AGNIRATH

STRATEGY MODULE

RECRUITMENT’25

APPLICATION FOR THE POSITION OF CREW MEMBER

AGNIRATH

General Instructions:

- Try to answer as many questions as possible.
- You can use the help of any online resources to answer the questions.
- We have attached a few references (hyperlinks in questions) which will be helpful to solve the questions.
- Try to be creative but feasible with your ideas.
- Explain your ideas/strategy wherever possible.
- Save your answer script as a pdf file with the name <Module Acronym>_<Roll Number>_<Name>.pdf Eg: ST_AE24xxxx_Rick.pdf (ST for Strategy module)
- Google forms for application submissions will be circulated later on smail and whatsapp.
- The deadline for submission is 28th September EOD.

agnirath.iitm

Kevin - +91 63665 77373 (Strategy)

Overview

Agnirath is IITM's pioneering solar car racing team, dedicated to conquering the World Solar Challenge - a grueling 3000km endurance race across Australia that tests the limits of solar-powered engineering and strategic innovation.

As a Strategy Module Member, Our mission goes beyond building a solar car - we're solving a complex, multi-variable optimization problem. Success demands mastering the intricate balance between solar energy capture, vehicle efficiency, and strategic race management across a 5-day, 3000km endurance challenge.

Core Objectives

- Predict Race Completion Probability
- Optimize Solar Energy Utilization
- Calculate Optimal Velocity Strategies
- Minimize Energy Consumption

Critical Analysis Parameters

- Aerodynamic Coefficient Evaluation
- Energy Loss Minimization
- Climate and Solar Radiation Modeling
- Battery Performance Optimization

Evaluation Metrics

This module is designed to assess applicants' inherent aptitude, strategic thinking, hardworking nature, and ability to make logical decision with limited data.

AGNIRATH

CONTENTS

AGNIRATH

Sunlight, Speed & Strategy

Forces - The Drama Queens of Motion

When you enter the team, the first task you will undertake is modelling the car. After all, everything else you do will be based on the equations you use to model the environment your car faces.
Our car will face a variety of forces, each of which affects the car in a different way.

Question 1

Think of the various factors affecting your car- both internal and external. To answer this question, it might help to think about

- Forces at constant velocity.
- Force while accelerating.

Find the expressions for these forces, and write down newton’s laws for the car.

Battery Anxiety Simulator

(Warning: Your battery percentage drops faster than a sophomore's GPA during midterms)
Now, before your solar chariot turns into a very expensive paperweight, you'll need to speak fluent Battery. Let's break down the drama:

Question 2

1. How do you visualize the “percentage” of a battery?
2. As we use batteries more and more, the chemicals inside the battery degrade, leading to a lesser capacity. What is used to indicate the “health” of a battery? And how would you express battery capacity in terms of the aforementioned parameters?
3. How would you model the “flow of energy” from the battery?

Solar Power - The Ultimate Side Hustle

Your battery’s on life support, the motor’s begging for juice, and guess who’s here to save the day?
That giant nuclear fusion reactor in the sky (yes, the Sun – A+ for naming, scientists).

Question 3

1. What are the different kinds of solar irradiation?
2. How does the solar radiation change with:
   a. Location
   b. Time of day
   c. Days of the year
3. Arrive at a comprehensive formula for computing solar irradiance at a given place at a given time.

AGNIRATH

Giving the Reality check

You’ve got solar panels slurping photons, a battery playing power banker, and a motor screaming for watts. But here’s the plot twist: this entire system is just physics playing Tetris with energy.

Question 4

Now, the battery and the forces acting on a car lead to a system of energy consumption and generation. What are the power inputs and outputs for the car? Write down any relevant mechanical constraints, and arrive to mathematical expressions for all power inputs and outputs.

Question 5

One benefit of a power - based approach is that using some simple math, we can easily arrive to the governing equations of our system. This approach utilizes Lagrangian mechanics(recall PH1010).
Derive the lagrangian for the car, and derive the equations of motion from the lagrangian and compare them to the newtonian force equations that you have. What are the differences? How would you express constraints?

AGNIRATH

Bits&Bytes may break your bones

In the World Solar Challenge, our car must conquer 3000km of relentless sun, sand, and stochasticity.
For us the Strategists, this isn't just a race—it's a high-stakes optimization duel where every joule is a bullet in our chamber; every second is a heartbeat we can't afford to lose.

The Computational Arms Race

The Heart of the Battle: Objective Function

Every constraint optimization problem has three sacred components:

1. Objective Function f(x) → What we brutally minimize (energy consumption)
2. Decision Variables x → What we control (velocity, battery allocation)
3. Constraints $g(x) \leq 0$ → What we obey (safety, physics)

Consider a simple constrained optimization:

- 10 decision variables (e.g., velocity points)
- Gradient descent with 1,000 iterations
- Finite differences (central difference: 2 calls per variable)

Total calls = (10 params) × (2 finite diffs) × (1,000 iterations) = 20,000 objective evaluations

Now scaling to Agnirath's reality:

- 240 velocity points (every 5min for 8hr for 5 days)
- 500 iterations (nonlinear constraints)
- 5 physics models per call

Total calls = 240 × 500 × 5 = 6 × 10^5 evaluations

A 1ms slowdown in your objective function → 10 minutes of lost compute.
"In solar racing, microseconds become minutes—and minutes become INF."

Question 1

```python
def objective(x):
    return 0.5*drag*(x**3) + (sun_gain - battery_loss)
```

Why can't we just differnentiate and solve directly? Your teammate suggests precomputing sun_gain for all 86,400 timesteps. What's the hidden cost?

AGNIRATH

Optimization Survival Guide

Before charging into battle, know your weapons. Every algorithm has a time complexity - its hidden cost when scaled to war. Time complexity (Big-O notation) is how we predict how long an algorithm will take as inputs grow.

Question 2

```python
def find_optimal_angle(panels):
    # Code that checks solar panel alignments
    best_angle = 0
    max_power = -1

    for angle in range(0, 180, 5):          # Loop A (36 iterations)
        current_power = 0
        for panel in panels:                 # Loop B (n panels)
            current_power += calculate(panel, angle)

        if current_power > max_power:
            max_power = current_power
            best_angle = angle

    return best_angle
```

```python
def balance_cells(cells):
    if len(cells) <= 1:
        return cells

    pivot = cells[len(cells)//2]  # Median pivot
    left = [x for x in cells if x < pivot]
    middle = [x for x in cells if x == pivot]
    right = [x for x in cells if x > pivot]

    return balance_cells(left) + middle + balance_cells(right)
```

Profile these functions like they're draining your battery - calculate their time complexities and break down your reasoning like a race engineer under fire.

Your Unwanted Companion Snake (Python)

Yes, C runs circles around Python. We know. But when 90% of ML libraries, AI tooling, and rapid prototyping ecosystems live in Python, we're stuck with our inefficient companion snake.

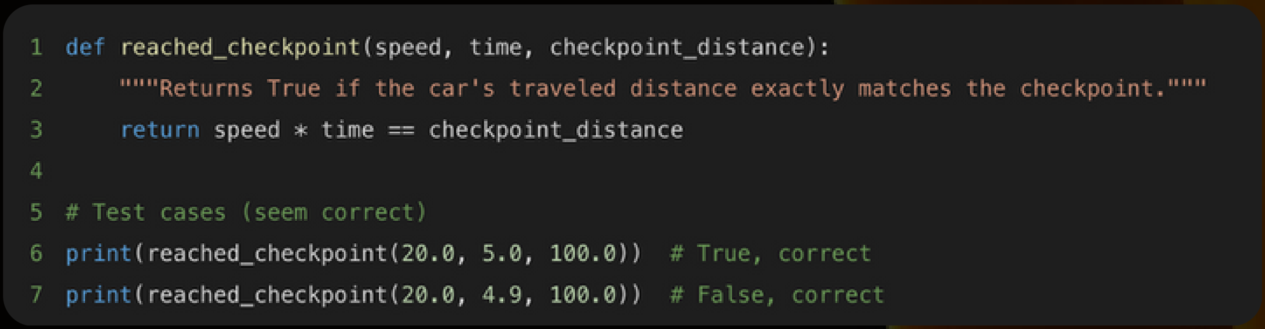
At this point you'll probably be wondering:
"Kevin, isn't it just as easy as not writing unoptimized code right?"
To which I can only reply:
"Yes, but sadly no 🥲".

Question 3

This function finds the sunniest panel segment. It’s called 100x/sec but runs slow—optimize it without changing behavior:

```python
def sunniest_segment(segments):
    max_irrad = 0
    best_segment = None
    for i in range(len(segments)):
        irrad = (segments[i].irradiance *
                 cos(radians(segments[i].angle - sun_angle)))
        if irrad > max_irrad:
            max_irrad = irrad
            best_segment = segments[i]
    return best_segment
```

Question 4



```python
def reached_checkpoint(speed, time, checkpoint_distance):
    """Returns True if the car's traveled distance exactly matches the checkpoint."""
    return speed * time == checkpoint_distance

# Test cases (seem correct)
print(reached_checkpoint(20.0, 5.0, 100.0))  # True, correct
print(reached_checkpoint(20.0, 4.9, 100.0))  # False, correct
```

Your Mission

- Find Inputs that unexpectedly return False despite correct math
- Explain why this happens in detail
- Fix the function to handle all real-world cases

To Infinity & Beyond

Question 5

Our solar car's telemetry system processes 500Hz sensor data (voltage, current, temperature) to compute real-time efficiency metrics. The current Python implementation is too slow for race conditions. Propose and compare different approaches to optimize this pipeline, considering:

- Performance improvements (orders of magnitude speedup)
- Implementation complexity (learning curve, integration effort)
- Hardware constraints (compatibility with embedded systems like Raspberry Pi)
- Maintainability (team expertise, debugging overhead)

Structure your answer by:

- Identifying optimization strategies (e.g., lower-level compilation, parallel computing, algorithmic improvements).
- Comparing tradeoffs for each method in terms of speed, development cost, and hardware requirements.
- Recommending the best approach for our solar racing application, justifying your choice based on real-world constraints.

Bound by the Sun

Now armed with physics and coding prowess, you move to build a model… but how do you build a model? How do you get to the optimal solution?

Generalized Constrained Optimisation Problem

Consider the following optimisation problem:

$$\min_{x \in R^n} f(x)$$

$$\text{s.t. } h_i(x) = 0, \quad g_j(x) \leq 0,$$

$$i \in E = \{1, \dots, m\}, \quad j \in I = \{1, \dots, p\}$$

- $f : R^n \to R$
- $h_i : R^n \to R$
- $g_i : R^n \to R$

Feasible Set: $F = \{x \in R^n | h_i(x) = 0, g_j(x) \leq 0, \forall i, j\}$

Key Challenges in Non-Convex Settings

- Constraint Qualifications: The KKT conditions require LICQ. If LICQ fails, Lagrange multipliers may not exist or be unique.
- Duality Gaps: Non-convex problems may have different primal/dual objectives at optimality.
- Algorithmic Stability: Projected gradient methods may suffer from the Maratos effect.

Question 1

Prove that if x⋆ is a local minimizer satisfying LICQ, then there exist:

$$\lambda^* \in R^m \quad \& \quad \mu^* \in R^p$$

such that:

1. **Stationary Condition:** $\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla h_i(x^*) + \sum_{j=1}^{p} \mu_j^* \nabla g_j(x^*) = 0$
2. **Complementary Slackness:** $\mu_j^* g_j(x^*) = 0 \quad \forall j$
3. Construct a counterexample where LICQ fails, making the KKT conditions insufficient

Question 2

Define the Lagrangian function (does it remind of anything wink wink):

$$L(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \mu^T g(x)$$

Show concavity of the dual function: $d(\lambda, \mu) = \inf_x L(x, \lambda, \mu)$

AGNIRATH

Question 4

Consider:

$$\min_x f(x) + \|h(x)\|^2,$$

$$\mathcal{L}_\rho = f(x) + \lambda^T h(x) + \frac{\rho}{2}\|h(x)\|^2.$$

1. Show that as $\rho \to \infty$, both enforce feasibility, but the augmented Lagrangian avoids ill-conditioning.
2. For $h(x) = x^2 - 1$, compute the Hessian condition number at x=1. Discuss the impact on Newton-type solvers.

Question 5

Argue whether interior-point methods, SQP, or augmented Lagrangians are best suited for this problem.
Further, for a greater idea about optimization, Read up on numerical integration schemes, and search based methods. Explain any one numerical integration scheme, and perform error analysis for it.

- Take the following methods of optimization
- A numerical integration scheme(solving ODEs)
- A gradient based search.
- An agentic search algorithm.

And compare their benefits and disadvantages. Focus on which situations they might be useful in.

AGNIRATH

Stayin' in Control

Think of the car in an environment as a dynamic system- it changes its state based on the environment and the control inputs you provide (if the system is controllable). You want to control the car to finish the race the fastest.
Note:

- The **"state"** of a system refers to the minimum set of variables that, when known at a particular time, describe the complete behaviour (of interest) of the system.
- **"Controls"** refer to the set of variables that as input can steer the systems behaviour.
- **"Environment"** refers to everything outside of the system that can influence the system's state and behaviour.

Our solar race car hence can be modelled as a dynamic system in the race environment.

Once you model the car as a control system, you wonder what to do next. Well, the next step is to design a controller that determines the best control inputs at each step. One approach is to frame this as a constrained optimization problem (do you remember where else you've seen this?), where the controller finds the optimal control inputs based on well-defined objectives, constraints and bounds.
Some terminology, you pickup while learning about controls-

- An objective is the desired goal the optimizer tries to achieve, typically expressed as a function to be minimized in code.
- Constraints refer to restrictions that must be satisfied while optimizing the objective function.
- Bounds refer to limits on the values of variables.

Question 1

1. How do you define the state of the solar race car? What could be the possible control inputs?
2. Also comment on how deterministic or stochastic your control execution would be and why? Does this depend on the environment?

Question 2

1. Define a clear objective function for the solar race car problem. Hint: The solar race car problem is a finite-horizon problem, that is, a terminal state exists. How do you leverage it?
2. What are the possible constraints and bounds? Constraints can either be tight/hard or flexible/soft depending on the penalty you desire to offer on its violation.
3. Can you convince me that your objective, constraints and bounds allow for a feasible and optimal solution?

At this point, you might realise that your strategy must depend on the actual state of the car. (Convince yourself why).

This means designing a closed-loop control system where the controller continuously reads the state of the car and updates control inputs based on real-time feedback. However, this introduces a challenge- online nature of control. A closed-loop controller must wait for the execution of current control, measure the new state and then optimize at each step.

This sequential process makes it slower and computationally challenging.
Think and answer about what can be done to remedy this.

AGNIRATH

Model Predictive Control - (no, it’s not ur mom)

Now that you have a solid understanding of control problem formulation, let's dive into one of the most widely used control strategies - Model Predictive Control (MPC).

MPC predicts open-loop trajectories for closed-loop control. Instead of making decisions for only the current timestep, MPC:

- Takes input a sequence of observation states (observation horizon)
- Predicts control inputs over a sequence of future states (prediction horizon)
- Executes a subset of those predicted controls (execution horizon)
- Repeats the process, continuously updating predictions based on new state measurements

Key Parameters:

- Control loop frequency = dt
- Observation horizon =T_o (How many past states the controller considers)
- Prediction horizon = $\hat{T}$ (How far into the future the controller predicts)
- Execution horizon = T_e(How many control actions are actually executed)
- State at time-step $i = x_i$
- Control variable at timestep $i = u_i$
- Current time-step = t

At each time-step t, the controller:

1. Observes past states:$\{x_i | i \in [t - T_0, t-1]\}$
2. Predicts control inputs for future states: $\{u_i | i \in [t+1, t+\hat{T}]\}$
3. Executes a subset of control actions: $\{u_i | i \in [t+1, t+\hat{T}]\}$
4. Measures the new system state x_t and repeats the process

Question 3

Model Predictive Control while highly capable (What are the pros and cons of using MPC?) requires careful handling of its various parameters. Some algorithms are heavily dependent on the parameters while others are quite robust.

Question 4

How long should dt be? If $\tau(< \tau_{max})$ be the time taken to optimize any given step, is there a relationship between the two? What decides τ ? How long should each of $T_o, T_e, \hat{T}$ be?

Question 5

Numerical methods of optimization often rely on a good initial guess to accelerate finding an extremum in the objective function.

- What’s a good initial guess for the model? Do you really need a fixed initial guess everytime the optimizer runs?
- Suppose you attempt at variable initial guesses at each cycle. What could it be then? Is the previous one still better?
- While a good initial guess makes it faster to find an extremum, it could land up in a local minimum. We desire to find the global minimum. What could be a method to enable this?

(Hint: Relate it with the Stochastic Gradient Descent method in Machine Learning.)

AGNIRATH

Hittin' the Home Run

This section is designed to assess your ability to bring ideas to life and execute them effectively within the context of the Strategy module.

Submission Requirements:

- All solutions must be submitted via a **single GitHub repository**.
- The repository should include:
  - **Well-structured** and **readable** code
  - **Clear comments and explanations** where necessary
  - An **organised folder structure** that makes navigation intuitive

Note:

- Familiarity with Git and GitHub is expected. (Don't worry, it ain't that hard and can always reach out for assitance)
- Part of the evaluation may be based on:
  - Code correctness
  - Clarity and documentation
  - Repository organisation

- GitHub Docs (official): https://docs.github.com/en/get-started
- Git Handbook (beginner-friendly): https://guides.github.com/introduction/git-handbook/
- Interactive Git Tutorial: https://learngitbranching.js.org/

Question 1

Your motor's winding temp is like that friend who takes forever to pick a Netflix show - iterative and stubborn. Write a function calculate_steady_state_temp that takes ambient temp T_a(K) and torque τ(Nm), then returns the steady-state winding temp T_w(K). Use these equations:

1. **Magnet temp**: $T_m = \frac{T_a+T_w}{2}$ *(It's the Switzerland of temperatures)*
2. **Remanence**: $B = 1.32 - 1.2 \times 10^{-3}(T_m - 293)$ *(B for "Better not melt")*
3. **Phase current**: $i = 0.561B\tau$ *(Current, not drama)*
4. **Resistance**: $R = 0.0575(1 + 0.0039(T_w - 293))$ *(Resist the urge to panic)*
5. **Copper loss**: $P_c = 3i^2R$ *(Power loss ≠ existential loss)*
6. **Eddy loss**: $P_e = \frac{9.602\times10^{-6}(B\tau)^2}{R}$ *(Eddy says "Hi!")*
7. **Update** $T_w = 0.455(P_c + P_e) + T_a$.

Iterate until T_w changes by <1 K. Start with $T_w = 323K$

Refer to Thermal modelling section in the motor's datasheet: https://drive.google.com/file/d/1P2ljpBRz3DZhVKbVxU7VVhPIzM3ZMt8v/view?usp=sharing

```python
def calculate_steady_state_temp(T_a, tau):
    T_w = 323  # Initial guess
    # Your iteration here...
    return round(T_w, 1)

```

Question 2

Now let's see if you can gather the kind of real-world data we need for high-level simulations.

Task: Retrieve geographic and environmental data for the road route from Chennai to Bangalore.

1. Identify a suitable public API (mapping, routing, or geospatial data provider).
2. Retrieve and save:
    - GPS coordinates along the route
    - Altitude (elevation) at each point
    - Direction (bearing) of each road segment
3. Choose a reasonable resolution for the road segment and justify why that resolution. Assume we'd actually be driving that route so accoridnly justify

Save your final data as a csv file

Question 3 - The FINAL Challenge

This is more of an open ended question, but also happens to the section most weightage (yes, answering this question well gives you a good shot at selection)**.**

Important:

- **Do not leave this question blank. If you're short on time or stuck, at least submit:**
    - **A design overview, OR**
    - **A high-level plan of how you would implement the model**

Let us see how you bring all the pieces together — this is where you show us your thinking, creativity, and execution.

Task:
Design a simple race strategy model that decides the optimal velocity profile over a race duration, broken into discrete time intervals.
This optimal velocity profile may be aiming for different things, choose one kinda of optimisation: (either, energy optimisation or time-minimization or distance-maximization); pick one.

You may:

- Use Python or C++ (preferred for evaluation)
- Use built-in optimizers (e.g. Adam, scikit-learn), or implement your own (e.g. gradient descent)
- Generate synthetic data as needed (Or contact me regarding data)

What to Submit:
A GitHub repo containing:

- Modular, readable, and well-commented code
- A README.md with your assumptions and model description
- Visual outputs (e.g., velocity profile, energy graphs)

I'll put out a few more things to give you a general idea;

AGNIRATH

Input:

Based on your answers to the previous questions, you should now have a clear idea of what inputs you need and what range and scale they fall in. You can either accordingly generate dataset or request/discuss with me about it.

You'd generally have solar irradiance based off the time of day, and a fixed battery capacity which discharges through the motor (as motor has a power consumption based on the velocity); think more from this

Output:

The key output I'm looking for is the **velocity profile**. Would be better, if additional graph showcasing details of this calculated optimal velocity profile add to it. I'd also appreciate insight as to the behaviour of the velocity profile or the model in general.

Additional Info:

- Make sure that the model converges/outputs in a sane amount of time. The cap is about **5-10min max for a medium sized dataset.**
- Try keeping the code modular and broken into parts, modules separate function, etc..
- For this question, it suffices to implement a predictive model; assume you don't have to worry about live data.

AGNIRATH