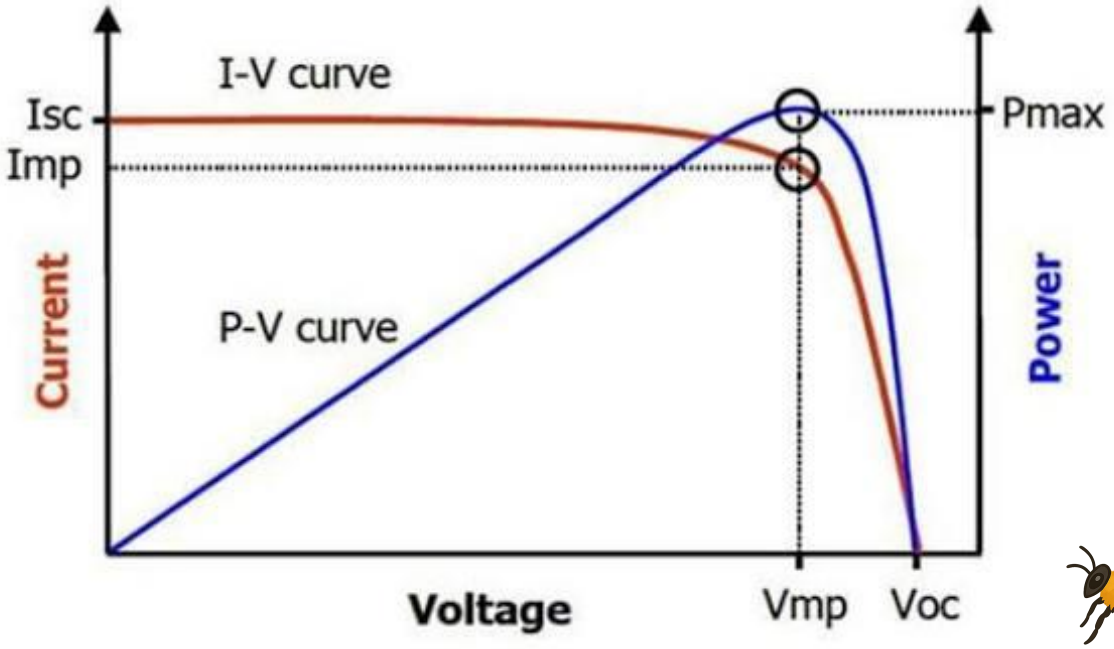
RAPPORT DE SÉANCE 8 :

Durant la **8ième séance** nous avons continué le **code Arduino**.
Dans un premier temps, nous avons **réfléchi** à la manière **d'optimiser la récupération d'énergie du panneau solaire**.
Nous nous sommes appuyés sur :

- la courbe **courant–tension (I – V) du panneau solaire**,
- la courbe **puissance–tension (P – V)**.



On sait que la puissance fournie par le panneau est donnée par : **P = V × I**
La **courbe P–V** présente un **point de puissance maximale**
(***MPP – Maximum Power Point***).
L'objectif est donc **d'adapter** la **charge vue par le panneau** afin de **fonctionner au plus près de ce point**.

Pour cela, nous agissons sur le **rapport cyclique α (*alpha*)** du **signal PWM** qui **pilote le MOSFET du système de chauffage**.

Modifier α revient à modifier l'impédance vue par le panneau :
- si **α augmente →** le **courant absorbé augmente**,
- si **α diminue →** le **courant absorbé diminue**.

Ainsi, en **ajustant progressivement α**, nous cherchons à **maximiser la puissance mesurée**.

Nous avons ajouté **l'initialisation du PWM** dans le **setup()** :

```
// PWM (LEDC)
ledcAttach(SIGNAL1, pwm_freq, pwm_res);
ledcAttach(SIGNAL2, pwm_freq, pwm_res);

ledcWrite(SIGNAL1, 0);
ledcWrite(SIGNAL2, 0);
```

Le microcontrôleur utilisé étant un ESP32, nous avons utilisé les fonctions LEDC (LED Controller) natives du microcontrôleur.

Nous avons choisi ledcWrite() plutôt que analogWrite() car :

- **analogWrite() n'est pas une vraie sortie analogique mais une PWM simplifiée.**
- **Sur ESP32, analogWrite() est moins flexible.**
- **ledcWrite() permet :**
  - **de choisir précisément la fréquence (ici 1000 Hz),**
  - **de choisir la résolution (ici 8 bits → 0 à 255),**
  - **d'utiliser plusieurs canaux indépendants.**

Cela permet un contrôle plus précis et plus adapté à une application de puissance.

Avant d'activer le **MPPT**, nous avons mis en place une **régulation simple en température** :

```
// ----- Hystérésis température -----
if (tRuche < (temperatureConsigne - bande)) {
  chauffe = true;
} else if (tRuche > (temperatureConsigne + bande)) {
  chauffe = false;
}
```

Une **bande d'hystérésis de ±1°C évite les commutations trop fréquentes** autour de la **consigne**. Le **chauffage** ne s'active que si la **température est inférieure** à la **consigne moins la bande**.

Nous avons ensuite **implémenté un algorithme** de **type Perturb & Observe (P&O)**.

À chaque boucle, **nous mesurons** : la **tension V**, la **puissance P**.

Puis nous **calculons** :
$\Delta P = P - P_{old}$
$\Delta V = V - V_{old}$

Selon le **signe de ΔP et ΔV** :

- Si la **puissance augmente** dans la **direction actuelle → on continue.**
- Si la **puissance diminue →** on **inverse la direction**.

```
if (dP > 0) {
    if (dV > 0) {
        direction = -1;
    }
    else if (dV < 0) {
        direction = 1;
    }
}
else if (dP < 0) {
    if (dV > 0) {
        direction = 1;
    }
    else if (dV < 0) {
        direction = -1;
    }
}
```

Cela permet de **se rapprocher progressivement du point de puissance maximale**.

Afin d'éviter que les **petites fluctuations de mesure** ne **perturbent l'algorithme**, nous avons introduit des **seuils** :

```
float seuil_dV = 0.01; // 10 mV
float seuil_dP = 50;   // 50 mW
```

Le **rapport cyclique** n'est modifié que si une **variation significative de la puissance** et de la **tension** est **détectée**. Cela permet d'éviter de confondre une **variation due au réglage de alpha** avec une **variation liée à un changement d'ensoleillement**.
Le paramètre **alpha** est **initialisé à 0,5** (**soit 50 %**) et ajusté par pas de **delta = 0,02**, correspondant à une **variation de ±2 % à chaque itération de l'algorithme**.
On **limite** ensuite **alpha** entre **5 %** et **95 %.**
Cela **évite** :

- une **saturation totale du MOSFET**,
- un **fonctionnement instable aux extrêmes**.

```
if (abs(dP) > seuil_dP && abs(dV) > seuil_dV) {
  // mettre à jour alpha
  alpha += direction * delta;
}
```

```
if (alpha > 0.95) alpha = 0.95;
if (alpha < 0.05) alpha = 0.05;
```

```
int duty = alpha * 255;

if (chauffe) {
  ledcWrite(SIGNAL1, duty);
  ledcWrite(SIGNAL2, duty);
}
else {
  ledcWrite(SIGNAL1, 0);
  ledcWrite(SIGNAL2, 0);
}

P_old = P;
V_old = V;
```

Enfin, si le chauffage est activé, ledcWrite() envoie le signal PWM au MOSFET, avec duty (0–255) proportionnel au rapport cyclique alpha.

Sinon, les sorties sont mises à zéro, coupant l'alimentation de la résistance et évitant tout échauffement inutile.