Laplace

LABORATOIRE PLASMA ET CONVERSION D'ÉNERGIE

Formation toolbox pour le design et contrôle des micro-reseaux

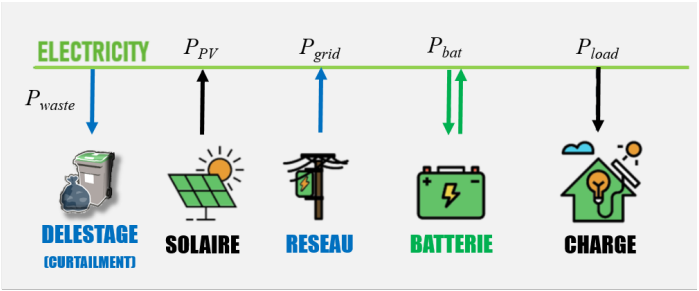
Partie 1 - Architechture, interractions, rule base et affichage

Corentin Bœnnec
Groupe Genesys, Laplace, Université de Toulouse

14 mars 2025

cnrs
TOULOUSE INP N7
UNIVERSITÉ TOULOUSE III PAUL SABATIER

Explication du problème avec un cas simple



Variables imposées (PV et Charge en flux de puissance)

Variables de décision (Flux pour le stockage batterie)

Variables de recours (Puissance réseau et déléstage)

Contrainte :

On cherche à satisfaire une charge énergétique à chaque instant.

Objectif :

Parmi les ensembles de variables de décision permettant de satisfaire la charge, trouver celui qui optimise le critère de notre choix (prix, autoconsommation...).

Équilibre des puissances

Contrainte :

$$P^{PV} + P^{bat} + P^{grid} = P^{load} + P^{waste} \quad (1)$$

Variables :

- $P^{PV} \geq 0$: Flux de puissance des panneaux solaires (unidirectionnel et positif)
- $P^{load} \geq 0$: Charge en puissance (unidirectionnel et positif)
- $P^{grid} \geq 0$: Flux de puissance du réseau (unidirectionnel et positif)
- $P^{waste} \geq 0$: Flux de puissance du délestage (unidirectionnel et positif)

P^{bat} : Flux de puissance batterie (bidirectionnel) $\begin{cases} \text{Charge,} & \text{si } P^{bat} < 0 \\ \text{Décharge,} & \text{si } P^{bat} > 0 \end{cases}$

Scénarios et Décisions

Scénarios :

Ici, P^{PV} et P^{load} sont imposés par des **scénarios**. Ceux-ci sont des ensembles de séries temporelles associant une valeur à un point dans le temps.
Les **scénarios** sont un élément essentiel permettant de définir le contexte de travail.

Décisions :

Si P^{PV} et P^{load} sont imposés et que P^{grid} et P^{waste} sont des variables de recours, alors seules les variables liées à la batterie représentent un degré de liberté.
Les décisions sont alors les flux d'entrée-sortie de la batterie pour chaque pas de temps d'opération.

Note :

Notons que le nombre de variables dépend du nombre de composants à opérer. Pour un système avec, par exemple, 2 systèmes de stockage, le nombre de variables de décision par pas de temps serait augmenté.

Décisions de dimmentionnement

Quelques précisions

Il est en réalité imprécis d'écrire que P^{PV} est imposé. En réalité, seul l'ensoleillement est imposé, P^{PV} dépend de la puissance crête du panneau solaire installé.

Le dimensionnement

En effet, un second type de décision sont les décisions de dimensionnement, on les prend à des intervalles de temps plus éloignés et elles déterminent la taille des composants du réseau.
Les décisions de contrôle/opération ont une dépendance aux décisions de dimensionnement, car ces dernières définissent un cadre.

Quelques exemples de décisions de design

- Puissance crête d'un PV.
- Capacité d'une batterie.
- Puissance souscrite sur un réseau électrique.
- Nombre et surface de cellules de pile à combustible (PàC).

Méthodes de Décision

Introduction

Pour prendre des décisions dans le cadre du dimensionnement (*sizing*) et de l'exploitation (*control*), différentes méthodes peuvent être utilisées, chacune reposant sur des hypothèses différentes.

Exploration des Options

- **Rule-Based Controller (RBC)** : Méthodes ad hoc spécifiques à une architecture de réseau donnée. Réagissent en temps réel aux entrées en utilisant un ensemble de règles pour donner des instructions de contrôle.
- **Manual Designer** : Attribution manuelle des valeurs pour le dimensionnement des éléments du système.

Remarque

Avant de plonger dans ces méthodes, nous explorerons plus en détail les architectures du package.

Le découpage principal

Les fichiers source du package sont structurés en 5 grands dossiers :

- **assets** - Contient l'ensemble des structures et fonctions relatives aux composants.
- **optimization** - Contient les fonctions relatives à la décision et l'optimisation pour le contrôle ainsi que le dimensionnement.
- **scenarios** - Contient l'ensemble des structures et fonctions relatives à la génération, la réduction et la transformation des variables d'environnement du problème.
- **simulation** - Contient les fonctions permettant la simulation du système une fois que tous les éléments variables sont décidés.
- **utils** - Contient les fonctions et structures permettant l'analyse et l'affichage des dynamiques et des métriques relatives à la simulation du système.

L'ensemble des fichiers de ces dossiers doit être exécuté pour être utilisé. Ensemble et encapsulés, ils constituent notre package.

Relations de base entre les entités du package

Déroulement d'un script

1. Les **assets** vont nous permettre de constituer un micro-réseau de la composition de notre choix.
2. D'autre part, on va créer ou importer des **scénarios** de données pour définir les variables d'environnement (la consommation, l'ensoleillement...)
3. Une fois l'architecture du réseau déclarée et les données connues, on peut faire appel conjointement à une méthode de **design** et une méthode de **contrôle** afin de prendre l'ensemble des décisions concernant le dimensionnement des éléments du réseau, mais aussi leur opération en décidant également des flux énergétiques à chaque pas de temps.
4. Grâce à ces décisions, il est alors possible d'évaluer l'architecture définie précédemment dans le contexte des **scénarios** avec pour décisions celles de nos **designer** et **controller**.
5. Enfin, nous pouvons utiliser les résultats de cette simulation pour évaluer les métriques et afficher les dynamiques du réseau au cours du temps.

Contenu de la structure micro-réseau et aperçu des sous-structures

La structure centrale : Microgrid

Cette structure est au cœur de tout le développement du package. Elle contient :

- les paramètres définissant le cadre de la simulation dans `parameters` (horizon temporel, nombre de scénarios...)
- Elle contient aussi sous forme de vecteur l'ensemble des composants du système répartis en 5 catégories.

Catégories de composants

- **Demands** - Les demandes modélisent la charge, pour chaque vecteur énergétique impliqué, que doit fournir le système à chaque pas de temps.
- **Generations** - Les générateurs modélisent la production d'énergie.
- **Storages** - Les stockers modélisent le stockage de l'énergie.
- **Converters** - Les convertisseurs modélisent la conversion entre les différents vecteurs énergétiques.
- **Grids** - Les réseaux modélisent un fournisseur externe d'énergie avec une souscription sous forme d'abonnement.

Quelques cas d'usage

Dimensionnement d'un réseau avec cahier des charges

- Rassemblement des données sous forme de variables d'environnement pour positionner notre microgrid dans un contexte concret. Cela inclut des prix spécifiques, le coût local de l'électricité, ainsi que des données de consommation et d'ensoleillement basées sur l'emplacement géographique.
- Dimensionnement optimal selon ces données et une liste de composants possibles.

Questionnement de l'émergence économique d'une technologie

Réalisation d'analyses pour évaluer le prix maximal pour l'émergence d'une technologie.

Comparaison d'architectures

Dimensionnement optimal de deux réseaux d'architectures différentes répondant à une charge commune et analyse des avantages de chacune.

Déclaration d'un Microgrid

Code associé

```julia
# Parameters of the simulation
const nh, ny, ns = 8760, 20, 1

# Initialize the microgrid
microgrid = Microgrid(parameters = GlobalParameters(nh, ny, ns, renewable_share = .5))

# Add the equipment to the microgrid
add!(microgrid, Demand(carrier = Electricity()), # Demands
                Solar(), # Generations
                Liion() # Storages
                Grid(carrier = Electricity())) # Grids
```

Les composants utilisent ici les paramètres par défaut de leur constructeur, nous explorerons plus tard comment ajuster ces paramètres et modèles pour répondre aux besoins spécifiques de notre application. Des explications détaillées seront fournies au cours 3, se concentrant principalement sur les modèles et leurs paramètres.

Déclaration des scénarios associés

Code associé

```
# Chargement de données préalablement générées
data_optim = JLD.load(joinpath("Examples", "data", "ausgrid_multi_grid_optim.jld"))

# Initialize scenarios (microgrid, data, repeated_years; seed) seed are optional
ω_d = Scenarios(microgrid, data_optim, true; seed=1:ns)
```

Les scénarios ici utilisés sont déjà générés et sont simplement chargés.

Designer et Controller (version la plus rudimentaire)

Code associé

```
# Déclaration des dictionnaires pour le dimensionnement du micro-réseau

generations = Dict("Solar" => 15.0)

storages = Dict("Liion" => 20.0)

subscribed_power = Dict("Electricity" => 10.0)

manual_designer = Manual(generations=generations, storages=storages, subscribed_power=subscribed_power)

# Affectation des valeurs choisies au designer
designer = initialize_designer!(microgrid, manual_designer, ω_d)

# Déclaration du controller et sélection de la règle d'opération numéro 2
controller = RBC(options = RBCOptions(policy_selection = 2))
```

Ici, les décisions de design sont prises manuellement (méthode : Manual) et les décisions de contrôle sont prises via un ensemble de règles prédéfinies pour cette architecture de réseau (méthode : RBC n°2). La RBC n°2 implique le calcul de la différence entre la production du PV et la charge de consommation des clients ($P^{PV} - P^{load}$), utilisant ce résultat pour piloter la charge ou décharge de la batterie.

Simulation, métriques et affichage

Simulation

```
#Simule le micro-reseau avec l'architechture, les scenarios et les décisions définies précédement
simulate!(microgrid, controller, designer, ω_d, options = Options(mode = "serial"))
```

Métriques

```
#Génère les métriques
metrics = Metrics(microgrid, designer)
```

Affichage

```
#Affiche les dynamiques du réseau selon plusieurs angles de vue
# et ce sur l'horizon temporel/les scenarios spécifiés (ici de l'année 1 à ny pour le scénario 1)
plot_operation(microgrid, y=1:ny, s=1:1)
```

Principe du contrôle à base de règle

Nous l'avons vu au début de cette présentation, un controller à base de règle (RBC) a pour mission d'affecter une valeur aux variables de décisions à chaque pas de temps.

Avantages

- Une des spécificités est que ces controller fonctionnent en temps réel et sont par conséquent facile à mettre en place dans des cas concrets ;
- Ils ont une complexité très faible.

Inconvénients

- Pour chaque architecture et pour chaque objectif poursuivi une nouvelle règle doit être défini, ce qui fait de ces méthodes ad-hoc des outils peu adaptables ;
- Pour des systèmes complexes les règles sont difficiles à écrire rendant l'opération fortement sous-optimale.

La boucle de simulation

Paramètres

On la vue à la slide 15, la fonction `simulate!` nécessite :

- Un micro-réseau ici `mg` ;
- Un controller (les décisions qu'il contient) ;
- Un designer (décision de dimensionnement) ;
- Un (ou des) scénarios ici ω_simu ;
- une option (on va ici considérer ici le mode séquentiel, les scénarios étant indépendants, on peut ici utiliser du multi-threading).

Fonctionnement (légèrement simplifié)

```
for s in 1:ns
    initialize_investments!(s, mg, designer) #initialise les modèles du réseau
    for y in 1:ny
        for h in 1:nh
            update_operation_informations!(h, y, s, mg, ω_simu) #MaJ des données du scénario pour h,y,s
            compute_operation_decisions!(h, y, s, mg, controller) #Récupère ou calcul les décisions de contrôle
            compute_operation_dynamics!(h, y, s, mg, controller) #Les modèles appliquent les décisions
            compute_power_balances!(h, y, s, mg) # calcul les variables de recours.
        end
    compute_investment_decisions!(y, s, mg, designer) # Calcul les décisions de remplacement.
    compute_investment_dynamics!(y, s, mg, designer) # Applique les décisions de remplacement.
    end
end
```

Précision sur la boucle de simulation

Les décisions de contrôle

En réalité, les méthodes de contrôle peuvent soit être anticipatives et avoir déjà pris les décisions en "connaissant" le futur. Soit, comme pour les RBC, prendre les décisions à la volée. Dans ce cas, les décisions de contrôle sont calculées durant la simulation, à l'exécution de la fonction `compute_operation_decisions!(h, y, s, mg, controller)`.

Précisions sur les flux énergétiques

Les flux du grid

Le délestage introduit par la variable P^{waste} se fait sur le grid avec un prix nul, on peut donc considérer la variable P^{grid} de la façon suivante :

$$P^{grid} \text{ : Flux de puissance du réseau (bidirectionnel)} \begin{cases} \text{Déléstage,} & \text{si } P^{grid} \leq 0 \\ \text{Achat,} & \text{si } P^{grid} > 0 \end{cases}$$

Note

De façon générale, on peut considérer les flux du point de vue de notre micro-réseau. Un flux d'énergie qui pointe vers le microgrid se caractérise par une valeur positive (P^{PV}...), un flux qui pointe hors du microgrid se caractérise par une valeur négative (charge de la batterie ...) .

TP1 - Rappel

Exécution d'un bloc de code

Il est possible d'exécuter tout un fichier Julia via la fonction `include("file.jl")`.
Il est également possible dans VScode, d'exécuter une portion de code, en sélectionnant à l'aide de la souris la portion que l'on désire exécuter et en pressant "CTRL + Entrer". Cette seconde façon va être très utile pour les TP car elle permet de faire de légère modification avant d'exécuter à nouveau un bloc de code.

Tuer/stopper la console

L'exécution dans la console peut être stoppée par la commande "CTRL + C" (en étant dans la console, sinon comme vous le savez, on "copie") et être terminée par la commande "CTRL + D"

TP1 - Les décisions 1

Objectifs

Définir une méthode de contrôle basée sur des règles (RBC) pour chacun des objectifs suivants :

- Maximiser l'autonomie du système.
- Minimiser le vieillissement de la batterie.
- Minimiser le coût opérationnel du système. (Pour différents modèles tarifaires)

Instructions

- Élaborez un ensemble de règles sur papier pour chaque objectif.
- Implémentez les méthodes basées sur des règles dans l'espace dédié de votre code.
- Utilisez les métriques et les graphiques du script TP1 pour observer les effets de vos méthodes.

Remarques

Les décisions pour la batterie sont stockées dans `controller.decisions.storages[1][h, y, s]`. Les variables environnementales sont disponibles à chaque instant sous la forme `mg.demands[1].carrier.power[h, y, s]` pour la charge et `mg.generations[1].carrier.power[h, y, s]` pour la puissance PV.

TP1 - Les décisions 2

Objectif

Utiliser et s'approprier le **designer** *Manual*.

Instructions

- Exécutez la simulation avec différentes valeurs pour le dimensionnement des équipements.
- Observez les variations sur le coût et l'auto-suffisance du système.

TP1 - Les analyses 1

Objectif

- Explorer et prendre en main la structure *microgrid*.
- Apprendre à mettre en place des métriques.
- Apprendre à mettre en place des affichages.

Instructions

En utilisant les informations de la diapo suivante, coder soit même 3 fonctions calculant chacune une métrique :

- D'auto-suffisance
- De vieillissement de la batterie
- De coût.

Puis réaliser une fonction qui retourne ces 3 métriques.

TP1 - Astuces de développement

La fonction sum

Il est également possible d'utiliser l'argument *dims* pour appliquer la somme sur la ou les dimensions désirées.

```
A = [[1,2,3] [2,3,4]] # 3x2 Matrix{Int64}
sum(A) # 15
sum(A, dims=1) #  1x2 Matrix{Int64}: 6 9
sum(A, dims=2) #  3x1 Matrix{Int64}: 3, 5, 7
```

La fonction max

La fonction max permet de comparer des valeurs et d'en retourner la plus grande. Une version dites *broadcast* peut permettre d'isoler les valeurs positives d'un vecteur.

```
max(1, 3) # 3
B = [[-1,2,-3] [-2,3,4]]
max.(0,B) # 3x2 Matrix{Int64}: [[0,2,0]  [0, 3, 4]]
```

Note

Penser à observer le contenu des structure de donnée que vous manipulez. `mg.grids[1].carrier.power[:, :, :]`
Contient par exemple des valeurs positives et négatives.

TP1 - Les analyses 2

Auto-suffisance

L'auto-suffisance du système s'exprime comme le complément de la part assurée par le réseau.

$$RES = 1 - \frac{P^{grid}}{P^{demand}} \tag{2}$$

vous trouverez les informations concernant P^{grid} et P^{demand} dans les structure *microgrid*.
`mg.grids[1].carrier.power[1:nh, 1:ny, 1:ns]` et `mg.demands[1].carrier.power[1:nh, 1:ny, 1:ns]`

Vieillissement

Le vieillissement d'un composant se mesure via son SoH. Vous trouverez son évolution dans la structure : `mg.storages[1].soh`

Coût

Le coût du système peut être abordé de plusieurs façon plus ou moins complexe.
Vous pouvez créer la métrique de votre choix, sachez que :

- Chaque grid possède un champ `.carrier.power[1:nh, 1:ny, 1:ns]` et un champ `.cost_in[1:nh, 1:ny, 1:ns]` ;
- Et chaque élément architectural du réseau possède un champ cost[ny, ns] et le designer peut vous renseigner sur les investissements et remplacements.

TP1 - Les analyses 3

Instructions

Réaliser des fonctions d'affichage d'abord séparé puis dans une fenêtre commune pour **la production du PV** et **la consommation Pload**.
Puis pour **le SoC de la batterie** et **le SoH de la batterie**. (⚠ les tableaux de SoC et de SoH sont définis sur `1:(nh+1), 1:(ny+1), 1:ns`).

Aide PyPlot (recommendé)

Une page avec des exemples utilisant la librairie *PyPlot.jl* qui découle de *matplotlib*. Cette approche semble moins attractive, mais les librairies offrent plus de possibilités.

Aide Plots.jl

Une page remplie d'exemples utilisant la librairie *Plots.jl*.