Laplace
LABORATOIRE PLASMA ET CONVERSION D'ÉNERGIE

Formation toolbox pour le design et controle des micro-reseaux

Gestion des modèles

Corentin Bœnnec
Groupe Genesys, Laplace, Université de Toulouse

17 mars 2025

cnrs TOULOUSE INP N7 UNIVERSITÉ TOULOUSE III PAUL SABATIER

Revue des modèles disponibles

Liste des composants disponibles

Un ensemble d'éléments est actuellement implémenté dans le package :

- Storages : **Liion**, H2 Tank, Thermal Storage
- Generation : Solar Panel
- Converter : Heater, **Electrolyzer**, **Fuel Cell**

Les éléments en gras sont déclinés en plusieurs niveaux de modélisation que l'on va détailler.

Les modèles de batterie - Rendement

Rendement fixe

```
FixedLiionEfficiency() #default η = 0.98
FixedLiionEfficiency(;η_ch = 0.9, η_dch = 0.92)
```

Ce premier modèle implémente simplement un rendement fixe. La valeur par défaut est 0.98 et peut être modifiée et différenciée entre charge et décharge.

Rendement Polynomial

```
PolynomialLiionEfficiency()
```

Ce second modèle nous vient de [1], le rendement est calculé via une équation du second degré fonction du C_{rate}. Les coefficients de l'équation sont différents en charge ou en décharge.

$$\eta^{ch,b}(C_{rate}) = 0.0033 \cdot {C_{rate}}^2 - 0.0297 \cdot C_{rate} + 0.99814 \quad (1)$$

$$\eta^{dch,b}(C_{rate}) = 0.002232 \cdot {C_{rate}}^2 - 0.0246 \cdot C_{rate} + 1 \quad (2)$$

Les modèles de batterie - Vieillissement

Durée de vie fixe

```
FixedLifetimeLiion()  #Default 12
FixedLifetimeLiion(;lifetime = 15)
```

Ce modèle considère que la batterie sera hors d'état après X années.

Énergie Échangée

```
EnergyThroughputLiion() #ncycle_ini default = 2500
# Utilise des données constructeur pour obtenir le nombre de cycles d'une profondeur de 60% réalisables
ncycle = Int(floor(fatigue_data.cycle[findfirst(fatigue_data.DoD .> (0.6))]))
EnergyThroughputLiion(;nCycle_ini = ncycle)
```

Ce modèle [2] considère que la batterie possède une quantité d'énergie maximale qu'elle peut échanger avant d'être hors d'état. Ensuite son état de santé (SoH) est actualisé en fonction de la fraction de l'énergie totale échangeable déjà échangée. A cela vient s'ajouter un vieillissement calendaire.

$$SoH^b_{h+1} = SoH^b_h - \Delta_{cyc}(h) - \Delta_{cal}(\Delta_h) \tag{3}$$

Les modèles de batterie - Vieillissement

Paramétrage cyclage Énergie Échangée

$$E_{tot}^{ex} = 2 \cdot N_{tot}^{cycle}(DoD^{param}) \cdot DoD^{param} \cdot E^{nom} \tag{4}$$

Avec DoD^{param} la profondeur supposée $\in [0, 1]$ des demi-cycles (charge ou décharge)
La dégradation due au cyclage sur une heure est alors obtenue via :

$$\Delta_{cyc}(h) = \frac{(|P_{ch,h}^{bat}| + P_{dch,h}^{bat}) \cdot \Delta_h}{E_{tot}^{ex}} \tag{5}$$

Paramétrage calendaire Énergie Échangée

On a extrait la fonction de dégradation due au temps de notre modèle le plus complexe [3] (dans les conditions de référence)

$$\Delta_{cal}(t) = 1 - e^{t \cdot c_{cal}} \tag{6}$$

avec $c_{cal} = 1.49 \cdot 10^{-6}$.

Les modèles de batterie - Vieillissement

Modèle Rainflow

```
RainflowLiion(fatigue_data = fatigue_data) #On utilise les même données que pour le modèle précédent
```

Ce modèle inspiré de [4] extrait d'un profil de charge une séquence de cycle/sous-cycle et en déduit une dégradation.

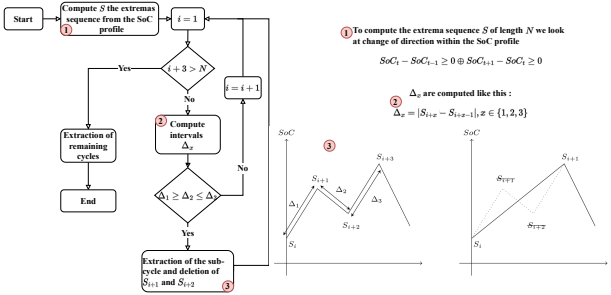


Figure – Diagramme de flux de l'extraction de cycle

Calcule de la dégradation

$$\Delta_{cyc}(m) = \sum_{i=1}^{|\overrightarrow{DoD}|} \frac{1}{2 \cdot NCF(\overrightarrow{DoD}_i)} \quad (7)$$

Mise à jour du SoH

$$SoH^{bat}_{m+1,1} = SoH^{bat}_{m,H} - \Delta_{cyc}(m) - \Delta_{cal}(\Delta_m) \quad (8)$$

Les modèles de batterie - Vieillissement

Modèle Semi-Empirique

```
SemiEmpiricalLiion()
```

Ce modèle [3] incorpore les dégradations précédemment décrites et en incorpore d'autres. Les cycles sont également extraits du profil de charge, mais sont ici traités selon leurs DoD, SoC moyen, durée et température. Ces données s'agglomèrent en un facteur d'usure noté fd

Calcul du facteur d'usure

L'usure unitaire d'un cycle est comptabilisé grâce à l'équation suivante.

$$fd^{unit} = [\frac{1}{2} S_{DoD}(DoD) + S_t(t)] S_{SoC}(\overline{SoC}) S_T(T) \quad (9)$$

Le facteur d'usure global est alors mis à jour pour un profil de charge via :

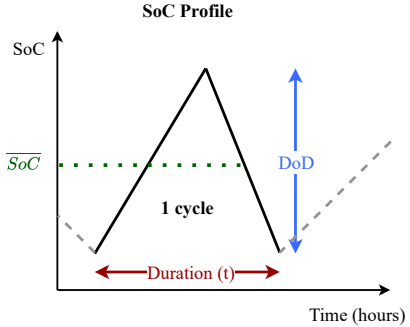
$$fd_{t+\Delta_m} = fd_t + \sum_{i=1}^{N_{cycle}(m)} fd_i^{unit} \quad (10)$$

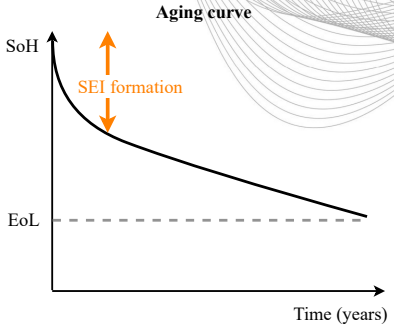
Actualisation du SoH

Enfin le SoH est mis à jour via l'équation suivante :

$$SoH_t^b = \alpha_{sei} \cdot e^{-\beta_{sei} \cdot fd_t} + (1 - \alpha_{sei}) e^{-fd_t} \quad (11)$$

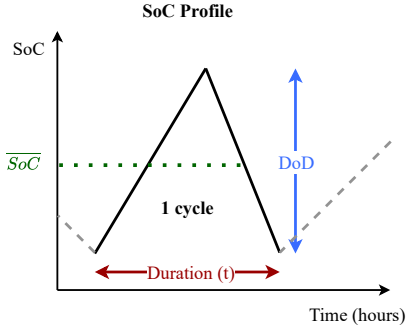
Semi-Empirical modèle : Principes

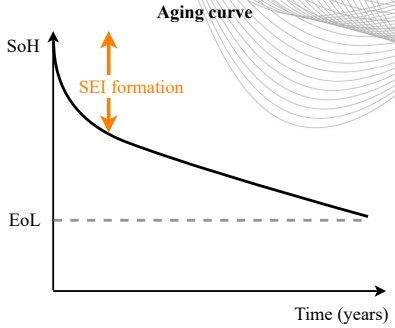




$$fd^{unit} = [\frac{1}{2}\mathbf{S_{DoD}(DoD)} + \mathbf{S_t(t)}]\mathbf{S_{SoC}}(\overline{\mathbf{SoC}})S_T(T)$$

Semi-Empirical modèle : Principes

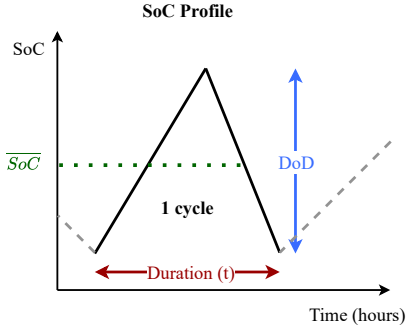


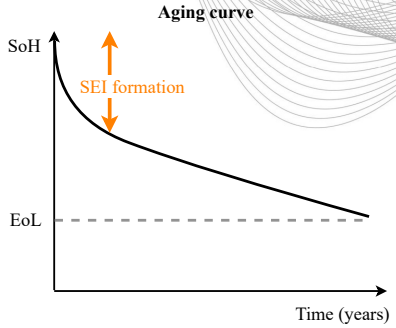


$$fd^{unit} = [\frac{1}{2}\mathbf{S_{DoD}(DoD)} + \mathbf{S_t(t)}]\mathbf{S_{SoC}}(\overline{\mathbf{SoC}})S_T(T)$$

$$fd_{t+\Delta_m} = fd_t + \sum_{i=1}^{N_{cycle}(t,\Delta_m)} fd_i^{unit}$$

Semi-Empirical modèle : Principes





$$fd^{unit} = [\frac{1}{2}\mathbf{S_{DoD}(DoD)} + \mathbf{S_t(t)}]\mathbf{S_{SoC}}(\overline{\mathbf{SoC}})S_T(T)$$

$$fd_{t+\Delta_m} = fd_t + \sum_{i=1}^{N_{cycle}(t,\Delta_m)} fd_i^{unit}$$

$$SoH_t^b = \alpha_{\mathbf{sei}} \cdot e^{-\beta_{\mathbf{sei}} \cdot fd_t} + (1 - \alpha_{\mathbf{sei}})e^{-fd_t}$$

Les modèles de batterie - Couplage

Définition

On nomme ici couplage, le fait que le vieillissement de la batterie agisse sur ses caractéristiques.

Les couplages implémentés

On propose 2 types d'effets activables indépendamment l'un de l'autre :

- Un couplage sur la capacité des batteries noté (E)
- Un couplage sur le rendement noté (R)

```
#On active le couplage (E) mais pas le (R)
SoC_m = PolynomialLiionEfficiency()
SoH_m = SemiEmpiricalLiion()
Liion(SoC_model = SoC_m, SoH_model = SoH_m, couplage = (E=true, R=false))
```

Calcul

- Mise à jour de la capacité : $E_t = E^{nom} \cdot SoH_t$.
- Mise à jour du rendement [5] : $\eta(SoH_t) = \eta^{ini} - (0.2303 \cdot (1 - SoH_t))$

Les modèles de Pile à Combustible - Rendement

Calcul de P(I)

$$P(J \times S^{FC}) = V(J \cdot S^{FC}) \cdot J \cdot S^{FC} \cdot Ncell^{FC} \quad \forall J \qquad (12)$$

Prise en compte des auxiliaires

$$P_{brut} = \frac{P_{net}}{1 - k_{aux}} \qquad (13)$$

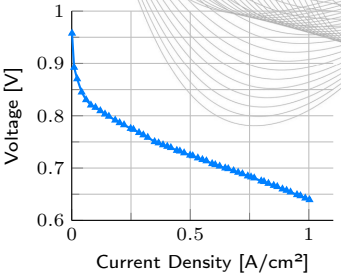
On considère $k_{aux} = 10\%$ de P_{brut}.

Rendement

$$\eta(I(P), P_{net}) = \frac{P_{net} \cdot 2F}{I(P_{brut}) \cdot Ncell^{FC} \cdot LHV \cdot M_{H_2} \cdot \lambda} \qquad (14)$$

Puissance minimum

Une puissance P_{min} est définie par $V(P_{min}) = 0.8V$.



- $I(P)$ la réciproque de P(I)
- P_{net} la puissance demandée
- $F = 26.80$ la constante de faraday en $Ah.mol^{-1}$
- $LHV = 33.33$ Le potentiel calorifique inférieur en $kWh.kg^{-1}$
- $M_{H_2} = 2.016$ la masse molaire du H_2
- $\lambda = 1.2$ la proportion stœchiométrique.

Les modèles de Pile à Combustible - Rendement

Modèle de rendement basé sur la courbe de polarisation

On le voit ici, la courbe de polarisation V(J) est un paramètre de la fuel cell plutôt que de son modèle de rendement, car la courbe de polarisation sert aussi au modèle de vieillissement.

```
eff_m = PolarizationFuelCellEfficiency()
age_m = PowerAgingFuelCell(;deg_params=deg)
FuelCell(;V_J_ini = V_J_FC, EffModel = eff_m, SoH_model = age_m)
```

Modèle de rendement linéarisé

On veut utiliser un rendement qui soit une fonction linéaire de la puissance. Pour cela on détermine P_{min} et P_{max}. Puis on calcule avec le modèle précédent $\eta_{max} = \eta(I(P), P_{min})$ et $\eta_{min} = \eta(I(P), P_{max})$.
Enfin on obtient la pente et l'ordonnée à l'origine de la façon suivante :

$$a_\eta = \frac{\eta(I(P), P_{max}) - \eta(I(P), P_{min})}{P_{max} - P_{min}} = \frac{\eta_{min} - \eta_{max}}{P_{max} - P_{min}} \tag{15}$$

$$b_\eta = \eta(I(P), P_{min}) - P_{min} \cdot a_\eta = \eta_{max} - P_{min} \cdot a_\eta \tag{16}$$

Pour finir on calcule le rendement comme :

$$\eta(P) = a_\eta \cdot P + b_\eta \tag{17}$$

Les modèles de Pile à Combustible - Rendement

Modèle de rendement linéarisé

```
        LinearFuelCellEfficiency()
```

Modèle de rendement fixe

Ici on peut remarquer que malgré un modèle de rendement qui n'utilise pas la courbe de polarisation celle ci est quand même déclaré comme paramètre. D'une part parce que celle ci peut être en même temps utilisée pour le modèle de vieillissement. D'autre part parce que la structure se doit d'être compatible avec tous les modèles.

```
    eff_m = FixedFuelCellEfficiency()
    age_m = PowerAgingFuelCell(;deg_params=deg)

    FuelCell(;V_J_ini = V_J_FC, EffModel = eff_m, SoH_model = age_m)
```

Les modèles de Pile à Combustible - Vieillissement

Principe général

Le vieillissement est décrit par l'évolution de la courbe de polarisation (baisse de tension). Parmi les mécanismes pouvant avoir un impact sur cette dernière, on considère ici la puissance d'utilisation et les marches/arrêts.
On néglige les variation de puissance $\frac{dP}{dt}$ sous l'hypothèse que ceux-ci sont "aplatis" et rendus négligeable grâce à l'hybridation avec la batterie

Fin de vie (EoL)

On décrète la fin de vie lorsque la tension pour une densité de courant donnée à diminué de 10%.

Dégradations dues aux Marches/Arrêts

La dégradation en tension due à un marche/arrêt est issue de [6] où chaque marche/arrêt retire $\lambda_{on/off} = 0.00196\%$ de la tension de la pile (pris au courant de référence).

Les modèles de Pile à Combustible - Vieillissement

La dégradation en fonction de la puissance

On utilise ici le travail de [7], la courbe de polarisation utilisée pour nos modèles, visible en slide 10, est extraite de ces travaux.
L'idée est de déterminer la perte en tension ΔV sur l'ensemble de la courbe de polarisation V(J) selon la puissance (et donc la densité de courant J_h) utilisée.

$$\Delta V_{J_h}(J) = a_{deg}(J_h) \cdot J + b_{deg} \qquad (18)$$

Le données expérimentales desquelles on propose d'extraire une loi via les coefficients $a_{deg}(J)$ et b_{deg} sont présentée ci-contre

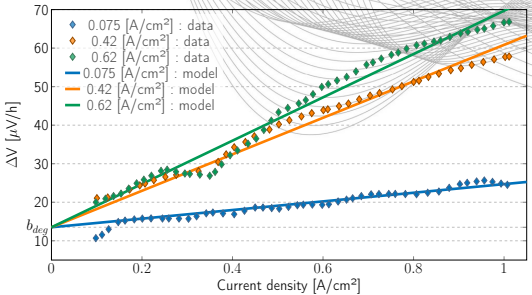
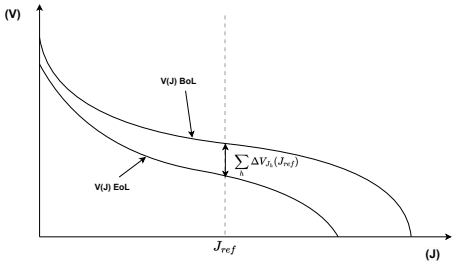


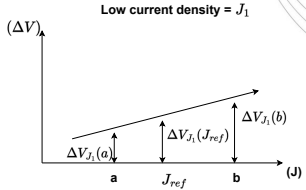
Figure – Fit des données de dégradations expérimentales de [7]. Les données initiales sont issues d'une campagne de vieillissement conduites à des niveaux de courant fixes. Ici nous cherchons à extraire les pentes ($a_{deg}(J)$) et l'ordonnée à l'origine (b_{deg}).

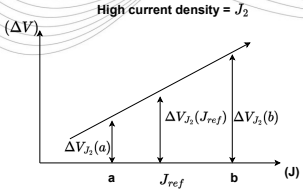
Illustration de la degradation en puissance

Évolution de V(J)

L'évolution de la courbe de polarisation est obtenue en soustrayant pas à pas des dégradations individuelles sur l'ensemble de la courbe. Ainsi la fin de vie est déclarée quand
$V_h(J_{ref}) \leq 0.9 \cdot V_0(J_{ref})$







Dégradation en fonction de la puissance

Ces dégradations sont fonction de la puissance et donc de la densité de courant à l'instant h, J_h. Plus la puissance est élevée plus la pente est raide.

Les modèles de Pile à Combustible - Couplage

Avec Couplage

L'évolution du vieillissement est caractérisé par la dégradation en tension de la courbe de polarisation. Pour tenir compte du couplage il suffit d'utiliser $V_h(J)$, la version actualisée de la courbe de polarisation, dans les modèles d'efficacité énergétique de la PaC.

Sans couplage

Au contraire, si l'on souhaite ne pas utiliser le couplage il faut alors utiliser $V_0(J)$, la courbe de polarisation initiale avant vieillissement.

Déclaration du Couplage

Par défaut le couplage est activé mais il peut être désactivé de la façon suivante :

```
FuelCell(;couplage = false, V_J_ini = V_J_FC, ...)
```

Les modèles d'Electrolyzer - Rendement

Note :

On s'intéresse ici à un électrolyzer de type PEM (Proton Exchange Membrane). Pour le modéliser, les concepts utilisés dans les modèles de piles à combustible vont être réemployés.

Prise en compte des auxiliaires

$$P_{net} = \frac{P_{brut}}{1 + k_{aux}} \tag{19}$$

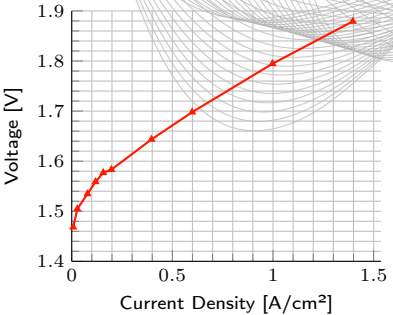
On considère $k_{aux} = 10\%$ de P_{net} [8].

Rendement

$$\eta(I(P), P_{brut}) = \frac{I(P_{net}) \cdot Ncell^{elyz} \cdot LHV \cdot M_{H_2}}{P_{brut} \cdot 2F} \tag{20}$$

Puissance minimum

La puissance minimale est généralement comprise entre 0 et 10% de la puissance nominale [9, 10]. Ici nous utilisons $P_{min} = 5\%$.



- $I(P)$ la réciproque de P(I)
- P_{brut} la puissance brute fournie
- $F = 26.80$ la constante de faraday en $Ah.mol^{-1}$
- $LHV = 33.33$ Le Potentiel Calorifique Inférieur (PCI) en $kWh.kg^{-1}$
- $M_{H_2} = 2.016$ la masse molaire du H_2

Les modèles d'Electrolyzer - Rendement

Rendement basé sur la courbe de polarisation

```
eff_m = PolarizationElectrolyzerEfficiency()
```

Ce premier modèle est l'application directe de ce qui a été montré à la slide précédente.

Linéarisation de l'efficacité énergétique

```
eff_m = LinearElectrolyzerEfficiency()
```

On veut utiliser un rendement qui soit une fonction linéaire de la puissance. Pour cela on détermine P_{min} et P_{max}. Puis, on calcule avec le modèle précédent $\eta_{max} = \eta(I(P), P(min))$ et $\eta_{min} = \eta(I(P), P(max))$. Enfin on obtient la pente et l'ordonnée à l'origine de la façon suivante :

$$a_\eta = \frac{\eta(I(P), P_{max}) - \eta(I(P), P_{min})}{P_{max} - P_{min}} = \frac{\eta_{min} - \eta_{max}}{P_{max} - P_{min}} \tag{21}$$

$$b_\eta = \eta(I(P), P_{min}) - P_{min} \cdot a_\eta = \eta_{max} - P_{min} \cdot a_\eta \tag{22}$$

Enfin le rendement est calculé comme suit :

$$\eta(P) = a_\eta \cdot P + b_\eta \tag{23}$$

Les modèles d'Electrolyzer - Rendement

Rendement fixe

```
eff_m = FixedElectrolyzerEfficiency()
```

Pour dernier modèle, on utilise un rendement fixe dont la valeur par défaut est la moyenne entre η_{min} et η_{max}

$$\eta_{fix} = a_\eta \cdot \frac{P_{max} + P_{min}}{2} + b_\eta \tag{24}$$

Les modèles d'Electrolyzer - Vieillissement

Note :

D'autres types de dégradation peuvent être considérés, c'est faute de données et de connaissances que nous proposons les 2 modèles suivants.

Basé sur les heures d'utilisation

```
eff_m = FunctHoursAgingElectrolyzer()
```

Un premier modèle qui considère que l'utilisation de l'électrolyzer induit une augmentation de $10\mu V/h$ de sa courbe de polarisation.
La fin de vie sera déclarée lorsque la tension à la densité de courant de référence J_{ref} aura cru de 20%.

Durée de vie fixe

```
eff_m = FixedLifetimeElectrolyzer()
```

Le modèle à durée de vie fixe considère simplement que l'electrolyser doit être remplacé après une durée prédéfinie.

Les modèles d'Electrolyzer - Couplage

Avec Couplage

L'évolution du vieillissement est caractérisé par la augmentation en tension de la courbe de polarisation. Pour tenir compte du couplage il suffit d'utiliser $V_h(J)$, la version actualisée de la courbe de polarisation, dans les modèles d'efficacité énergétique de l'électrolyser.

Sans couplage

Au contraire, si l'on souhaite ne pas utiliser le couplage il faut alors utiliser $V_0(J)$, la courbe de polarisation initiale avant vieillissement.

Déclaration du Couplage

Par défaut le couplage est activé mais il peut être désactivé de la façon suivante :

```
Electrolyzer(;couplage = false, V_J_ini = V_J_Elyz, ...)
```

Déterminer la compatibilité des méthodes de décision

Définition formelle

Si on défini C l'ensemble des composant du réseau et M l'ensemble des modèles décrivant des composants. On défini également l'opérateur $m(C)$ qui donne l'ensemble des modèles de M utilisé pour décrire C.
Enfin pour un modèle $M_x \in M$, $\alpha(M_x)$ renvoi l'ensemble des méthodes compatibles avec ce modèle. On peut alors écrire que la compatibilité (des méthodes de décision pour le dimentionnement et l'opération) d'un ensemble de n modèles décrivant un réseau est égale à l'intersection des compatibilité de chaque modèle évaluée individuellement.

$$\alpha(m(C)) = \bigcap_{i=1}^{n} \alpha(m(C_i)) \tag{25}$$

Définition informelle

En d'autres termes, c'est le modèle le plus restrictif qui détermine les méthodes utilisables pour le contrôle du système.

Compatibilité modèles / méthode de contrôle

Modèle	RB	Prog. mathématique (MI)NLP	(MI)QP	(MI)LP
FixedLiionEfficiency	✓	✓	✓	✓
PolynomialLiionEfficiency	✓	✓	X	X
FixedLifetimeLiion	✓	✓	✓	✓
EnergyThroughputLiion	✓	✓	✓	✓
RainflowLiion	✓	X	X	X
SemiEmpiricalLiion	✓	X	X	X
Couplage (E)	✓	✓	X	X
Couplage (R)	✓	✓	X	X

Figure – Compatibilité des modèles de composant batterie avec les méthodes de contrôle. **Cela ne tient pas compte des approximations et reformulation qui pourrait être faites**.

Compatibilité modèles / méthode de contrôle

		Prog. mathématique		
Modèle	RB	(MI)NLP	(MI)QP	(MI)LP
Rendement fixe	√	√	√	√
Rendement linéaire	√	√	X	X
Courbe de Polarisation	√	X | √	X	X
Durée de vie fixe	√	√	√	√
Heures de fonctionnement	√	√	√	√
Fonction de la puissance	√	√	X	X
Puissance et Marches/Arrêts	√	√	X	X
Couplage	√	√	X	X

Figure – Compatibilité des modèles de composant Pile à Combustible/électrolyser avec les méthodes de contrôle.

Compatibilité modèles / méthodes de design

Programmation Mathématique

Pour pouvoir utiliser la programmation mathématique pour le dimensionnement, il faut que l'ensemble des modèles de composants le permette. Faute de simplifications, le problème résultant (si de forme fermée) sera généralement de catégorie MINLP et de grande taille, ce qui implique un temps de résolution déraisonnable.

Méhaheuristiques

Les métaheuristiques font appel à n'importe quelle méthode de contrôle et s'en servent comme fonction d'évaluation, elles sont donc compatibles avec toutes les méthodes de contrôle et donc tous les modèles de composants.

Manual

Bien entendu, le designer manuel est compatible avec l'ensemble des modèles de contrôle car il en est totalement indépendant.

Focus sur l'implémentation des batteries

Rôle :

Une batterie Li-ion est avant tout ici considérée comme un système de stockage d'énergie électrique. Son rôle est donc de traiter les flux de puissance demandés ou reçus tout en maintenant à jour son état de charge et de santé.

- Le `SoC_model` permet de calculer la variation d'état de charge en fonction de la décision de puissance en entrée ou sortie.
- Le `SoH_model` permet de mettre à jour l'état de santé de la batterie (souvent en fonction de la façon dont elle est opérée).

Structure : (la liste des champs est non exhaustive)

```
mutable struct Liion <: AbstractLiion  #<: AbstractStorage
    SoC_model::AbstractLiionEffModel #Efficiency model
    SoH_model::AbstractLiionAgingModel # Aging model

    soc::AbstractArray{Float64,3} #3 dim matrix (h,y,s) containing the state of charge [0-1]
    soh::AbstractArray{Float64,3} #3 dim matrix (h,y,s) containing the state of health [0-1]

    carrier::Electricity #Type of energy
end
```

Focus sur l'implémentation des batteries - SoH model

Modèle à énergie échangée

```
mutable struct EnergyThroughputLiion <: AbstractLiionAgingModel

    calendar::Bool # Activation du vieillissement calendaire
    nCycle::Int64 # Nombre de cycle réalisable ajusté
    nCycle_ini::Int64 #Nombre de cycle réalisable
    Δcal::Float64 #dégradation calendaire par pas horaire
        #Constructeur par défaut
    EnergyThroughputLiion(;calendar = true,
        nCycle = 2500.,
        nCycle_ini = 2500.,
        Δcal = (1 - exp(- 4.14e-10 * 3600))
        ) = new(calendar, nCycle, nCycle_ini, Δcal)
end
```

Focus sur l'implémentation des batteries - SoC model

Modèle à rendement fixe

```
mutable struct FixedLiionEfficiency <: AbstractLiionEffModel

    η_ch::Float64 #Charging efficiency
    η_dch::Float64 #Discharging efficiency
    η_deg_coef::Float64 #The efficiency degradation coefficient
    couplage::NamedTuple{(:E, :R), Tuple{Bool, Bool}}  #a boolean tuple to tell wether or not the soh
    #should influence the other parameters.
    α_p_ch::Float64 #C_rate max
    α_p_dch::Float64 #C_rate max
    η_self::Float64 #Auto discarge factor

    FixedLiionEfficiency(;η_ch = 0.98,
        η_dch = 0.98,
        η_deg_coef = 0.2303, # ref : Redondo Iglesias - Efficiency Degradation Model of Lithium-Ion
    #Batteries for Electric Vehicles
        couplage = (E = true, R = true),
        α_p_ch = 1.5,
        α_p_dch = 1.5,
        η_self = 0.0005,
        ) = new(η_ch, η_dch, η_deg_coef, couplage, α_p_ch, α_p_dch ,η_self)

end
```

Focus sur l'implémentation des batteries

Gestion des décisions

Le traitement des décisions passe par la fonction `compute_operation_dynamics!` (on notera le "!" qui, dans la convention de nommage, signifie que les variables d'entrées, ici la structure `liion` seront modifiées).

Implémentation standard

```
function compute_operation_dynamics!(h::Int64, y::Int64, s::Int64, liion::Liion, decision::Float64, Δh::Int64)
    #Ici c'est liion.SoC_model qui va indiquer l'implémentation de compute_operation_soh à utiliser
    liion.soc[h+1,y,s], p_ch, p_dch = compute_operation_soc(liion, liion.SoC_model, h ,y ,s, decision, Δh)

    liion.carrier.power[h,y,s] = p_ch + p_dch
    #Ici c'est liion.SoH_model qui va indiquer l'implémentation de compute_operation_soh à utiliser
    liion.soh[h+1,y,s] = compute_operation_soh(liion,  liion.SoH_model, h ,y ,s, Δh)
end
```

Flexibilité via le multiple dispatch

Les fonctions `compute_operation_soh` et `compute_operation_soc` possèdent plusieurs implémentations, qui ont des retours similaires mais des entrées de types différents. En effet, le multiple dispatch (une forme de polymorphisme) permet plusieurs implémentations d'une même fonction, la fonction exécutée sera sélectionnée via les types de sa liste de paramètres.

Focus sur l'implémentation des batteries

Modèle Rainflow

```
RainflowLiion(fatigue_data = fatigue_data) #On utilise les même données que pour le modèle précédent
```

Ce modèle inspiré de [4] extrait d'un profile de charge une séquence de cycle/sous-cycle et en déduit une dégradation.

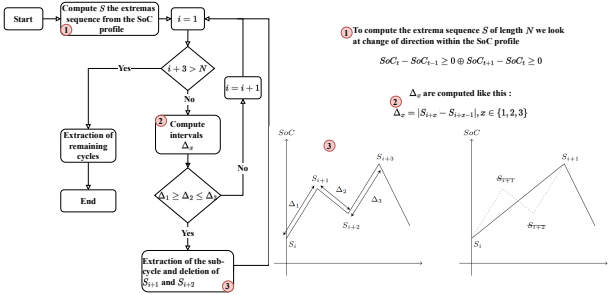


Figure – Diagramme de flux de l'extraction de cycle

Calcule de la dégradation

$$\Delta_{cyc}(m) = \sum_{i=1}^{|\overrightarrow{DoD}|} \frac{1}{2 \cdot NCF(\overrightarrow{DoD}_i)} \tag{26}$$

Mise à jour du SoH

$$SoH^{bat}_{m+1,1} = SoH^{bat}_{m,H} - \Delta_{cyc}(m) - \Delta_{cal}(\Delta_m) \tag{27}$$

Focus sur l'implémentation des batteries

Implémentation du modèle Rainflow

```
function compute_operation_soh(liion::Liion, model::RainflowLiion, h::Int64, y::Int64, s::Int64, Δh::Int64)

    h_between_update = convert(Int64,floor(8760/model.update_by_year))
    #test if its time to update
    if (h%h_between_update) != 0
        next_soh = liion.soh[h,y,s] #No changes
    else #rainflow computaion
        interval = (h-h_between_update+1):h

        ΔSoH = compute_operation_soh_rainflow(liion, model, Δh, liion.soc[interval,y,s])

        #Calendar part
        if model.calendar == true
            ΔSoH += h_between_update * (1 - exp(- 4.14e-10 * 3600 ))
        end

        next_soh = liion.soh[h,y,s] - ΔSoH
    end

    return next_soh
end
```

Focus sur l'implémentation des batteries

Implémentation du modèle Rainflow

```
function compute_operation_soh_rainflow(liion::Liion, model::RainflowLiion, Δh::Int64, soc::Vector{Float64})
    soc_peak, _ = get_soc_peaks(soc) #Gather peaks from the soc profil
    #Then compute the DoD sequence by extracting the subcycles DoD
    DoD_seq, i = Float64[], 1 #Sequence of all the charging and decharging half cycles DoDs
    while i+3 <= length(soc_peak)
        #Define your 3 deltas with 4 consecutives points
        δ1,δ2,δ3 = abs(soc_peak[i+1]-soc_peak[i]), abs(soc_peak[i+2]-soc_peak[i+1]), abs(soc_peak[i+3]-soc_peak[i+2])
        if δ2 <= δ1 && δ2 <= δ3 #rainflow sub-cycle criterion
                push!(DoD_seq, δ2) #1 half cycle of DoD δ2
                push!(DoD_seq, δ2) #1 half cycle
                deleteat!(soc_peak, i+2) #start with the second or you will delete i+1 and i+3
                deleteat!(soc_peak, i+1)
        else #else use the following point insequence
            i = i+1
        end
    end
    for i in 1:(length(soc_peak)-1)#Then add the englobing cycles to the DoD sequence
        push!(DoD_seq, abs(soc_peak[i+1]-soc_peak[i]))
    end
    fatigue = sum(1/(2*ϕ(DoD_seq[i], model.fatigue_data)) for i in 1:length(DoD_seq))
    #Compute fatigue with ϕ function applied to all the half cycles DoD factor 2 refer to half cycles
    return fatigue
end
```

Focus sur l'implémentation des batteries

Implémentation du modèle à rendements fixes

```
function compute_operation_soc(liion::Liion, model::FixedLiionEfficiency, h::Int64,  y::Int64,
s::Int64, decision::Float64, Δh::Int64)
    #Charge or discharge
    decision >= 0 ? η_ini = model.η_dch : η_ini = model.η_ch
    #Coupling on capacity
    model.couplage.E ? Erated = liion.Erated[y,s] * liion.soh[h,y,s] : Erated = liion.Erated[y,s]
    #Coupling on efficiency
    model.couplage.R ? η = η_ini - (model.η_deg_coef * (1-liion.soh[h,y,s])) : η = η_ini

    p_dch = max(0., #result have to be null or positiv
    min(decision, #The decision is bounded by
    model.α_p_dch * Erated, #C-rate limit
    liion.soh[h,y,s] * liion.Erated[y,s] / Δh, #capacity limit
    η * (liion.soc[h,y,s] - liion.α_soc_min) * Erated / Δh)) #minimum SoC limit

    p_ch = min(0., #result have to be null or negativ
    max(decision, #The decision is bounded by
    -model.α_p_ch * Erated, #C-rate limit
    -liion.soh[h,y,s] * liion.Erated[y,s] / Δh, #Capacity limit
    (liion.soc[h,y,s] - liion.α_soc_max) * Erated / Δh / η)) #Max SoC limit

    return (1-model.η_self) * liion.soc[h,y,s] - (p_ch * η + p_dch / η) * Δh / Erated, p_ch, p_dch
end
```

Focus sur l'implémentation des batteries

L'initialisation et la réinitialisation

Dans le code suivant, les décisions sont des décisions de design. Sous la forme d'une matrice ($ny \times ns$) elles contiennent une instruction sur la capacité à installer pour le scénario et l'année correspondante. 0 Signifiant pas de remplacement.

```
function initialize_investments!(s::Int64, liion::Liion, decision::Union{Float64, Int64})
   liion.Erated[1,s] = decision
   liion.soc[1,1,s] = liion.soc_ini
   liion.soh[1,1,s] = liion.soh_ini
end
function compute_investment_dynamics(liion::Liion, state::NamedTuple{(:Erated, :soc, :soh), Tuple{Float64, Float64, Flo
    if decision > 1e-2
        Erated_next = decision
        soc_next = liion.soc_ini
        soh_next =  1.
        if liion.SoH_model isa SemiEmpiricalLiion
            liion.SoH_model.Sum_fd[s] = 0.
        end
    else
        Erated_next = state.Erated
        soc_next = state.soc
        soh_next = state.soh
    end
    return Erated_next, soc_next, soh_next
end
```

Éléments à implémenter / modifier pour ajouter un modèle

Ajout d'un nouveau modèle de SoC

La structure batterie ayant un champ `SoC_model::AbstractLiionEffModel`, il suffit de créer une structure héritant de `AbstractLiionEffModel` et d'implémenter une fonction

`compute_operation_soc(liion::Liion, model::Model, h::Int64, y::Int64, s::Int64, decision::Float64, Δh::Int64)`

dont le retour sera un triplé : SoC, P_{ch}, P_{dch}.

Ajout d'un nouveau modèle de SoH

La structure batterie ayant un champ `SoH_model::AbstractLiionAgingModel`, il suffit de créer une structure héritant de `AbstractLiionAgingModel` et d'implémenter une fonction

`compute_operation_soh(liion::Liion, model::Model, h::Int64,y::Int64 ,s::Int64 , Δh::Int64)` dont le retour sera la nouvelle valeur de SoH

Éléments à implémenter / modifier pour ajouter un composant

Création des fichiers

1. Il faut commencer par créer un fichier *mon_composant.jl* dans lequel on va mettre la structure *Mon_composant* qui représente le composant.
   - Cette structure doit hériter de la grande catégorie à laquelle elle appartient.
   - Un convertisseur énergétique transformant énergie électrique en énergie chimique sera donc un héritier de la catégorie **AbstractConverter** et déclaré comme suit :

     ```
     mutable struct Electrolyzer <: AbstractConverter
     ```

2. Puis on va ajouter ce fichier au fichier principal qui importe l'ensemble du code du projet. (Genesys2.jl)

```
include(joinpath("assets","mon_composant","mon_composant.jl"))
```

Éléments à implémenter / modifier pour ajouter un composant

Interactions du composant

Le composant doit implémenter au minimum 4 fonctions fondamentales.

1. `preallocate!` pour allouer la taille adéquate aux structures de données selon les données du problème (exemple le bon nombre d'année ou de scénarios).
2. `initialize_investments!` pour mettre le composant dans son état initial, lorsqu'il est neuf. (par exemple le SoC initial à 50%, le SoH à 100%...)
3. `compute_operation_dynamics!` pour appliquer les décisions d'opération pour une heure, année, scénario. Une version `compute_operation_dynamics` non modificatrice de la structure destinée l'usage des RB sera généralement implémentée. Elle aura pour valeur de retour les puissances respectives associées à chaque vecteur d'énergie. **Cette fonction est capitale, dans l'exemple d'une batterie, elle traite la décision en puissance, actualise le SoC et le SoH de la batterie**.
4. `compute_investment_dynamics!` doit être implémenté pour appliquer les décisions de design (remplacement).

Éléments à implémenter / modifier pour ajouter un composant

Contrôle

Il est difficile de décrire dans le cas général les ajouts à faire vis-à-vis de l'opération, cependant :

- il est claires que des équations relatives au fonctionnement du composant devront être intégrées aux modèles d'optimisation de Math. Prog. pour pouvoir faire du contrôle optimale.
- Pour ce qui est des RB, il faudra prévoir d'inclure la place du composant dans la stratégie de RB correspondante.

Scénarios

Ajouter le coût du composant (par unité, exemple : €/kwh) à la création de scénario mais surtout dans les données des scénarios.

TP3 - Objectifs d'apprentissage

Compétences acquises

On va ici chercher à acquérir des compétences pour pouvoir développer de nouvelles briques d'architecture du réseau. Ces compétences sont :

- La capacité à ajouter et modifier des modèles pour des composants existant. Exemple : un nouveau modèle de rendement pour la batterie.
- La capacité à ajouter de nouveau composants au package. Exemple : un nouveau système de stockage.

Listes des tâches

Afin de pratiquer ces nouvelles compétences on va implémenter :

- un nouveau comportement pour la batterie à travers deux modèles
  - Un pour le rendement
  - Un pour le vieillissement
- Un nouveau système de stockage.

Note :

Ces modèles ne sont absolument pas tirés d'une quelconque littérature et servent uniquement comme exemple pour pratiquer l'implémentation de nouveaux composants et leur modèles.

TP3.1 - Création d'un modèle simple de batterie

Le rendement

Faites un rendement par palier qui se comporte différemment en fonction du C-rate. Le C-rate représente une vitesse de charge relative à la taille de la batterie et s'exprime en multiple de la capacité. On peut l'exprimer comme l'inverse du nombre d'heures nécessaires pour vider/remplir la batterie.

- Vous avec le choix dans le nombre de palier et les équations associées ($\eta = a$ étant une forme d'équation acceptée).
- Le rendement doit respecter les bornes suivantes : $\eta \in [0 - 1]$
- La structure `Liion` possède deux paramètres définissant les limite du SoC : `α_soc_max = 0.8` et `α_soc_min = 0.2`. Si ces limites sont conservés pour le test des modèles, avec un pas de temps horaire une conséquence implicite sera : C-rate $\in [0 - 0.6]$

TP3.1 - Création d'un modèle simple de batterie

Le vieillissement

Créez un modèle dont le vieillissement accélère pour chaque pas de temps passé en régime de charge ou de décharge.

- Vous devez utiliser la fonction `get_nb_consecutif(soc, h, y, s)` pour récupérer depuis le profil de soc le nombre d'heures de charge/décharge consécutif.
- Vous devez également créer une fonction qui traitera ce nombre.
- Enfin vous devrez appliquer cette dégradation au SoH.

Conseils

Prenez bien sûr exemple sur les slides concernées (35), mais aussi sur les structures existantes et fonctionnelles que vous trouverez dans le code. En partant des plus simples.

TP3.1 - Création d'un modèle simple de batterie

Instructions :

- Implémenter les structures décrites dans les deux slides précédente.
- Implémenter les fonctions interagissant avec ces structures.
- Créer un microréseaux avec ces nouveaux modèles et un avec des modèles préexistant. Puis valider le comportement de la batterie en étudiant l'évolution de son SoC et de son SoH à travers les fonctions d'affichages.

Note :

Il est recommandé pour observer l'évolution du SoH de la batterie d'utiliser un cas sur plusieurs années $ny > 1$

TP3.2 Ajout d'un nouveau composant : Retenue d'eau

Note :

La représentation de la retenue d'eau qui suit n'est issue d'aucun travail de recherche et n'a aucune autre prétention que celle de servir d'exemple pédagogique à ce cours.

Description de l'élément :

La retenue d'eau est considérée comme un élément de stockage dans la mesure où il est possible :

- De stocker de l'énergie potentielle en pompant de l'eau en contrebas vers un bassin dont l'altitude est supérieure.
- Réciproquement il est possible de récupérer de l'énergie électrique par turbinage en déversant une partie du bassin supérieur vers le cours d'eau en contrebas.

TP3.2 Ajout d'un nouveau composant : Retenue d'eau

Les caractéristiques à modéliser

Elle est caractérisée par :

- Le volume de son bassin supérieur (initial et en 2D [y,s])
- Le remplissage de ce dernier (qui sera représenté par une variable `soc` $\in [0-1]$ en 3D [nh,ny,ns]. ainsi qu'une variable d'initialisation `soc_ini`.
- Une durée de vie (qui sera représenté par une variable `lifetime::Int64`, obligatoire si pas de variable soh)
- Un facteur de conversion $m^3 \rightarrow Wh$ et un facteur réciproque $Wh \rightarrow m^3$
- Une limite au débit de pompage et de turbinage.
- Un `carrier` pour stocker les flux électriques
- Une variable `cost` en 2D [ny,ns] pour stocker le prix en €$/m^3$
- Une variable pour stocker les données d'ensoleillement en 3D [nh,ny,ns]

TP3.2 Ajout d'un nouveau composant : Retenue d'eau

Consignes 1 :

Implémenter les structures et fonctions suivantes :

```
mutable struct Barrage <: AbstractStorage
preallocate!(barrage::Barrage, nh::Int64, ny::Int64, ns::Int64)
initialize_investments!(s::Int64, barrage::Barrage, decision::Union{Float64, Int64})
compute_operation_dynamics!(h::Int64, y::Int64, s::Int64, barrage::Barrage, decision::Float64, Δh::Int64)
compute_operation_dynamics(h::Int64, y::Int64, s::Int64, barrage::Barrage, decision::Float64, Δh::Int64)
compute_investment_dynamics!(y::Int64, s::Int64, barrage::Barrage, decision::Union{Float64, Int64})
compute_investment_dynamics(barrage::Barrage, state::NamedTuple{(:volume, :soc), Tuple{Float64, Float64}}, decision
compute_operation_decisions!(h::Int64, y::Int64, s::Int64, mg::Microgrid, controller::RBC)
RB_barrage(h::Int64, y::Int64, s::Int64, mg::Microgrid, controller::RBC)
```

Prenez exemple sur les fonctions implémentées pour les autres composants de stockage afin de construire les vôtres.
Les données relatives au coût du barrage ont été intégrées via une redéfinition de la fonction de scénario.

Consignes 2 :

- Créez un micro réseaux avec pour système de stockage le Barrage pour remplacer la batterie.
- Faites varier les dimensions pour vérifier la cohérence et la robustesse du nouveau modèle. Identifier les éventuels éléments disfonctionnels.

Consignes Bonus et facultatives - vers le TP4 :

Utilisation des scénarios

La retenue d'eau peut être affectée par la météo, le TP 4 portera sur les scénarios et la gestion des variables définissant l'environnement.

L'évaporation

- Ajoutez une notion d'évaporation en fonction du soleil.
- Utilisez la fonction `get_evaporation` pour réduire le soc

La pluie

- Ajoutez une notion de remplissage qui viendrait de la pluie.
- Utilisez la fonction `get_pluie` pour réduire augmenter le soc lorsqu'il pleut.

References I

[1] Kaiyuan Li et King Jet Tseng. "Energy efficiency of lithium-ion battery used as energy storage devices in micro-grid". In : *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*. 2015, p. 005235-005240. doi : 10.1109/IECON.2015.7392923.

[2] H Bindner, T Cronin et P Lundsager. *Lifetime modelling of lead acid batteries*. Avr. 2005.

[3] Bolun Xu et al. "Modeling of Lithium-Ion Battery Degradation for Cell Life Assessment". en. In : *IEEE Transactions on Smart Grid* 9.2 (mars 2018), p. 1131-1140. issn : 1949-3053, 1949-3061. doi : 10.1109/TSG.2016.2578950.

[4] Yuanyuan Shi et al. "Optimal Battery Control Under Cycle Aging Mechanisms in Pay for Performance Settings". In : *IEEE Transactions on Automatic Control* 64.6 (2019), p. 2324-2339. doi : 10.1109/TAC.2018.2867507.

References II

[5] Wiljan Vermeer, Gautham Ram Chandra Mouli et Pavol Bauer. "Optimal Sizing and Control of a PV-EV-BES Charging System Including Primary Frequency Control and Component Degradation". In : *IEEE Open Journal of the Industrial Electronics Society* 3 (2022), p. 236-251. doi : 10.1109/OJIES.2022.3161091.

[6] P Pei, Q Chang et T Tang. "A quick evaluating method for automotive fuel cell lifetime". en. In : *International Journal of Hydrogen Energy* 33.14 (juill. 2008), p. 3829-3836. issn : 03603199. doi : 10.1016/j.ijhydene.2008.04.048. url : https://linkinghub.elsevier.com/retrieve/pii/S036031990800476X (visité le 19/10/2023).

[7] PESSOT Alexandra. "Modélisation des performances et du vieillissement des piles à combustible PEM basses températures en vue d'applications". fr. In : (). url : https://theses.hal.science/tel-04163683.

References III

[8] Ragnhild Hancke, Thomas Holm et Øystein Ulleberg. "The case for high-pressure PEM water electrolysis". en. In : *Energy Conversion and Management* 261 (juin 2022), p. 115642. issn : 01968904. doi : 10.1016/j.enconman.2022.115642. url : https://linkinghub.elsevier.com/retrieve/pii/S0196890422004381 (visité le 29/11/2023).

[9] Luca Bertuccioli et al. "Development of Water Electrolysis in the European Union". en. In : *Journal of Power and Energy Engineering* (). url : https://www.h2knowledgecentre.com/content/researchpaper1120.

[10] T. Nguyen et al. "Grid-connected hydrogen production via large-scale water electrolysis". en. In : *Energy Conversion and Management* 200 (nov. 2019), p. 112108. issn : 01968904. doi : 10.1016/j.enconman.2019.112108. url : https://linkinghub.elsevier.com/retrieve/pii/S0196890419311148 (visité le 30/11/2023).