Climb Trajectory Simulation

1) Model Scope & Assumptions

- Mission segment: climb from initial altitude h_0 to a fixed target altitude h_{target} .
- State variables: altitude h [m], true airspeed V [m/s], mass m [kg], time t [s].
- Controls: a commanded specific-energy magnitude $\dot{E}_{\text{cmd}}$ [m/s] and a strategy that allocates it between climb and acceleration via weights (w_c, w_s) with $w_c + w_s = 1$.
- Atmosphere: `Atmosphere` provides $T, P, \rho, g(h)$. Speed of sound $a = \sqrt{\gamma R T}$ with $\gamma = 1.4, R = 287.05\ \mathrm{J/(kg\ K)}$.
- Aerodynamics: quasi-steady lift balance to compute C_L (During the calculation of the Cl it was assumed that the L=W, as the AOA of the AC is small); parabolic drag polar $C_D = C_{D0} + \frac{C_L^2}{\pi A R e}$.
- Propulsion: `pyengine` gives per-engine thrust for given lever $[0, 1]$, Mach, and altitude (ft). TSFC used for fuel burn.
- Engines: total thrust $T_{\text{total}} = N_{\text{eng}} \cdot T_{\text{per-eng}}$.

---

2) Specific Energy Formulation

Specific energy (per unit mass):

$$E = h + \frac{V^2}{2 g_0}$$

Rate of change:

$$\dot{E} = \frac{dh}{dt} + \frac{V}{g}\frac{dV}{dt}$$

Strategy split (normalized): a strategy returns raw weights (c_w, s_w) . They are normalized to (w_c, w_s) with $w_c + w_s = 1$. With commanded $\dot{E}_{\text{cmd}}$:

$$\frac{dh}{dt} = w_c\, \dot{E}_{\text{cmd}}, \qquad \frac{dV}{dt} = \frac{g}{V}\, w_s\, \dot{E}_{\text{cmd}}$$

Constant-Mach marker:

Starting from $M = V/a$ with $a = \sqrt{\gamma R T}$, differentiation gives:

$$\frac{dM}{dt} = \frac{1}{a}\frac{dV}{dt} - \frac{V}{a^2}\frac{da}{dt}$$

Solving for $\dot{V}$:

$$\frac{dV}{dt} = a\frac{dM}{dt} + \frac{V}{a}\frac{da}{dt}$$

Since $\frac{da}{dT} = \frac{a}{2T}$ and $\frac{da}{dt} = \frac{a}{2T}\frac{dT}{dh}\frac{dh}{dt}$, the general relation becomes:

$$\frac{dV}{dt} = a\frac{dM}{dt} + \frac{V}{2T}\frac{dT}{dh}\frac{dh}{dt}$$

For the constant-Mach case $(\dot{M} \approx 0)$:

$$\boxed{\frac{dV}{dt} = \frac{V}{2T}\frac{dT}{dh}\frac{dh}{dt}}$$

with $\frac{dT}{dh}$ evaluated numerically from the atmosphere model.

---

3) Aerodynamics

Lift balance:

$$C_L = \frac{2W}{\rho V^2 S}, \qquad W = m\,g(h)$$

Drag polar:

$$C_D = C_{D0} + \frac{C_L^2}{\pi A R e}$$

Drag:

$$D = \tfrac{1}{2}\rho V^2 S C_D$$

Constants in the current code: $S = 122.4\,\mathrm{m}^2$, $C_{D0} = 0.02$, $AR = 9.5$, $e = 0.85$.

(Aircraft Design lecture notes are taken as reference here)

---

4) Power Balance and Required Thrust

Purpose:
Relates the aircraft's aerodynamic drag, its rate of change of specific energy, and the thrust required to sustain the commanded climb/acceleration profile.

4.1 Excess Power Concept

The **excess power per unit weight** formulation comes from the aircraft energy balance:

$$\frac{(T - D)\,V}{W} = \dot{E}$$

Where:

- T = total thrust [N]
- D = total aerodynamic drag [N]
- V = true airspeed [m/s]
- $W = m \cdot g$ = aircraft weight [N]
- $\dot{E}$ = rate of change of **specific energy height** [m/s]

This equation states:

- **Left-hand side:** net propulsive power available per unit weight (power = force × velocity).
- **Right-hand side:** rate at which the aircraft's total specific energy changes.

4.2 Specific Energy Height

Specific energy height is defined as:

$$E = h + \frac{V^2}{2g}$$

Where:

- h = altitude [m]
- V = true airspeed [m/s]
- g = gravitational acceleration [m/s²]

Its time derivative is:

$$\dot{E} = \frac{dh}{dt} + \frac{V}{g} \cdot \frac{dV}{dt}$$

- $\frac{dh}{dt}$ = climb rate [m/s]
- $\frac{dV}{dt}$ = acceleration [m/s²]

These rates are determined by the strategy function (via w_c and w_s) and the commanded specific energy rate $E_{\text{DOT\_cmd}}$, then adjusted for special cases like constant-Mach climbs.

4.3 How $\dot{E}$ is Set and Used

1. **Commanded Specific Energy Rate ($E_{\text{DOT\_cmd}}$)**
   A fixed target magnitude of specific energy change (currently $6.5\ \mathrm{m/s}$) defines how aggressively the climb profile should change total specific energy height.
2. **Split into Climb and Speed Shares**
   The active strategy function returns raw weights (c_w, s_w) , which are normalized so $w_c + w_s = 1$.
   - Climb rate: $\frac{dh}{dt} = w_c \cdot E_{\text{DOT\_cmd}}$
   - Acceleration rate: $\frac{dV}{dt} = \frac{g}{V} \cdot (w_s \cdot E_{\text{DOT\_cmd}})$ in standard mode, or adjusted in constant-Mach mode.
3. **Actual Specific Energy Rate ($\dot{E}$)**
   Once $\frac{dh}{dt}$ and $\frac{dV}{dt}$ are determined, the actual rate of change of specific energy is calculated:

   $$\dot{E} = \frac{dh}{dt} + \frac{V}{g} \cdot \frac{dV}{dt}$$

   This value may differ from $E_{\text{DOT\_cmd}}$ due to constant-Mach logic, thrust limitations, or numerical effects.

4. **Use in Power Balance**
   The calculated $\dot{E}$ is inserted into the power balance equation:

$$F_{\text{req}} = D + \frac{W \cdot \dot{E}}{V}$$

   This produces the total thrust requirement for the current state, which is then sent to the lever solver to determine the appropriate engine lever position.

4.4 Solving for Required Thrust

1. Start from:

$$\frac{(T - D)V}{W} = \dot{E}$$

2. Multiply through by W/V :

$$T - D = \frac{W \cdot \dot{E}}{V}$$

3. Add D to both sides:

$$\boxed{F_{\text{req}} = D + \frac{W \cdot \dot{E}}{V}}$$

4.5 Physical Interpretation

- **Drag term (D)**: thrust required to overcome aerodynamic resistance at the current speed.
- **Energy term ($\frac{W \cdot \dot{E}}{V}$)**: additional thrust required to produce the commanded climb rate and/or acceleration.
- **Sum**: total thrust required from all engines to achieve the desired climb/acceleration state.

---

5) Engine Model and Lever Solve

Purpose:
Given the current flight state (altitude, Mach) and the total thrust demand F_{req} from the power balance, determine the **engine lever position** $\ell \in [0, 1]$ that delivers the required thrust per engine.

5.1 Engine Query Bounds

Concept:
The engine performance tables are only valid inside a fixed envelope:

$$\text{Mach range: } [0.00,\ 0.94], \quad \text{Altitude range: } [0,\ 14{,}000] \text{ ft}.$$

Before any engine table lookup, Mach and altitude are **clipped** to these ranges to avoid invalid queries.

Code logic:

- The clipping in your current structure is performed **before** calling the lever solver inside `simulate_climb_path()`.
- The solver itself assumes Mach and altitude are already safe and directly queries the engine model.

5.2 Convert Demand to Per-Engine Thrust

Concept:
With N_{eng} engines installed, the total required thrust is split evenly:

$$T_{\text{req,per}} = \frac{F_{\text{req}}}{N_{\text{eng}}}.$$

This is the target thrust **per engine** that the solver will try to match.

Code logic:

```
T_req = float(required_thrust_total) / float(N_ENGINES)
```

This value is used throughout the solver to compare against thrust table results.

5.3 Idle and Max Thrust Checks

Concept:
Check whether the required thrust is already satisfied at idle, or is beyond maximum capability.

Code logic:

1. **Idle check** (`lever=0.0`):
   - `T_idle = eng.get_thrust_with_lever_position(0.0, mach, alt_ft)`
   - If `T_idle >= T_req` → return `(0.0, T_idle)`.
2. **Max check** (`lever=1.0`):
   - `T_max = eng.get_thrust_with_lever_position(1.0, mach, alt_ft)`
   - If `T_max < T_req` → return `(1.0, T_max)` and set thrust-limited flag.

This step is a **fast exit** and mirrors real FADEC behavior.

5.4 Sampling Thrust vs. Lever

Concept:
If the required thrust is between idle and maximum available thrust, the simulation mimics a **FADEC-like** process: it samples thrust output at several lever positions for the current Mach and altitude, then determines the lever that best matches the thrust demand.

Code logic:

- Define a fixed **lever grid** that spans the full range from idle to full throttle.
  - In the updated version, `lever_grid = np.linspace(0.0, 1.0, 21)` produces 20 points between 0 and 1.
- For each lever value in the grid:

```
thrusts = [safe_thrust(lv) for lv in lever_grid]
```

where `safe_thrust()` queries the engine model (`eng.get_thrust_with_lever_position`) at the current Mach and altitude, and returns `None` if the data is missing, non-finite, or negative.

- Weakly enforce **non-decreasing thrust** vs. lever to smooth out small table noise from the engine model.
- Store only valid `(lever, thrust)` pairs to use in interpolation. Invalid points are skipped but endpoints are checked explicitly for idle and maximum conditions.
- Once the sampled thrust data is collected:
  - If idle meets or exceeds the demand → select lever = 0.0.
  - If maximum available thrust is still below demand → select lever at maximum available thrust and flag the case as **thrust-limited**.
  - Otherwise, find the **bracketing interval** around the required thrust and use linear interpolation (plus one optional refine query) to determine the lever position that meets demand.
- If no clean bracket is found due to gaps in valid data, select the **closest valid lever** in terms of thrust difference.

5.5 Linear Interpolation

Concept:
Identify the first grid interval where the required thrust lies between two valid thrust points, and interpolate linearly to estimate the lever.

Code logic:

```
for i in range(len(lever_grid) - 1):
    if T_i <= T_req <= T_ip1:
        lv_star = li + (T_req - Ti) * (lj - li) / (Tip1 - Ti)
```

This produces a continuous lever value ℓ^* between the two grid points.

5.6 Optional Refine

Concept:
Make one more engine call at the interpolated lever to align the thrust and TSFC with the actual lever setting.

Code logic:

```
if allow_refine:
    Tstar = safe_thrust(lv_star)
    if Tstar is not None:
        return lv_star, Tstar
```

If refinement fails, the solver falls back to whichever endpoint is closer in thrust.

5.7 Fallback

Concept:
If no bracketing interval is found (e.g., due to table holes), pick the lever with the smallest thrust error.

Code logic:

```
diffs = [(abs(T - T_req), lv, T) for lv, T in valid_points]
diffs.sort(key=lambda x: x[0])
return diffs[0][1], diffs[0][2]
```

This ensures the solver always returns something unless all data is invalid.

5.8 TSFC Alignment

Concept:
After deciding the lever, ensure the TSFC is read at that exact lever setting.

Code logic:

- Call `eng.get_tsfc()` immediately after the final thrust query.
- Unit heuristic: if `TSFC > 1e-3`, assume kg/(N·hr) and convert to kg/(N·s) by dividing by 3600.

---

6) Fuel Burn and Mass Update

Per-engine TSFC is read from the engine at the current state. If it appears to be in kg/(N·hr), it is divided by 3600 to convert to kg/(N·s).

Total fuel flow:

$$\dot{m}_{\text{fuel,tot}} = N_{\text{eng}} \cdot TSFC \cdot T_{\text{per-eng}}.$$

Mass update:

$$m_{k+1} = \max(m_k - \dot{m}_{\text{fuel,tot}}\, dt,\ 0)\,.$$

---

7) Time Integration and Termination

Explicit Euler with fixed dt (default 0.2 s):

$$h_{k+1} = h_k + \frac{dh}{dt}\, dt, \qquad V_{k+1} = V_k + \frac{dV}{dt}\, dt.$$

If h_{k+1} would overshoot h_{target}, a partial step is used:

$$dt_{\text{last}} = \frac{h_{\text{target}} - h_k}{\max\left(\frac{dh}{dt}, 10^{-9}\right)},$$

and t, V are advanced with dt_{last}. Final h is set exactly to h_{target}.

---

8) Strategy Profiles

All strategies return raw (c_w, s_w); they are normalized internally to (w_c, w_s).

FixedEnergy.Linear.profile(h, V, af):

$$c_w = af, \qquad s_w = 1 - af.$$

FixedEnergy.Exponential (with $h_t = h_{\text{target}}$, $af \in (0,1)$):

- Increasing climb:

$$c_w = af\, e^{h/h_t}, \qquad s_w = (1 - af)\, e^{-h/h_t}.$$

- Decreasing climb:

$$c_w = af\, e^{-h/h_t}, \qquad s_w = (1 - af)\, e^{h/h_t}.$$

- Increasing speed:

$$s_w = af\, e^{h/h_t}, \qquad c_w = (1 - af)\, e^{-h/h_t}.$$

- Decreasing speed:

$$s_w = af\, e^{-h/h_t}, \qquad c_w = (1 - af)\, e^{h/h_t}.$$

ConstantRates.constant_speed:

- Returns $(1, 0)$ (all energy into climb). After normalization, $dV/dt = 0$.

ConstantRates.constant_mach_marker:

- Returns a tagged function that signals the integrator to use the constant-Mach kinematics (the special dV/dt relation above).

generate_strategy(profile):

- For `"linear"` and all `"exponential_*"`: generates five scenarios with $af \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.
- For `"constant_speed"` and `"constant_mach"`: returns a single scenario (with `af=None`).
- Lambdas bind `af` via default arguments to avoid late binding.

---

9) Engine Call Envelope

Before every engine query:

$$M \leftarrow \mathrm{clip}(M, 0.00, 0.94), \qquad h_{\mathrm{ft}} \leftarrow \mathrm{clip}(h\,[\mathrm{ft}], 0, 14000).$$

---

10) Diagnostics and Edge Cases

- Lever-limit markers: timestamps when lever $= 1.0$ (computed or clamped) are stored in `limit_times` and highlighted in the plots.
- No engine data: if thrust cannot be obtained at both bounds, or repeated failures occur in the table interior, the solver returns the closest valid value or `None`; times are stored in `none_lever_times` and marked.
- Numerical guards: divisions use `max(x, tiny)` (e.g., for V, ρ, $\frac{dh}{dt}$); TSFC unit heuristic avoids hr-to-s mistakes.

---

11) simulate_climb_path Outputs

Return tuple:

1. `t` — time [s]
2. `h` — altitude [m]
3. `V` — velocity [m/s]
4. `lever_positions` — lever time series (floats or `None`)
5. `final_results` (dict):
   - `"Final Altitude"` [m]
   - `"Final Velocity"` [m/s]
   - `"Total Climb Time"` [s]
   - `"Final Lever Position"`
   - `"Final Mass (kg)"`

- `"Total Fuel Burned (kg)"`
- `"Engines"` (int)

6. `diagnostics` (dict):
    - `"altitudes"`, `"velocities"`, `"times"`
    - `"lever_positions"`, `"none_lever_times"`, `"limit_times"`
    - `"fuel_flow_kg_s"` (total), `"fuel_burn_step_kg"`
    - `"mass_kg"` (aligned; last omitted to match `t`)

---

12) Parameters and Constants (current values)

- `N_ENGINES = 2`
- `S_ref = 122.4` m^2
- `CD0 = 0.02`
- `AR = 9.5`
- `e = 0.85`
- `initial_mass_kg = 60000.0`
- `initial_altitude = 0` m
- `initial_speed = 75` m/s
- `target_altitude = 4267.2` m
- `dt = 0.2` s
- `altitude_fractions = linspace(0.1, 0.9, 5)`
- Engine envelopes: Mach $[0, 0.94]$, altitude $[0, 14000]$ ft
- $\dot{E}_{\mathrm{cmd}}$ is currently set as a constant inside the integrator (6.5 m/s).

---

13) Known Behaviors

- Lever pinned at 1.0: thrust-limited timestep. Many such points suggest the scenario is not feasible for that profile/AF.
- `none_lever_times` non-empty: engine map did not provide thrust for the queried condition; the solver returned the closest valid lever or `None`.

Literature Survey

Summary of Climb Performance Concepts (Raymer, Chapter 17.3)

This summary consolidates key concepts and equations related to **steady climb and descent** from Daniel Raymer's *Aircraft Design: A Conceptual Approach*, with a focus on climb gradient, best angle/rate of climb, and time/fuel to climb.

Steady Climbing Flight and Climb Gradient

- **Climb gradient** G is the ratio of vertical to horizontal distance traveled.
- It is equivalent to $\sin(\gamma)$, where γ is the climb angle:

$$\gamma = \sin^{-1}\left(\frac{T-D}{W}\right) = \sin^{-1}\left(\frac{T}{W} - \frac{1}{L/D}\right) \qquad \text{(Eq. 17.38)}$$

- The **vertical velocity** or **rate of climb** V_v is:

$$V_v = V\sin(\gamma) = V\sqrt{\frac{T}{W} - \frac{1}{L/D}} \qquad \text{(Eq. 17.39)}$$

- Force balances used:

$$\sum F_x = T - D - W\sin(\gamma) \qquad \text{(Eq. 17.6)}$$

$$\sum F_z = L - W\cos(\gamma) \qquad \text{(Eq. 17.7)}$$

Graphical Method: Best Angle and Rate of Climb

- **Best rate of climb** maximizes vertical velocity V_v.

- **Best angle of climb** maximizes altitude gain per unit horizontal distance (i.e., max γ).
- Plot V_v vs airspeed (using Eq. 17.39) and superimpose thrust/drag data to identify:
  - **Peak of the curve**: Best rate of climb.
  - **Tangency from origin**: Best angle of climb.
    (Refer to Fig. 17.4 in Raymer)

Jet Aircraft: Best Climb Conditions

- For jets, thrust T is mostly constant with speed.
- Best rate of climb is found by maximizing:

$$V_v = V\left(\frac{T}{W} - \frac{\rho V^2 C_D}{2(W/S)} - \frac{2K}{\rho V}\left(\frac{W}{S}\right)\right) \qquad \text{(Eq. 17.42)}$$

- Setting $\frac{dV_v}{dV} = 0$ and solving gives:

$$V = \sqrt{\frac{W/S}{3\rho C_{D_0}}\left(\frac{T}{W} + \sqrt{\left(\frac{T}{W}\right)^2 + 12C_{D_0}K}\right)} \qquad \text{(Eq. 17.43)}$$

- Example: The B-70 has a best climb speed of 583 kt (≈1080 km/h).

Time and Fuel to Climb

- Time to climb a small height dh:

$$dt = \frac{dh}{V_v} \qquad \text{(Eq. 17.46)}$$

- Fuel burn over that time:

$$dW_f = -C_T T\, dt \qquad \text{(Eq. 17.47)}$$

- Since V_v varies with altitude, it can be linearly approximated:

$$V_v = V_{v_i} - a(h_{i+1} - h_i) \tag{Eq. 17.48}$$

$$a = \frac{V_{v_2} - V_{v_1}}{h_2 - h_1} \tag{Eq. 17.49}$$

- Total time and fuel between two altitudes:

$$t_{i+1} - t_i = \frac{1}{a} \ln\left(\frac{V_{v_i}}{V_{v_{i+1}}}\right) \tag{Eq. 17.50}$$

$$\Delta W_{\text{fuel}} = -(CT)_{\text{avg}}(t_{i+1} - t_i) \tag{Eq. 17.51}$$

- Improved accuracy can be achieved via **iteration**, updating W after each step.

Reference

Raymer, D. P. (2021). *Aircraft Design: A Conceptual Approach*, 6th ed., AIAA Education Series, Chapter 17.3.

Future Considerations and Open Questions

As development continues, several physical constraints and operational factors need to be addressed:

1. Thrust Limitations

- The climb energy rate is ultimately limited by the available engine thrust at a given altitude.
- To model this accurately, engine performance data (e.g., thrust vs. altitude) is needed.

2. TSFC Clarification

- The thrust-specific fuel consumption (TSFC) is often provided in units such as lb fuel / lb thrust / hr.
- A consistent unit system should be defined, and conversions should be handled clearly for modeling fuel flow.

3. Angle of Attack Constraints

- The maximum achievable angle of attack limits climb steepness and lift.
- For steady climb, the angle of attack can be derived using Raymer's Equation 17.38 (refer to “Aircraft Design: A Conceptual Approach”).

4. Scenario Planning

We may consider simulating under different mission or design contexts:

- Minimum fuel climb
- Minimum time climb
- Constant Mach
- Engine-out or degraded thrust condition

5. Boundary Conditions

We should consider the boundary conditions defined for each segment.