Fuel-Optimal PDG via Convex Optimization

Technical Foundations and Implementation

Casper J.P. van der Wouden[1]

December 2025

Abstract

This note provides the derivations underlying the fuel-optimal powered descent guidance implementation. I present the lossless convexification approach of Açıkmeşe and Ploen (2007), which reformulates the non-convex rocket landing problem as a second-order cone program (SOCP). The document details the continuous-time formulation, exact zero-order-hold discretization, free-final-time handling via grid search, and successive convex programming (SCP) refinement for mass-dependent bounds. A proof of losslessness is provided. The document corresponds directly to the accompanying Python codebase: `config.py`, `dynamics.py`, `problem.py`, `solver.py`, `plots.py`, `sensitivity.py`, and `main.py`.

Contents

[1]Maastricht University SBE, Netherlands. Email: c.vanderwouden@student.maastrichtuniversity.nl

1 Introduction & Motivation

The problem of optimally landing a rocket, minimizing fuel consumption while satisfying safety and physical constraints, represents an important challenge in aerospace guidance and control.

This problem has gained renewed practical significance with the advent of reusable launch vehicles, most notably SpaceX's Falcon 9 booster, which performs autonomous propulsive landings..

From a mathematical perspective, powered descent guidance is a constrained optimal control problem. The objective is to determine the thrust vector profile $\boldsymbol{T}(t)$ that transfers the vehicle from an initial state (position, velocity) to a terminal state (soft landing at target) while minimizing propellant consumption. The problem is subject to several constraints arising from physics and engineering requirements:

- **Dynamics**: The vehicle (rocket) obeys Newton's laws under gravity and thrust.
- **Thrust bounds**: Rocket engines have both maximum and minimum thrust limits. The minimum arises because many engines cannot throttle below a certain threshold without combustion instability.
- **Glideslope constraint**: The vehicle must remain above a conical boundary to avoid terrain collision during descent. Crashing is not so good..
- **Pointing constraint**: The thrust vector cannot deviate too far from vertical due to gimbal limits.
- **Terminal conditions**: The vehicle must reach the target with zero velocity.

Two standard numerical approaches exist for such problems:

(i) **Direct collocation / shooting**: Transcribe to nonlinear program (NLP), solve with gradient-based methods. May converge to local minima.

(ii) **Dynamic programming**: Exponential complexity in state dimension (curse of dimensionality).

The fundamental difficulty is that the thrust magnitude constraint $\|\boldsymbol{T}(t)\| \geq T_{\min}$ is nonconvex. This destroys the convexity of the feasible set and, in principle, requires global optimization methods that do not scale well and cannot guarantee real-time performance.

The seminal contribution of Açıkmeşe and Ploen (2007) was demonstrating that the nonconvex powered descent problem admits a *lossless convexification*. By introducing a slack variable representing thrust magnitude and relaxing the equality constraint to an inequality, the problem becomes a second-order cone program (SOCP), a class of convex optimization problems solvable in polynomial time with interior-point methods. The key theoretical result is that this relaxation is *tight*: at any optimal solution of the relaxed problem, the relaxed inequality constraint is satisfied with equality. This remarkable property enables real-time optimal guidance computation.

Implementation Reference

Project entry points:

- `main.py`: `run_full()` executes grid search, plotting, sensitivity, and writes `output/results.json` + `output/trajectory.npz`.
- `main.py`: `run_quick()` runs a single fixed-t_f solve for rapid testing.
- Problem data: `config.py` (`DescentConfig`, `DEFAULT_CONFIG`).

2 Problem Formulation

2.1 Coordinate System and State Variables

We work in a Cartesian frame with z upward and target at the origin. Gravity is constant..

$$\boldsymbol{g} = (0, 0, -g_0)^\top, \quad g_0 > 0. \tag{1}$$

Let

$$\boldsymbol{r}(t) \in \mathbb{R}^3, \quad \boldsymbol{v}(t) \in \mathbb{R}^3, \quad m(t) \in \mathbb{R}_{>0}, \quad \boldsymbol{T}(t) \in \mathbb{R}^3 \tag{2}$$

denote position, velocity, mass, and thrust in Newtons. The state vector is $\boldsymbol{x}(t) = (\boldsymbol{r}(t), \boldsymbol{v}(t), m(t)) \in \mathbb{R}^7$. The control input is the thrust vector $\boldsymbol{T}(t) \in \mathbb{R}^3$.

2.2 Equations of Motion

The translational dynamics follow from Newton's second law:

$$\dot{\boldsymbol{r}}(t) = \boldsymbol{v}(t), \tag{3}$$

$$\dot{\boldsymbol{v}}(t) = \frac{\boldsymbol{T}(t)}{m(t)} + \boldsymbol{g}. \tag{4}$$

The mass dynamics follow from the rocket equation. Assuming constant specific impulse I_{sp} and exhaust velocity $v_e = I_{\text{sp}} \cdot g_0$:

$$\dot{m}(t) = -\frac{\|\boldsymbol{T}(t)\|}{v_e} = -\alpha \|\boldsymbol{T}(t)\|, \tag{5}$$

where $\alpha = 1/v_e > 0$ is the mass depletion rate per unit thrust.

Remark 2.1. The mass dynamics (5) couple the fuel consumption directly to the thrust magnitude. Minimizing $\int_0^{t_f} \|\boldsymbol{T}(t)\| \, dt$ is equivalent to minimizing fuel consumption (maximizing final mass).

2.3 Boundary Conditions

Initial conditions at $t = 0$:

$$\boldsymbol{r}(0) = \boldsymbol{r}_0, \quad \boldsymbol{v}(0) = \boldsymbol{v}_0, \quad m(0) = m_0. \tag{6}$$

Terminal conditions at final time t_f:

$$\boldsymbol{r}(t_f) = \boldsymbol{0}, \quad \boldsymbol{v}(t_f) = \boldsymbol{0}. \tag{7}$$

The final mass $m(t_f)$ is free and will be maximized.. (equivalently, fuel consumption minimized)

2.4 Thrust Constraints

Rocket engines are subject to both upper and lower thrust bounds:

$$0 < T_{\min} \leq \|\boldsymbol{T}(t)\| \leq T_{\max}. \tag{8}$$

The lower bound $T_{\min} > 0$ is the critical non-convex constraint. It arises because many engines cannot throttle below $\sim$30% to $\sim$70% without combustion instability. If $T_{\min} = 0$ were allowed, the problem would be convex.

2.5 Glideslope Constraint

To ensure terrain avoidance, the vehicle must remain within a conical region centered on the target. Let $\gamma_{\text{gs}} \in (0, \pi/2)$ be the glideslope angle measured from the horizontal plane. The constraint follows as..

$$r_z(t) \geq \tan(\gamma_{\text{gs}}) \cdot \|(r_x(t), r_y(t))\| . \tag{9}$$

Equivalently, $\|(r_x, r_y)\| \leq \cot(\gamma_{\text{gs}}) \cdot r_z$. This is a second-order cone constraint and is convex.

Remark 2.2 (Angle Convention). The glideslope angle γ_{gs} is measured from horizontal:

- **Small** $\gamma_{\text{gs}} \Rightarrow$ wide cone $\Rightarrow$ permissive (can fly more horizontally)
- **Large** $\gamma_{\text{gs}} \Rightarrow$ narrow cone $\Rightarrow$ restrictive (must stay near vertical)

2.6 Pointing Constraint

The thrust vector cannot deviate too far from the vertical axis due to gimbal limitations. Let $\theta_{\max} \in (0, \pi/2)$ be the maximum angle from vertical. Then:

$$T_z(t) \geq \cos(\theta_{\max}) \cdot \|\boldsymbol{T}(t)\| . \tag{10}$$

This is also a second-order cone constraint and is convex..

2.7 Mass Bounds

The mass must remain positive and cannot exceed the initial wet mass:

$$m_{\text{dry}} \leq m(t) \leq m_0, \tag{11}$$

where m_{dry} is the dry mass (vehicle mass without propellant).

2.8 Fuel Objective

Propellant used is:

$$m_0 - m(t_f) = \frac{1}{v_e} \int_0^{t_f} \|\boldsymbol{T}(t)\| \, dt. \tag{12}$$

Minimizing this integral is equivalent to maximizing final mass.

2.9 The Optimal Control Problem

Assembling all elements:

Problem 1 (Powered Descent Guidance)**.**

$$\begin{aligned}
\underset{\boldsymbol{T}(\cdot), t_f}{\text{minimize}} \quad & \int_0^{t_f} \|\boldsymbol{T}(t)\| \, dt && (13)\\
\text{subject to} \quad & \dot{\boldsymbol{r}}(t) = \boldsymbol{v}(t) && (14)\\
& \dot{\boldsymbol{v}}(t) = \frac{\boldsymbol{T}(t)}{m(t)} + \boldsymbol{g} && (15)\\
& \dot{m}(t) = -\alpha \|\boldsymbol{T}(t)\| && (16)\\
& \boldsymbol{r}(0) = \boldsymbol{r}_0, \ \boldsymbol{v}(0) = \boldsymbol{v}_0, \ m(0) = m_0 && (17)\\
& \boldsymbol{r}(t_f) = \boldsymbol{0}, \ \boldsymbol{v}(t_f) = \boldsymbol{0} && (18)\\
& T_{\min} \leq \|\boldsymbol{T}(t)\| \leq T_{\max} && (19)\\
& r_z(t) \geq \tan(\gamma_{\text{gs}}) \, \|(r_x(t), r_y(t))\| && (20)\\
& T_z(t) \geq \cos(\theta_{\max}) \, \|\boldsymbol{T}(t)\| && (21)\\
& m_{\text{dry}} \leq m(t) \leq m_0 && (22)
\end{aligned}$$

Implementation Reference

Constraint implementation in `problem.py`:

- Glideslope: `r[k,2] >= tan_gs * cp.norm(r[k,:2])`
- Pointing: `u[k,2] >= cos_point * sigma[k]`
- Bounds: `config.py` provides `rho_min`, `rho_max`

3 Analysis of Non-Convexity

Problem 1 exhibits two sources of non-convexity..

3.1 Thrust Lower Bound

The constraint $\|\boldsymbol{T}(t)\| \geq T_{\min}$ defines the exterior of a ball, which is non-convex. Consider the feasible set for $\boldsymbol{T} \in \mathbb{R}^3$:

$$\mathcal{T} = \{\boldsymbol{T} \in \mathbb{R}^3 : T_{\min} \leq \|\boldsymbol{T}\| \leq T_{\max}\}. \tag{23}$$

This is an annular region, a spherical shell.. It is non-convex because the line segment connecting two points on opposite sides of the shell passes through the interior (the "hole").

Proposition 3.1. *The set* $\mathcal{T} = \{\boldsymbol{T} \in \mathbb{R}^3 : T_{\min} \leq \|\boldsymbol{T}\| \leq T_{\max}\}$ *is non-convex for* $T_{\min} > 0$.

Proof. Consider $\boldsymbol{T}_1 = (T_{\min}, 0, 0)^\top$ and $\boldsymbol{T}_2 = (-T_{\min}, 0, 0)^\top$. Both satisfy $\|\boldsymbol{T}_i\| = T_{\min}$, so $\boldsymbol{T}_1, \boldsymbol{T}_2 \in \mathcal{T}$. However, the midpoint $\frac{1}{2}(\boldsymbol{T}_1 + \boldsymbol{T}_2) = \boldsymbol{0}$ has $\|\boldsymbol{0}\| = 0 < T_{\min}$, so the midpoint is not in $\mathcal{T}$. □

3.2 Mass-Thrust Coupling

The dynamics $\dot{\boldsymbol{v}} = \boldsymbol{T}/m + \boldsymbol{g}$ involve division by mass, making the system nonlinear in a way that couples control and state. This can be addressed via a change of variables (Section 4).

4 Lossless Convexification

The breakthrough of lossless convexification is that, for this specific problem, convex relaxation yields the global optimum.

4.1 Change of Variables

Define thrust-acceleration and thrust magnitude surrogate:

$$\boldsymbol{u}(t) = \frac{\boldsymbol{T}(t)}{m(t)} \in \mathbb{R}^3, \quad \sigma(t) = \frac{\|\boldsymbol{T}(t)\|}{m(t)} \in \mathbb{R}_{\geq 0}. \tag{24}$$

Note that $\|\boldsymbol{u}(t)\| = \sigma(t)$.. The velocity dynamics become linear:

$$\dot{\boldsymbol{v}}(t) = \boldsymbol{u}(t) + \boldsymbol{g}. \tag{25}$$

Define log-mass:

$$z(t) = \ln m(t). \tag{26}$$

Then:

$$\dot{z}(t) = \frac{\dot{m}(t)}{m(t)} = -\alpha\sigma(t). \tag{27}$$

The thrust bounds become:

$$\frac{T_{\min}}{m(t)} \leq \sigma(t) \leq \frac{T_{\max}}{m(t)}. \tag{28}$$

Define mass-dependent bounds

$$\rho_{\min}(t) = T_{\min} e^{-z(t)}, \quad \rho_{\max}(t) = T_{\max} e^{-z(t)}. \tag{29}$$

4.2 Reformulated Problem

In the new variables, the problem becomes:

Problem 2 (Reformulated)**.**

$$\begin{aligned}
\underset{\boldsymbol{u}(\cdot),\sigma(\cdot),t_f}{\text{minimize}} \quad & \int_0^{t_f} \sigma(t)\, dt && (30)\\
\text{subject to} \quad & \dot{\boldsymbol{r}} = \boldsymbol{v},\ \dot{\boldsymbol{v}} = \boldsymbol{u} + \boldsymbol{g},\ \dot{z} = -\alpha\sigma && (31)\\
& \boldsymbol{r}(0) = \boldsymbol{r}_0,\ \boldsymbol{v}(0) = \boldsymbol{v}_0,\ z(0) = \ln m_0 && (32)\\
& \boldsymbol{r}(t_f) = \boldsymbol{0},\ \boldsymbol{v}(t_f) = \boldsymbol{0} && (33)\\
& \|\boldsymbol{u}(t)\| = \sigma(t) && (34)\\
& \rho_{\min}(t) \leq \sigma(t) \leq \rho_{\max}(t) && (35)\\
& r_z(t) \geq \tan(\gamma_{\text{gs}})\, \|(r_x(t), r_y(t))\| && (36)\\
& u_z(t) \geq \cos(\theta_{\max})\sigma(t) && (37)\\
& \ln(m_{\text{dry}}) \leq z(t) \leq \ln(m_0) && (38)
\end{aligned}$$

The constraint (34) is problematic as it states that the magnitude of $\boldsymbol{u}$ equals σ, which together with $\sigma \geq \rho_{\min} > 0$ creates non-convexity.

4.3 The Convex Relaxation

The key insight is to relax the equality to an inequality:

Problem 3 (Relaxed)**.**

$$\begin{aligned}
\underset{\boldsymbol{u}(\cdot),\sigma(\cdot),t_f}{\text{minimize}} \quad & \int_0^{t_f} \sigma(t)\, dt && (39)\\
\text{subject to} \quad & \text{(all constraints from Problem 2, except (34))} && (40)\\
& \|\boldsymbol{u}(t)\| \leq \sigma(t). && (41)
\end{aligned}$$

The constraint (41) is a second-order cone constraint:

$$\|\boldsymbol{u}\| \leq \sigma \iff (\boldsymbol{u}, \sigma) \in \mathcal{Q}^4, \tag{42}$$

where $\mathcal{Q}^4 = \{(\boldsymbol{x}, t) \in \mathbb{R}^3 \times \mathbb{R} : \|\boldsymbol{x}\| \leq t\}$ is the Lorentz cone.

The feasible set of Problem 3 strictly contains that of Problem 2. Any feasible solution of the original problem is feasible for the relaxed problem, so:

$$J^*_{\text{relaxed}} \leq J^*_{\text{original}}. \tag{43}$$

4.4 Losslessness of the Relaxation

Scope of the bounds. In this section I treat $\rho_{\min}(\cdot)$ and $\rho_{\max}(\cdot)$ as *prescribed* (known) bound functions during a single convex solve (e.g., conservative constants, or bounds frozen from a reference log-mass trajectory in an SCP outer loop). Under this assumption, the relaxed problem is convex and the following slack-removal argument applies..

The remarkable result is that the relaxation is tight: at optimality, $\|\boldsymbol{u}^*(t)\| = \sigma^*(t)$ for almost all t.

Theorem 4.1 (Lossless Convexification)**.** *Let $(\boldsymbol{u}^*, \sigma^*, t_f^*)$ be an optimal solution of the relaxed Problem 3. Then:*

$$\|\boldsymbol{u}^*(t)\| = \sigma^*(t) \quad \text{for almost all } t \in [0, t_f^*]. \tag{44}$$

Consequently, (u^*, σ^*, t_f^*) is optimal for the corresponding equality-constrained formulation obtained by replacing $\|u(t)\| \le \sigma(t)$ with $\|u(t)\| = \sigma(t)$ while keeping the same prescribed bounds $\rho_{\min}(\cdot)$ and $\rho_{\max}(\cdot)$, and the optimal values coincide.

We first establish a key lemma.

Lemma 4.2 (Strict Optimum Improvement)**.** *Suppose $(\boldsymbol{u}, \sigma)$ is feasible for Problem 3 and there exists a set $S \subset [0, t_f]$ of positive measure such that $\|\boldsymbol{u}(t)\| < \sigma(t)$ for $t \in S$. Then there exists a feasible solution $(\tilde{\boldsymbol{u}}, \tilde{\sigma})$ with strictly smaller objective value.*

Proof. On the set S, define:

$$\tilde{\sigma}(t) = \max\{\|\boldsymbol{u}(t)\|, \rho_{\min}(t)\} \tag{45}$$

and $\tilde{\boldsymbol{u}}(t) = \boldsymbol{u}(t)$. On $[0, t_f] \setminus S$, keep $\tilde{\sigma}(t) = \sigma(t)$ and $\tilde{\boldsymbol{u}}(t) = \boldsymbol{u}(t)$.

Feasibility verification:

- $\|\tilde{\boldsymbol{u}}(t)\| \le \tilde{\sigma}(t)$ by construction.
- $\tilde{\sigma}(t) \ge \rho_{\min}(t)$ by construction.
- $\tilde{\sigma}(t) \le \rho_{\max}(t)$ since $\|\boldsymbol{u}(t)\| < \sigma(t) \le \rho_{\max}(t)$ on S and $\rho_{\min}(t) \le \rho_{\max}(t)$.
- The (r, v) dynamics are unchanged since $\tilde{u} = u$.
- The log-mass dynamics are satisfied by construction by integrating $\dot{z}(t) = -\alpha\tilde{\sigma}(t)$ with the same initial condition $z(0) = \ln m_0$.

For the objective, on S:

$$\tilde{\sigma}(t) = \max\{\|\boldsymbol{u}(t)\|, \rho_{\min}(t)\} < \sigma(t), \tag{46}$$

since $\|\boldsymbol{u}(t)\| < \sigma(t)$ and $\rho_{\min}(t) \le \sigma(t)$. Thus:

$$\int_0^{t_f} \tilde{\sigma}(t)\, dt < \int_0^{t_f} \sigma(t)\, dt. \tag{47}$$

□

Proof of Theorem 4.1. Suppose for contradiction that the optimal solution $(\boldsymbol{u}^*, \sigma^*)$ has $\|\boldsymbol{u}^*(t)\| < \sigma^*(t)$ on a set S of positive measure. By Lemma 4.2, there exists a feasible solution with strictly smaller objective value, contradicting optimality. Therefore, $\|\boldsymbol{u}^*(t)\| = \sigma^*(t)$ almost everywhere.

Since Problem 3 has a larger feasible set, $J^*_{\text{relaxed}} \le J^*_{\text{original}}$. But it has now been shown that the optimal solution of the relaxed problem is feasible for the original problem, so..

$$J^*_{\text{original}} \le J^*_{\text{relaxed}} \ \& \ \text{Therefore } J^*_{\text{relaxed}} = J^*_{\text{original}}$$

□

Remark 4.3 (Economic Intuition). The key intuition is: in the relaxed problem, we penalize σ in the objective while only requiring $\|\boldsymbol{u}\| \leq \sigma$. But σ appears in the mass dynamics as $\dot{z} = -\alpha\sigma$. Any "slack" in the constraint (where $\|\boldsymbol{u}\| < \sigma$) means we are depleting mass faster than necessary for the achieved acceleration. The optimizer will never choose this.. It will always set $\sigma = \|\boldsymbol{u}\|$ to minimize fuel consumption.

4.5 Handling Mass-Dependent Bounds

The constraints involving $e^{-z(t)}$ are not jointly convex in $(\boldsymbol{u}, z)$. Two approaches can be taken:

Approach 1: Conservative Fixed Bounds. Use worst-case mass values:

$$\rho_{\min} = \frac{T_{\min}}{m_0}, \quad \rho_{\max} = \frac{T_{\max}}{m_{\text{dry}}}. \tag{48}$$

These are constants that guarantee feasibility for all mass values but are conservative.

Approach 2: Successive Convexification. Solve iteratively, using the previous solution's mass trajectory to refine bounds:

$$\rho^{(i)}_{\min,k} = T_{\min} e^{-z_k^{(i-1)}}, \quad \rho^{(i)}_{\max,k} = T_{\max} e^{-z_k^{(i-1)}} \tag{49}$$

This is a form of sequential convex programming (SCP) that typically converges in few iterations..

Implementation Reference

Mass-dependent bounds in `problem.py`:

- Base solve: Uses conservative bounds from `config.thrust_bounds_conservative()`
- SCP option: `build_problem(..., use_scp_bounds=True, z_ref=...)` computes time-varying bounds

4.6 Final Convex Formulation

The complete convexified problem is:

Problem 4 (Convex Powered Descent)**.**

$$\begin{aligned}
\underset{\boldsymbol{u}(\cdot),\sigma(\cdot),t_f}{\text{minimize}} \quad & \int_0^{t_f} \sigma(t)\, dt && (50)\\
\text{subject to} \quad & \dot{\boldsymbol{r}}(t) = \boldsymbol{v}(t) && (51)\\
& \dot{\boldsymbol{v}}(t) = \boldsymbol{u}(t) + \boldsymbol{g} && (52)\\
& \dot{z}(t) = -\alpha\sigma(t) && (53)\\
& \boldsymbol{r}(0) = \boldsymbol{r}_0,\ \boldsymbol{v}(0) = \boldsymbol{v}_0,\ z(0) = \ln m_0 && (54)\\
& \boldsymbol{r}(t_f) = \boldsymbol{0},\ \boldsymbol{v}(t_f) = \boldsymbol{0} && (55)\\
& \|\boldsymbol{u}(t)\| \leq \sigma(t) && (56)\\
& \rho_{\min} \leq \sigma(t) \leq \rho_{\max} && (57)\\
& r_z(t) \geq \tan(\gamma_{\text{gs}})\, \|(r_x(t), r_y(t))\| && (58)\\
& u_z(t) \geq \cos(\theta_{\max})\sigma(t) && (59)\\
& \ln(m_{\text{dry}}) \leq z(t) \leq \ln(m_0) && (60)
\end{aligned}$$

Here $\rho_{\min}, \rho_{\max}$ are constant thrust-acceleration bounds chosen conservatively as

$$\rho_{\min} = \frac{T_{\min}}{m_0}, \qquad \rho_{\max} = \frac{T_{\max}}{m_{\text{dry}}}. \tag{61}$$

For fixed t_f and conservative bounds, this is a convex infinite-dimensional problem.

5 Discretization for Numerical Implementation

To solve Problem 4 numerically, time is discretized using an exact zero-order-hold (ZOH) scheme.

5.1 Time Discretization

Let t_f be fixed and divide $[0, t_f]$ into N intervals:

$$t_k = k\Delta t, \quad k = 0, 1, \ldots, N, \quad \Delta t = \frac{t_f}{N}. \tag{62}$$

State samples:

$$\boldsymbol{r}_k \approx \boldsymbol{r}(t_k), \quad \boldsymbol{v}_k \approx \boldsymbol{v}(t_k), \quad z_k \approx z(t_k). \tag{63}$$

Control is assumed constant over each interval (ZOH):

$$\boldsymbol{u}(t) \equiv \boldsymbol{u}_k, \quad \sigma(t) \equiv \sigma_k, \quad \text{for } t \in [t_k, t_{k+1}). \tag{64}$$

5.2 Exact Discrete-Time Dynamics

Integrating $\dot{\boldsymbol{v}} = \boldsymbol{u}_k + \boldsymbol{g}$ gives:

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + (\boldsymbol{u}_k + \boldsymbol{g})\Delta t. \tag{65}$$

Integrating $\dot{\boldsymbol{r}} = \boldsymbol{v}(t)$ gives:

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \boldsymbol{v}_k\Delta t + \frac{1}{2}(\boldsymbol{u}_k + \boldsymbol{g})\Delta t^2. \tag{66}$$

Integrating $\dot{z} = -\alpha\sigma_k$ gives:

$$z_{k+1} = z_k - \alpha\sigma_k\Delta t. \tag{67}$$

These are exact under the ZOH assumption.

5.3 Matrix Form

Let $\boldsymbol{x}_k = (\boldsymbol{r}_k, \boldsymbol{v}_k, z_k) \in \mathbb{R}^7$ and $\boldsymbol{w}_k = (\boldsymbol{u}_k, \sigma_k) \in \mathbb{R}^4$. Then:

$$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{w}_k + \boldsymbol{c}, \tag{68}$$

where:

$$A = \begin{pmatrix} I_3 & \Delta t \cdot I_3 & 0 \\ 0 & I_3 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} \frac{1}{2}\Delta t^2 \cdot I_3 & 0 \\ \Delta t \cdot I_3 & 0 \\ 0 & -\alpha\Delta t \end{pmatrix}, \quad \boldsymbol{c} = \begin{pmatrix} \frac{1}{2}\Delta t^2 \boldsymbol{g} \\ \Delta t \boldsymbol{g} \\ 0 \end{pmatrix}. \tag{69}$$

Implementation Reference

Discrete dynamics in `dynamics.py`:

- `build_state_transition_matrices(config, dt)` returns $(A, B, \boldsymbol{c})$
- `problem.py` uses these matrices to impose $\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{w}_k + \boldsymbol{c}$

5.4 Discrete SOCP (Fixed t_f)

Decision variables:

$$\{\boldsymbol{r}_k, \boldsymbol{v}_k, z_k\}_{k=0}^{N}, \quad \{\boldsymbol{u}_k, \sigma_k\}_{k=0}^{N-1}. \tag{70}$$

Objective (exact for ZOH σ):

$$J = \sum_{k=0}^{N-1} \sigma_k \Delta t. \tag{71}$$

Constraints:

$$\boldsymbol{r}_0 = \boldsymbol{r}_0,\ \boldsymbol{v}_0 = \boldsymbol{v}_0,\ z_0 = \ln m_0, \tag{72}$$

$$\boldsymbol{r}_N = \boldsymbol{0},\ \boldsymbol{v}_N = \boldsymbol{0}, \tag{73}$$

$$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{w}_k + \boldsymbol{c}, \quad k = 0, \ldots, N-1, \tag{74}$$

$$\|\boldsymbol{u}_k\| \le \sigma_k, \quad k = 0, \ldots, N-1, \tag{75}$$

$$\rho_{\min,k} \le \sigma_k \le \rho_{\max,k}, \quad k = 0, \ldots, N-1, \tag{76}$$

$$u_{k,z} \ge \cos(\theta_{\max})\sigma_k, \quad k = 0, \ldots, N-1, \tag{77}$$

$$r_{k,z} \ge \tan(\gamma_{\text{gs}}) \left\|(r_{k,x}, r_{k,y})\right\|, \quad k = 0, \ldots, N, \tag{78}$$

$$\ln(m_{\text{dry}}) \le z_k \le \ln(m_0), \quad k = 0, \ldots, N. \tag{79}$$

If $(\rho_{\min,k}, \rho_{\max,k})$ are constants (conservative bounds), this is a finite-dimensional SOCP.

Proposition 5.1. *The discrete problem (71)–(79) is an SOCP: objective is linear; constraints are affine or second-order cone constraints..*

Proof. All dynamics constraints (74) are affine equalities. Thrust bounds and mass bounds are affine inequalities. The constraints $\|\boldsymbol{u}_k\| \le \sigma_k$, $u_{k,z} \ge \cos(\theta_{\max})\sigma_k$, and the glideslope constraint are SOC-representable. □

6 Free-Final-Time Handling

6.1 Why Not Optimize t_f Directly?

Naively letting t_f be a decision variable makes $\Delta t = t_f/N$ multiply $\boldsymbol{u}_k$ in the dynamics (e.g., $\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + (\boldsymbol{u}_k + \boldsymbol{g})t_f/N$), introducing bilinear terms and destroying convexity.

6.2 Grid Search Meta-Algorithm

Choose a grid $\{t_f^{(i)}\}_{i=1}^{M}$. For each $t_f^{(i)}$, solve the SOCP and obtain optimal value $J^{(i)}$ if feasible. Pick:

$$t_f^\star \in \arg \min_{t_f^{(i)} \text{ feasible}} J^{(i)}. \tag{80}$$

This yields a discrete approximation of a free-final-time optimum and (importantly) reveals the feasibility window. The grid can be refined around the optimum.

Complexity: Each solve is polynomial time; total complexity is $O(M \cdot N^3)$. In practice, $M \sim 30$ and $N \sim 50$ solve in seconds.

Implementation Reference

Free-final-time logic in `solver.py`:

- `solve_grid_search(config, tf_min, tf_max, n_grid)` loops over t_f values
- `plots.py`: `plot_grid_search(grid_results)` visualizes fuel vs. t_f

7 SCP Refinement for Mass-Dependent Bounds

7.1 SCP Subproblem

Given a reference log-mass trajectory z_k^{ref} (from previous iteration), define mass-dependent bounds:

$$\rho_{\min,k} = T_{\min} e^{-z_k^{\text{ref}}}, \quad \rho_{\max,k} = T_{\max} e^{-z_k^{\text{ref}}}. \tag{81}$$

Solve the discrete SOCP with these bounds. Update reference $z^{\text{ref}} \leftarrow z^\star$ and repeat.

7.2 Convergence Criterion

Stop when the relative change in fuel consumption falls below tolerance:

$$\frac{|J^{(j)} - J^{(j-1)}|}{\max\{1, |J^{(j-1)}|\}} \leq \varepsilon, \tag{82}$$

or when a maximum iteration count is reached..

7.3 Trust Regions – *Optional*

For more robust convergence, add trust region constraints:

$$|z_k - z_k^{\text{ref}}| \leq \delta_z, \tag{83}$$

to prevent large deviations from the linearization point..

Implementation Reference

SCP loop in `solver.py`:

- `solve_with_scp(...)` iterates, passing `use_scp_bounds=True` and `z_ref` into `problem.build_problem`

8 Implementation Notes (CVXPY/SOCP)

8.1 Decision Arrays and Sizes

For fixed (N, t_f):

$$\boldsymbol{r} \in \mathbb{R}^{(N+1)\times 3},\ \boldsymbol{v} \in \mathbb{R}^{(N+1)\times 3},\ z \in \mathbb{R}^{N+1},\ \boldsymbol{u} \in \mathbb{R}^{N\times 3},\ \sigma \in \mathbb{R}^{N}. \tag{84}$$

Scalar dimension: $(3+3+1)(N+1) + (3+1)N = 11N + 7$.

8.2 Conic Constraints

Each SOC constraint is represented in CVXPY using `cp.norm()`:

- **Thrust cone:** `cp.norm(u[k]) <= sigma[k]` (Lorentz cone in $\mathbb{R}^4$)
- **Glideslope:** `r[k,2] >= tan_gs * cp.norm(r[k,:2])` (after rearranging)
- **Pointing:** `u[k,2] >= cos_point * sigma[k]` (half-space intersection)

8.3 Solvers

Any SOCP solver supported by CVXPY applies:

- **Clarabel**: Modern open-source, recommended default
- **ECOS**: Embedded-friendly, stable but slower
- **MOSEK**: Commercial, best accuracy and speed
- **SCS**: First-order method, good for large-scale

Numerical tolerances determine observed slack $\varepsilon_{\max}$. Tight tolerances achieve $\varepsilon_{\max} \leq 10^{-6}$

8.4 Losslessness Certificate

After solving, verify relaxation tightness:

$$\varepsilon_k = \begin{cases} \frac{\sigma_k - \|\boldsymbol{u}_k\|}{\sigma_k}, & \sigma_k > 0, \\ 0, & \sigma_k = 0, \end{cases} \qquad \varepsilon_{\max} = \max_k \varepsilon_k. \tag{85}$$

If $\varepsilon_{\max} \leq \text{tol}$ (e.g., 10^{-6}), the relaxation is tight in practice.

Implementation Reference

Losslessness checking in `problem.py`:

- `verify_losslessness(sol)` computes the max relative slack and returns a boolean flag
- `plots.py`: `plot_losslessness(...)` produces tightness and slack plots

8.5 Code Structure

Module	Purpose
`config.py`	Parameters, derived quantities, validation
`dynamics.py`	State transition matrices $(A, B, \boldsymbol{c})$, propagation
`problem.py`	CVXPY SOCP formulation, losslessness verification
`solver.py`	Fixed-t_f solve, grid search, SCP loop
`plots.py`	Trajectory and sensitivity visualization
`sensitivity.py`	Parameter sweeps (glideslope, $T_{\min}$, altitude)
`main.py`	End-to-end pipeline, result serialization, calls all visuals

Table 1: Code module organization.

8.6 Output Artifacts

Outputs include:

- `results.json`: Summary (best t_f, fuel, feasibility array, slack metrics)
- `trajectory.npz`: Arrays $(\boldsymbol{r}, \boldsymbol{v}, z, \boldsymbol{u}, \sigma, t)$
- `figures/`: PNG visualizations of trajectory, time series, grid search, sensitivity

9 Numerical Example

Considering a powered descent scenario inspired by SpaceX Falcon 9 first-stage landing..

9.1 Problem Parameters

Parameter	Value	Description
m_0	25,000 kg	Initial (wet) mass
m_{dry}	22,000 kg	Dry mass
I_{sp}	282 s	Specific impulse (sea level)
$T_{\max}$	845,000 N	Maximum thrust (single Merlin)
$T_{\min}$	250,000 N	Minimum thrust ($\sim$30% throttle)
γ_{gs}	6°	Glideslope angle from horizontal
$\theta_{\max}$	20°	Max thrust pointing from vertical
$\boldsymbol{r}_0$	$(500, 100, 1500)$ m	Initial position
$\boldsymbol{v}_0$	$(20, 5, -75)$ m/s	Initial velocity

Table 2: Problem parameters for numerical example.

9.2 Derived Quantities

$$v_e = I_{\text{sp}} \cdot g_0 = 282 \times 9.807 = 2765.6 \text{ m/s} \tag{86}$$

$$\alpha = 1/v_e = 3.616 \times 10^{-4} \text{ s/m} \tag{87}$$

$$\rho_{\min,0} = T_{\min}/m_0 = 10.0 \text{ m/s}^2 \tag{88}$$

$$\rho_{\max,0} = T_{\max}/m_0 = 33.8 \text{ m/s}^2 \tag{89}$$

9.3 Expected Results

For this configuration, the optimal trajectory exhibits:

Trajectory shape: Curved path from initial position to landing site, with glideslope constraint potentially active near the end.

Thrust profile: Due to the minimum thrust constraint, the engine cannot throttle to zero. The optimal profile shows a characteristic *bang-bang* structure: thrust near $T_{\max}$ during initial braking, decreasing to $T_{\min}$ during the coast phase, then returning to $T_{\max}$ for terminal braking. This matches predictions from Pontryagin's Maximum Principle.

Fuel consumption: Optimal fuel consumption is approximately 2,800 to 3,000 kg, corresponding to a final mass of $\sim$22,100 kg.

Time of flight: For the given initial conditions, the optimal landing time is approximately 25-27 seconds.

Implementation Reference

Numerical example in `config.py`:

- `DEFAULT_CONFIG` stores the parameters in Table 2
- `solver.py`: `solve_grid_search()` finds optimal t_f in range $[22, 32]$ s
- `plots.py`: Generates trajectory and time series figures

10 Sensitivity Analyses

10.1 General Sweep Template

Let λ be a scalar parameter (e.g., γ_{gs} or $T_{\min}$). For each λ_i:

1. Modify config to $\lambda = \lambda_i$
2. Solve fixed t_f (typically $t_f = t_f^\star$ from grid search)
3. Record feasibility, objective, fuel, slack metrics

10.2 Glideslope Sweep

Sweep $\gamma_{\text{gs}} \in [3, 20]$ from horizontal. Smaller angles are more permissive (allow more horizontal flight); larger angles are more restrictive (must stay near vertical!).

Result: Fuel consumption increases with more restrictive glideslope, often with a knee where the constraint transitions from inactive to active.

10.3 Minimum Thrust Sweep

Sweep $T_{\min}$ as percentage of $T_{\max}$ (e.g., 20-60%). Higher $T_{\min}$ reduces throttle authority and can cause infeasibility.

Result: Fuel consumption increases with $T_{\min}$. There exists a threshold $\bar{T}_{\min}$ beyond which landing is infeasible for given initial conditions. This demonstrates the "engineering cliff-edge" where small parameter changes cause catastrophic failure.

See animations to visualize some crashes!!

10.4 Initial Altitude Sweep

Sweep $r_{0,z} \in [1000, 4000]$ m. Scale t_f search range proportionally.

Result: Fuel consumption scales roughly linearly with initial altitude. Optimal t_f also scales approximately linearly.

Implementation Reference

Sensitivity analysis in `sensitivity.py`:

- `sweep_glideslope()`, `sweep_tmin()`, `sweep_initial_altitude()`
- `plot_sensitivity_glideslope()`, `plot_sensitivity_tmin()`
- `run_all_sensitivity()` called by `main.py`

11 Extensions and Practical Considerations

11.1 Successive Convexification for Complex Constraints

For problems with aerodynamic drag, attitude dynamics, or obstacle avoidance, successive convexification (SCvx) provides a general framework:

Algorithm 1 Successive Convexification

```
Initialize reference trajectory (x^(0), u^(0))
for i = 1, 2, ... until convergence do
    Linearize nonlinear constraints around (x^(i−1), u^(i−1))
    Solve convex subproblem with trust regions
    Update (x^(i), u^(i)) ← solution
end for
```

Trust regions prevent large deviations:

$$\left\|\boldsymbol{x}_k - \boldsymbol{x}_k^{\text{ref}}\right\| \leq \delta_x, \quad \left\|\boldsymbol{u}_k - \boldsymbol{u}_k^{\text{ref}}\right\| \leq \delta_u. \tag{90}$$

11.2 Minimum-Landing-Error Formulation

If the target cannot be reached exactly (insufficient fuel), minimize:

$$\|\boldsymbol{r}(t_f)\|^2 + \lambda \|\boldsymbol{v}(t_f)\|^2 . \tag{91}$$

This is a convex quadratic objective.

11.3 Robustness to Uncertainty

Real systems face uncertainty in:

- Initial state (navigation errors)
- Thrust magnitude and direction
- Aerodynamic effects
- Gravity model

Approaches include robust optimization (worst-case), stochastic optimization (expected performance), and Model Predictive Control (re-solve at each time step).

11.4 Comparison with Other Guidance Laws

- **Gravity Turn**: Simple but suboptimal; does not minimize fuel.
- **Polynomial Guidance**: Fits trajectory to low-order polynomial; may violate constraints.
- **ZEM/ZEV**: Linear feedback; near-optimal for unconstrained case but cannot handle inequality constraints.
- **Apollo Guidance**: ZEM/ZEV with heuristics; historically successful but not provably optimal.

The convex optimization approach is superior: guarantees global optimality (via lossless convexification), guarantees constraint satisfaction, handles complex constraint geometry, and provides polynomial-time computational guarantees.

12 Conclusion

This note has presented a comprehensive treatment of fuel-optimal powered descent guidance using convex optimization. Key contributions..

1. **Complete problem formulation** with all physical constraints.
2. **Rigor of lossless convexification**, showing the SOCP relaxation is tight at optimality; the optimizer never "wastes" fuel by having slack in the relaxed constraint.
3. **ZOH discretization** preserving convexity with state transition matrices.
4. **Practical algorithms**: grid search for free-final-time and SCP for mass-dependent bounds.
5. **Implementation mapping** from mathematics to Python/CVXPY (see Appendix D).

The lossless convexification technique transforms a fundamentally non-convex problem (one requiring global optimization) into a convex problem with polynomial-time solvability. This enables computation of truly optimal trajectories in real-time, a capability demonstrated operationally in autonomous rocket landings.

Future extensions include incorporating attitude dynamics, handling multiple flight phases, and developing robust formulations for navigation and actuation uncertainty.

A Second-Order Cone Constraint Derivations

A.1 Glideslope as SOC

The glideslope constraint is:

$$r_z \geq \tan(\gamma_{\text{gs}})\sqrt{r_x^2 + r_y^2}. \tag{92}$$

Define $c = \cot(\gamma_{\text{gs}}) = 1/\tan(\gamma_{\text{gs}})$. Then:

$$\|(r_x, r_y)\| \leq c \cdot r_z \iff ((r_x, r_y), c \cdot r_z) \in \mathcal{Q}^3, \tag{93}$$

which is a standard Lorentz cone constraint.

A.2 Pointing Constraint as SOC

The pointing constraint is:

$$u_z \geq \cos(\theta_{\max}) \cdot \sigma. \tag{94}$$

Combined with $\|\boldsymbol{u}\| \leq \sigma$, this is the intersection of a Lorentz cone with a half-space, which remains convex.

B Pontryagin's Maximum Principle Analysis

For the continuous-time problem, PMP provides necessary optimality conditions. The Hamiltonian is:

$$H = \sigma + \boldsymbol{\lambda}_r^\top \boldsymbol{v} + \boldsymbol{\lambda}_v^\top (\boldsymbol{u} + \boldsymbol{g}) + \lambda_z(-\alpha\sigma). \tag{95}$$

B.1 Co-state Dynamics

$$\dot{\boldsymbol{\lambda}}_r = -\frac{\partial H}{\partial \boldsymbol{r}} = \mathbf{0}, \tag{96}$$

$$\dot{\boldsymbol{\lambda}}_v = -\frac{\partial H}{\partial \boldsymbol{v}} = -\boldsymbol{\lambda}_r, \tag{97}$$

$$\dot{\lambda}_z = -\frac{\partial H}{\partial z} = 0. \tag{98}$$

Thus $\boldsymbol{\lambda}_r$ and λ_z are constant, and $\boldsymbol{\lambda}_v(t) = \boldsymbol{\lambda}_v(0) - \boldsymbol{\lambda}_r t$.

B.2 Optimal Control Law

The Hamiltonian is linear in $\boldsymbol{u}$ and σ.

Thrust direction: Since H contains $\boldsymbol{\lambda}_v^\top \boldsymbol{u}$, the optimal $\boldsymbol{u}$ points opposite to $\boldsymbol{\lambda}_v$ (subject to pointing constraint):

$$\boldsymbol{u}^* = -\sigma \frac{\boldsymbol{\lambda}_v}{\|\boldsymbol{\lambda}_v\|}. \tag{99}$$

Thrust magnitude: The coefficient of σ in H is the switching function:

$$S(t) = 1 - \alpha\lambda_z - \|\boldsymbol{\lambda}_v(t)\|. \tag{100}$$

- If $S(t) < 0$: $\sigma = \rho_{\max}$ (maximum thrust)
- If $S(t) > 0$: $\sigma = \rho_{\min}$ (minimum thrust)
- If $S(t) = 0$: Singular arc (intermediate thrust)

This analysis explains the bang-bang thrust profile observed in numerical solutions.

C CVXPY Implementation Schematic

```
import cvxpy as cp
import numpy as np

# Decision variables
r = cp.Variable((N+1, 3))
v = cp.Variable((N+1, 3))
z = cp.Variable(N+1)
u = cp.Variable((N, 3))
sigma = cp.Variable(N)

# Objective
objective = cp.Minimize(cp.sum(sigma) * dt)

# Constraints
constraints = [
    r[0] == r0, v[0] == v0, z[0] == np.log(m0),
    r[N] == 0, v[N] == 0,
    z >= np.log(m_dry), z <= np.log(m0)
]

for k in range(N):
    constraints += [
        # Dynamics
        r[k+1] == r[k] + v[k]*dt + 0.5*(u[k] + g)*dt**2,
        v[k+1] == v[k] + (u[k] + g)*dt,
        z[k+1] == z[k] - alpha*sigma[k]*dt,
        # SOC constraints
        cp.norm(u[k]) <= sigma[k],
        sigma[k] >= rho_min, sigma[k] <= rho_max,
        u[k,2] >= cos_point * sigma[k],  # Pointing
        r[k,2] >= tan_gs * cp.norm(r[k,:2])  # Glideslope
    ]

prob = cp.Problem(objective, constraints)
prob.solve(solver=cp.CLARABEL)
```

C.1 Common Pitfalls

- **Numerical scaling**: Normalize variables to $O(1)$ (e.g., position in km)
- **Infeasibility**: Usually indicates insufficient propellant or too restrictive t_f range
- **Solver accuracy**: Use Clarabel or ECOS for tight tolerances; SCS may report spurious infeasibility

D Script-to-Notation Mapping

Python Symbol	Symbol	Description
`config.m0`	m_0	Initial mass
`config.m_dry`	m_{dry}	Dry mass
`config.Isp`	I_{sp}	Specific impulse
`config.T_max`	$T_{\max}$	Maximum thrust
`config.T_min`	$T_{\min}$	Minimum thrust
`config.g0`	g_0	Gravitational acceleration
`config.gamma_gs_deg`	γ_{gs}	Glideslope angle (deg, from horizontal)
`config.theta_max_deg`	$\theta_{\max}$	Pointing angle (deg, from vertical)
`config.r0`	$\boldsymbol{r}_0$	Initial position
`config.v0`	$\boldsymbol{v}_0$	Initial velocity
`config.v_e`	v_e	Exhaust velocity
`config.alpha`	α	Mass depletion rate
`config.tan_gs`	$\tan(\gamma_{\text{gs}})$	Glideslope tangent
`config.cos_point`	$\cos(\theta_{\max})$	Pointing cosine
`N`	N	Number of discretization steps
`tf`	t_f	Final time
`dt`	Δt	Time step
`r[k]`	$\boldsymbol{r}_k$	Position at step k
`v[k]`	$\boldsymbol{v}_k$	Velocity at step k
`u[k]`	$\boldsymbol{u}_k$	Thrust acceleration at step k
`sigma[k]`	σ_k	Thrust acceleration magnitude at step k
`z[k]`	z_k	Log-mass at step k

Table 3: One-to-one mapping from Python code to mathematical notation.

References

Açıkmeşe, B. and Ploen, S. R. (2007). Convex programming approach to powered descent guidance for Mars landing. *Journal of Guidance, Control, and Dynamics*, 30(5):1353–1366.

Açıkmeşe, B., Carson, J. M., and Blackmore, L. (2013). Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem. *IEEE Transactions on Control Systems Technology*, 21(6):2104–2113.

Blackmore, L., Açıkmeşe, B., and Scharf, D. P. (2010). Minimum-landing-error powered-flight guidance for Mars landing using convex optimization. *Journal of Guidance, Control, and Dynamics*, 33(4):1161–1171.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Mao, Y., Szmuk, M., and Açıkmeşe, B. (2016). Successive convexification of non-convex optimal control problems and its convergence properties. In *IEEE Conference on Decision and Control*, pages 3636–3641.

Szmuk, M. and Açıkmeşe, B. (2018). Successive convexification for 6-DoF Mars rocket powered landing with free-final-time. In *AIAA Guidance, Navigation, and Control Conference*.