FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

# Atmosphärenbremsung bei Marsmissionen

## Erstellung eines Modells zur Simulation von Atmosphärenbremsung

Jason Cedric Kalscheuer
FH Aachen
Fachbereich 6, Luft- und Raumfahrttechnik

# Bedeutung der Atmosphärenbremsung

- Erschließung des Mars als nächstes großes Raumfahrtziel
- Ankunftsgeschwindigkeit > Zielgeschwindigkeit
- Geschwindigkeitsreduktion benötigt mehrere Tonnen Treibstoff (Magellan-Sonde)
- Einsparmöglichkeiten für Treibstoff als Forschungsziel
- Luftwiderstand der Atmosphäre ist eine naheliegende Möglichkeit zur Energiedissipation
- Computersimulationen essenziell für den Erfolg solcher Manöver

# Überblick über das Programm

- Einteilung in elf Regionen

```
#region User input···

#region Prepare the program···

#region Atmosphere···

#region Mission Phases···

#region Equations of Motion···

#region Simulation termination conditions···

#region Simulate Trajectory···

#region Iterate B-Plane-Offset···

#region Optimize Thrustduration···

#region Final Simulation of best configuarations···

#region Multithreading & Plots···
```

# Überblick über das Programm

- Berechnungszeit extrem abhängig von der Genauigkeit der Berechnung

```python
#region Iterate B-Plane-Offset
def find_optimal_b(initial_b, r_p):
    global best_fuel_usage, best_b, best_r_thrust_descend, best_r_thrust_ascend, best_configs, best_trajectory

    b = initial_b
    tolerance = 0.2  # Allowed difference between r_p and 2nd periapsis of trajectory in km
```

```python
#region Final Simulation of best configuarations
def run_full_trajectory(config):
    global stable_orbit_time, aerobrake, slowdown_start_time, stabilization_start_time

    r_p = config['r_p']
    b = config['b']
    r_thrust_descend = config['r_thrust_descend']
    r_thrust_ascend = config['r_thrust_ascend']
    r_p_slow = config['r_p_slow']
    thrust_only = config['thrust_only']

    # Set aerobrake to false if it's a thrust_only run
    if thrust_only:
        original_aerobrake = aerobrake
        aerobrake = False

    final_sim_start_time = tm.time()
    if r_p-R <= 80:
        _, _, full_trajectory, _, final_mass = simulate_trajectory(b, r_thrust_descend, r_thrust_ascend, r_p_slow, short_run=False, max_step=0.2)
    elif r_p-R <=90:
        _, _, full_trajectory, _, final_mass = simulate_trajectory(b, r_thrust_descend, r_thrust_ascend, r_p_slow, short_run=False, max_step=0.3)
    elif r_p-R <=95:
        _, _, full_trajectory, _, final_mass = simulate_trajectory(b, r_thrust_descend, r_thrust_ascend, r_p_slow, short_run=False, max_step=0.5)
    else:
        _, _, full_trajectory, _, final_mass = simulate_trajectory(b, r_thrust_descend, r_thrust_ascend, r_p_slow, short_run=False, max_step=1.25)
```

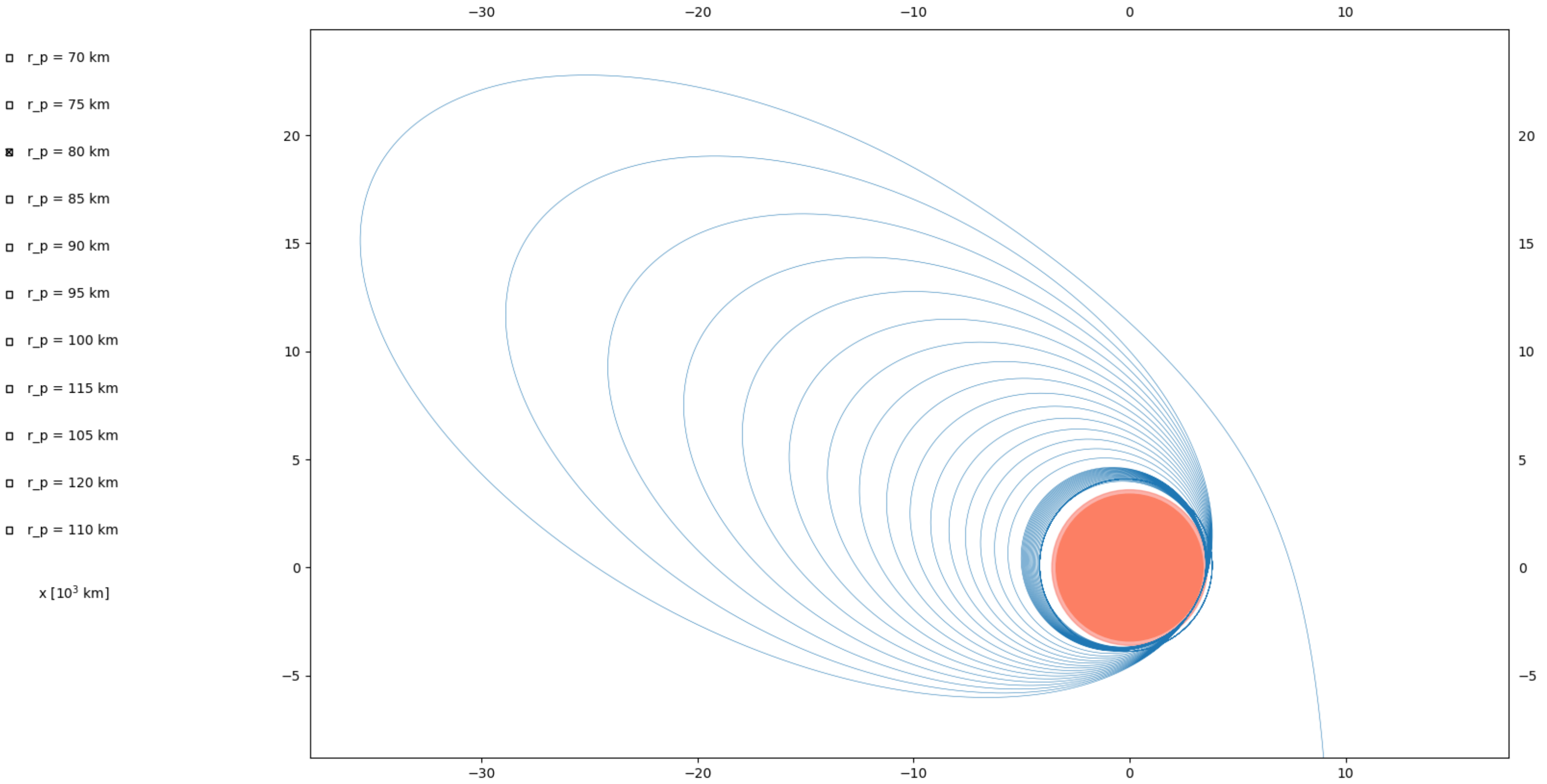
# Benutzereingabe

- Benötigt 3 Arten von Eingaben: Raketenparameter, Orbitparameter und Ausgabeeinstellungen

```
#region User input
# === Rocketparameters ===========================
m_0                          = 3725.0          # Initial spacecraft mass in [kg]
m_p                          = 3000.0          # Propellant mass in [kg]
thrust                       = 425.0           # Thrust in [N]
c_3                          = 13.78           # Characteristic energy in [km^2/s^2]
Isp                          = 321.0           # Specific impulse
c_d                          = 2.2             # Drag coefficient
A                            = 29.3            # Cross-sectional area of the spacecraft in [m^2]
# === Orbitparameters ===========================
tint                         = 175             # Integration time in days
r_p_lowest                   = 75              # Lowest Periapsis tested
r_p_highest                  = 96              # Highest Periapsis tested
r_p_step_size                = 5               # Step size between r_p_lowest and r_p_highest
r_p_orbit                    = 400             # Periapsis of desired Orbit in [km]
r_a                          = 800             # Apoapsis of desired Orbit in [km]
r_a_limit_aero               = 150000          # Apoapsis limit to avoid excessive orbit duration for aerobraking runs in [km]
r_a_limit_thrust             = 300000          # Apoapsis limit to avoid excessive orbit duration for thrust_only run in [km]
# === Outputparameters ===========================
Plot_Trajectory              = False           # Plot spacecraft trajectory?
Plot_Atmosphere              = False           # Plot atmospheric density over height?
Plot_Values                  = True            # Plot panel of values?
Orbit_Count                  = False           # Plot over orbit count if True, otherwise over time
Optimize_Thrust_Range        = True            # Optimize R_Thrust_Ascend?
Thrust_Only_Run              = True            # Calculate Thrust Only Run?
Plot_Comparison              = True            # Plot propellant consumed vs time to stable orbit?
# === End user input    ===========================
#endregion
```

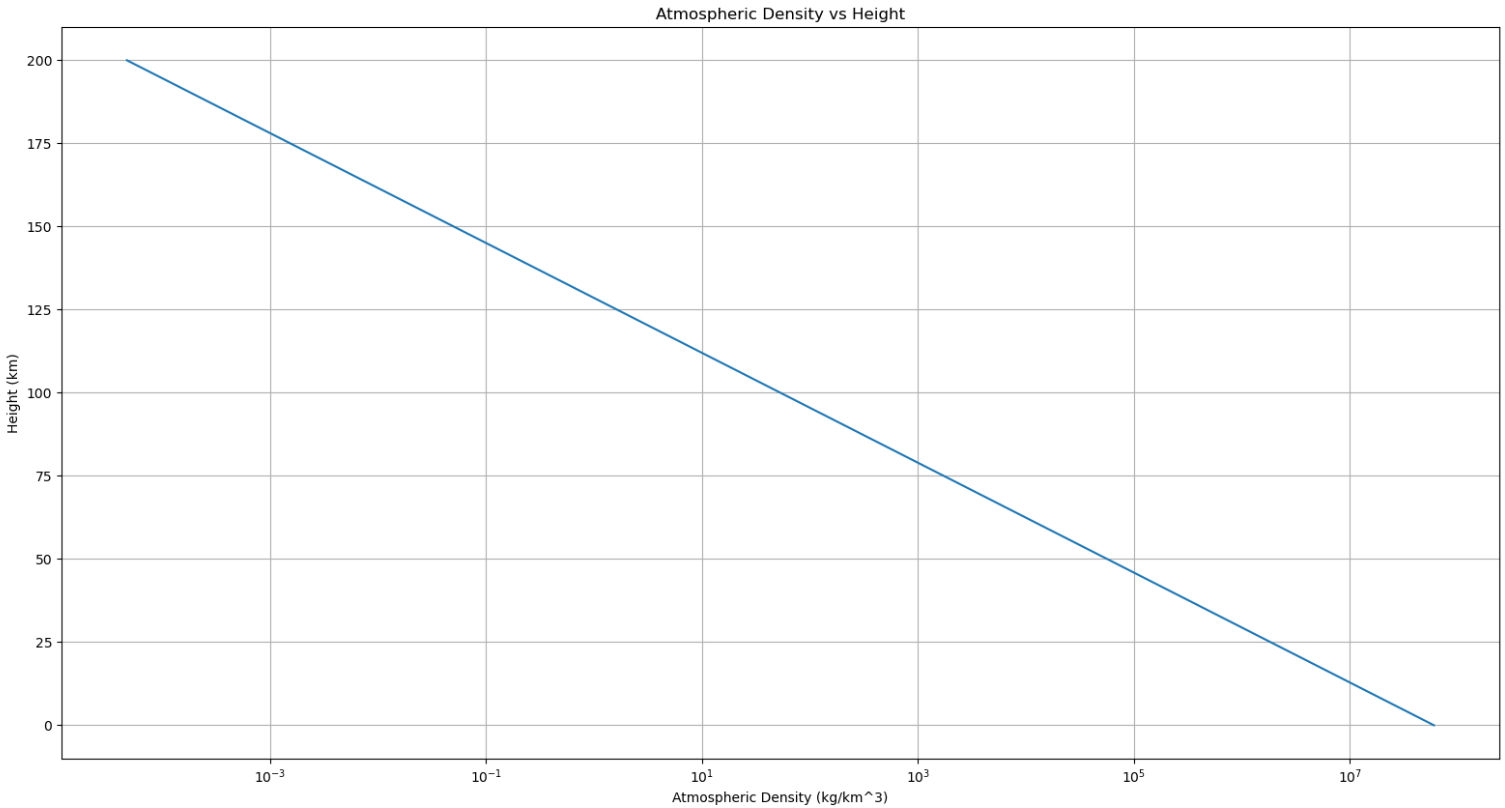
# Benutzereingabe

- Flugbahn anzeigen



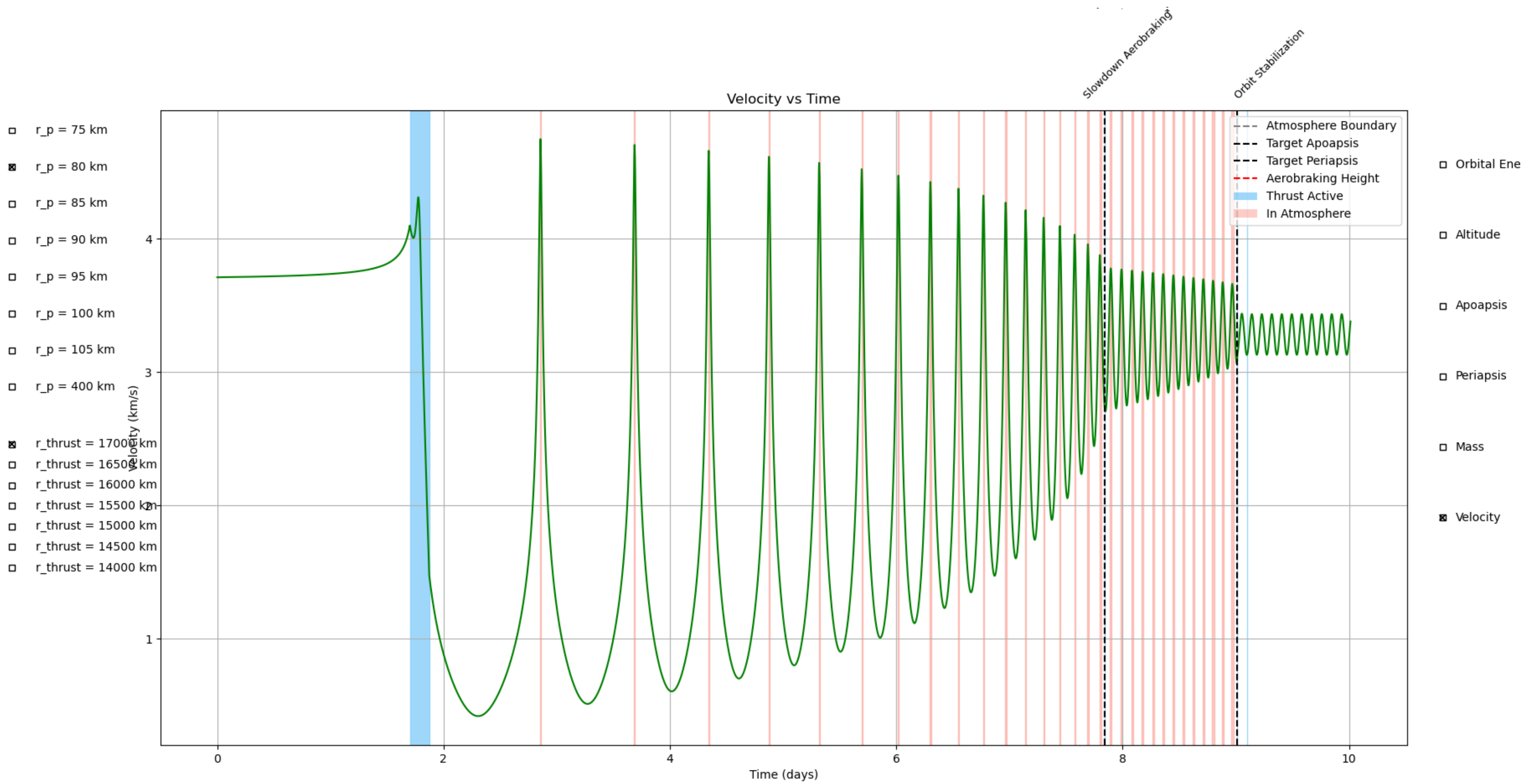
# Benutzereingabe

- Atmosphärenmodell anzeigen



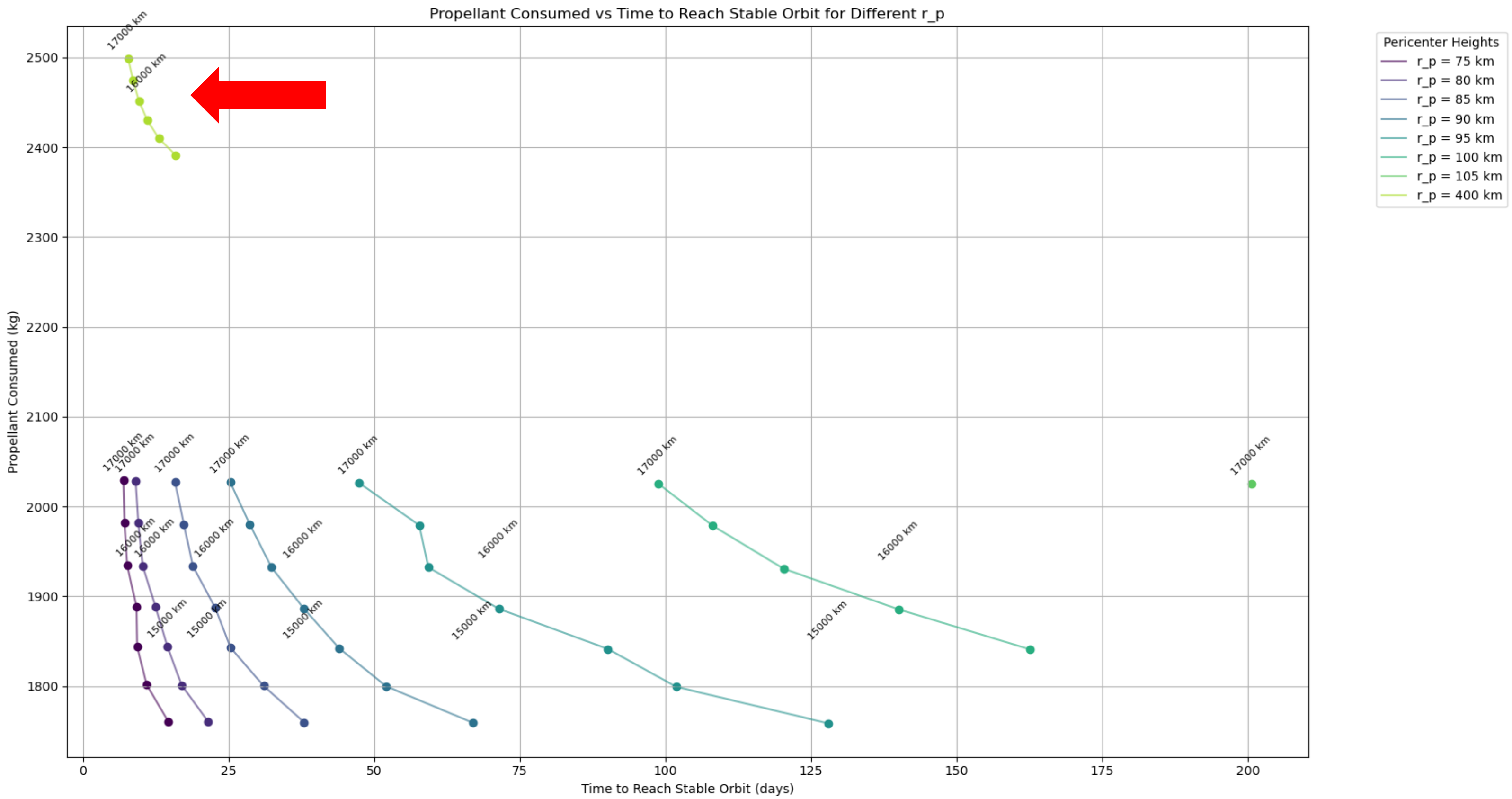
# Benutzereingabe

- Datenverlauf anzeigen



# Benutzereingabe

- Schubdauerreduzierung & Referenzsimulation anzeigen



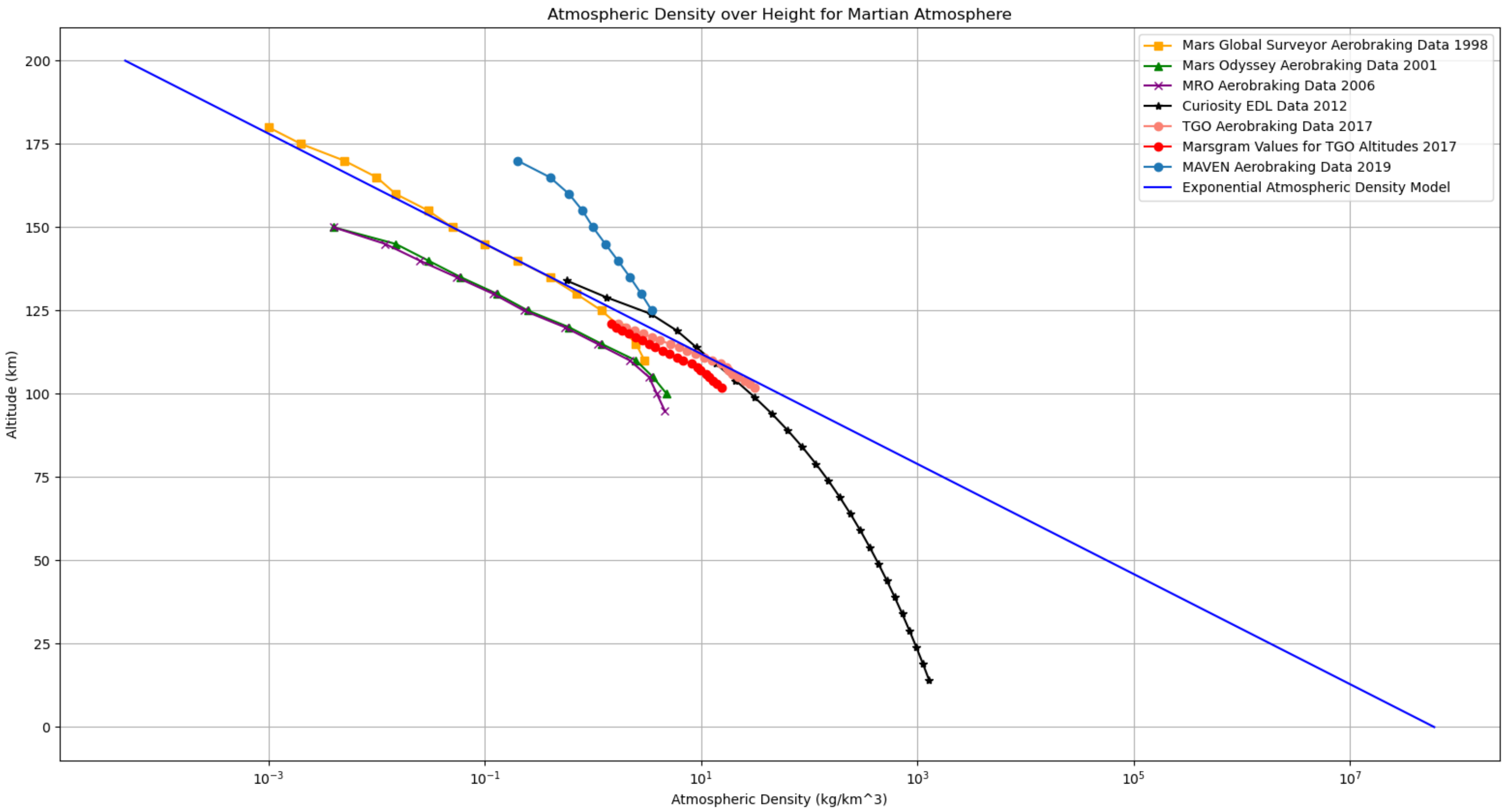
# Atmosphärenmodell

- Einfaches Exponentialmodell
- Benötigte Angaben: Höhe der Atmosphäre, Dichte auf Marsoberfläche und in bestimmter Höhe (195km)

```python
#region Atmosphere
atmo_height     = 200                                        # Height of the atmosphere boundary in [km]
def atmospheric_density(r):
    height          = r - R                                  # Altitude above Mars' surface in km
    atmo_density    = 0.06 * 1e9                             # Mars Atmosphere density in [kg/m^3]
    atmo_density_200= 0.095 * 1e-3                           # Density at 195 km
    H = 195 / np.log(atmo_density / atmo_density_200)        # Scale height based on the new model
    return atmo_density * np.exp(-height / H) if height <= atmo_height else 0
#endregion
```

# Atmosphärenmodell

- Vergleich zu vorherigen Marsmissionen



# Missionsphasen und Bewegungsgleichungen

- 4 Missionsphasen

```
#region Mission Phases
> def handle_orbit_insertion_maneuver(r, phi, rhor, rhophi, m, mdry, t, r_thrust_descend, r_thrust_ascend):···

> def handle_slowdown_aerobraking(r, phi, rhor, rhophi, m, mdry, apoapsis, periapsis, t, r_p_slow):···

> def handle_orbit_stabilization(r, phi, rhor, rhophi, m, mdry, apoapsis, periapsis, t):···

> def handle_thrust_only(r, phi, rhor, rhophi, m, mdry, apoapsis, periapsis, t):···
#endregion
```

- 5 Bewegungsgleichungen

Zentrifugalkraft (Kein Inertialsystem)

Gravitation

Luftwiderstand

Triebwerksschub

Korioliskraft

$$\frac{dr}{dt} = \rho_r$$

$$\frac{d\phi}{dt} = \rho_\phi$$

$$\frac{d\rho_r}{dt} = r\rho_\phi^2 - \frac{\mu}{r^2} - \frac{D}{m}\cos(\alpha) + \frac{\text{throttle} \cdot \text{thrust}}{m}\cos(\beta)$$

$$\frac{d\rho_\phi}{dt} = \frac{-2\rho_r\rho_\phi - \frac{D}{m}\sin(\alpha) + \frac{\text{throttle} \cdot \text{thrust}}{m}\sin(\beta)}{r}$$

$$\frac{dm}{dt} = -\frac{\text{throttle} \cdot \text{thrust}}{c_{\text{eff}}}$$

# Missionsphasen und Bewegungsgleichungen

- 1. Missionsphase: Umlaufbahneinbringung

```python
def handle_orbit_insertion_maneuver(r, phi, rhor, rhophi, m, mdry, t, r_thrust_descend, r_thrust_ascend):
    global is_descending, thrust_active, orbit_insertion_maneuver
    if is_descending and r < r_thrust_descend and m > mdry:
        return 1.0  # Throttle fully open to capture the Spacecraft
    elif not is_descending and r > r_thrust_ascend:
        orbit_insertion_maneuver = False  # End of Orbit Insertion Maneuver
        return 0.0  # Throttle fully closed
    return 1.0 if thrust_active else 0.0  # Maintain current state
```

# Missionsphasen und Bewegungsgleichungen

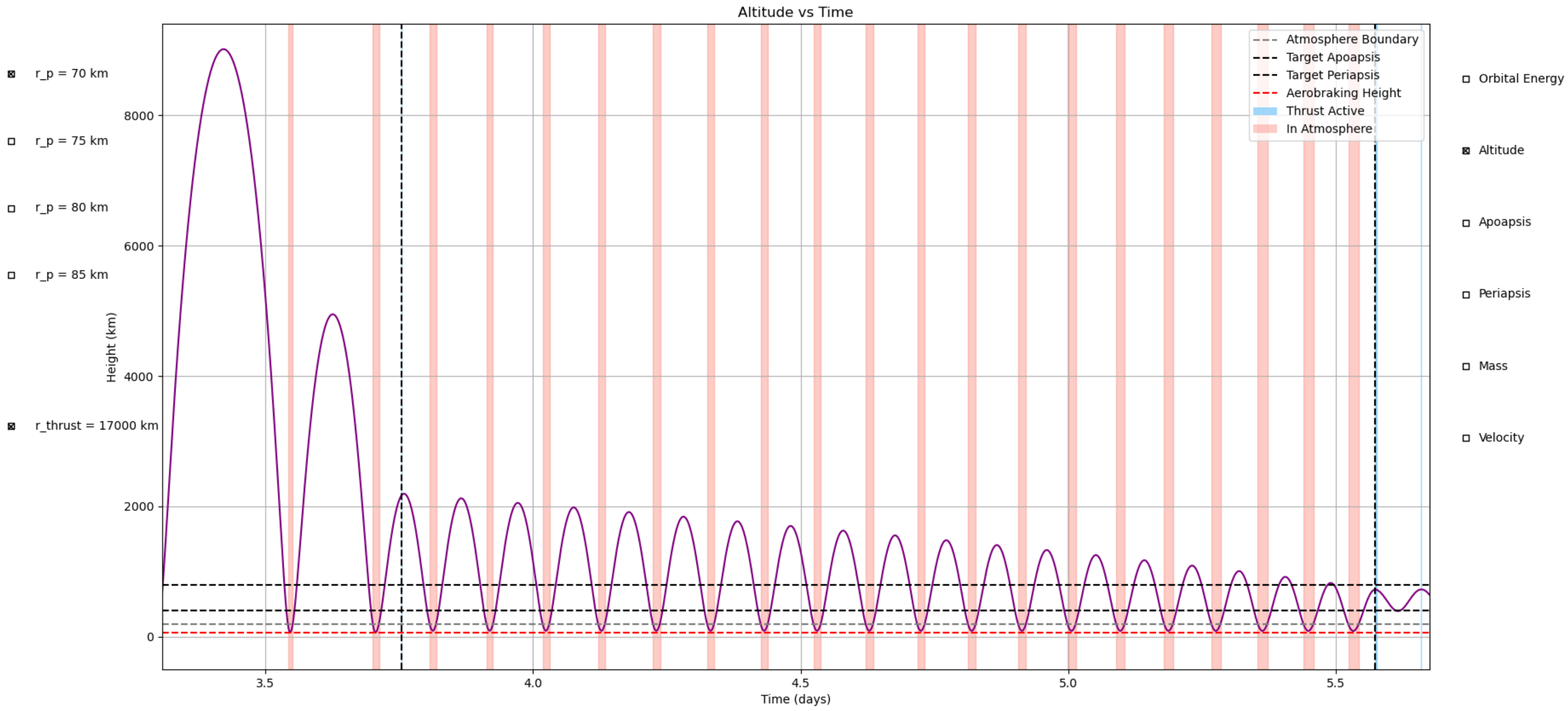
- 2. Missionsphase (nur bei Triebwerksnutzung)

```python
def handle_thrust_only(r, phi, rhor, rhophi, m, mdry, apoapsis, periapsis, t):
    global stable_orbit_time, full_aerobraking
    if apoapsis >= r_a and abs(r - periapsis) <= 10 and m > mdry:
        return 1.0  # Throttle fully open to reach the desired orbit
    elif apoapsis < r_a and abs(r - periapsis) <= 0.1:
        full_aerobraking = False
    return 0.0
```

- Alternativ Phase der Atmosphärenbremsung

# Missionsphasen und Bewegungsgleichungen

- 3. Missionsphase: Bremsreduzierung (nur für Atmosphärenbremsung)



# Missionsphasen und Bewegungsgleichungen

- 4. Missionsphase: Orbitstabilisierung

```python
def handle_orbit_stabilization(r, phi, rhor, rhophi, m, mdry, apoapsis, periapsis, t):
    global thrust_active, stable_orbit_time, stabilization_start_time
    if apoapsis <= r_a and abs(r - apoapsis) < 0.1 and periapsis < r_p_orbit and m > mdry:
        if stabilization_start_time is None:
            stabilization_start_time = t
        return 1.0  # Throttle fully open at apoapsis once the desired apoapsis is reached to increase periapsis
    elif periapsis >= (r_p_orbit-5):
        if stable_orbit_time is None:
            stable_orbit_time = t
        return 0.0  # Throttle fully closed once the desired periapsis (and with that the final orbit) is reached
    return 1.0 if thrust_active else 0.0  # Maintain current state
```

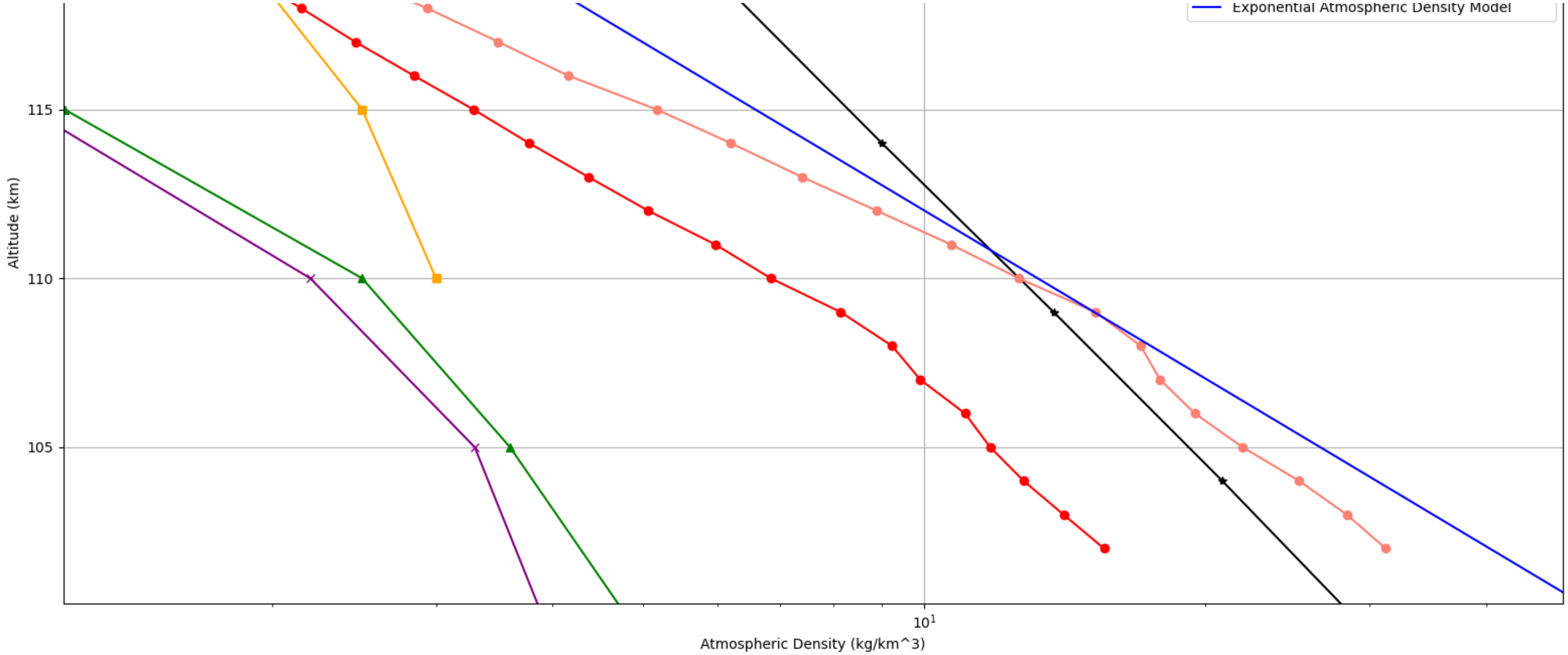
# Ergebnisdarstellung

- Ausgabe von Plots und Text in der Konsole
- Abhängig von Nutzereinstellung

```
Iteration: 1, Target r_p: 70.0 km, Distance to Target: 80.4629, max_step: 5.0
Iteration: 2, Target r_p: 70.0 km, Distance to Target: -8.2053, max_step: 5.0
Iteration: 3, Target r_p: 70.0 km, Distance to Target: 5.4708, max_step: 1.0
Iteration: 4, Target r_p: 70.0 km, Distance to Target: 0.8007, max_step: 1.0
Iteration: 5, Target r_p: 70.0 km, Distance to Target: -0.1512, max_step: 0.2
    New best found! Fuel used: 1991.3599 kg for r_p: 70 km
Found initial best b: 10119.0403 km for r_p: 70 km
```

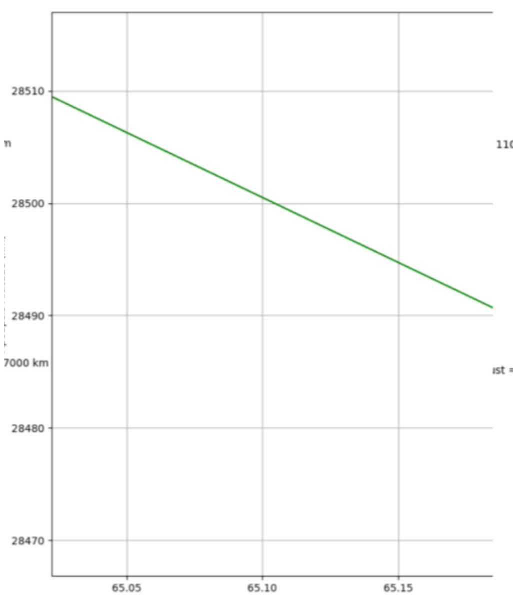
# Validierung des Programms

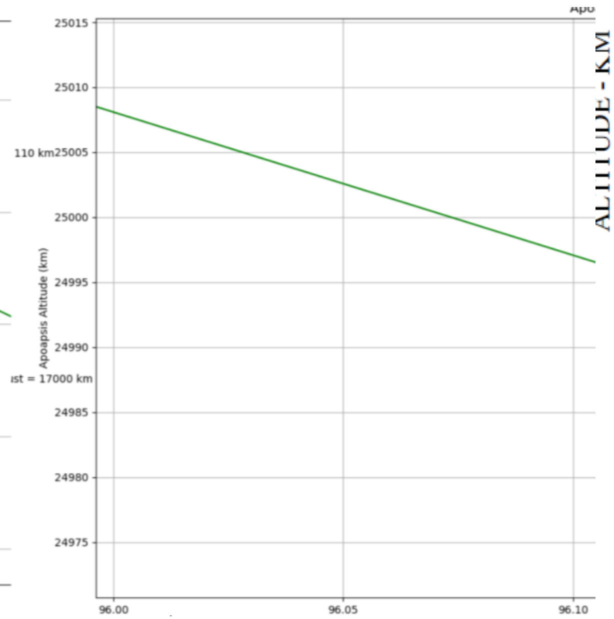
- Vergleich zur ExoMars Trace Gas Orbiter Mission der ESA von 2017
- Benutzereingabe entspricht Missionsdaten der Mission
- Atmosphärenmodell beinahe identisch im Bereich von 100 - 115 km

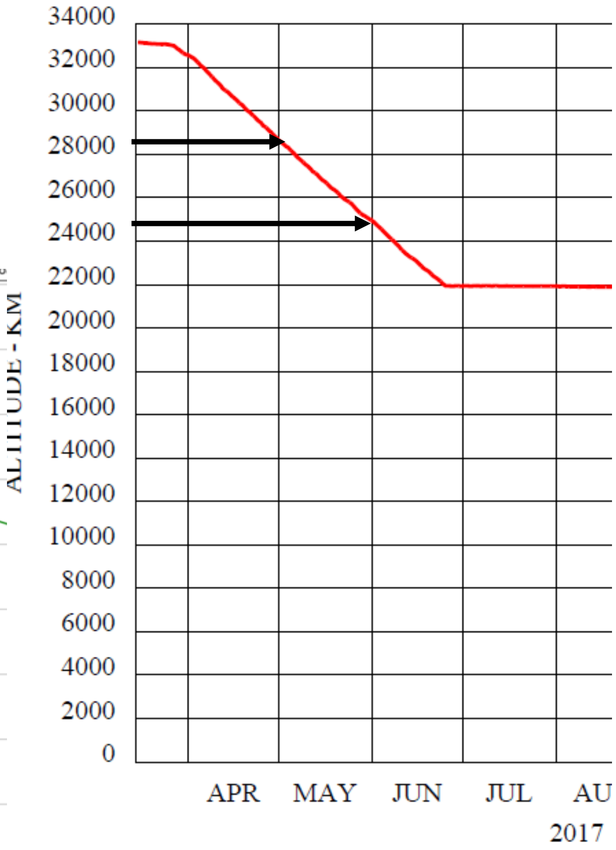


# Validierung des Programms

- Apoapsis der ExoMars TGO Mission in rot
- Apoapsis der Simulation in grün
- Betrachteter Zeitraum: Mai



# Optimierung der Rechenzeit

- Abbruchsbedingungen der Simulation
- Einteilung in kurzen und finalen Durchlauf
- Adaptive Schrittgröße, je nach Missionsdauer (Periapsishöhe)
- Gleichzeitige Berechnung verschiedener Simulationen

```python
#region Simulation termination conditions
def planet_crash(t, state, *args): ...
planet_crash.terminal    = True
def stable_orbit_reached(t, state, *args): ...
stable_orbit_reached.terminal = True
stable_orbit_reached.direction = 1
#endregion
```

```python
if r_p-R <= 80:
    _, _, full_trajectory, _, final_mass = simulate_trajectory(b, r_thrust_descend, r_thrust_ascend, r_p_slow, short_run=False, max_step=0.2)
elif r_p-R <=90:
    _, _, full_trajectory, _, final_mass = simulate_trajectory(b, r_thrust_descend, r_thrust_ascend, r_p_slow, short_run=False, max_step=0.3)
elif r_p-R <=95:
    _, _, full_trajectory, _, final_mass = simulate_trajectory(b, r_thrust_descend, r_thrust_ascend, r_p_slow, short_run=False, max_step=0.5)
else:
    _, _, full_trajectory, _, final_mass = simulate_trajectory(b, r_thrust_descend, r_thrust_ascend, r_p_slow, short_run=False, max_step=1.25)
```

# Schlussfolgerung und Ausblick

- Haupterkenntnisse: Zusammenhänge zwischen Höhe der Atmosphärenbremsung, Treibstoffersparnis, Dauer des Manövers
- Bei hoher Genauigkeit: Anwendung zur Missionsanalyse und Planung
- Bei geringer Genauigkeit: erster Eindruck zur Nutzungsmöglichkeit der Atmosphärenbremsung
- Anpassbar für andere Planeten
- Zukünftige Erweiterungen (z.B. Hitzeentwicklung) möglich

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

# Referenzen


- Bruinsma, S., Forbes, Jeffrey M. (2020). ExoMars Trace Gas Orbiter aerobraking densities
  https://scholar.colorado.edu/concern/datasets/zp38wd65j
- ESA. (2017). Hang 10 over Mars.
  https://blogs.esa.int/rocketscience/2017/03/16/hanging-10-over-mars/
- Jet Propulsion Laboratory. (1993). Magellan Descends Into Venus‘ Atmosphere.
  https://www.jpl.nasa.gov/news/magellan-descends-into-venus-atmosphere
- NASA. (2024). Planetary Data System – Mars Orbiter
  https://pds-atmospheres.nmsu.edu/data_and_services/atmospheres_data/MARS/mars_orbiter.html