*Damery, Noah (S&T-Student)*

*Missouri S&T Rocket Design Team | Rolla, MO*

*Full rocket simulation Handbook*

A description of the functions and processes involved in the MATLAB-based rocket simulation

Table of Contents

[Simulation Improvement Plan 2](#_Toc178980738)

[Core Simulation Functions 3](#_Toc178980739)

[Get Optimization Matrix 3](#_Toc178980740)

[Get Altitude 3](#_Toc178980741)

[Get flow 5](#_Toc178980742)

[Get material properties 6](#_Toc178980743)

[Get bolt shear stress 7](#_Toc178980744)

[Get rho 8](#_Toc178980745)

[Get thermal expansion (NOT INCLUDED IN CURRENT VERSION) 8](#_Toc178980746)

[Get coolant properties (NOT INCLUDED IN CURRENT VERSION) 9](#_Toc178980747)

[Get gas coefficient (NOT INCLUDED IN CURRENT VERSION) 9](#_Toc178980748)

[Get passage pressure drop (NOT INCLUDED IN CURRENT VERSION) 10](#_Toc178980749)

# Simulation Improvement Plan

|  |  |  |
| --- | --- | --- |
| **Function Name** | **Update Plan** | **Current Version Ready for V1 Release** |
| Get material properties | Finish documenting data sources |  |
| Get thermal expansion | Think through more thoroughly |  |
| Get coolant properties | Finish documenting and consider adding pressure dependence |  |
| Altitude simulator | Document and release  Consider making a MATLAB live version for rapid iteration? | 9/4/2024 |
| Get gas coefficient | Document and port over |  |
| Get passage pressure drop | Document and port over |  |
| Get liquid coefficient | Document and port over |  |
| Get wall temps | Document and port over |  |
| Get final coolant temperature | Rewrite |  |
| Get optimization matrix | Document | 9/4/2024 |
| UI Altitude Simulator | Done | 9/4/2024 |
| Get max stress | Document and port over |  |
| Get rho | Done | 9/2/2024 |
| Get bolt shear stress | Done | 9/2/2024 |
| Get flow | Document | 9/28/2024 |
| Test units | Document | 9/2/2024 |

# Core Simulation Functions

## Get Optimization Matrix

GETOPTIMIZATIONMATRIX - Returns a set of named matrices that can be used for optimization analyses like the getAltitude function.

This function generates an n-dimensional matrix where n is the number of rows in the ranges vector. It generates matrices such that each one varies in only one axis - i.e. each dimension corresponds to a variable with some meaning. This would allow you to run samples^n simulations at once without needing to iterate through all the possibilities.

names - Names of each set of variables

ranges - n x 2 matrix where the first and second elements of each row is the range that is correlated with the corresponding value in the names field.

samples - The number of samples to be taken between the range endpoints for each set of values.

Warning: these output matrices can get big very fast. A reasonable value for matrix elements seems to be around 5,000,000, but this depends a lot on how complicated of a sim you're working with.

## Get Altitude

GETALTITUDE - Returns an estimate of the rocket's apogee based on given input conditions.

[r, inp] = getAltitude(opt, const, crit)

This function is designed to be able to be used as an altitude optimizer. It can take in a variety of input condition ranges and output altitudes for all of them. You can then find the input conditions that produced the best results and build a rocket based on those.

In order to do an optimization, you could input an n-dimensional matrix for each field where there are n variables being optimized and each direction in the matrix corresponds to changes in one variable. This function will run all of the simulations simultaneously to improve calculation time and spit out an n-dimensional matrix of solutions. Such a matrix can be generated quite easily using the getOptimixationMatrix function.

If the min TWR condition is not met for a given case then it's apogee will be set to zero - excluding it from coming out of the optimization as a reasonable choice.

The r field is a struct containing all the function results.

A timestep of 0.01 seconds is recommended for this function. Higher timesteps result in higher calculation times, but you will start getting noticeable error in the thousands of feet by decreasing your sim timestep to ~1 second.

Required Values:

arg.dragCoefficient - Coefficient of drag (-)

arg.diameter - Rocket diameter (m)

arg.startAltitude - Starting altitude above MSL (m)

arg.isp - Rocket specific impulse (s)

arg.ispDecay - Specific impulse lost per second (s/s)

arg.flightAngle - Angle of flight (deg)

arg.railLength - Launch rail length (m)

arg.railButtonDist - Rail button distance (m)

arg.m\_leftover - Propellant remaining in tanks (kg)

arg.dt - Timestep for running simulation (s)

arg.thrust - Rocket starting thrust (N)

arg.massFraction - Mass fraction {Prop mass/total mass} (-)

arg.propMass - Propellant mass (kg)

arg.thrustDecay - Thrust lost per second due to blow down (N/s)

arg.minTWR - Min TWR

Values Returned:

r.delta\_h - Apogee (m)

r.t\_burn - Total burn time (s)

r.v\_max - Maximum velocity (m/s)

r.v\_rail - Velocity when first rail button leaves rail (m/s)

r.a\_max - Maximum acceleration (m/s^2)

r.de\_max - Maximum deceleration (m/s^2)

r.D\_max - Maximum drag (N)

r.h\_max - Max altitude {apogee + starting altitude} (m)

r.l - Lateral distance traveled at given angle (m)

r.t\_apogee - Time to apogee (s)

r.initialTWR - TWR at start of burn (-)

inp.T - Input thrust (N)

inp.zeta - Input mass fraction (-)

inp.C\_D - Drag coefficient (-)

inp.m\_prop - Propellant mass (kg)

inp.m\_dry - Dry mass (kg)

inp.m\_total - Total mass (kg)

## Get flow

Solves for the flow properties of the injector orifices

outp = getFlow(arg)

Returns a struct of flow properties for a given set of input properties. If mass flow rate is passed in, then this function solves for orifice area and diameter. If orifice area is passed in, then this function solves for the mass flow rate. The math for this function is based on RPE Equations 8-1 and 8-2 which can be found on page 280 of the 9th Edition. These equations are technically only valid for single phase flow.

Required Values:

arg.mDot\* - Total orifice group mass flow rate (kg/s)

arg.A\* - Individual orifice area (m^2)

arg.C\_d - Orifice discharge coefficient (-)

arg.rho - Fluid density at orifice (kg/m^3)

arg.deltaP – Pressure drop across the orifices (Pa)

arg.N\_orifices – Number of orifices in group (-)

Values Returned:

outp.A\* - Individual orifice area (m^2)

outp.d\* - Individual orifice diameter (m)

outp.mDot\* - Total orifice group mass flow rate (kg/s)

outp.Q - Total orifice group volume flow rate (m^3/s)

outp.v - Orifice exit velocity (m/s)

All values marked with an asterisk can be either input or output values depending on which variables are passed into the function, as described above.

*For more information, see the extended documentation*

Expressed symbolically, the equations governing orifice mass flow rate are as follows:

(RPE Equation 8-1)

(RPE Equation 8-2)

The quantity is an important value that can be determined by flow testing an injector with any fluid. You can use this experimentally determined value to predict mass flow rates at different pressure drops with different fluids.

## Get material properties

Returns material properties of given material

materialMatrix = getMaterialProperties(materialName)

Returns a struct of material properties of the material given in the “materialName” parameter, including the following variables. All values are returned in SI units, when applicable (temperatures in Celsius). All properties except yield strength are either given for a constant temperature of approximately 20 C (293 K, 68 F) or they are given for more conservative temperatures.

Values Returned:

Kappa - Conductivity (W/m-K)

rho - Density (kg/m^3)

a - Thermal expansion coefficient (m/m-C)

v - Poisson’s ratio (-)

solidus – Solidus melting point (C)

yield - [INTERNAL] Yield values across temperatures (Pa)

temps - [INTERNAL] Temperature values for yields (C)

Functions Returned:

getYieldStrength(temperature)

* Returns the yield strength of the material at the given temperature in Pa.
* Temperature inputs must be in C.

getYoungsModulus(temperature)

* Returns the Young’s modulus of the material at the given temperature in Pa.
* Temperature inputs must be in C.

Supported Materials and their Abbreviations:

Stainless Steel 304 – “steel304”

Stainless Steel 303 – “steel303”

Alloy Steel 4340 – “steel4340”

Aluminum 6061-T6 – “aluminum6061”

Aluminum 7075-T6 – “aluminum7075”

Data for this function was sourced from the links listed below.

**Stainless Steel 303 Data Sources**

**Stainless Steel 304 Data Sources**

* <https://www.matweb.com/search/DataSheet.aspx?MatGUID=abc4415b0f8b490387e3c922237098da>
* <https://nickelinstitute.org/media/1699/high_temperaturecharacteristicsofstainlesssteel_9004_.pdf>  (Page 15)

**Alloy Steel 4340 Data Sources**

**Aluminum 6061-T6 Data Sources**

* <https://www.matweb.com/search/datasheet_print.aspx?matguid=1b8c06d0ca7c456694c7777d9e10be5b> (This uses data from the Aluminum Design Manual)

**Aluminum 7075-T6 Data Sources:**

* The Aluminum Design Manual, Part V, Page 39 (7075-T6)
* <https://www.matweb.com/search/DataSheet.aspx?MatGUID=4f19a42be94546b686bbf43f79c51b7d> (This uses data from the Aluminum Design Manual)
* Data for Young’s modulus: Eurocode 9, page 21 – general properties for two hour exposure.

## Get bolt shear stress

Calculates the stresses in bolts under given conditions

outp = getBoltShearStress(inp)

Returns a struct containing the bearing stress and shear stress of the bolt setup in pascals. This function assumes that there is only one shear plane for each bolt. Make sure that the value for the wall thickness is the thinner of the two walls that are in contact.

Required Values:

arg.d - Diameter of bolt holes (m)

arg.t - Thickness of wall (m)

arg.F - Total force (N)

arg.n - Number of bolts (-)

Values Returned:

outp.bearingStress - Bearing stress on hole face (Pa)

outp.shearStress - Shear stress on bolt (Pa)

Link to similar calculator:

<https://www.engineersedge.com/material_science/bolt_single_shear_calcs.htm>

## Get rho

Gets the density at a given height in the atmosphere.

rho = getRho(h)

Returns the atmospheric density in pascals for a given altitude in meters. Valid for altitudes from 0-80,000 m. This function uses geopotential altitude, At altitudes above 80,000 m, this function returns 0. This function’s source gives density at a given geopotential altitude – not at a given geometric altitude, but the differences should be small enough to be negligible.

Required Value:

h - altitude above sea level (m)

Value Returned:

rho - density at given altitude (kg/m^3)

For details on data sources, see the extended documentation:

Standard atmosphere data is from the following website:

<https://www.engineeringtoolbox.com/standard-atmosphere-d_604.html>

## Get thermal expansion (NOT INCLUDED IN CURRENT VERSION)

Returns the linear thermal expansion of an element subjected to a temperature change.

expansion = getThermalExpansion(args)

Returns the linear thermal expansion present in an object that is heated under the given conditions. Args is a struct that must contain the following properties:

L – Initial length of the object

a - Thermal expansion coefficient

deltaT - Temperature difference from start length to current condition.

All values are given and returned in consistent units.

The equation is given on Wikipedia (and countless others online sources) as follows:

A black background with white text

Description automatically generated

From here, we can derive the equation,

Where L is the change in length of the element, L is the original length, is the linear thermal expansion coefficient of the material (which can be found online or via the GETMATERIALPROPERTIES function), and is the change in temperature of the element from its initial temperature to its current temperature.

## Get coolant properties (NOT INCLUDED IN CURRENT VERSION)

Gets fluid properties for a given coolant (fuel) selection.

coolant = getCoolantProperties(coolantName)

Returns a struct containing coolant properties of the coolant specified by the coolantName parameter. The properties returned are as follows: constant pressure specific heat (c), density (rho), dynamic or absolute viscosity (mu), thermal conductivity (kappa), and boiling temperature at 1 atm (T\_boil). All values are returned in SI units, when applicable, and the temperature values are returned in kelvin. All temperature dependent values are given for a temperature of approximately 20 Celsius (293 K, 68 F).

Supported Coolant Names:

Ethanol – “ethanol”

Isopropanol – “isopropanol”

Data Sources:

Properties for ethanol are from:

<https://www.engineeringtoolbox.com/ethanol-ethyl-alcohol-properties-C2H6O-d_2027.html>.

Properties for isopropanol are from: <https://www.matweb.com/search/datasheet.aspx?matguid=7d8c7f1164124ddc827eb662d6da7943&ckck=1>.

## Get gas coefficient (NOT INCLUDED IN CURRENT VERSION)

Calculates the estimated value of the combustion gas heat transfer coefficient.

{h\_g, ~Pr, ~Nu} = getGasCoefficient(args)

Returns the value of the gas heat transfer coefficient h\_g for a given set of input parameters stored in the struct args. This function uses RPE equation 8-20 on page 313 in the 9th edition, although similar formulas can be found in other rocketry texts. This formula is called out in the detailed documentation. The Nusselt number is calculated as a function of the Reynold’s number and the Prandtl number, and then the Nusselt number’s formula can be solved for h\_g. To perform all these calculations, the following parameters are required:

rho - Gas density

v - Gas velocity

kappa - Gas thermal conductivity

mu - Gas absolute (dynamic) viscosity

c\_p - Gas specific heat at constant pressure

D - Interior diameter of chamber/nozzle

All input and output values can be evaluated using any consistent unit system.

The Prandtl number and the Nusselt number are also available as optional output values.

The formulas for these calculations are as follows:

Nu is the Nusslet number, Re is the Reynold's number, and Pr is the Prandtl number. We can express each of these numbers as a combination of the properties of the system.

A black background with white text

Description automatically generated

Pr = A black and white text on a black background

Description automatically generated

A black background with white text

Description automatically generated

We can rearrange this last equation to isolate the convective heat transfer coefficient h\_g.

A black and white symbol

Description automatically generated

## Get passage pressure drop (NOT INCLUDED IN CURRENT VERSION)

(Orifice pressure drop -> page 280)

Typical values for the pressure drop are dependent on the friction coefficient , which is dependent on Reynold’s number and usually between 0.02 and 0.05.

RPE Also notes that a typical pressure drop in coolant passages is between 5-25%

According to RPE Equation 8-11 (Edition 9, page 296), the pressure drop in a cooling passage is given by the following equation:

A mathematical equation with numbers

Description automatically generated

Where is the pressure drop, is the density of the fluid, is a dimensionless friction coefficient, is the fluid velocity, is the length of the passage, and is the equivalent diameter.