programming solvers, which require linear constraints. To overcome this, we utilize *Outer Linearization* (also known as the Cutting Plane method).

The core concept is to approximate the smooth, non-linear delay surface by constructing a "polyhedral envelope" underneath it. This envelope is formed by the intersection of many flat tangent planes. We use decision variables z_{NS} and z_{EW} representing the delay for each direction and constrain them to be greater than or equal to the height of their respective tangent planes.

3.1 2D Sampling

The delay in a particular direction is not directly linked to the time the light is green in the opposite direction, instead these are linked in the simplex by C (Cycle Length). This allows us to generate approximation constraints for the North-South and East-West directions separately using 2D sample pairs (g_i, C_i). These two independent samples are then coupled together in the Simplex method via the Cycle Length (C).

3.2 Tangent Plane Derivation

For a given direction (e.g., North-South), we verify sample points (g_i, C_i) within the feasible domain. The equation of a plane tangent to the delay surface at point (g_i, C_i) is given by the first-order Taylor expansion:

$$z_{NS} \geq D(g_i, C_i) + \frac{\partial D}{\partial g_{NS}}(g_{NS} - g_i) + \frac{\partial D}{\partial C}(C - C_i) \tag{3}$$

Since the delay z_{NS} is independent of g_{EW}, the partial derivative with respect to g_{EW} is zero. To implement this in the Simplex method (which accepts constraints in the form $Ax \leq b$), we rearrange the terms into a row vector format.

$$\underbrace{\left[\frac{\partial D}{\partial g_{NS}}\right]}_{m_g} \cdot g_{NS} + \mathbf{0} \cdot g_{EW} + \underbrace{\left[\frac{\partial D}{\partial C}\right]}_{m_c} \cdot C - 1 \cdot z_{NS} \leq \underbrace{-\left(D(g_i, C_i) - m_g g_i - m_c C_i\right)}_{k_i} \tag{4}$$

Where k_i is the derived constant for the right hand side. This process is repeated for the East-West direction, creating a second set of constraints where g_{NS} has a zero coefficient.

3.3 Example Constraint Generation

To illustrate, consider a single sample point for the North-South direction:

- **Sampled Point:** Cycle $C_0 = 90$s, Green $g_{NS} = 45$s.
- **Calculated Delay (D_0):** 30.0 seconds.
- **Calculated Slopes:** $m_g = -0.5$ (delay decreases as green increases), $m_c = 0.2$ (delay increases as cycle increases).

The plane equation becomes:

$$z_{NS} \geq 30 - 0.5(g_{NS} - 45) + 0.2(C - 90) \tag{5}$$

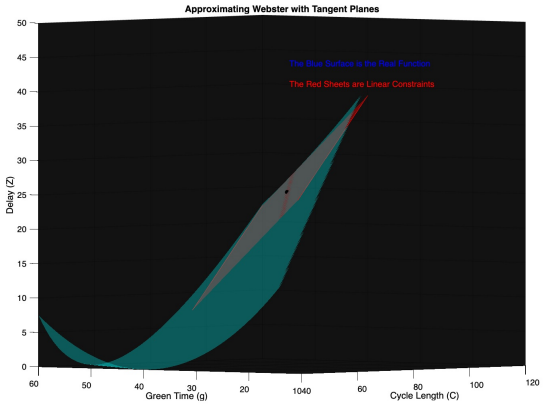
Expanding and rearranging:

$$\begin{aligned} z_{NS} &\geq 30 - 0.5g_{NS} + 22.5 + 0.2C - 18 \\ 0.5g_{NS} - 0.2C - z_{NS} &\leq -34.5 \end{aligned}$$

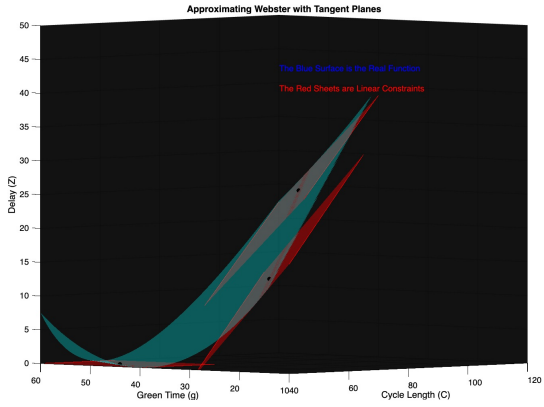
This inequality is added as a row to the constraint matrix A. Simplex satisfies this constraint (and many others) simultaneously for both directions, effectively optimizing the trade-off between the two separable delay functions.

3.4 Visualization of Approximation Density

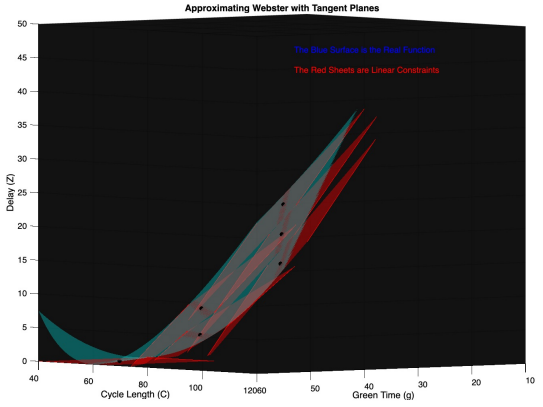
The accuracy of this method depends on the density of the constraints. Figure 2 illustrates how increasing the number of tangent planes creates a tighter "envelope" around the true non-linear function.



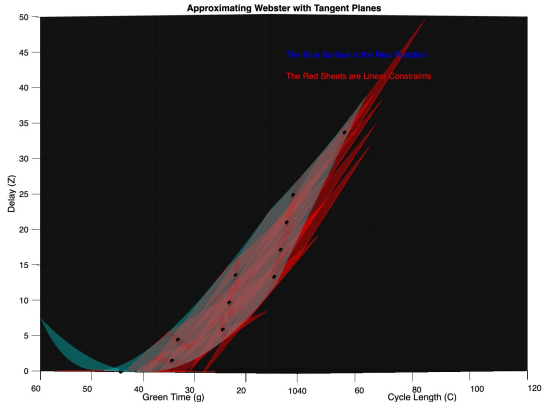
(a) 1 Tangent plane approximation



(b) 3 Tangent planes approximation



(c) 9 Tangent planes approximation



(d) 16 Tangent planes approximation

Figure 2: Visualization of tangent planes (in red) approximating Webster formula of North-South direction (in blue)

3.5 How the Simplex Method finds optimal solution with the planes

The Simplex algorithm seeks the optimal corner of the feasible region defined by the intersection of the planes. These vertices naturally occur *between* sampled points.

- If we sample at $C = 90$ and $C = 100$, we generate two planes.
- The Simplex method will slide along the surface of the $C = 90$ plane until it hits the $C = 100$ plane.
- The resulting corner might exist at $C = 94.5$.

3.6 The Simplex Formulation

The final problem formulation is as follows:

$$\textbf{Minimize } J = q_{NS} \cdot z_{NS} + q_{EW} \cdot z_{EW} \tag{6}$$

Subject to:

1. **Cycle Length:** The green times must equal the total cycle length.

$$g_{NS} + g_{EW} - C = 0 \tag{7}$$

2. **Webster Approximations:** A set of inequality constraints derived from the tangent planes (generated via MATLAB sampling loop).

$$A \cdot X \leq b \tag{8}$$

3. **Bounds:** $0 \leq g_{min} \leq g$, $0 \leq C_{min} \leq C$, and $z \geq 0$.

4 Results

The optimization was performed using MATLAB (All the code is available in the Appendix).

4.1 Optimal Signal Strategy

Table 1 compares the performance of the Simplex method with respect to number of constraints.

Number of Constraints	**Cycle Length (C)**	**Green NS (g_{NS})**	**Green EW (g_{EW})**	**Total Delay**
1	120.0	110.0	10.0	9.8
15	40.0	23.0	17.0	2.18
120	40.0	22.0	18.0	2.18
4120	40.0	22.6	17.4	2.17

Table 1: Comparison of Total Delay (measured in combined hours waited by vehicles per hour) with number of tangent plane linear constraints

4.2 Verification and Error Analysis

To verify the accuracy of the piecewise linear approximation, we compared the delay predicted by the Simplex method against the actual value from the non-linear Webster formula at the optimal coordinates.

Number of Constraints	**Simplex Delay Prediction**	**Actual Webster Delay**
1	0.0	9.8
15	2.04	2.18
120	2.15	2.18
4120	2.17	2.17

Table 2: Comparison of Simplex delay prediction with actual Webster delay at different constraints

In most cases, the Simplex method underestimates the delay. This is an expected consequence of using flat planes to approximate a convex bowl, the linear planes cut underneath the curve. However, we do see the simplex method better map the delay function as the number of constraints jump dramatically.

4.3 Accuracy Against True Minimum

To confirm the accuracy of our methodology, we benchmarked the result against the exact minimum of the webster (calculated analytically).

Parameter	**(Best) Simplex Result**	**Exact**	**Difference**
Cycle Length	40.0000	40.0000	0.0000
Green NS	22.6010	22.6072	0.0062
Green EW	17.3990	17.3928	0.0062

Table 3: Benchmarking Simplex vs. Non-Linear Solver

As shown in Table 3, the Simplex method found the exact optimal Cycle Length and was within 0.0062 seconds of the exact Green split. This confirms that the linearization approach can be an highly effective method for optimizing traffic signal timings.

5 Conclusion

This project demonstrated that non-linear traffic optimization problems can be successfully solved using Linear Programming. By approximating the Webster Delay formula with over 4,000 linear constraints, we transformed a complex convex problem into a standard Simplex problem. The resulting model provides an optimal signal strategy that matches the theoretical non-linear optimum with high precision.

References

[1] LLC Gurobi Optimization. Optimization with linear programming: Examples, tips, and use cases, 2021.

[2] William L. Hosch. George dantzig. Encyclopædia Britannica Online, 2025.

[3] Wikipedia contributors. Simplex algorithm. Wikipedia, The Free Encyclopedia, 2023.

6 Appendix

6.1 Main Optimization Code

This code uses 4120 constraints (change lines 26 and 27 to change number of constraints)

```
clear; clc; close all;

% Intersection parameters
L = 0;
sNS = 1800;
qNS = 600;
sEW = 1800;
qEW = 500;
Cmin = 40;
Cmax = 120;
gmin = 10;

% Symbolic math for Webster formula
syms g c q s real
x = (q * c) / (s * g);
webster = (c * (1 - g/c)^2) / (2 * (1 - q/s)) + (x^2) / (2 * q * (1 - x));
dg = diff(webster, g);
dc = diff(webster, c);
D = matlabFunction(webster, 'Vars', {g, c, q, s});
mg = matlabFunction(dg, 'Vars', {g, c, q, s});
mc = matlabFunction(dc, 'Vars', {g, c, q, s});
disp('Symbolic math setup complete.');

% Generating Linear Constraints (A*x <= b)
A = []; b = [];
Cs = linspace(Cmin, Cmax, 20);
Sp = 0.1:0.01:1.8;
i = 0;
for Cval = Cs
    for s = Sp
        % NS Direction
        gval = Cval * s;
        xNS = (qNS * Cval) / (sNS * gval);
        if xNS < 0.95 && gval >= gmin
            D0 = D(gval, Cval, qNS, sNS);
            m_g = mg(gval, Cval, qNS, sNS);
            m_c = mc(gval, Cval, qNS, sNS);
            row = [m_g, 0, m_c, -1, 0];
            rhs = -(D0 - m_g*gval - m_c*Cval);
            A = [A; row];
            b = [b; rhs];
            i = i + 1;
        end

        % EW Direction
        gval_ew = Cval * (1-s) - L;
        xEW = (qEW * Cval) / (sEW * gval_ew);
        if xEW < 0.95 && gval_ew >= gmin
            D0 = D(gval_ew, Cval, qEW, sEW);
            m_g = mg(gval_ew, Cval, qEW, sEW);
```

```
            m_c = mc(gval_ew, Cval, qEW, sEW);
            row = [0, m_g, m_c, 0, -1];
            rhs = -(D0 - m_g*gval_ew - m_c*Cval);
            A = [A; row];
            b = [b; rhs];
            i = i + 1;
        end
    end
end
disp(['Generated ', num2str(i), ' constraints.']);

% Simplex (linprog) Setup
% X = [gNS; gEW; C; zNS; zEW]
f = [0; 0; 0; qNS; qEW];
Aeq = [1, 1, -1, 0, 0];
beq = -L;
lb = [gmin; gmin; Cmin; 0; 0];
ub = [Cmax; Cmax; Cmax; Inf; Inf];
opt = optimoptions('linprog','Display','off');

% Run Optimization
[x, val, flag] = linprog(f, A, b, Aeq, beq, lb, ub, opt);

% Display Results
gNS = x(1);
gEW = x(2);
C_opt = x(3);
zNS = x(4);
zEW = x(5);
fprintf('\n--- Simplex Results ---\n');
fprintf('Optimal C: %.2f s\n', C_opt);
fprintf('Optimal gNS: %.4f s\n', gNS);
fprintf('Optimal gEW: %.4f s\n', gEW);
fprintf('Total Delay: %.2f h/h\n', val/3600);

% Error Check vs Real Curve
realZNS = D(gNS, C_opt, qNS, sNS);
fprintf('NS Delay Error: %.4f sec\n', realZNS - zNS);

% Comparison with real minimum
fun = @(x_nl) qNS * D(x_nl(1), x_nl(3), qNS, sNS) + qEW * D(x_nl(2), x_nl(3), qEW,
    sEW);
Aeq_nl = [1, 1, -1]; beq_nl = -L;
lb_nl = [gmin, gmin, Cmin];
ub_nl = [Inf, Inf, Cmax];
x0 = [40, 40, 90];
opt_nl = optimoptions('fmincon', 'Display', 'off');
x_exact = fmincon(fun, x0, [], [], Aeq_nl, beq_nl, lb_nl, ub_nl, [], opt_nl);

fprintf('\n--- Exact vs Simplex Comparison ---\n');
fprintf('Solver    | C (s) | gNS (s) | gEW (s)\n');
fprintf('----------|-------|---------|--------\n');
fprintf('Simplex   | %5.2f | %7.4f | %7.4f\n', C_opt, gNS, gEW);
fprintf('fmincon   | %5.2f | %7.4f | %7.4f\n', x_exact(3), x_exact(1), x_exact(2));
```

6.2 Tangent Plane Visualization Code

This code plots 16 tangent planes against the Webster formula.

```
clear; clc; close all;

% num of tangent planes = (density x density)
density = 4;
```

```
5
6
7  s = 1800;
8  q = 600;
9  Cmin = 40;
10 Cmax = 120;
11 gmin = 10;
12
13
14 syms g c real
15 x = (q * c) / (s * g);
16 webster = (c * (1 - g/c)^2) / (2 * (1 - q/s)) + (x^2) / (2 * q * (1 - x));
17 dg = diff(webster, g);
18 dc = diff(webster, c);
19 D = matlabFunction(webster, 'Vars', {g, c});
20 mg = matlabFunction(dg, 'Vars', {g, c});
21 mc = matlabFunction(dc, 'Vars', {g, c});
22
23 % Real delay surface
24 figure('Name', 'Webster␣Linearization', 'Color', 'w');
25 [Cm, gm] = meshgrid(linspace(Cmin, Cmax, 60), linspace(gmin, 60, 60));
26 Z = zeros(size(Cm));
27
28 for i = 1:numel(Cm)
29     if (q * Cm(i)) / (s * gm(i)) < 0.98
30         Z(i) = D(gm(i), Cm(i));
31     else
32         Z(i) = NaN;
33     end
34 end
35
36 surf(Cm, gm, Z, 'FaceColor', 'c', 'FaceAlpha', 0.3, 'EdgeColor', 'none');
37 hold on;
38
39 % Sample points
40 sampleC = linspace(50, 110, density);
41 sampleG = linspace(20, 50, density);
42
43 % Plot the tangent planes
44 for C0 = sampleC
45     for g0 = sampleG
46
47         if (q * C0) / (s * g0) < 0.95
48
49             D0 = D(g0, C0);
50             m_g = mg(g0, C0);
51             m_c = mc(g0, C0);
52
53             sC = 15;
54             sG = 10;
55             [Cp, gp] = meshgrid([C0-sC, C0+sC], [g0-sG, g0+sG]);
56
57             % Plane equation: Z = D0 + m_g*(g - g0) + m_c*(C - C0)
58             Zp = D0 + m_g.*(gp - g0) + m_c.*(Cp - C0);
59
60             surf(Cp, gp, Zp, 'FaceColor', 'r', 'FaceAlpha', 0.4, 'EdgeColor', 'r', '
                   MeshStyle', 'row');
61             plot3(C0, g0, D0, 'k.', 'MarkerSize', 15);
62         end
63     end
64 end
65
66
```

```
title('Linear Approximation of Non-Linear Delay');
xlabel('Cycle Length (C)');
ylabel('Green Time (g)');
zlabel('Delay (Z)');
axis([Cmin Cmax gmin 60 0 50]);
grid on;
view(120, 30);
rotate3d on;
```