

IMPULSE ESTIMATION

Physics Basis: Newton's Second Law

Considerations:

- **Newton's Second Law:** $F=ma$
- **Kinematics:** For altitude from acceleration
- **Energy conservation:** For coast phase apogee
- **Drag correction:** Through empirical $F_Z(N)$ factor

JULIA code (written by Ogweno):

DESCRIPTION

using Interpolations # Load package to perform interpolation (used to estimate drag loss factor F_Z)

using Plots # Load package for plotting data (e.g., N vs F_Z)

CONSTANTS AND INITIAL VALUES

$m = 22.5$ # Total estimated rocket mass during burn (kg)

$t = 3.6$ # Estimated burn time of SRM (s)

$g = 9.8$ # Gravitational acceleration (m/s^2)

$cd = 0.4$ # Drag coefficient (estimated, dimensionless)

$D = 11$ # Max rocket diameter (cm) **recommendation- convert to meters**

$md = 17.5$ # Rocket dry mass (without propellant grains) (kg)

DRAG CORRECTION TABLE

$N_values = [0.0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]$

$F_Z_values = [1.0, 0.82, 0.7, 0.62, 0.56, 0.52, 0.48, 0.44, 0.4, 0.38, 0.38]$

- N : Drag influence number, a function of velocity, drag, and geometry.
- F_Z : Corresponding drag loss factor — this reduces actual altitude due to air resistance.

A high N means greater drag influence, hence smaller F_Z (larger altitude loss).

LINEAR INTERPOLATION OF FZ

```
itp = LinearInterpolation(N_values, Fz_values)
```

```
function interpolate_Fz(N)
```

```
    return itp(N)
```

```
end
```

- Given a computed N , you can estimate the drag correction factor F_Z using linear interpolation.

- Accuracy can be improved by using spline interpolation (`QuadraticSplineInterpolation`)

BURNOUT ALTITUDE

```
function burnout(F, m, g, t)
```

```
    z1 = 0.5 * ((F / m) - g) * t^2
```

```
    return z1
```

```
end
```

Consider Newton's Second Law:

$$F = ma$$

Where

$$a = \frac{F}{m} - g$$

using kinematics:

$$z_1 = \frac{1}{2}at^2$$

This gives **altitude at the end of thrust phase**.

MAX ALTITUDE/ APOGEE (z2)

```
function apogee(F, z1, m, g)
```

```
    z2 = (F * z1) / (m * g)
```

```
    return z2
```

end

Converts kinetic energy at burnout to additional altitude.

Using:

$$V = \sqrt{2az_1}$$

$$\Delta z = \frac{v_1^2}{2g}$$

THRUST SWEEP AND DRAG LOSS MODELING

```
function thrust_generator()
```

```
    z1 = 0 # Initialize burnout altitude
```

```
    z2 = 0 # Initialize total apogee altitude
```

```
    N_plots = []
```

```
    Fz_plots = []
```

Initialization of data for plotting and result storage.

```
    for F in 0:25:3500
```

```
        z1 = burnout(F, m, g, t)
```

```
        z2 = apogee(F, z1, m, g)
```

Iterate over thrust values from 0 to 3500 N in steps of 25 N.

BURNOUT VELOCITY AND DRAG INFLUENCE NUMBER

```
    v1 = sqrt(((2*z1)/m) * (F - m*g))
```

A derived approximation for burnout velocity based on net thrust and mass.

This can be better expressed as:

$$v_1 = a * t = \left(\frac{F}{m} - g \right)$$

```
    N = (cd*D^2*v1^2)/(1000*md) # Drag influence number
```

Custom drag-related nondimensional number N :

$$N = \frac{c_d * D^2 * v_1^2}{1000 * m_{\text{dry}}}$$

Units must be consistent — **D should be converted to meters:**

$D_{\text{m}} = D / 100$ # cm to m

CORRECTED APOGEE WITH DRAG LOSS FACTOR

$F_z = \text{interpolate_Fz}(N)$

$z2_{\text{actual}} = F_z * z2$

- Use interpolated drag factor F_z to correct the raw apogee estimate ($z2_{\text{actual}}$ is the final altitude).
- If $F_z = 0.4$, you lose 60% of apogee to drag.

OUTPUT

```
# Uncomment to stop once target apogee is reached
# if z2_actual >= 4000
#   println("Thrust: ", F)
#   break
# end
```

OUTPUT & PLOTTING

```
println("Iteration", F/25)
println("Thrust: ", F)
println("Altitude at burnout: ", z1)
println("Maximum altitude (apogee): ", z2_actual)
println("velocity at burnout v1: ", v1)
println("Drag influence number N: ", N)
println("\n")

push!(N_plots, N)
push!(Fz_plots, Fz)
```

end

plot(N_plots, Fz_plots)

end

thrust_generator() # Start simulation

Stores and plots N vs F_z to visualize drag effects across thrust range.

RECOMMENDATIONS

Convert diameter to meters for N :

$D = 0.11$ # meters

Improve burnout velocity formula:

$v_1 = (F / m - g) * t$

Model mass loss over burn time

- Split t into timesteps, and reduce m over time
- Integrate using numerical methods

Include **air density** and **dynamic pressure drag**