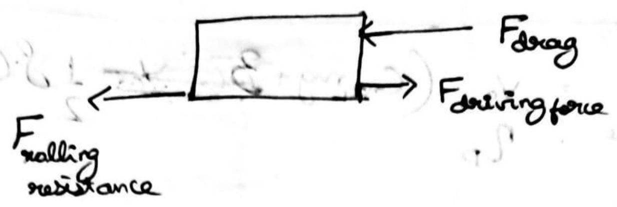
# *VELOCITY OPTIMIZATION MODEL FOR SOLAR CAR*

**-S.BHUVANESH**
**ED23B063**

## *GETTING TO KNOW ABOUT YOUR CAR:*

### *POWER CONSUMED BY THE CAR:*

~~* give V~~



$$\eta_D P_{drive\ power} = F_{drive} \times V$$

$\eta_D \Rightarrow$ efficiency of drive system

$$F_{rolling\ resistance} = C_{roll} mg \; \cancel{[illegible]} + \cancel{\frac{1}{2}\rho C_d A v^2}$$

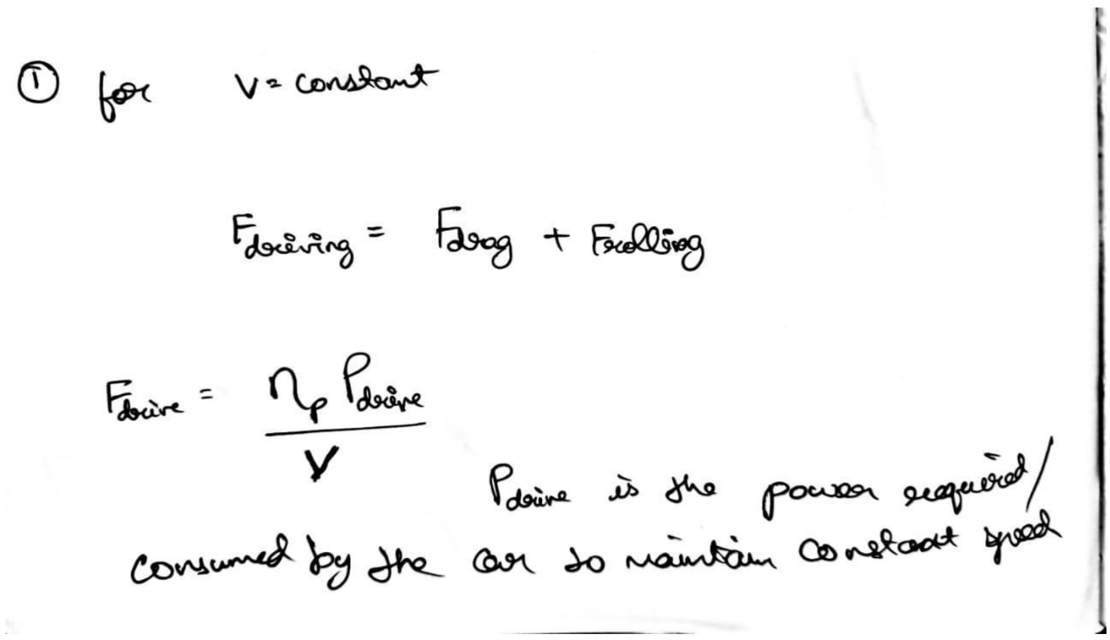
[ $C_{roll} \Rightarrow$ rolling resistance coefficients ]

$$F_{drag} = \frac{1}{2}\rho C_d A V^2$$

$$F_{driving} - \left(F_{drag} + F_{rolling\ resistance}\right) = ma$$

***When velocity is constant :***

- Assuming there is no gradient in the road:



① for V = constant

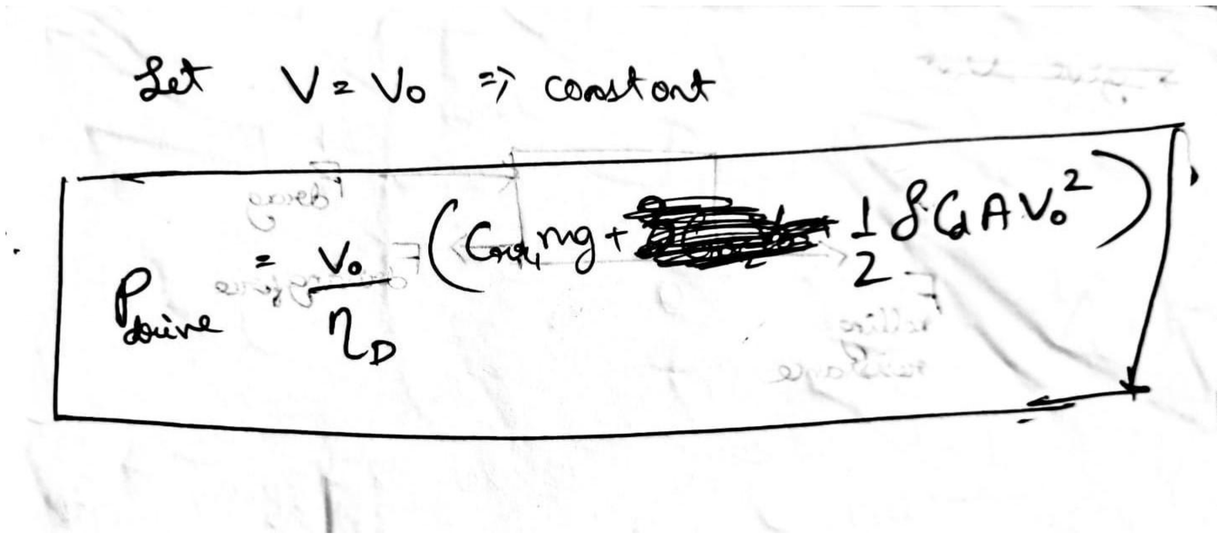
$$F_{driving} = F_{drag} + F_{rolling}$$

$$F_{drive} = \frac{\eta_p P_{drive}}{V}$$

$P_{drive}$ is the power required/consumed by the car to maintain constant speed

The net driving power given by the motor should be equal in magnitude and opposite in direction to the net resitive forces.Through this equation we derive that:

“ The power consumed by a car at constant speed is constant and is related to the speed by following equation:



Let $V = V_0 \Rightarrow$ constant

$$\boxed{P_{drive} = \frac{V_0}{\eta_D}\left(C_{rr} mg + \frac{1}{2}\rho C_d A V_0^2\right)}$$

”

***<u>POWER CONSUMED DURING CONSTANT ACCELERATION AND DECELERATION:</u>***

② when $a = \frac{dV}{dt}$ is constant :- $a = a_0 \Rightarrow$ constant

$$F_{driving} - (F_{ror} + F_{drag}) = ma_0$$

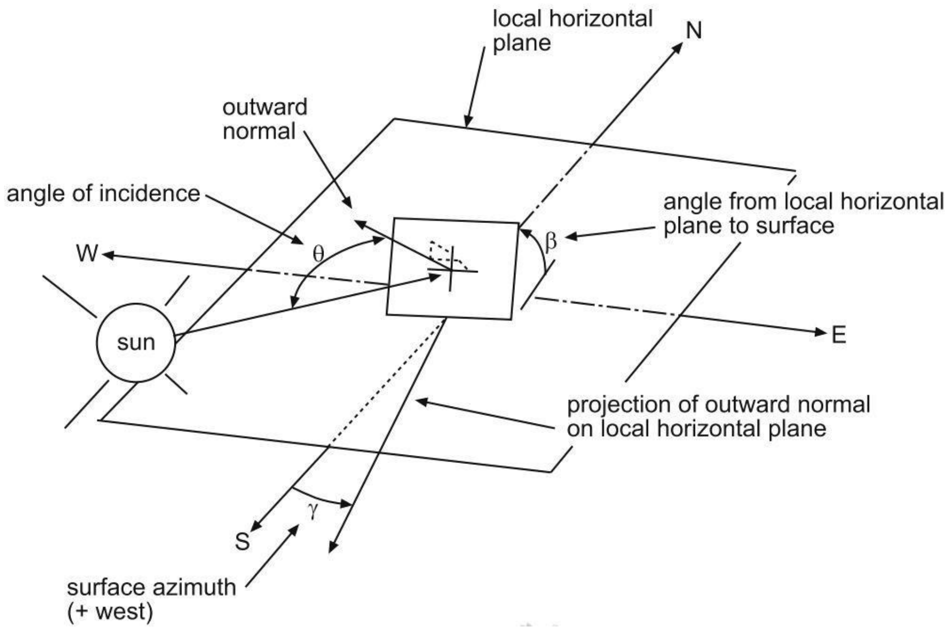
$\frac{\eta_0}{V} P_{driving} = ma_0 + C_{ror} mg +$ [redacted] $+ \frac{1}{2}\rho C_d A V^2$

$\boxed{P_{driving} = \eta_0 \left[ ma_0 V + C_{ror} mgV + \right.}$ [redacted] $\boxed{\left. \frac{1}{2}\rho C_d A V^3 \right]}$

During acceleration $a_0$ is positive, whereas during deceleration it is negative, that's the only difference.

## POWER GAINED BY THE CAR:

### Calculating power gained by solar panel at a given tilt at a given time:


local horizontal plane
N
outward normal
angle of incidence
θ
W
angle from local horizontal plane to surface
β
sun
E
projection of outward normal on local horizontal plane
γ
S
surface azimuth (+ west)

$P_{gain} = I_0 A \cos\theta$

$\theta \rightarrow$ angle of inclination

By using a relation given by Benford and Bock:-

$$\boxed{\begin{aligned}\cos\theta = {} & \sin\delta \sin La \cos\beta - \sin\delta \cos La \sin\beta \cos\gamma \\ & + \cos\delta \cos La \cos\beta \cos\omega + \cos\delta \sin La \sin\beta \cos\gamma \cos\omega \\ & + \cos\delta \sin\beta \sin\gamma \sin\omega\end{aligned}}$$

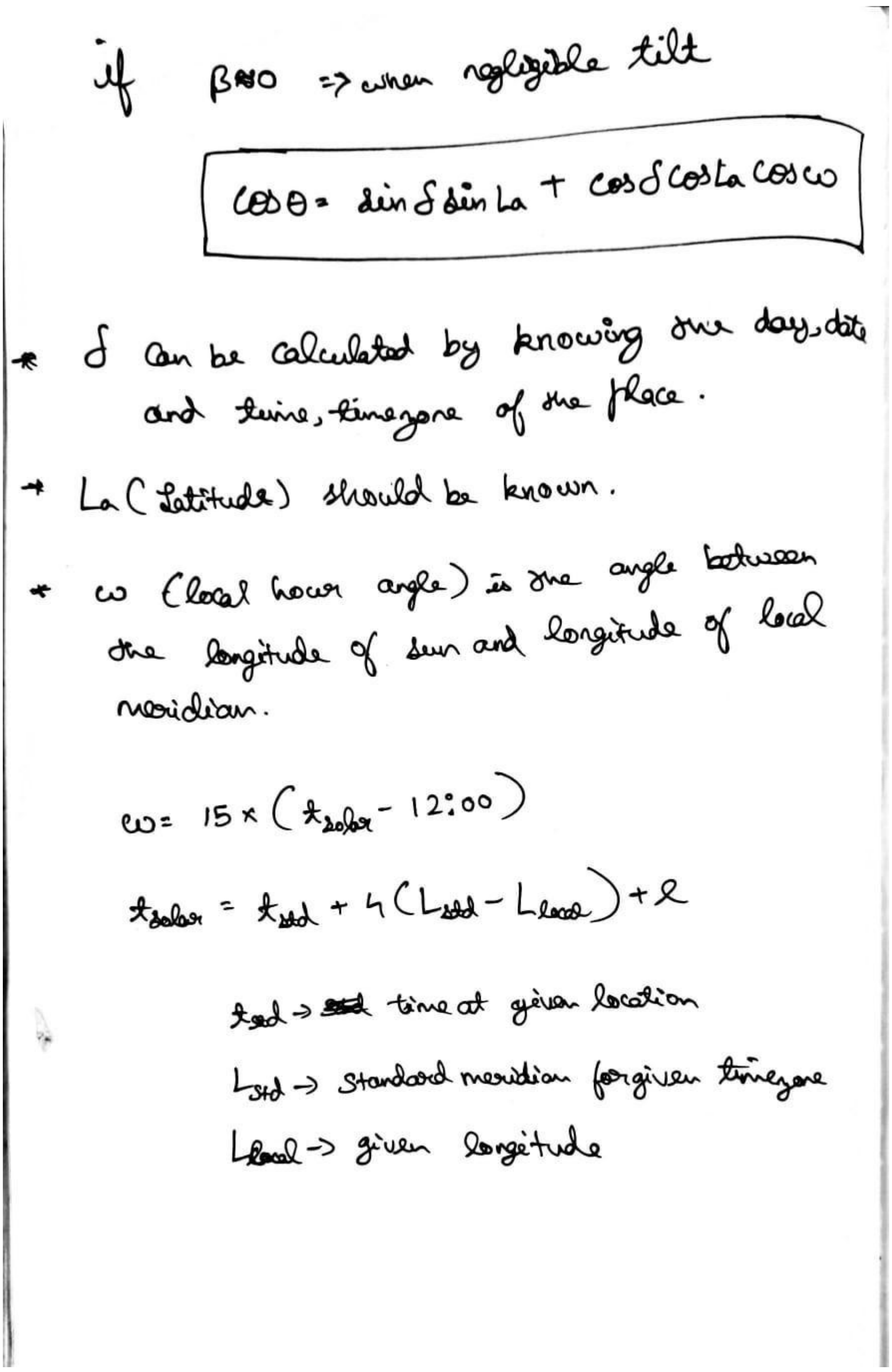
La - Latitude

$\beta$ - tilt angle

$\gamma$ - surface azimuth angle

$\delta$ (angle of declination) - angle between rays of sun and plane of equator.

$\omega$ - angle between the longitude of solar panel and the longitude at which the sun currently lies.



if $\beta \approx 0$ => when negligible tilt

$$\boxed{\cos\theta = \sin\delta \sin L_a + \cos\delta \cos L_a \cos\omega}$$

* $\delta$ can be calculated by knowing the day, date and time, timezone of the place.
* $L_a$ (Latitude) should be known.
* $\omega$ (local hour angle) is the angle between the longitude of sun and longitude of local meridian.

$\omega = 15 \times (t_{solar} - 12{:}00)$

$t_{solar} = t_{std} + 4(L_{std} - L_{local}) + E$

$t_{std}$ -> ~~std~~ time at given location

$L_{std}$ -> standard meridian for given timezone

$L_{local}$ -> given longitude

Integrate this power gained through a fixed time interval to get the energy gained.

## ***Power gained through regenerative Braking:***

Regeneration should be used to slow the car before applying the brakes. This recovers some of the kinetic energy stored in the car and reduces wear on the brakes.

$$\frac{1}{2}mv_1^2 + mgh_1 = \Delta E + \frac{1}{2}mv_2^2 + mgh_2 + W + Q$$

$\Delta E \rightarrow$ energy regained through regenerative braking

$Q \rightarrow$ energy lost as heat

$W \rightarrow$ work done against drag + rolling resistance.

$$\Rightarrow \Delta E + W + Q = \frac{1}{2}m(v_1^2 - v_2^2) + mg(h_1 - h_2)$$

assuming $h_1 \approx h_2$

$$\Rightarrow \Delta E + W + Q = \frac{1}{2}m(v_1^2 - v_2^2)$$

$$\boxed{\Delta E + Q = \frac{1}{2}m(v_1^2 - v_2^2) - W}$$

$$\boxed{\Delta E = \frac{1}{2}m(v_1^2 - v_2^2) - W - Q}$$

$$P_{gained} = \cancel{\frac{1}{2}} \times \cancel{2} \times m(v_1 a_1 - v_2 a_2) - P_{resistive\ forces} - P_{heat\ lost}$$

$$\boxed{P_{gain} = m(v_1 a_1 - v_2 a_2) - P_{resist} - P_{heat\ lost}}$$

Here we cant calculatively say about the power due to heat lost, power lost due to resistive forces.We can only find all those with the help of datas from sensors such as heat sensor(thermistor) , to record the heat loss values.

## *POWER STORED IN THE CAR*

***soc:*** The State of Charge (SoC) of a battery is typically defined as the ratio of the current charge level to the maximum charge capacity of the battery, expressed as a percentage.soc can also found out in terms of energy, that is , it can also be defined as the ratio of the current energy level of the battery and the maximum energy which can be stored in the battery.

If no energy is being drained from the battery:

$$Soc = \frac{E_{current}}{E_{max}} \times 100$$

If energy is getting drained from battery**(E_drain>0)** or if battery is getting recharged by solar energy**(E_drain<0)**:

$$100 \times \left( \frac{E_{current} - E_{drain}}{E_{capacity}} \right) = SoC$$

**SOH:**

The SOH tells us about the degree of health of the battery.It indicates how much the battery's capacity or performance has degraded over time due to factors such as usage, aging, and environmental conditions.

SOH= (Current capacity/ original capacity)*100

- soc provides real-time information about the current charge level of the battery. By predicting the soc, the energy management system of the solar car can optimize the use of stored energy, ensuring efficient operation and extending the vehicle's range.Predicting SoC enables better estimation of the driving range remaining before the battery needs recharging.knowing the soc can prevent overcharging and overdischarging of the battery.

  The SOH indicates the health of the battery, knwoing its real time value allows us to know when the battery might wear down and be prepared for it ,instead of facing a sudden battery breakdown problem.

- Battery temperaature is useful to be known in a strat model.Because it significantly affects battery performance and battery life.Monitoring the battery temperature can help prevent thermal runaways and helps optimizing charging and discharging strategies.Voltage and current also are another set of useful parameters to be known in the model, since we can easily get the power output from the battery using these.

***POWER LOST MECHANICALLY BY THE CAR***

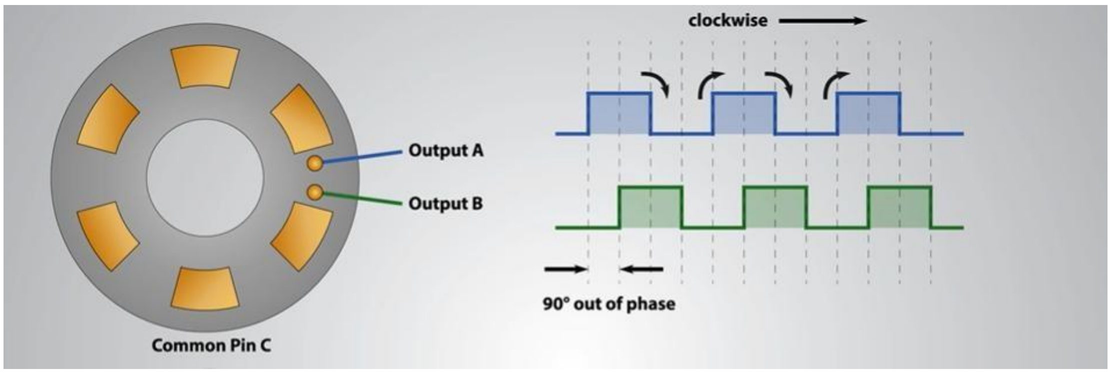
# ***GETTING TO KNOW YOUR DATA***

## ***MOTOR***

We can get data of

- Speed
- Direction of rotation

Using **encoder**.



The above is an incremental encoder.It consists of a disc with many slits placed between a photodetector sensor and light ray source. When light passes through slit as motor rotates, output is high and when light is blocked output is low.The direction is determined by the phase difference in channels A and B.Both channels are always 90 degrees out of phase.If it rotates CW, then A leads B. If CCW , B leads A.

We can  get data of

- Temperature by connecting thermistor to motor windings.
- Voltage output from motor controller

Additionally PID controller is also connected to the motor, which recieves feedback from various sensors like encoders.PID stands for Proportional-Integral-Derivative, which are the three terms that make up the controller.

The P block produces a signal proportional to the magnitude of the errorIn motor control, the proportional action adjusts the motor speed based on how far the actual speed is from the desired speed.
The integral term of the PID controller accounts for past errors over time and works to eliminate steady-state error. It continuously integrates the error signal over time and applies a corrective action proportional to the accumulated error. Integral control ensures that the system eventually reaches and maintains the desired setpoint.The longer the error and greater the amount , greater is the integral output.

The derivative term of the PID controller predicts future error based on the rate of change of the error signal. It anticipates the system's behavior and applies a corrective action proportional to the rate of change of the error.The faster the error changes, the larger the derivative output.

## *SOLAR*

For data acquisition for predictive model I would get my information on solar data from:
**Solar/weather forecast models**.From here we can aquire the possible irradiance,shadow factor(how much a cell gets shadowed) since shadowing of cells reduced the solar cell output.

Test run records of the solar cell to find its actual efficiency.And we can also use the datasheet of the solar cell.

For data acquisition of real time model I would use:

**Pyranometer.**It tracks the irradiance of the sun.It directly measures the total solar radiation striking a surface.

**MPPT.**It tracks the maximum power point of the solar cell by constantly monitoring the **dP/dV** value. Thus we can get the power output and voltage of the solar cell from the MPPT.

## *BATTERY*

We use the BMS to obtain data about the battery.

A Battery Management System (BMS) is essentially the brain of the solar car’s battery, constantly monitoring and controlling its operation to ensure safe, healthy, and optimal performance. It’s functions are monitoring:

**SOC:** It estimates the remaining total capacity of the battery based on monitoring and using the sensor data of voltage, current, and temperature data.

**cell voltages:** The BMS continuously measures the voltage of each individual cell within the battery pack.

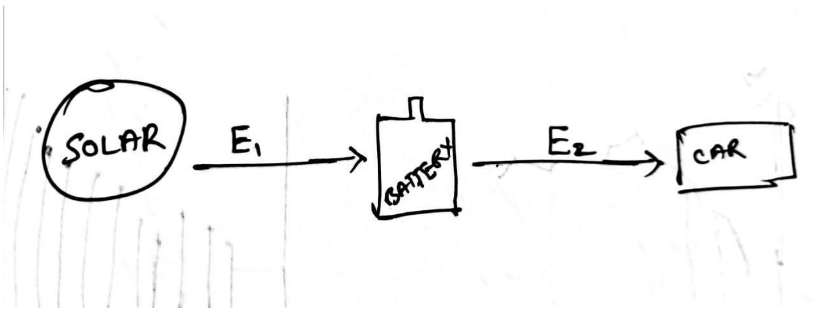
**Battery temperature:** It keeps track of the battery's temperature. It activates the cooling systems based on temperature readings.

**cell current:** It monitors the current flowing in and out of the battery, preventing ‘overcharging’ and ‘over-discharging’ by checking and regulating the flow.

**Cell balancing:** During charging or discharging, individual cell voltages can become unbalanced. The BMS performs a crucial role of cell balancing, transferring energy between cells to maintain equal voltages and hence maintaining optimal performance and long life for the battery.

**Safety features:** The BMS can trigger safety measures like shutting down the battery in case of critical issues like overheating, excessive current, or internal short circuits or, for prolonging the lifespan; cuts off the voltage supply from the battery after reaching the over-discharging limit.

In my opinion we can compare the data obtained from the BMS with the data of the power cosumed by car and solar data using this logic:



Let us say the battery has energy **E** right now. Solar provides **E1** and car consumes **E2.** So the net energy drained from battery is :

**E** drain **=E2 - E1**

$$P_{drain} = \frac{V}{\eta_D}\left(C_{ron} mg + \frac{1}{2}\delta C_d A V^2\right) - IA\cos\theta$$

So battery energy decreases:

**E - E** drain **=> Energy remaining in battery**

This gives the net energy consumption of the car. Using this we can find the Soc of the battery, without involving any other battery parameters except battery capacity , and battery voltage.

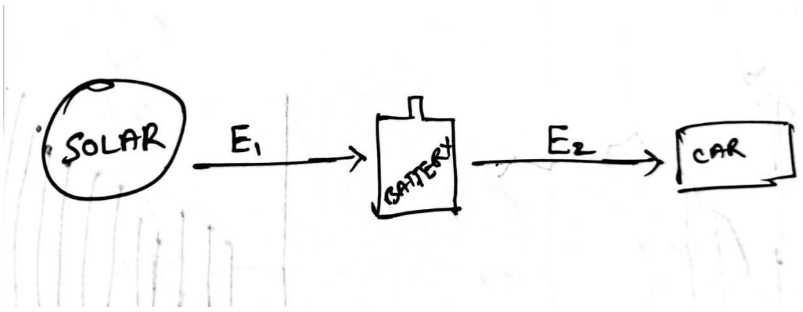
## ***THERMAL DATA***

Temperature data of the solar panel is really important, since solar panels efficiency are greatly affected by temperatures. Their efficiency decreases with an increase in temperature.So this affects the power gained by the battery through solar panels. The temperature data of the battery is also really crucial since it tells us about the performance and battery life.Monitoring the battery temperature can help prevent thermal runaways and helps optimizing charging and discharging strategies.This data can be obtained with the help of BMS, which gets its data from various sensors such as thermistor or thermocouples or infrared sensors, etc…

## DECIDING THE HIGH LEVEL OF THE MODEL

## OBJECTIVE FUNCTION

I would prefer to use the power flow from the battery as my objective function.I would want to minimize it.Since using this data we can optimize velocity as well as at the same time get the soc data of the car at any given time easily.



Let us say the battery has energy **E** right now. Solar provides **E1** and car consumes **E2.** So the net energy drained from battery is :

**E** $_{\text{drain}}$ **=E2 - E1**

$$P_{drain} = \frac{v}{\eta_D}\left(C_{rr} m g + \frac{1}{2} \rho C_d A v^2\right) - I A \cos\theta$$

So battery energy decreases:

**E - E** $_{\text{drain}}$

This gives the net energy consumption of the car. Using this we can find the Soc of the battery, without involving any other battery parameters except battery capacity , and battery voltage.

$$P_{drain} = P_2 - P_1$$

$$P_{drain} = \frac{v}{\eta_D}\left(C_{rr} mg + \frac{1}{2}\rho C_d A v^2\right) - IA\cos\theta$$

$$f(x) = P_{drain} \Rightarrow \text{objective function}$$

$$\int_{t_1}^{t_2} P_{drain}\, dt \Rightarrow E_{drain}$$

We know :-

Battery Capacity = 4300 Ah
voltage = 150 V

(I got this data [redacted] after asking many people)

Energy (which can be stored) = Capacity × Voltage

$$\boxed{Energy_{capacity} = 645000}$$

$$100 \times \left( \frac{E_{current} - E_{drain}}{E_{capacity}} \right) = SOC$$

2 $$100 \left( \frac{E_{current} - E_{drain}}{\cancel{E_{capa}}\,645000} \right) = SOC$$

$E_{current} \rightarrow$ Energy in the battery when we are measuring it.

For the code I assumed that the battery starts at full charge.

## *CONSTRAINTS*

The constraints which we can apply are:

* $V_{initial} = 0$
* $V \geq 0$
* $0 \leq SOC \leq 100$

Darwin:

Latitude: Darwin is located at approximately 12.4634° S.
Longitude: Darwin is located at approximately 130.8456° E.

Adelaide:

Latitude: Adelaide is located at approximately 34.9285° S.
Longitude: Adelaide is located at approximately 138.6007° E.

So the latitude range is (12,35).longitude range is (130,139). [approx].

The soc constraint is used so that we don’t have a battery which can be negatively charged or can be charged above 100 percent. The latitude and longitude constraints are used to find the angle of inclination of the sun to the solar panel, which is used to calculate the solar power.

## ***DECISION VARIABLES:***

Decision variables or control variables are variables, which we can adjust to optimize our objective function to minimum or maximum.We can control velocity by Formulating velocity control as an optimization problem and solve it iteratively to find the optimal velocity trajectory that maximizes energy efficiency, minimizes travel time, or achieves other objectives. Optimization-based control techniques can incorporate constraints and objective functionss to generate optimal control actions.

We can also use PID controller which can adjust the velocity by measuring the difference between the desired velocity and actual car velocity by getting data from sensors.The controller calculates control signals to accelerate or decelerate the car, aiming to minimize the error and maintain the desired velocity.

## ***OPTIMISATION TYPE:***

For predictive optimisation we can use gradient descent:

**Gradient descent** is an algorithm to minimize a function by optimizing its parameters. It basically works on this one formula:

New value = current value - (step size * gradient)

Here:

Step size is the value by how much we are to shift the current value,and in coding terms we refer to it as **learning rate.**
The gradient descent works because it follows the direction of the negative gradient of the function.if it crosses the minima and goes to the direction of positive gradient; the negative of the gradient is negative.this signifies that it wants to travel towards the direction of negative gradient, that is towards the minima.

We can use feedback

## SOME MATH?

### ***LAGRANGIAN***

we used the method of Lagrange multipliers. The method
says that the local extreme values of a function ƒ(x, y, z) whose variables are subject to a constraint g(x, y, z) = 0 are to be found on the surface
g = 0 among the points where ;

$\nabla f = \alpha \nabla g$

for some scalar α(called a Lagrange multiplier)

**L =** $\nabla f + \alpha \nabla g$

**L,** is the Lagrangian

We use Lagrange multipliers to solve constrained optimization problems.A Lagrangian is properly defined only in the presence of constraints.

### GRADIENT DESCENT

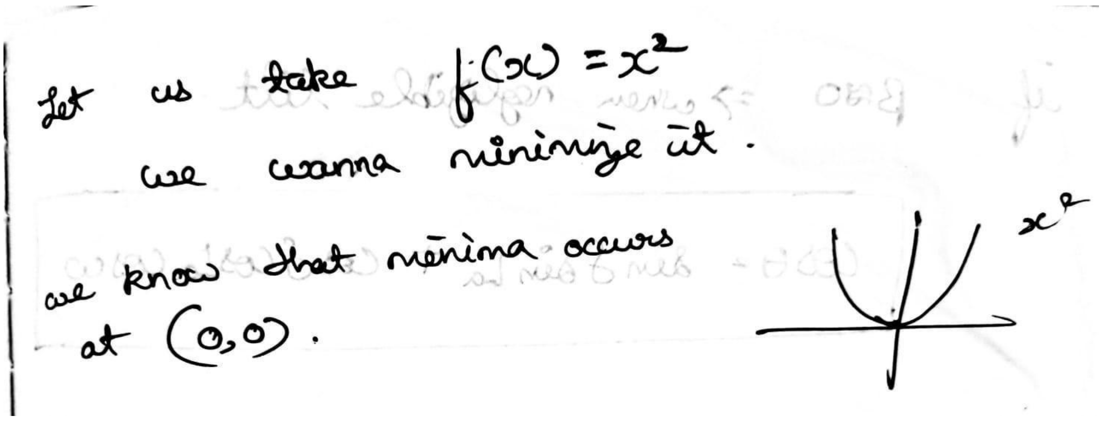
**Gradient descent** is an algorithm to minimize a function by optimizing its parameters. It basically works on this one formula:

New value = current value - (step size * gradient)

Here:

Step size is the value by how much we are to shift the current value,and in coding terms we refer to it as **learning rate.**

Let us see it’s working step by step with an example:


Let us take $f(x) = x^2$

we wanna minimize it.

we know that minima occurs at $(0,0)$.

- We can start by taking any random value of x;
  x_initial=x1.
- Gradient= df(x)/dx = 2x
- X2=x1-learning rate*2x1
- In the working of gradient descent , when we take x=x1, if gradient is negative at that point , the algorithm knows to move towards right since minima occurs at some point on the right, but if gradient is positive it moves towards left.But how much it shifts depends on the learning rate which we decide.

In case of a constraint based optimisation:

Lets take for the same example:

Constraint: g(x)=x=1 => x-1=0

**L=** f(x) + α*g(x)
**L**=x^2 + α*(x-1)

Using lagrangian makes this objective function with a constraint, like a function without any constrraint.And hence now we can optimize **L** using normal gradient descent method.

$$x_{new} = x_0 - \eta \nabla L_x$$

$$\alpha_{new} = \alpha_0 - \eta \nabla L_\alpha$$

The gradient descent works because it follows the direction of the negative gradient of the function.if it crosses the minima and goes to the direction of positive gradient; the negative of the gradient is negative.this signifies that it wants to travel towards the direction of negative gradient, that is towards the minima.

## ***ADAM OPTIMIZER:***

Adam optimizer uses adaptive learning rates based on the estimates of the first and second moments of the gradients to automatically adjust the step size during optimization.It works based on the following:

$$\theta_{t+1} = \theta_t - \frac{\eta \times \hat{m}_t}{\sqrt{\hat{v}_t + e}}$$

$$\hat{m}_t = \frac{m_t}{1-(\beta_1)^t} \qquad \hat{v}_t = \frac{v_t}{1-(\beta_2)^t}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \times \text{gradient}$$

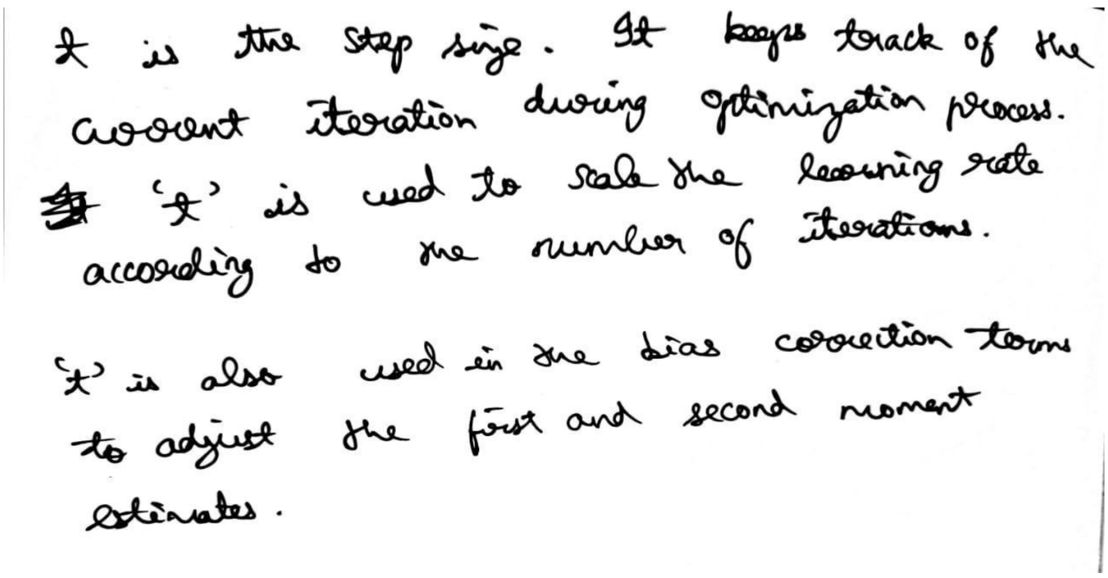
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) \times (\text{gradient})^2$$

$m_t$, $v_t$ are the moment estimates.

These two terms ~~[illegible]~~ keep track of the gradients and gradient squares through iterations.

$m_t$, $v_t$ are initialized as 0. Hence in order to prevent them from being biased towards 0; the biased correction terms :- $\hat{m}_t$ + $\hat{v}_t$ are used.

Here $\eta$ is the learning rate ⇒ determines the size of steps taken during parameter updates.



t is the step size. It keeps track of the current iteration during optimization process.
't' is used to scale the learning rate according to the number of iterations.

't' is also used in the bias correction terms to adjust the first and second moment estimates.

I forgot to explain above that the term 'e' is present to avoid division by 0 error.

## ***DIFFERENTIAL EQUATIONS***

the order of a differential equation refers to the highest order derivative present in the equation, while the degree of a differential equation refers to the highest power to which the highest order derivative is raised.

Let us take :-

given :-

$$f(x, y, y', y'', \ldots y^n) = 0$$

now we should assign new variables to each derivative of $y$ till order $(n-1)$.

$y_1 = y$

$y_2 = y'$

$y_3 = y''$ . . .

$y_n = y^{n-1}$

now we can process these as :-

$\frac{dy_1}{dx} = y_2$ , $\frac{dy_2}{dx} = y_3$ , . . . $\frac{dy_n}{dx} = y^n$

These are the set of adjoint eqns.

these adjoint eqns are ~~a~~ system of first order eqns using the new variables and their derivatives.

we can obtain the solution by ~~to~~ solving all this first order eqns.

Lets take an example:-

$y''(x) + 2y'(x) + y(x) = 0$

To reduce order, we introduce new variables :-

$y_1 = y$

$y_2 = y'$

$y_1^{*} = y_1$

$y_2^{*} = \frac{dy_1}{dx} = y'$

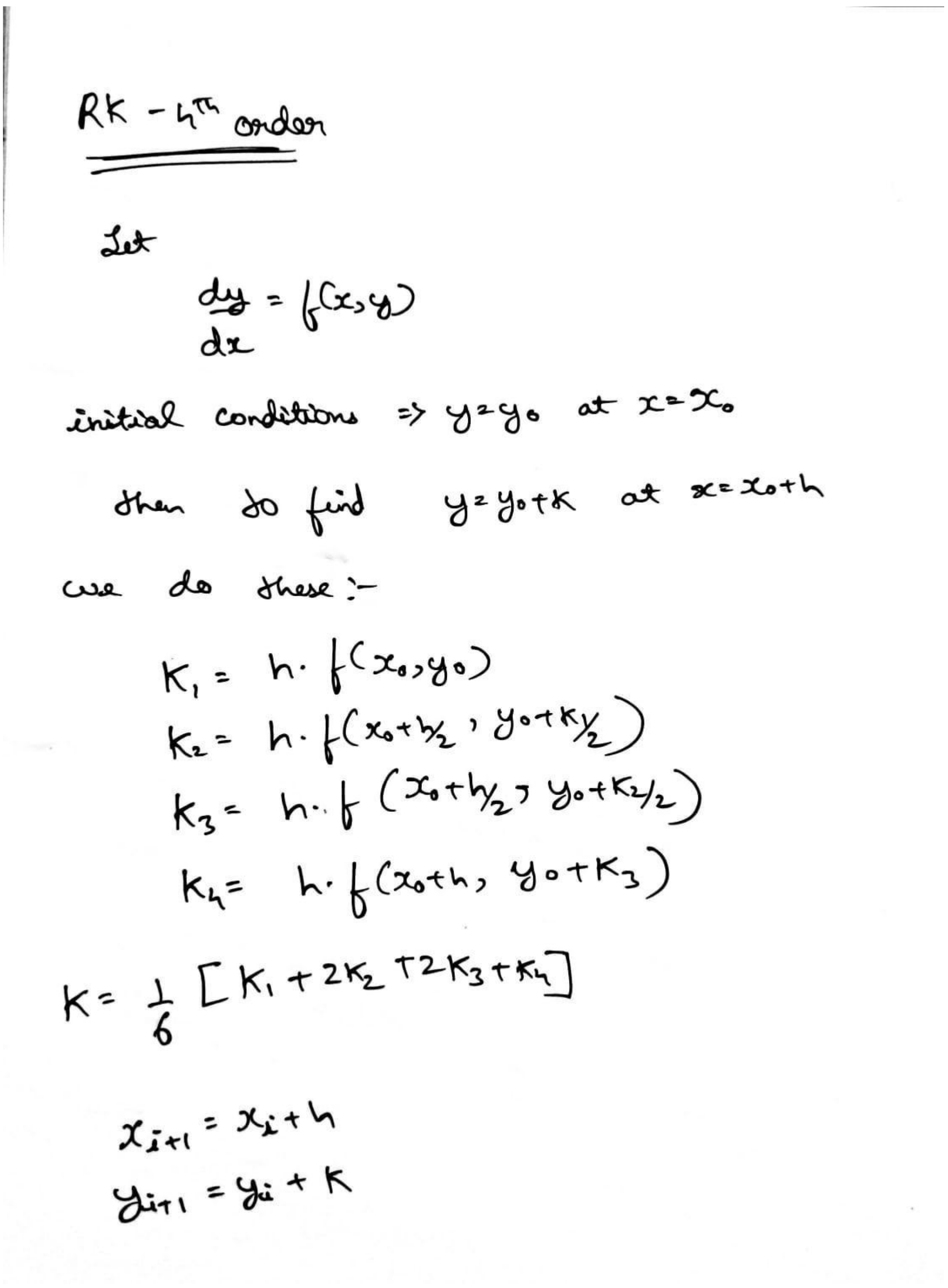
$y'' = \frac{dy_2}{dx}$

using these we get :-

$\frac{dy_2}{dx} + 2y_2 + y_1 = 0$

$\frac{dy_1}{dx} = y_2$

now if we have the initial conditions for $y$, $y'$ we can obtain conditions for $y_1$, $y_2$ and solve this system of equations.



## RK - 4th order

Let

$$\frac{dy}{dx} = f(x, y)$$

initial conditions $\Rightarrow$ $y = y_0$ at $x = x_0$

then to find $y = y_0 + K$ at $x = x_0 + h$

we do these :-

$$K_1 = h \cdot f(x_0, y_0)$$
$$K_2 = h \cdot f(x_0 + h/2, y_0 + K_1/2)$$
$$K_3 = h \cdot f(x_0 + h/2, y_0 + K_2/2)$$
$$K_4 = h \cdot f(x_0 + h, y_0 + K_3)$$

$$K = \frac{1}{6}\left[K_1 + 2K_2 + 2K_3 + K_4\right]$$

$$x_{i+1} = x_i + h$$
$$y_{i+1} = y_i + K$$

## STOCHASTIC GRADIENT DESCENT

Take an example: when we have to optimize a function with a given dataset of 1000 samples, and each samples have n parameters.If you use normal gradient descent, you have to optimize for all the 1000 samples and that too each of the 15 parameters in each sample.And this is too costly to execute.

So to avoid this we use SDG. In SDG we just take a random value out of these 1000 samples and perform normal gradient descent on that.But this too has its disadvantages.There is a possibility that the one value we select is not in sync with the other samples, which would lead to our final optimized value being biased.

In order to avoid this we use something called as mini batch SGD. Here we take k samples in random out of these 1000 samples and do gradient descent on all k samples.

Since we are selecting the samples in random the optimisation path follows a zigzag path , unlike gradient descent which is almost smooth.

SGD processes only a subset of data points (or a mini-batch) in each iteration, making it computationally more efficient than GD, especially for large datasets.

## *CODING THE MODEL YOURSELF*

## ***The code for the model is in the github link below:***

### Repository link:

https://github.com/argonaut031205/ED23B063_STRATEGYAGNIRATH_MODEL.git

### Code direct link:

https://github.com/argonaut031205/ED23B063_STRATEGYAGNIRATH_MODEL/blob/main/ED23B063_STRATMODEL_BHUVANESH.S.py

### Code download link:

https://argonaut031205.github.io/ED23B063_STRATEGYAGNIRATH_MODEL/ED23B063_STRATMODEL_BHUVANESH.S.py

Im just paranoid , so I put 3 links :)

### ***INPUTS AND OUTPUTS:***

I am assuming that the race takes place on 30th June 2025.
I am starting the predictive model at noon(assumption).

This gives me the data on angle of declination(δ) to calculate θ(angle of inclination).I can calculate ω using all this data and additional data of longitude, which is given below.

$\cos(\theta)=\sin(\delta)\cdot\sin(\text{latitude})+\cos(\delta)\cdot\cos(\omega)\cdot\cos(\text{latitude})$

Darwin:

Latitude: Darwin is located at approximately 12.4634° S.
Longitude: Darwin is located at approximately 130.8456° E.

Adelaide:

Latitude: Adelaide is located at approximately 34.9285° S.
Longitude: Adelaide is located at approximately 138.6007° E.

- I am taking solar irradiance input in the range 1300 to 1400

I used the adam optimizing algorithm for my model.I took my objective function as:

$$P_{drain} = \frac{v}{\eta_D}\left(C_{rr} mg + \frac{1}{2}\rho C_d A v^2\right) - IA\cos\theta$$

I am giving:

- Solar irradiance at each time
- Latitude,longitudes at each time, δ,ω
- Predicted velocity
- Front area of car,$C_{rr}$,air density,area of solar panel, mass of car
- Instantaneous acceleration due to predicted velocity

As inputs to the model to get the output as:

- Optimized velocity profile for minimum power consumption from battery.
- Soc of the battery vs time
- Energy consumption at different times.

## ***LOGIC OF IMPLEMENTATION OF MODEL:***

I am optimizing my objective function using adam optimizer.

- First I am importing random,matplotlib for graph plotting, numpy to work with arrays in vector form as well as to implement maths functions.
- Now I am inputting all the necessary inputs as mentioned above.
- To find the cos of angle of inclination I find delta value by the date,time and place.
- I find the local hour angle using formula of tsolar:

$$\omega = 15 \times (t_{solar} - 12{:}00)$$

$$t_{solar} = t_{std} + 4(L_{std} - L_{local}) + E$$

v_values is an array containing velocity values.
I, t, and a are initialized as empty lists.

- Now I Convert lists I and a to numpy arrays for vector operations.
- Now I define the power_drain, gradient functions.

- Now I Define the Adam optimizer function adam_optimizer, which optimizes the velocity (velocity) using the Adam optimization algorithm.
- Inside the optimizer function:

❖ Initialize moment estimates m and v.
❖ Iterate for a specified number of iterations:
❖ Calculate the gradient of the function.
❖ Update moment estimates m and v.
❖ Calculate bias-corrected moment estimates m_hat and v_hat.

❖ Update the optimized velocity .
❖ Check for convergence using the tolerance condition.
❖ Return the optimized velocity.

- Now I input the total energy capacity of the battery to find soc.my assumptions are, it is a 4300mAh ,150V battery (I got these values from a friend of mine in agnirath).
- Energy_max= 4300*150=645000
- I initialize an empty array energy=[].then I append the energy values for each time interval into the array.
- Then I find soc using this and add all soc values to the soc array.

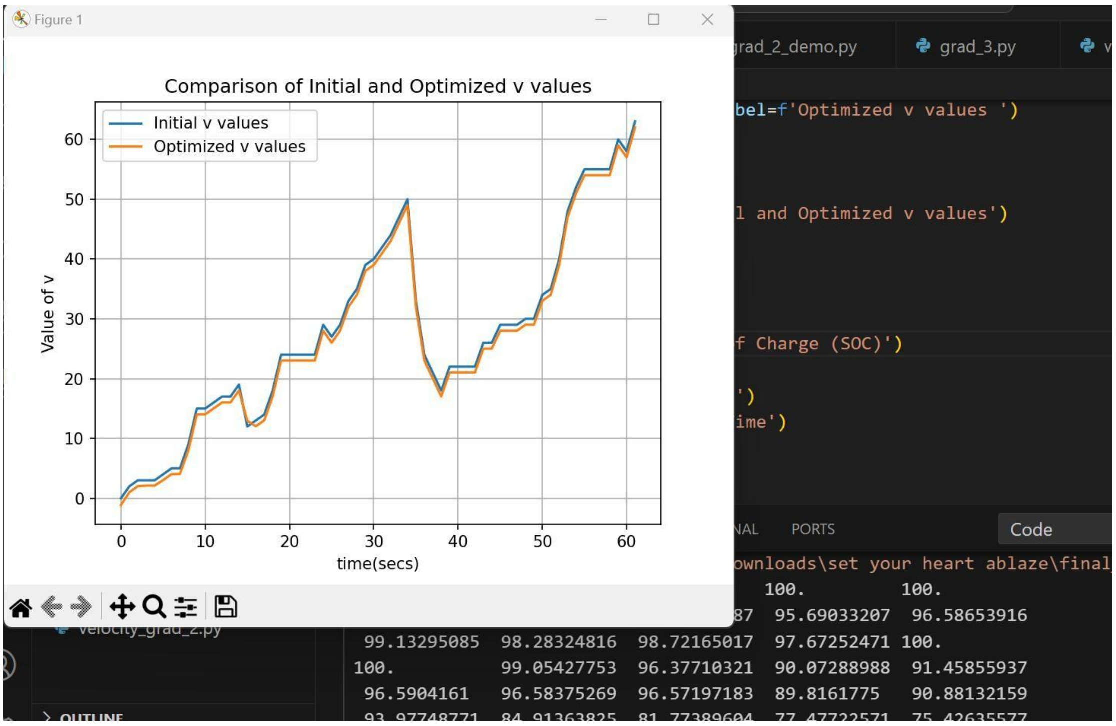
- Now I start plotting the graphs using matplotlib

***<u>REFERENCE CODE:</u>***

```
import numpy as np
import random
import matplotlib.pyplot as plt

#I am assuming that the competition takes place on 30th june 2025,(just assuming a random date)

M = 300
a = []  # acceleration (del v / del t)
c1 = 0.004  # coefficient
g = 9.81  # m/s^2
d = 1.225  # kg/m^3
A = 2  # m^2 (frontal area)
efficiency = 0.65
# I = 1000  # Solar irradiance
A1 = 4  # Panel area

learning_rate = 0.01
num_iterations = 10000
tolerance = 1e-6

v_values = np.array([0, 2, 3, 3, 3, 4, 5, 5, 9, 15, 15, 16, 17, 17, 19, 12, 13, 14, 18, 24, 24, 24, 24, 24, 29,
27, 29, 33, 35, 39, 40, 42, 44, 47, 50, 33, 24, 21, 18, 22, 22, 22, 22, 26, 26, 29, 29, 29, 30, 30, 34,
35,40,48,52,55,55,55,55,60,58,63])

I = []
t = []
optimized_v = []

#the next few lines of code is used in finding the cos of angle of inclination
delta=23.20 #degrees (found out from declination angle calculator in google)
La= np.linspace(-18,-15,62) #latitude
Lo= np.linspace(132,136,62) #longitude

for i in range(len(v_values)):
   I.append(random.randint(1300, 1400))
```

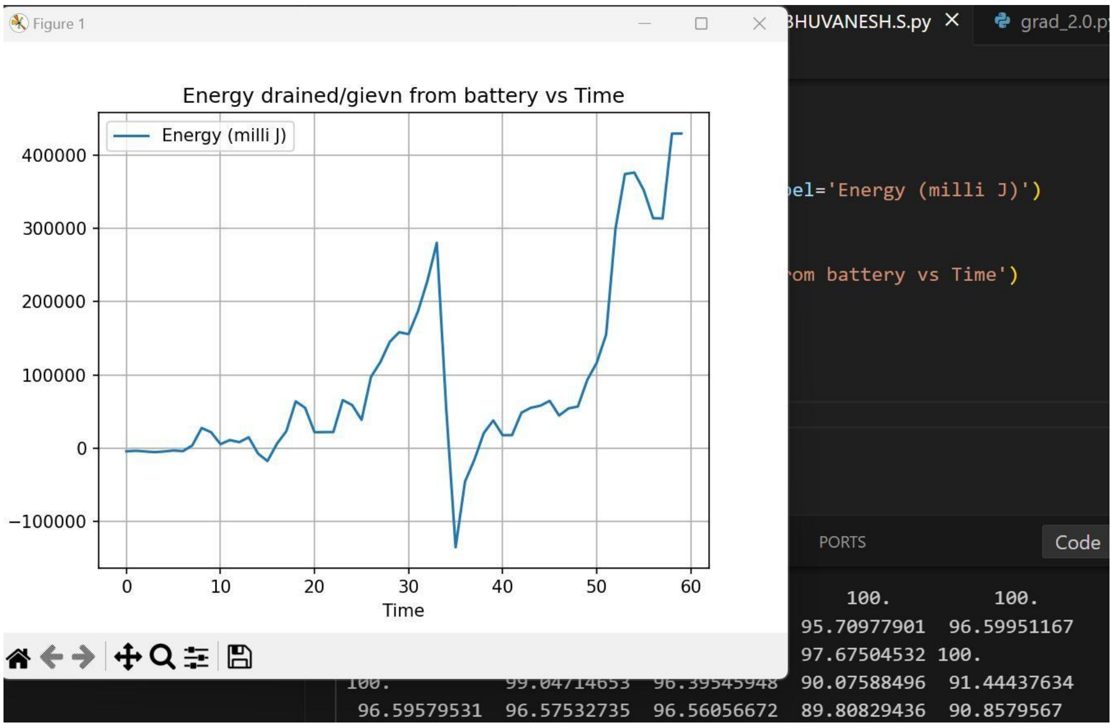
```
for i in range(len(v_values)):
    t.append(i)#here im just creating an array for time
    tsolar= 4*(136-Lo) + t[i]#calculating solar time, this is the formula.if this is not clear i have explained
about this in previous qns
    if i == 0:
        a.append(v_values[i] / 1)
    else:
        delta_v = v_values[i] - v_values[i - 1]
        delta_t = t[i] - t[i - 1]
        if delta_t == 0:
            a.append(0)  # Avoid division by zero
        else:
            a.append(delta_v / delta_t)


omega= 15*(np.array(tsolar)-0)/60 #we use solar time to calculate omega
cos_theta=np.zeros_like(t)# Angle of inclination of sun assuming that the tilted angle of the solar
panel is 0

cos_theta=((np.sin(delta)*np.sin(La))+np.cos(delta)*np.cos(La)*np.cos(omega))
for i in range(len(cos_theta)):
    if cos_theta[i]<0:
        cos_theta[i]=-cos_theta[i]#preventing negative vaues of costheta


I = np.array(I)
a = np.array(a)

def power_drained(v, a, I, costheta):#this the power drained from battery
    return (v * (M * a + c1 * M * g + 0.5 * A * d * (v) * (v)) / efficiency) - ((I) * A1 * costheta)



# Define the gradient of f(x, i) with respect to x
def grad(v, a):
    return (M * np.array(a) + c1 * M * g + 3.5 * A * d * v * v) / efficiency

#defining a function for the adam optimizer
def adam_optimizer(velocity, learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8,
num_iterations=1000,
                tolerance=1e-6):

    m = np.zeros_like(velocity)  # First moment estimate
    v = np.zeros_like(velocity)  # Second moment estimate
    t = 0  # Time step
    v_optimized = []  # Optimized v values

    for _ in range(num_iterations):
        t += 1
        gradient = grad(velocity, a)  # Gradient of power_drained

        m = beta1 * m + (1 - beta1 ) * gradient  # Update first moment estimate
        v = beta2 * v + (1 - beta2 ) * (gradient * gradient)  # Update second moment estimate

        m_hat = m / (1 - np.power(beta1,t) )  # Bias-corrected first moment estimate
        v_hat = v / (1 - np.power(beta2,t) )  # Bias-corrected second moment estimate

        velocity = velocity.astype(float)  # Convert to float to allow for floating-point arithmetic
        velocity -= learning_rate * m_hat / (np.sqrt(v_hat) + epsilon)  # Update velocity

        if np.all(np.abs(gradient)) < tolerance:
            break
    v_optimized.append(velocity)
    return v_optimized

# Optimizing x using Adam optimizer
optimized_v_values = adam_optimizer(v_values)
energy_drain=[]
soc=[]
energy=0
#print(power_drained(v_values,a,I,theta))
p=power_drained(v_values,a,I,cos_theta)
for i in range(0,62):
    if i<60:
        energy_interval = 0.5 * (p[i] + p[i + 1]) * (t[i + 1] - t[i])
        #energy += energy_interval
        #energy_drain.append(power_drained(v_values,a,I,theta))
        energy_drain.append(energy_interval)
    if i==61:
        energy_interval = 0.5 * (p[i]) * (t[i])

# Initialize soc array with zeros
soc = np.zeros(len(t))

for i, j in enumerate(energy_drain):
    ratio = (j)/ 645000
    soc[i] = 100 - (ratio * 100)
    if soc[i]>100:
        soc[i]=100

print("soc:",soc,"\n")
# Print optimized x values
print("Optimized v values:", optimized_v_values,"\n")
print("energy drained:",energy_drain,"\n")
plt.plot(t,v_values, label='Initial v values')

# Plotting optimized x values
for i, v_optimized in enumerate(optimized_v_values):
    plt.plot(t, v_optimized, label=f'Optimized v values ')

plt.xlabel('time(secs)')
plt.ylabel('Value of v')
plt.title('Comparison of Initial and Optimized v values')
plt.legend()
plt.grid(True)
plt.show()

plt.plot(t, soc, label='State of Charge (SOC)')
plt.xlabel('Time')
plt.ylabel('State of Charge (%)')
plt.title('State of Charge vs Time')
plt.legend()
plt.grid(True)
plt.show()
plt.plot(t[:-2], energy_drain, label='Energy (milli J)')
plt.xlabel('Time')
plt.ylabel('State of Charge (%)')
plt.title('Energy drained/gievn from battery vs Time')
plt.legend()
plt.grid(True)
plt.show()
```

## ***OUTPUT:***



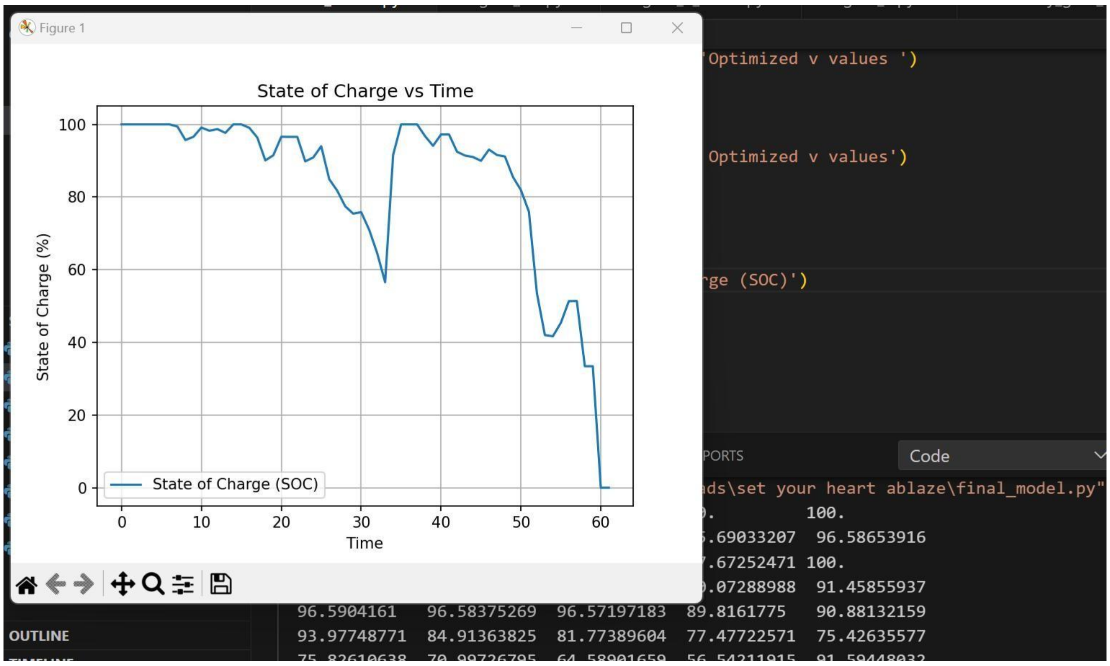
```
bel=f'Optimized v values ')
l and Optimized v values')
f Charge (SOC)')
')
ime')
```

```
ownloads\set your heart ablaze\final
             100.          100.
87   95.69033207  96.58653916
 99.13295085  98.28324816  98.72165017  97.67252471 100.
100.          99.05427753  96.37710321  90.07288988  91.45855937
 96.5904161   96.58375269  96.57197183  89.8161775   90.88132159
 93.97748771  84.91363825  81.77389604  77.47722571  75.42635577
```



```
bel='Energy (milli J)')
om battery vs Time')
```

```
                100.          100.
              95.70977901  96.59951167
              97.67504532 100.
100.          99.04714653  96.39545948  90.07588496  91.44437634
 96.59579531  96.57532735  96.56056672  89.80829436  90.8579567
```



```
'Optimized v values ')

 Optimized v values')

rge (SOC)')
```

```
ads\set your heart ablaze\final_model.py"
).          100.
5.69033207  96.58653916
7.67252471 100.
0.07288988  91.45855937
96.5904161   96.58375269  96.57197183  89.8161775   90.88132159
93.97748771  84.91363825  81.77389604  77.47722571  75.42635577
75.82610638  70.99726795  64.58901659  56.54211915  91.59448032
```