**Top Force and Speed Method**

**Parameters:**

**Current:**

step (seconds): Time step between iterations

total\_time (seconds): Used for creating long enough arrays

wheel\_radius (meters): Radius of wheel of bike

gearing (ratio): gear ratio from motor to wheel

rider\_mass(kg): Mass of rider

bike\_mass(kg): Mass of bike

gravity(m/s^2): Acceleration of Gravity

air\_resistance or drag coefficient(dimensionless): resistance of air considering shape of bike

air\_density(kg/m^2): density of air

frontal\_area(m^2): Frontal cross sectional area of the bike, the area of the bike the air will hit

rolling\_resistance(dimensionless): coefficient of friction given to material of tire and ground

top\_torque(Nm): top amount of force the motor can output.

top\_rpm(rpm): top rpm of motor

chain\_efficiency: power efficiency of the chain

battery\_efficiency: power efficiency of the batteries

motor\_torque\_constant(Nm/amps rms): ratio to convert amperes to motor torque of motor

motor\_rpm\_constant(rpm/voltage): ratio to convert voltage to rpm

series\_cells: number of cells in series in the bike.

max\_amphour(amphour): max amphours in the bike. Capacity

batt\_max\_current: max Current out of the batteries (used for power limiting)

**Old:**

efficiency: percentage of energy used by batteries that is used by the bike

**Output Arrays:**

time (seconds): Cumulative time at each time step

distance(meters): Cumulative distance at each time step

l\_speed(m/s): look up speed at each time step

t\_speed(m/s): Top speed, the speed after being compared to top speed

c\_force(newton): compare force, the force before compared to top force

speed(m/s): speed after all comparisons, official speed for time step

force(newton): force after all comparisons, official force for time step

power(watts): power at each time step not including inefficiency

energy(watt/hour): Cumulative energy at each time step

acceleration(newton): force associated to acceleration at each time step

drag(newton): force associated to drag at each time step

slope(ratio): slope of the rode at each time stop

incline(newton): force associated to the inclination of the road at each time step

rolling(newton): force associated to rolling resistance at each time stop.

motor\_torque(Nm): motor torque used for efficiency

motor\_rpm(rpm): rpm of the motor used for efficiency

motor\_loss(watts): power loss due to motor inefficiency

motor\_controller\_loss(watts): power loss due to motor controller inefficiency

chain\_loss(watts): power loss due to the chain inefficiency

battery\_loss(watts): power loss due to battery inefficiency

total\_power(watts): power to overcome including inefficiency

arms(amps rms): amps rms out of motor controller used for efficiency

vrms(volts rms): voltage rms out of the motor controller used for efficiency

motor\_efficiency(%): motor efficiency

motor\_controller\_efficiency(%): motor controller efficiency

chain\_power(watts): power required at chain, after chain losses

motor\_power(watts): power required at motor, after motor and chain losses

motor\_controller\_power(watts): power required at motor controller after motor,chain,and controller losses

battery\_power(watts): power required at batteries given all losses

voltage(volts): traction pack voltage

top\_force(newton): top allowable force at that time (given motor)

top\_speed(m/s): top allowable speed at that time (given motor)

top\_power(watts): top allowable power at that time (given motor and battery pack)

amphour

**Logic Flow:**

1) Time:

Step time by time step

2) Distance:

Determine current distance given time step and previous speed

If over max distance end simulation

3) Look Up Speed:

Look up speed using a distance to speed look up table

For IOM use distance to speed table of a faster bike

4)Top Speed Check:

Check if look up speed is higher than top speed of simulated bike.

If higher than stop speed make speed equal to top speed

5) Force:

Calculate force necessary for bike to apply to ground to overcome forces (drag, rolling resistance, inclination, acceleration)

6)Top Force Check:

Check if force is higher than top force of simulated bike

If higher find speed that gives top force and set force equal to top force

7) Power:

Calculate power from force, speed, and efficiency

8) Power Check:

Check power given motor and battery limits

9) Energy

Calculate Energy (KW/hour) from power and time step

**Code:**

notation: [n] means previous step [n+1] means current step

1)Time

*time[n+1] = time[n] + step*

2)Distance

*distance[n+1] = distance[n] + speed[n]\*step*

3) Look up

*l\_speed[n+1] = distancetospeed\_lookup(distance[n+1])*

4) Check To Speed

*if l\_speed[n+1] > top\_speed:*

*t\_speed[n+1] = top\_speed*

*else:*

*t\_speed[n+1] = l\_speed[n+1]*

5) Force

*acceleration[n+1] = mass\*((speed[n+1] - speed[n])/step)*

*drag[n+1] = 0.5 \* drag\_area\*air\_density\*speed[n+1] \*\*2*

*altitude[n+1] = distancetoaltitude\_lookup(distance[n+1])*

*slope[n+1] = (altitude[n+1] - altitude[n])/(distance[n+1] - distance[n])*

*incline[n+1] = mass\*gravity\*slope[n+1]*

*rolling[n+1] = mass\*gravity\*rolling\_resistance*

*force[n+1] = acceleration[n+1] + drag[n+1] + incline[n+1]*

6) Check Top Force

*if c\_force[n+1] > top\_force:*

*#find speed that gives top force*

*speed[n+1] = (opt.fsolve(force\_solve,t\_speed[n+1],n))[0]*

*force[n+1] = Force(speed[n+1],n)*

*else:*

*speed[n+1] = t\_speed[n+1]*

*force[n+1] = c\_force[n+1]*

#make sure force is not negative

*force[n+1] = np.max([0,force[n+1]])*

7) Power

*power[n+1] = (force[n+1] \* speed[n+1])*

*motor\_rpm[n+1] = ((speed[n+1])/(wheel\_radius\*2\*np.pi)) \* gearing \* 60*

*motor\_torque[n+1] = (force[n+1] \* wheel\_radius)/gearing*

*arms[n+1] = motor\_torque[n+1]/motor\_torque\_constant*

*vrms[n+1] = motor\_rpm[n+1]/(motor\_rpm\_constant)\*(1/(sqrt2))*

#power loss calculated using efficiency mapping

*motor\_loss[n+1] = power[n+1]\*(1-motor\_eff\_grid[np.int(np.around(motor\_rpm[n+1]))] [np.int(np.around(motor\_torque[n+1]))])*

#power loss calculated using efficiency mapping

*motor\_controller\_loss[n+1] = power[n+1]\*(1-motor\_controller\_eff\_grid[np.int(np.around(vrms[n+1]))][np.int(np.around(arms[n+1]))])*

*chain\_loss[n+1] = power[n+1]\*(1-chain\_efficiency)*

*battery\_loss[n+1] = power[n+1]\*(1-battery\_efficiency)*

*Efficiency = motor\_loss[n+1] + motor\_controller\_loss[n+1] + chain\_loss[n+1] + battery\_loss[n+1]*

*total\_power[n+1] = Efficiency(n) + power[n+1]*

8) Energy

*energy[n+1] = energy[n] + power[n+1]\*(step/(60\*60))*