Optimisation de l'Allocation de Couple pour Véhicule Électrique

Résumé de l'Étude

4 janvier 2026

1 Définition du Problème

L'objectif est de déterminer la consigne de couple optimale pour chaque moteur d'un véhicule électrique à 4 roues motrices indépendantes, afin de maximiser le rendement global ($\cos\phi$), sous contraintes de dynamique longitudinale.

1.1 Variables et Données

- **Entrées :**
  - Consigne de couple global véhicule : C_{req} (Nm).
  - Vitesse du véhicule : V (rpm ou rad/s).
  - Cartographie d'efficacité (Nuage de points) : $f(C, V) \to \cos\phi$.
- **Sorties :** Couple par roue C_{av} (avant) et C_{ar} (arrière).

1.2 Hypothèses et Contraintes

1. **Mouvement rectiligne :** Les couples gauche et droit sont identiques sur un même essieu.
2. **Bilan des forces :** La somme des couples aux roues doit égaler la demande conducteur :
$$2 \cdot C_{av} + 2 \cdot C_{ar} = C_{req} \tag{1}$$
Ce qui permet de réduire le problème à une seule variable indépendante C_{av} :
$$C_{ar} = \frac{C_{req}}{2} - C_{av} \tag{2}$$
3. **Limites physiques :** $0 \leq C_{av}, C_{ar} \leq C_{max\_moteur}$.

2 Stratégie d'Optimisation

Pour respecter la consigne de **priorité à l'essieu avant** (maximiser le $\cos\phi$ avant en priorité), nous formulons une fonction objectif pondérée.

Soit $\eta(C, V)$ la fonction d'interpolation du $\cos\phi$. Nous cherchons à **maximiser** la fonction J :

$$J(C_{av}) = K_p \cdot \eta(C_{av}, V) + \eta(C_{ar}, V) \tag{3}$$

Où K_p est un facteur de priorité (ex : $K_p = 10$).

— Si $K_p > 1$, l'optimiseur favorisera un rendement élevé sur l'avant, quitte à sacrifier légèrement l'arrière.

— Cela force naturellement une distribution de type "Tout à l'avant" (ex : 19Nm/1Nm) si l'efficacité est meilleure à forte charge.

Comme les algorithmes standards minimisent, nous minimiserons $-J(C_{av})$.

3 Implémentation Python

Le code suivant utilise `scipy.interpolate.LinearNDInterpolator` pour générer la surface d'efficacité à partir des vecteurs de données brutes, et `scipy.optimize.minimize_scalar` pour trouver la répartition idéale.

```python
import numpy as np
from scipy.interpolate import LinearNDInterpolator
from scipy.optimize import minimize_scalar

class EVTorqueOptimizer:
    def __init__(self, T_vec, V_vec, Phi_vec, max_torque=120.0):
        self.max_torque = max_torque
        # Construction de l'interpolateur 2D
        points = np.column_stack((T_vec, V_vec))
        self.model = LinearNDInterpolator(points, Phi_vec, fill_value=0)

    def get_efficiency(self, torque, speed):
        if torque <= 0.1: return 0.0
        val = self.model(torque, speed)
        return float(val) if not np.isnan(val) else 0.0

    def cost_function(self, c_front, c_global, speed, priority):
        # Contrainte d'egalite : c_arriere est deduit de c_avant
        c_rear = (c_global / 2.0) - c_front

        # Penalites si hors limites
        if c_rear < 0 or c_rear > self.max_torque:
            return 1e6

        eff_front = self.get_efficiency(c_front, speed)
        eff_rear = self.get_efficiency(c_rear, speed)

        # Maximisation ponderee (on minimise l'oppose)
        return - (priority * eff_front + eff_rear)

    def compute_optimal_torques(self, global_cmd, speed, priority=10):
        # Bornes dynamiques pour c_front
        min_f = max(0, (global_cmd - 2 * self.max_torque) / 2)
        max_f = min(self.max_torque, global_cmd / 2)

        if min_f > max_f: return None # Impossible

        res = minimize_scalar(
            self.cost_function,
            bounds=(min_f, max_f),
```

```
            args=(global_cmd, speed, priority),
            method='bounded'
        )

        c_f_opt = res.x
        c_r_opt = (global_cmd / 2.0) - c_f_opt

        return {
            "C_AV_roue": round(c_f_opt, 2),
            "C_AR_roue": round(c_r_opt, 2),
            "Eff_AV": round(self.get_efficiency(c_f_opt, speed), 4)
        }
```

Listing 1 – Code d'Allocation de Couple Optimale