MINIMUM SNAP TRAJECTORY GENERATION & TIME SEGMENT OPTIMIZATION

Tejal Barnwal (tejalbarnwal@gmail.com)

Formulation of spline:

Given a list of waypoints (& other constraints like maximum or minimum velocity), a minimum snap trajectory generation tends to interpolate the trajectory between consecutive waypoints using some function $x = f(t)$. Without loss generality we assume $f: R \mapsto R$.
In a 3D environment, the location of the quadrotor can be written as $\vec{r}: R \mapsto R^3$, $\vec{r}(t) = [\,x(t)\; y(t)\; z(t)]^T$, where each direction $x(t),\ y(t)$ and $z(t)$ will be interpolated using a seperate function.

Minimzing snap involves minimizing the magnitude of snap along the entire trajectory
Therefore, we minimize the square of the snap(4th derivative) of the quadrotor position from time 0 to T.

$$x_n(t) = \arg\min \int_0^T \left(\ddddot{x}_n(t)\right)^2 dt = \arg\min \int_0^T L(t)\, dt \tag{1}$$

From Langrangian Principal's, the solution to such minimization problem must satisfy the principal of least action, that is:

$$\frac{\partial L}{\partial x_n} - \frac{d}{dt}\frac{\partial L}{\partial \dot{x}_n} + \frac{d^2}{dt^2}\frac{\partial L}{\partial \ddot{x}_n} + \ldots + (-1)^k \frac{d^k}{dt^k}\frac{\partial L}{\partial x_n^k} = 0 \tag{2}$$

Substituting $L(t) = \left(\ddddot{x}_n(t)\right)^2$,

$$\frac{d^4}{dt^4}\frac{\partial L}{\partial \ddddot{x}_n} = 2 \times \frac{d^4}{dt^4}\left(\ddddot{x}_n(t)\right) = x_n^{(8)}(t) = 0 \tag{3}$$

which indicates, $x(t)$ atmost can be a 7-degree polynomial.
Since, we are minimizing the snap of the trajectory, therefore we would want the fourth derivative of the trajectory to be smooth(continuous), this relates to the trajectory being at least a 4 degree polynomial.
Further, for each segment, we generally would have 8 boundary conditions for position, velocity , acceleration and jerk(which might be either free or 0) for the two waypoints the polynomial is connecting.
Therefore, $x(t)$ which interpolates between waypoints is a 7-degree polynomial between each pair of consecutive waypoints. That is $x(t)$ is a 7th order spline.
Therefore, minimum snap algorithm essentially uses a 7th order spline to interpolate between each pair of consecutive waypoints, while minimizing the square of the 4th derivative of spline subject some additional constraints.

Set up the Optimization Problem:

Let us say we have 3(N) given waypoints: $(X_0,\ Y_0,\ Z_0)$ at time t_0, $(X_1,\ Y_1,\ Z_1)$ at time t_1, $(X_2,\ Y_2,\ Z_2)$ at time t_2
Thus, we have $M = N - 1$ trajectories or trajectory segments, where each of them can be represented by a 7 degree polynomial.
Let $x(t)_{t0}^{t1} = x_0(t)$ and $x(t)_{t1}^{t2} = x_1(t)$
The general equation $x_i(t)$ that is the i^{th} trajectory segment is given by:

$$x_i(t) = c_{i7}\, t^7 + c_{i6}\, t^6 + c_{i5}\, t^5 + c_{i4}\, t^4 + c_{i3}\, t^3 + c_{i2}\, t^2 + c_{i1}\, t + c_{i0} \tag{4}$$

Now, we wish to setup an optimization problem such that

$$x(t) = \arg\min \int_{t0}^{t2} (\dddot{x}(t))^2\, dt = \int_{t0}^{t1} (\dddot{x}(t))^2\, dt + \int_{t1}^{t2} (\dddot{x}(t))^2\, dt \tag{5}$$

<u>Define Constraints</u>

Since we want our trajectory to be "smooth"(i.e. trajectory is continuous adn differential till atleast 2nd order), we must define some constraints:

1. Position Constraints:

$a)\ x_0(t_0) = X_0$
$b)\ x_0(t_1) = X_1$
$c)\ x_1(t_1) = X_1$
$d)\ x_1(t_2) = X_2$

2. Velocity Constraints:

$a)\ \dot{x}_0(t_0) = 0$
$b)\ \dot{x}_1(t_2) = 0$
$c)\ \dot{x}_0(t_1) = \dot{x}_1(t_1)$

3. Acceleration Constraints:

$a)\ \ddot{x}_0(t_0) = 0$
$b)\ \ddot{x}_2(t_2) = 0$
$c)\ \ddot{x}_0(t_1) = \ddot{x}_1(t_1)$

Since in total each 7 degree polynomial has 8 coefficients, so we would need to find $16\ (8\times\ M = 2)$ coefficient variables in order to define the trajectory.

Let us write each constraint in equation form:

<u>1. Position Constraints:</u>

$a)\ c_{07}\, t_0^7 + c_{06}\, t_0^6 + c_{05}\, t_0^5 + c_{04}\, t_0^4 + c_{03}\, t_0^3 + c_{02}\, {t_0}^2 + c_{01}\, t_0 + c_{00} = X_0$
$b)\ c_{17}\, t_2^7 + c_{16}\, t_2^6 + c_{15}\, t_2^5 + c_{14}\, t_2^4 + c_{13}\, t_2^3 + c_{12}\, {t_2}^2 + c_{11}\, t_2 + c_{10} = X_2$
$c)\ c_{07}\, t_1^7 + c_{06}\, t_1^6 + c_{05}\, t_1^5 + c_{04}\, t_1^4 + c_{03}\, t_1^3 + c_{02}\, {t_1}^2 + c_{01}\, t_1 + c_{00} = X_1$
$d)\ c_{17}\, t_1^7 + c_{16}\, t_1^6 + c_{15}\, t_1^5 + c_{14}\, t_1^4 + c_{13}\, t_1^3 + c_{12}\, {t_1}^2 + c_{11}\, t_1 + c_{10} = X_1$

<u>2. Velocity Constraints:</u>

$a)\ 7\, c_{07}\, t_0^6 + 6\, c_{06}\, t_0^5 + 5\, c_{05}\, t_0^4 + 4\, c_{04}\, t_0^3 + 3\, c_{03}\, t_0^2 + 2\, c_{02}\, t_0 + c_{01} = 0$
$b)\ 7\, c_{17}\, t_2^6 + 6\, c_{16}\, t_2^5 + 5\, c_{15}\, t_2^4 + 4\, c_{14}\, t_2^3 + 3\, c_{13}\, t_2^2 + 2\, c_{12}\, t_2 + c_{11} = 0$

$$c)\ \begin{aligned} &7\, c_{07}\, t_1^6 + 6\, c_{06}\, t_1^5 + 5\, c_{05}\, t_1^4 + 4\, c_{04}\, t_1^3 + 3\, c_{03}\, t_1^2 + 2\, c_{02}\, t_1 + c_{01} - \\ &\left(7\, c_{17}\, t_1^6 + 6\, c_{16}\, t_1^5 + 5\, c_{15}\, t_1^4 + 4\, c_{14}\, t_1^3 + 3\, c_{13}\, t_1^2 + 2\, c_{12}\, t_1 + c_{11}\right) = 0\end{aligned}$$

<u>3. Acceleration Constraints:</u>

$a)\ 42\, c_{07}\, t_0^5 + 30\, c_{06}\, t_0^4 + 20\, c_{05}\, t_0^3 + 12\, c_{04}\, t_0^2 + 6\, c_{03}\, t_0 + 2\, c_{02} = 0$
$b)\ 42\, c_{17}\, t_2^5 + 30\, c_{16}\, t_2^4 + 20\, c_{15}\, t_2^3 + 12\, c_{14}\, t_2^2 + 6\, c_{13}\, t_2 + 2\, c_{12} = 0$

$$c)\ \begin{aligned} &42\, c_{07}\, t_1^5 + 30\, c_{06}\, t_1^4 + 20\, c_{05}\, t_1^3 + 12\, c_{04}\, t_1^2 + 6\, c_{03}\, t_1 + 2\, c_{02} - \\ &\left(42\, c_{17}\, t_1^5 + 30\, c_{16}\, t_1^4 + 20\, c_{15}\, t_1^3 + 12\, c_{14}\, t_1^2 + 6\, c_{13}\, t_1 + 2\, c_{12}\right) = 0\end{aligned}$$

This gives rise to a total of $2\ (N-2) + 2$ position contstriants, N velocity constraints and N acceleration contrsints, therefore a total of $4N-2$ constraint equations

Now, since each constraint is an equality constraint in terms of coefficients c_{ij}

We can stack all these constraints in the matrix form to become $Ac = b$
In the above case when $N = 3$, so we have $8 \times 2 = 16$ length vector

$$c = \begin{bmatrix} c_{00} \\ c_{01} \\ c_{02} \\ c_{03} \\ c_{04} \\ c_{05} \\ c_{06} \\ c_{07} \\ c_{10} \\ c_{11} \\ c_{12} \\ c_{13} \\ c_{14} \\ c_{15} \\ c_{16} \\ c_{17} \end{bmatrix}$$

There are 10 constraints with 16 variables, this gives rise to 10×16 sized matrix A,

$$A = \left[\begin{array}{cccccccc|cccccccc}
1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 & t_0^6 & t_0^7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & t_2 & t_2^2 & t_2^3 & t_2^4 & t_2^5 & t_2^6 & t_2^7 \\
1 & t_1 & t_1^2 & t_1^3 & t_1^4 & t_1^5 & t_1^6 & t_1^7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & t_1 & t_1^2 & t_1^3 & t_1^4 & t_1^5 & t_1^6 & t_1^7 \\
0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 & 6t_0^5 & 7t_0^6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2t_2 & 3t_2^2 & 4t_2^3 & 5t_2^4 & 6t_2^5 & 7t_2^6 \\
0 & 1 & 2t_1 & 3t_1^2 & 4t_1^3 & 5t_1^4 & 6t_1^5 & 7t_1^6 & 0 & -1 & -2t_1 & -3t_1^2 & -4t_1^3 & -5t_1^4 & -6t_1^5 & -7t_1^6 \\
0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 & 30t_0^4 & 42t_0^5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 6t_2 & 12t_2^2 & 20t_2^3 & 30t_2^4 & 42t_2^5 \\
0 & 0 & 2 & 6t_1 & 12t_1^2 & 20t_1^3 & 30t_1^4 & 42t_1^5 & 0 & 0 & -2 & -6t_1 & -12t_1^2 & -20t_1^3 & -30t_1^4 & -42t_1^5
\end{array}\right]$$

$$b = \begin{bmatrix} X_0 \\ X_2 \\ X_1 \\ X_1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Therefore, we now have the above 10 constraints in the form of $Ac = b$.

Define Objective Cost

Now, we move to deriving the objective function of the optimization problem

Lets take a sample i^{th} spline into consideration and try to optimization objective function for it,

$p_i(t) = c_7 t^7 + c_6 t^6 + c_5 t^5 + c_4 t^4 + c_3 t^3 + c_2 t^2 + c_1 t + c_0$

Therefore, $\overset{....}{p_i}(t) = \frac{7!}{3!} t^3 c_7 + \frac{6!}{2!} t^2 c_6 + \frac{5!}{1!} t c_5 + \frac{4!}{0!} c_4$

$$Objective\ Cost = \int_{t_0}^{t_n} \left(\overset{....}{x}(t)\right)^2 dt = \sum_{i=0}^{i=M-1} \int_{t_i}^{t_{i+1}} \left(\overset{....}{p_i}(t)\right)^2 dt \tag{6}$$

$$\begin{aligned}
\int_{t_i}^{t_{i+1}} \left(\overset{....}{p_i}(t)\right)^2 dt &= \int_{t_i}^{t_{i+1}} \left(\frac{7!}{3!} t^3 c_7 + \frac{6!}{2!} t^2 c_6 + \frac{5!}{1!} t c_5 + \frac{4!}{0!} c_4 \right)^2 dt \\
&= \int_{t_i}^{t_{i+1}} \left(\frac{7!}{3!}\frac{7!}{3!} t^3 t^3 \left(c_7^2\right) + \frac{7!}{3!}\frac{6!}{2!} t^3 t^2 (c_7 c_6) + \dots + \frac{6!}{2!}\frac{7!}{3!} t^2 t^3 (c_6 c_7) + \dots \right) dt \\
&= \int_{t_i}^{t_{i+1}} \left(\sum_{r=4,\, s=4}^{r=7,\, s=7} \frac{r!}{(r-4)!} \frac{s!}{(s-4)!} t^{r-4} t^{s-4}\, c_r c_s \right) dt \\
&= \sum_{r=4,\, s=4}^{r=7,\, s=7} \frac{r!}{(r-4)!} \frac{s!}{(s-4)!} c_r c_s \left(\int_{t_i}^{t_{i+1}} t^{r+s-8}\, dt \right) \\
&= \sum_{r=4,\, s=4}^{r=7,\, s=7} \frac{r!}{(r-4)!} \frac{s!}{(s-4)!} \frac{\left({t_{i+1}}^{r+s-7} - {t_i}^{r+s-7} \right)}{(r+s-7)} c_r c_s
\end{aligned}$$

Let $q_{i,\, r,\, s} = \dfrac{r!}{(r-4)!} \dfrac{s!}{(s-4)!} \dfrac{\left({t_{i+1}}^{r+s-7} - {t_i}^{r+s-7} \right)}{(r+s-7)}$

Side note:

Now, any expression of the form $x^2 a + xy(b+c) + y^2 d$ can be written as $z^T Q z$, where

$z = \begin{bmatrix} x \\ y \end{bmatrix},\ Q = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

Therefore, we define Q matrix, such that $\left(c^T Q\, c\right)$ gives expression in specified in equation 6

For $N = 3$ case, $i = 0,\ 1$:

$$Q = \left[\begin{array}{cccccccc|cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & q_{0,4,4} & q_{0,4,5} & q_{0,4,6} & q_{0,4,7} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & q_{0,5,4} & q_{0,5,5} & q_{0,5,6} & q_{0,5,7} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & q_{0,6,4} & q_{0,6,5} & q_{0,6,6} & q_{0,6,7} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & q_{0,7,4} & q_{0,7,5} & q_{0,7,6} & q_{0,7,7} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & q_{1,4,4} & q_{1,4,5} & q_{1,4,6} & q_{1,4,7} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & q_{1,5,4} & q_{1,5,5} & q_{1,5,6} & q_{1,5,7} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & q_{1,6,4} & q_{1,6,5} & q_{1,6,6} & q_{1,6,7} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & q_{1,7,4} & q_{1,7,5} & q_{1,7,6} & q_{1,7,7}
\end{array}\right]$$

Now that constraints and objective function is defined,

<u>Final Setup</u>

The problem becomes:

$$\begin{gathered} c = \arg\min \left(c^T Q\, c\right) \\ s.\ t.: Ac = b \end{gathered} \tag{7}$$

Which then becomes a QP problem to solve, solving which result in the spline trajectory that interpolates between the waypoints. We solve the problem three times in order to solve for each of three X, Y and Z directions.

<u>Qstns to ponder:</u>

1. Why not add jerk continuity constraints? Or snap continuity constraints?? Are these ensured indirectly?

<u>What I think:</u> Yes, I think jerk continuity constraints should be imposed at segment boundaries. For snap, I think they MIGHT be ensured indirectly. We will test it with some implementation.

2. How much difference(wrt control inputs or let say the acceleration profile) does it cause in the resultant spline if its jerk minimized versus when it is snap minimized?

<u>**Optimize Travelling Time:**</u>

In the previous settings, we had $M = N - 1$ waypoints(excluding the intial position) reaching from its previous waypoints in time intervals $T_1,\ T_2,\ T_3,\ \ldots,\ T_M$, where $T_i = t_i - t_{i-1}$, where reaching times $t_0,\ t_1,\ t_2,\ \ldots\ ,\ t_N$ can be set arbitrary set by user. However we may also develop algoirthm to obtain an optimal total time for the minimum snap trajectory.

<u>Part 1: Optimizing time for each segment:</u>

Lets take the first aspect of the problem.

Given a total time T, we first aim to solve the problem of optimally distributing the time T accross M different segments to acheive the minimum snap.

Therefore, we intend to find the optimal T_i for each segment $i = 1,\ 2,\ 3,\ \ldots,\ m.$

Assume $\boldsymbol{T} = [T_1\ T_2\ \ T_3 \ldots\ T_m]^T$, and $f(\boldsymbol{T})$ is the optimal value of the optimization problem in equation (7). In particular, in 3 dimensional space, we would consider $f(\boldsymbol{T})$ as some of average of optimal value in equation (7) performed in all three axes. Under this assumption, the optimal $\boldsymbol{T}$ satisfies the following optimization problem:

$$\boldsymbol{T} = \arg\min f(\boldsymbol{T}) \quad (8)$$
$$s.t. \sum_{i=1}^{m} T_i = T$$
$$T_i \geqslant 0$$

The above optimization problem is solved using constrained gradient descent method.

```
Algorithm 1: Constrained Gradient Descent on T
Input  : $\mathbf{T}_0, \varepsilon_0, \varepsilon_1$, some small number $h$
Output : Optimal $\mathbf{T} = \mathbf{T}_{\text{opt}}$
$\mathbf{T} := \mathbf{T}_0$
while solving is not timed out and $\|\mathbf{T} - \mathbf{T}_{\text{old}}\|_2 \geq \varepsilon_0$ do
    $\nabla f(\mathbf{T}) := \sum_{i=1}^{m} \frac{f(\mathbf{T}+h\mathbf{g}_i)-f(\mathbf{T})}{h}\mathbf{g}_i$, where $\mathbf{g}_i = [\frac{-1}{m-1}, \ldots, 1, \ldots, \frac{-1}{m-1}]$ where the $i^{\text{th}}$ entry is 1;
    Search for $\mathbf{T}_{\text{new}}$ using constrained backtracking line search;
    $\mathbf{T}_{\text{old}} := \mathbf{T}$;
    $\mathbf{T} := \mathbf{T}_{\text{new}}$;
end
return $\mathbf{T}$;
```

Qstns to ponder:

1. Why use gradient descent? We might need to comment on properties such as convexity to prove that problem is solvable with gradient descent, how do we go about that?

What I think: Maybe because the objective functions contains sum of squares

2. How is the second constraint about $T_i \geqslant 0$ satisfied?

Conversation with chatGPT: The discussion revealed a lot of other insights as well. Like, when we find $\boldsymbol{T}_{new}$ with gradient descent, who ensures that sum of elements in $\boldsymbol{T}_{new}$ still gives T.

Some notes about constrained backtracking search:

Gradient Descent: Iterative optimization algorithm used to find the minimum of a function. It updates the parameters in the direction of the negative gradient of the function to minimize it. The basic rule for gradient descent is:

$x_{k+1} = x_k - \alpha \nabla f(x_k)$

In many optimization problems, the variables are subject to constraints, which can be equality constraints $h(x) = 0$ or inequaility constraints $g(x) \leqslant 0$. Incorporating these constraints into the gradient descent algorithm requires ensuring that each update satisfies these constraints.

Backtracking line search: It is a method to determine the step size α dynamically to ensure sufficient decrease in the objective function. The backtracking method starts with a relatively large initial step length (e.g., 1 for Newton method), then iteratively shrinking it by a contraction factor until the Armijo (sufficient decrease) condition is satisfied. The advantage of this approach is that the curvature condition needs not be considered, and the step length found at each line search iterate is ensured to be short enough to satisfy sufficient decrease but large enough to still allow the algorithm to make reasonable progress towards convergence.

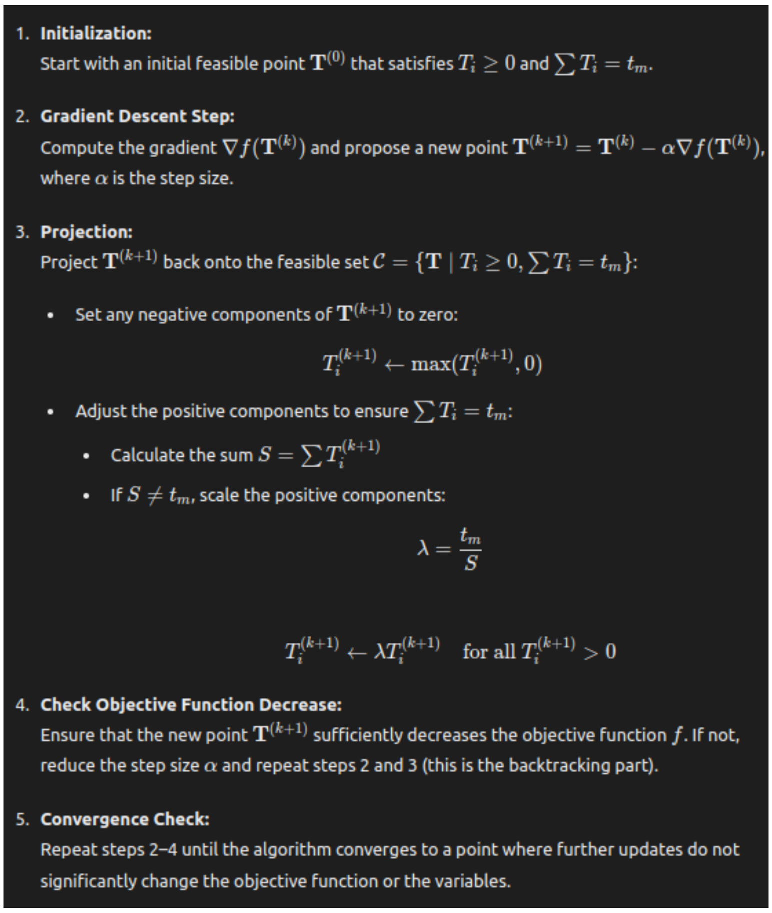
The process involves:

```
1. Starting with an initial step size $\alpha = \alpha_0 > 0, \rho \in (0,\ 1),\ c \in (0,\ 1)$
2. While $f(x_k + \alpha p_k) > f(x_k) + c\alpha\nabla f_k^T p_k$
       $\alpha \leftarrow \rho\alpha$
   End while
   Return $\alpha_k = \alpha$
```

Here, p_k is the descent direction, generally $-\nabla f_k$

References:
https://optimization.cbe.cornell.edu/index.php?title=Line_search_methods
https://www.stat.cmu.edu/~ryantibs/convexopt-S15/scribes/05-grad-descent-scribed.pdf

Handling Constraints: This involves projecting the update onto the feasible region defined by $h(x)$ and $g(x)$.
Steps in the gradient descent method:



1. **Initialization:**
   Start with an initial feasible point $\mathbf{T}^{(0)}$ that satisfies $T_i \geq 0$ and $\sum T_i = t_m$.
2. **Gradient Descent Step:**
   Compute the gradient $\nabla f(\mathbf{T}^{(k)})$ and propose a new point $\mathbf{T}^{(k+1)} = \mathbf{T}^{(k)} - \alpha \nabla f(\mathbf{T}^{(k)})$, where α is the step size.
3. **Projection:**
   Project $\mathbf{T}^{(k+1)}$ back onto the feasible set $\mathcal{C} = \{\mathbf{T} \mid T_i \geq 0, \sum T_i = t_m\}$:
   - Set any negative components of $\mathbf{T}^{(k+1)}$ to zero:
     $$T_i^{(k+1)} \leftarrow \max(T_i^{(k+1)}, 0)$$
   - Adjust the positive components to ensure $\sum T_i = t_m$:
     - Calculate the sum $S = \sum T_i^{(k+1)}$
     - If $S \neq t_m$, scale the positive components:
     $$\lambda = \frac{t_m}{S}$$
     $$T_i^{(k+1)} \leftarrow \lambda T_i^{(k+1)} \quad \text{for all } T_i^{(k+1)} > 0$$
4. **Check Objective Function Decrease:**
   Ensure that the new point $\mathbf{T}^{(k+1)}$ sufficiently decreases the objective function f. If not, reduce the step size α and repeat steps 2 and 3 (this is the backtracking part).
5. **Convergence Check:**
   Repeat steps 2–4 until the algorithm converges to a point where further updates do not significantly change the objective function or the variables.

3. Why not use langrangian multipliers in order to solve the problem?
Langrangian method handles equality constraints and not inequality. You can use penality methods. Explore more about difference in langrangian and penalty methods to solve optimization, especially incase of equality constraints.

Part 2: Optimizing total time and time for each segment

When solving for optimizing total time, use the following optimization problem formulation:

$$\boldsymbol{T} = \arg\min f(\boldsymbol{T})$$
$$0 \leqslant \boldsymbol{T} \leqslant \boldsymbol{T}_u$$

Or
Maybe we can also use the formulation in equation (8), with a transformed objective function as
$f(\boldsymbol{T}) + \lambda(T_1 + T_2 + ... + T_m)$

Qstns to Ponder:
1. What optimization algorithms can be used? Any non-linear ones maybe?
References:
https://core.ac.uk/download/pdf/76383736.pdf