KSPython

Release 0.1.0

Luiz Frederico Villalobos

Sep 29, 2020

CONTENTS:

CHAPTER

ONE

INTRODUCTION

This is the documentation of a library called KSPython, made to help people design Kerbal Space Program rockets. It features quick to use syntax for prototyping and simulations.

More details about the library, including examples, can be found at https://github.com/fred1608/KSPython.

CHAPTER

TWO

KSPYTHON

2.1 KSPython package

2.1.1 KSPython module

class KSPython.**Stage**

Stage class, that incorporate parts and is inserted into a rocket.

The stage class is one of the basic classes of this project. It is used as a collection of parts, and represents a section of the rocket.

In the stage, a form of engine and fuel must be present. Other parts can be represented as extra mass and extra cost.

Notes

- Different fuel types or engine types (solid or liquid) cannot be placed on the same stage.
- Only use one type of solid booster per stage.

add_extra_cost(*cost*)

Add extra cost to an stage. Used mainly to add the other parts that are not engines or fuel tanks.

Parameters

cost [*int/float*] Cost to be added.

add_extra_mass(*mass*)

Add extra mass to an stage. Used mainly to add the other parts that are not engines or fuel tanks.

Parameters

mass [*int/float*] Mass to be added [ton].

add_part(*part*)

Add a part to an stage.

Parameters

part [*part*] Part to be added to a stage.

add_parts(*parts*)

Add parts to an stage.

Parameters

parts [*list of parts*] Parts to be added to a stage.

calculate_cost()

Calculate the full cost of a stage.

Return

cost_sum [*float*] Total cost of stage.

calculate_empty_mass()
Calculate the mass of the stage while it is empty.

Return

mass_sum [*float*] Total mass of empty stage [ton].

calculate_full_mass()
Calculate the mass of the stage while it is full.

Return

mass_sum [*float*] Total mass of full stage [ton].

get_engine_performance(*loc='atm'*)
Calculate the relative thrust and isp of all engines within this stage.

Return

thrust [*float*] Relative thrust for all engines of this stage [kN].

isp [*float*] Relative ISP value for all engies of this stage [s].

get_fuel_type()
Returns the fuel type being used within the same stage.

list_parts()
Prints all parts present in a stage.

class KSPython.**Rocket**(*name=None*)
Rocket class, it is where most of the calculations occur, it also receives stages as inputs.

The rocket activates each stage in order that it has been inserted, burning its fuel, turning its engine on and discarding older stages.

It is possible to have engines fire before their stage by using *schedule_engine* method. Fuel will not be consumed by the later stages, and will only be used by the one being fired (assuming they share a type and it is possible to do so).

If ths is not the intended operation, fuel flow can also be restricted.

Parameter

name (optional) [*string*] Name of the rocket.

add_stage(*stage*)
Add a stage to an rocket.

Stages must be added in order, from first (ascension) to last

Parameters

stage [*stage*] Stage to be added to the rocket.

add_stages(*stages*)
Add stages to an rocket.

Stages must be added in order, from first (ascension) to last

Parameters

stages [*list of stages*] Stages to be added to the rocket.

adjusted_dV(*dV_out=2500*)
Calculates the true delta-V present in the rocket, by adjusting for the total required for leaving the atmosphere.

Parameters

dV_out [*int/float*]

Delta-V required to leave the atmosphere of a given body [m/s].

- 2500 - Kerbin

Return

dV [*float*] Delta V of the rocket [m/s].

calculate_dV(*loc='atm'*)
Calculates the delta-V present in the rocket.

Parameters

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

dV [*float*] Delta V of the rocket [m/s].

calculate_group_performance(*stage_num*, *loc='atm'*)
Calculates the total thrust and relative ISP of all engines firing at a given stage that share common fuel.

Parameters

stage_num [*int*] Stage to be analysed.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

total_thrust [*float*] Total thrust of engines [kN].

isp [*float*] Relative ISP of engines [s].

calculate_isp(*stage_num*, *loc='atm'*)
Calculates the relative ISP of all engines firing at a given stage.

Parameters

stage_num [*int*] Stage to be analysed.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

isp [*float*] Relative ISP of engines [s].

calculate_stage_dV(*stage_num*, *loc='atm'*)
Calculates the delta-V present in a single stage.

Parameters

stage_num [*int*] Stage to be analysed.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

dV [*float*] Delta V of the stage [m/s].

calculate_thrust(*stage_num*, *loc='atm'*)
Calculates the total thrust of all engines firing at a given stage.

Parameters

stage_num [*int*] Stage to be analysed.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

thrust [*float*] Total thrust of engines [kN].

calculate_total_cost()
Total cost of the rocket.

Return

total_cost [*float*] Total cost of the rocket.

calculate_total_mass()
Total mass of the rocket full.

Return

total_mass [*float*] Total mass of the rocket [ton].

calculate_twr(*stage_num*, *g=9.81*, *loc='atm'*)
Calculates the thrust o weight ratio of the rocket for a given stage.

Parameters

stage_num [*int*] Stage to be analysed.

g [*float*] Gravity (default for Kerbin).

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

twr [*float*] Thrust to weight ratio.

calculate_upper_mass(*stage_num*)
Total mass of the rocket above the stage being analysed (without including it).

Parameters

stage_num [*int*] Stage to be analysed.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

upper_mass [*float*] Upper mass of the rocket [ton].

change_payload(*payload*)
Adds a payload (or change its value, in case this method was used before) that will be carried by the rocket.

Parameters

payload [*int/float*] Payload to be added to a rocket [ton].

check_mass_lost(*stage_num*, *mass_loss*)
Verifies how much fuel mass a stage has, and if it is greater than a test value at any given stage.

Raises exception if test fails.

Parameters

stage_num [*int*] Stage to be analysed.

mass_loss [*float*] Mass to be verified if greather than fuel mass.

engine_burn_time(*stage_num*, *loc='atm'*)
This method calculates the total time an stage will spend burning at maximum thrust.

Note that if an engine was started before its stage started, and it wasn't able to receive fuel from upper stages, the fuel lost before it started will be considered in decreasing total burn time.

Parameters

stage_num [*int*] Stage to be analysed.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

burn_time [*float*] Total burn time [kN].

find_when_engine_fired(*stage_num*)
If the stage being analysed has been scheduled to fire, return when. Else it returns itself.

Parameters

stage_num [*int*] Stage to be analysed.

Return

stage_fire [*int*] Stage where engines fire.

generate_report(*g=9.81*)
Prints a report with the most important informations of a rocket.

Parameters

g [*float*] Gravity (default for Kerbin).

num_stages()
Number of stages in a rocket.

Return

num_stage [*int*] Number of stages in a rocket.

performance_engines_firing(*stage_num*, *stage_max=None*, *loc='atm'*)
This method gets engine peformance from all engines that are firing together in more complex stagings, at the time of stage_num.

It can also be restricted to return only values up to a limit stage, defined by stage_max. This is useful when calculating fuel flow with fuel restrictions.

Parameters

stage_num [*int*] Stage to be analysed.

stage_max [*int*] Maximum stage to which results will be brough (inclusing stage_max).

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

thrust_list [*list of thusts*] List the thurst of the engines [kN].

isp_list [*list of ISPs*] List the ISP of the engines [s].

prestage_mass_loss(*stage_num*, *loc='atm'*)
Calculates how much mass an stage has lost before the rocket staged into it.

Parameters

stage_num [*int*] Stage to be analysed.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

mass_loss [*float*] Total mass lost by the stage [ton].

rem_fuel_flow(*stage_num*)
Retrict the fuel flow in the rocket between the assigned stage and next one.

In normal operation, fuel will always be passed from smaller stages to the next automatically when applicable. Using this will prevent the rocket from moving fuel upstage. This action is not required when fuel flow is impossible, for example when using solid rocket boosters.

Parameters

stage_num [*int*] Stage where the operation will be executed.

schedule_engine(*stage_fire*, *stage_present*)
Schedule engines to fire before their normal stage.

Parameters

stage_fire [*int*] Stage to fire engines.

stage_present [*int*] Stage which is to fire their engines.

time_between_stages(*stage_ini*, *stage_end*, *loc='atm'*)
This method returns the total cumulative time between the start of two stages.

Parameters

stage_ini [*int*] Initial stage.

stage_end [*int*] Final stage.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

total_time [*float*] Total time between stages [s].

total_poststage_mass_loss(*stage_num*, *loc='atm'*)
Calculates how much mass the whole rocket has lost in stages above the stage being analysed when the stage ended.

Parameters

stage_num [*int*] Stage to be analysed.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

total_mass_lost [*float*] Total mass lost by the rocket [ton].

total_prestage_mass_loss(*stage_num*, *loc='atm'*)
Calculates how much mass the whole rocket has lost in stages above the stage being analysed when it started.

Parameters

stage_num [*int*] Stage to be analysed.

loc [*{'atm', 'vac'}*] Location where the method will be performed.

Return

total_mass_lost [*float*] Total mass lost by the rocket [ton].

class KSPython.**LiquidEngine**(*name*, *mass*, *cost*, *thrust_atm*, *thrust_vac*, *isp_atm*, *isp_vac*)

Liquid engine class for generating new parts.

Parameters

name [*string*] The name of the part.

mass [*float/int*] The mass of the part.

cost [*float/int*] Part cost.

thrust_atm [*float/int*] Atmospheric engine thrust, in kN.

thrust_vac [*float/int*] Vaccum engine thrust, in kN.

isp_atm [*float/int*] Atmospheric engine ISP, in s.

isp_vac [*float/int*] Vaccum engine ISP, in s.

Example

```
>>> REM3 = LiquidEngine('RE-M3 "Mainsail" Liquid Fuel Engine', 6, 13000, 1379.
↪03, 1500, 285, 310)
```

Note

- Basic parts have already been inserted through LiquidEngineParts, but new ones can be made by utilising this class.

class KSPython.**SolidEngine**(*name*, *mass_full*, *mass_empty*, *cost*, *thrust_atm*, *thrust_vac*, *isp_atm*, *isp_vac*)

Liquid engine class for generating new parts.

Parameters

name [*string*] The name of the part.

mass_full [*float/int*] The mass of the part when it is full.

mass_empty [*float/int*] The mass of the part when it is empty.

cost [*float/int*] Part cost.

thrust_atm [*float/int*] Atmospheric engine thrust, in kN.

thrust_vac [*float/int*] Vaccum engine thrust, in kN.

isp_atm [*float/int*] Atmospheric engine ISP, in s.

isp_vac [*float/int*] Vaccum engine ISP, in s.

Example

```
>>> RT10 = SolidEngine('RT-10 "Hammer" Solid Fuel Booster', 3.56, 0.75, 400,␣
↪197.9, 227, 170, 195)
```

Note

- Basic parts have already been inserted through BoosterParts, but new ones can be made by utilising this class.

class KSPython.**RocketFuelTank**(*name*, *mass_full*, *mass_empty*, *cost*)

Liquid fuel tank class for generating new parts.

Parameters

name [*string*] The name of the part.

mass_full [*float/int*] The mass of the part when it is full.

mass_empty [*float/int*] The mass of the part when it is empty.

cost [*float/int*] Part cost.

Example

```
>>> Jumbo64 = RocketFuelTank("Rockomax Jumbo-64 Fuel Tank", 36, 4, 5750)
```

Note

- Basic parts have already been inserted through RocketFuelTankParts, but new ones can be made by utilising this class.

2.1.2 KSPython.LiquidEngineParts module

This submodule is responsible to house liquid rocket engines to be used on simulation.

Note:

- Id Name is the name assigned to the part to be imported and inserted into the code.
- LVN 'Nerv' Engine is not-supported, as the calculation currently does not differentiate between oxidizer and liquid fuel.
- KR12 is divided into two parts, one for engine and one for fuel, both parts must be added if using it.

Id Name	Name	Mass [ton]	Cost	Thrust atm	Thrust vac	ISP atm	ISP vac
LV1R	LV-1R “Spider” Liquid Fuel Engine	0.02	120	1.79	2	260	290
E2477	24-77 “Twitch” Liquid Fuel Engine	0.02	230	15.17	16	275	290
Mk55	Mk-55 “Thud” Liquid Fuel Engine	0.9	820	108.2	120	275	305
LV1	LV-1 “Ant” Liquid Fuel Engine	0.02	110	0.51	2	80	315
E487S	48-7S “Spark” Liquid Fuel Engine	0.13	240	16.56	20	265	320
LV909	LV-909 “Terrier” Liquid Fuel Engine	0.5	390	14.78	60	85	345
LVT30	LV-T30 “Reliant” Liquid Fuel Engine	1.25	1100	205.16	240	265	310
LVT45	LV-T45 “Swivel” Liquid Fuel Engine	1.5	1200	167.97	215	250	320
S3KS25	S3 KS-25 “Vector” Liquid Fuel Engine	4	18000	936.51	1000	295	315
T1	T-1 Toroidal Aerospike “Dart”	1	3850	153.53	180	290	340
REL10	RE-L10 “Poodle” Liquid Fuel Engine	1.75	1300	64.29	250	90	350
REI5	RE-I5 “Skipper” Liquid Fuel Engine	3	5300	568.75	650	280	320
REM3	RE-M3 “Mainsail” Liquid Fuel Engine	6	13000	1379.03	1500	285	310
KR12_e	LFB KR-1x2 “Twin-Boar” Liquid Engine	0	0	1866.67	2000	280	300
KR2L	Kerbodyne KR-2L+ “Rhino”	9	25000	1205.88	2000	205	340
S3KS254	S3 KS-25x4 “Mammoth” Liquid Engine	15	39000	3746.03	4000	295	315
CR7	CR-7 R.A.P.I.E.R. Engine	2	6000	162.3	180	275	305

2.1.3 KSPython.RocketFuelTankParts module

This submodule is responsible to house liquid rocket fuel tanks to be used on simulation.

Note:

- Id Name is the name assigned to the part to be imported and inserted into the code.
- KR12 is divided into two parts, one for engine and one for fuel, both parts must be added if using it.

Id Name	Name	Mass Full [ton]	Mass Empty [ton]	Cost
R4	R-4 ‘Dumpling’ External Tank	0.1238	0.0138	50
R11	R-11 ‘Baguette’ External Tank	0.3038	0.03338	50
R12	R-12 ‘Doughnut’ External Tank	0.3375	0.0375	147
OscarB	Oscar-B Fuel Tank	0.225	0.025	70
FLT100	FL-T100 Fuel Tank	0.5625	0.0625	150
FLT200	FL-T200 Fuel Tank	1.125	0.125	275
FLT400	FL-T400 Fuel Tank	2.25	0.25	500
FLT800	FL-T800 Fuel Tank	4.5	0.5	800
X2008	Rockomax X200-8 Fuel Tank	4.5	0.5	800
X20016	Rockomax X200-16 Fuel Tank	9	1	1550

continues on next page

Table 1 – continued from previous page

Id Name	Name	Mass Full [ton]	Mass Empty [ton]	Cost
X20032	Rockomax X200-32 Fuel Tank	18	2	3000
Jumbo64	Rockomax Jumbo-64 Fuel Tank	36	4	5750
S33600	Kerbodyne S3-3600 Tank	20.25	2.25	3250
S37200	Kerbodyne S3-7200 Tank	40.5	4.5	6500
S314400	Kerbodyne S3-14400 Tank	81	9	13000
KR12_ft	LFB KR-1x2 “Twin-Boar” Liquid Engine	42.5	10.5	17000
Mk2RS	Mk2 Rocket Fuel Fuselage Short	2.29	0.29	750
Mk2R	Mk2 Rocket Fuel Fuselage	4.57	0.57	1450
Mk3RS	Mk3 Rocket Fuel Fuselage Short	14.29	1.79	2500
Mk3R	Mk3 Rocket Fuel Fuselage	28.57	3.57	5000
Mk3RL	Mk3 Rocket Fuel Fuselage Long	57.14	7.14	10000
C7BA	C7 Brand Adapter - 2.5m to 1.25m	4.57	0.57	800
C7BAS	C7 Adapter Slanted - 2.5m to 1.25m	4.57	0.57	800
Mk2125	Mk2 to 1.25m Adapter Long	4.57	0.57	1050
Mk2125L	Mk2 Bicoupler	2.29	0.29	860
Mk2Bi	Kerbodyne S3-14400 Tank	81	9	13000
A25Mk2	2.5m to Mk2 Adapter	4.57	0.57	800
Mk3Mk2	Mk3 to Mk2 Adapter	11.43	1.43	2200
Mk325	Mk3 to 2.5m Adapter	14.29	1.79	2500
Mk325S	Mk3 to 2.5m Adapter Slanted	14.29	1.79	2500
Mk3375	Mk3 to 3.75m Adapter	14.29	1.79	2500
ADTP23	Kerbodyne ADTP-2-3	16.88	1.88	1623

2.1.4 KSPython.BoosterParts module

This submodule is responsible to house solid rocket engines to be used on simulation.

Note:

- Id Name is the name assigned to the part to be imported and inserted into the code.

Id Name	Name	Mass Full [ton]	Mass Empty [ton]	Cost	Thrust atm	Thrust vac	ISP atm	ISP vac
RT5	RT-5 “Flea” Solid Fuel Booster	1.5	0.45	200	162.91	192	140	165
RT10	RT-10 “Hammer” Solid Fuel Booster	3.56	0.75	400	197.9	227	170	195
BACC	BACC “Thumper” Solid Fuel Booster	7.65	1.5	850	250	300	175	210
S1	S1 SRB-KD25k “Kick-back”	24	4.5	2700	593.86	670	195	220
Sepra-tron	Sepratron I	0.1	0	75	13.79	18	118	154
FM1	FM1 “Mite” Solid Fuel Booster	0.375	0.075	75	11.012	12.5	185	210
F3S0	F3S0 “Shrimp” Solid Fuel Booster	0.875	0.155	150	26.512	30	195	215
S217	S2-17 “Thoroughbred” Solid Booster	70	10	9000	1515.217	1700	205	230
FM1	FM1 “Mite” Solid Fuel Booster	144	21	18500	2948.936	3300	210	235

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

k

INDEX

T