# DATTES : Logiciel libre et ouvert d'analyse de données expérimentales sur les batteries

Eduardo Redondo-Iglesias[1,3], Marwan Hassini[1,2,3] Pascal Venet [2,3], Serge Pelissier[1,3]

[1]*LICIT-ECO7 Lab, Univ Eiffel, ENTPE, Univ Lyon, 69500 Bron, France*

[2]*Univ Lyon, Université Claude Bernard Lyon 1, INSA Lyon, Ecole Centrale de Lyon, CNRS, Ampère, UMR5005, 69622 Villeurbanne, France*

[3]*ERC GEST (Licit-Eco7/Ampère Équipe de Recherche Commune Gestion de l'Énergie et du Stockage pour les Transports), 69500 Bron, France*

**RÉSUMÉ - Les expériences sont essentielles pour comprendre le comportement et les performances des systèmes de stockage d'énergie. Dans ce domaine, une quantité considérable de données expérimentales est générée et le traitement des données est une tâche fastidieuse. À ce jour, les équipes de recherche travaillant dans le domaine du stockage de l'énergie ont tendance à privilégier le développement de leurs propres outils d'analyse plutôt que les logiciels ouverts existants. Cette stratégie peut nuire à la qualité et à la reproductibilité de la recherche. Cet article présente DATTES, un logiciel libre et ouvert d'analyse de données expérimentales sur les batteries. Le logiciel fournit une boite à outils complète et personnalisable pour l'extraction, l'analyse et la visualisation des données expérimentales. Il crée également des passerelles avec d'autres logiciels et outils ouverts. DATTES permet ainsi aux utilisateurs de tirer le meilleur parti de leurs données expérimentales et de s'engager dans une démarche de science ouverte et reproductible.**

***Mots-clés - Stockage de l'énergie, Batterie, Expériences, MATLAB, GNU Octave, Analyse des données, FAIR, Science Ouverte***

## 1. INTRODUCTION

Le traitement des données joue un rôle de plus en plus important dans les études scientifiques. Les logiciels tels que MATLAB, Python ou R sont au cœur de cette activité et selon une étude de Hettrick et al. 70% des scientifiques reconnaissent que leurs recherches ne seraient pas réalisables sans eux [1]. Cette activité est également chronophage pour les scientifiques puisqu'ils y consacrent généralement 30% ou plus de leur temps de recherche [2].

Dans le domaine du stockage de l'énergie, une grande diversité d'outils et de méthodologies sont utilisés pour produire les données expérimentales [3]. Ainsi les données expérimentales produites sont difficilement comparables entre elles. De plus, la plupart des équipes de recherche ont développé leurs propres outils de traitement des données adaptés à leurs besoins et jeux de données spécifiques. Si cette organisation s'explique par le besoin des chercheurs de traiter rapidement les données de leurs projets, une telle stratégie de développement limite la reproductibilité des travaux et tend à augmenter de manière significative le temps consacré par les chercheurs à la programmation. Par ailleurs, étant donné que la grande majorité des chercheurs sont des développeurs autodidactes, ils peuvent manquer d'expérience en matière de bonnes pratiques de développement de logiciels. Ainsi, la qualité des résultats de recherche peut se retrouver dégradée. La comparaison entre les études de deux laboratoires différents est également entravée puisque chaque équipe est susceptible d'utiliser sa propre méthodologie pour la production et le traitement des données. Cette absence d'outils communs pour le traitement des données constitue donc un obstacle majeur au partage et la comparaison des résultats expérimentaux.

Dans le domaine du stockage de l'énergie, les logiciels libres suscitent depuis peu un intérêt croissant, car ils permettent de mutualiser les efforts de programmation, les meilleures pratiques et les méthodologies d'analyse. Dans le paysage actuel des logiciels libres pour les systèmes de stockage de l'énergie, le langage Python prédomine. BEEP, cellpy, impedance.py ou Pybamm sont quelques exemples de logiciels populaires dans le domaine des batteries [4, 5, 6]. Ils sont tous développés en Python. Ce monopole peut s'expliquer par les qualités de ce langage de programmation polyvalent. Python est gratuit et open source, il met l'accent sur la lisibilité du code et est proposé avec une bibliothèque très complète d'outils pour l'analyse des données, tel que Scipy, Numpy ou Matplotlib. Néanmoins, bien que ce langage gagne peu à peu en intérêt dans la communauté du stockage de l'énergie, MATLAB reste de loin le langage le plus populaire. À titre d'exemple, Google Scholar a indexé plus de 200 000 publications faisant mention des mots-clés "batterie" et "MATLAB". C'est trois fois plus que celles en lien avec "Python". La popularité de MATLAB s'explique par la diversité d'applications que le logiciel propose. Ainsi que par son utilisation de matrices et de tableaux qui facilite son emploi dans la résolution de problèmes scientifiques. Enfin, MATLAB est également à la base de Simulink, un environnement de diagramme de blocs pour la modélisation, la simulation et l'analyse des systèmes dynamiques multi-physiques.

Malgré la popularité de MATLAB, à ce jour aucun logiciel ouvert de traitement des données expérimentales n'avait été proposé dans ce langage. Cet article présente DATTES (de l'anglais *Data Analysis Tools for Tests on Energy Storage*) qui est le premier logiciel ouvert et écrit sur MATLAB pour le traitement des données expérimentales de batteries.

## 2. PRÉSENTATION DE DATTES

DATTES est un logiciel open source écrit en code MATLAB et compatible avec GNU Octave qui vise à faciliter l'analyse de données pour les systèmes de stockage d'énergie [7]. Comme ces langages de programmation sont très populaires dans le domaine, le logiciel peut permettre à une grande partie de la communauté du stockage de l'énergie d'utiliser un outil ouvert de traitement des données.

Le logiciel a été conçu selon les principes du logiciel libre, à code ouvert (*FOSS*, *Free and Open Source Software*) et du *FAIR* (*Findable Accessible, Interoperable, Reusable*), et d'importants efforts ont été réalisés pour rédiger une documentation claire et complète. Les développeurs espèrent que la communauté tirera profit du logiciel. De plus, les nouveaux utilisateurs sont considérés comme des évaluateurs et contributeurs potentiels qui seront susceptibles de détecter les bugs plus tôt, de demander ou développer de nouvelles fonctionnalités, de documenter le code et d'accélérer la mise en place de standards de traitement des données.

### 2.1. *Processus d'analyse avec DATTES*

Avec DATTES, L'analyse de données expérimentales se fait en quatre étapes :

1. *dattes_import*
2. *dattes_structure*
3. *dattes_configure*
4. *dattes_analyse*

La première étape du processus consiste à standardiser le format des données expérimentales. La fonction *dattes_import* permet de convertir les données brutes issues d'un cycleur de batteries dans un format standard, le format XML, selon la structuration de données de la bibliothèque VEHLIB [8].

La deuxième étape consiste à enrichir les données expérimentales avec des métadonnées et à structurer l'information. Les métadonnées permettent l'enrichissement des bases de données, par exemple lors de campagnes de vieillissement avec des données concernant chaque test (expérimentateur, équipement utilisé, etc.) ou concernant la batterie (caractéristiques nominales, identifiant de cellule, etc.). Au cours de cette étape, la fonction *dattes_structure* permet de découper le test en phases caractéristiques selon les différents modes de sollicitation de la batterie (repos, courant constant, tension constante, etc). Cette fonction permet également de déterminer à chaque instant du test l'état de charge de la batterie.

Au cours de la troisième étape du processus de traitement des données, l'utilisateur peut adapter la méthodologie d'analyse à ses besoins. La fonction *dattes_configure* permet de modifier la configuration d'analyse définie par défaut dans DATTES et qui consiste à identifier et déterminer les grandeurs caractéristiques de la batteries (capacité, impédance, comportement à faible courant) sur l'ensemble des séquences où cela est possible. La modification de la méthode d'analyse de l'impédance peut par exemple permettre à l'utilisateur de choisir de travailler uniquement avec les pulses de courant d'une durée de son choix ou encore de sélectionner le type de modèle en circuit électrique équivalent qu'il souhaite identifier.

Finalement, la quatrième étape est l'étape d'analyse des données. Grâce à la fonction *dattes_analyse*, l'utilisateur peut analyser l'ensemble des grandeurs caractéristiques d'une batterie. La capacité peut être déterminée en charge ou en décharge sur une phase à courant constant associée ou non à une phase à tension constante. La résistance de la batterie peut être calculée à différents instants ($R_{2sec}$, $R_{10sec}$, etc.) et sur le front montant ou descendant du pulse de courant. L'impédance peut être déterminée grâce à un pulse de courant ou à l'aide de la méthode fréquentielle de la spectrométrie d'impédance (en anglais, *Electrochemical Impedance Spectroscopy* ou *EIS*). Le logiciel peut également servir à identifier les paramètres d'un modèle de la batterie en circuit électrique équivalent. De plus, DATTES permet d'analyser le comportement de la batterie à faible courant d'une part en identifiant la tension en circuit ouvert (en anglais, *Open Circuit Voltage* ou *OCV*), mais aussi grâce à ses fonctions permettant l'analyse incrémentale de la capacité (en anglais, *Incremental Capacity Analysis* ou *ICA*) et l'analyse différentielle de la tension (en anglais, *Differential Voltage Analysis* ou *DVA*).

### 2.2. *Outils de visualisation*

La fonction *dattes_plot* permet de visualiser les résultats des différentes étapes du processus d'analyse. La figure 1 présente par exemple les profils de tension et courant en fonction du temps. Les différentes phases d'usages (courant constant, tension constante, repos, etc.) sont également mises en valeur sur ce graphique. DATTES permet également à l'utilisateur de visualiser l'évolution du courant ou de la température. L'état de charge et la profondeur de décharge au cours du temps peuvent également être tracés.

### 2.3. *Atouts de DATTES*

Si DATTES se distingue comme étant le premier logiciel de traitement des données expérimentales de batteries sur MATLAB, certaines de ses fonctionnalités sont également uniques [3].

Le nombre et la diversité de formats de données d'entrée que DATTES permet de manipuler est remarquable. Les données produites par les cycleurs Arbin, Bitrode, Biologic, Digatron et Neware sont quelques exemples de formats déjà acceptés par DATTES.

DATTES propose aussi une conversion des données expérimentales brutes dans un format standard, le XML. Les fichiers convertis sont produits avec une structuration de données compatible avec la bibliothèque VEHLIB. DATTES est ainsi nativement compatible avec le logiciel de gestion de l'énergie VEHLIB [8].

À ce jour, DATTES est le logiciel ouvert de traitement des données expérimentales qui propose le plus grand nombre d'outils d'analyse. Cette faculté à traiter les différents types de test s'explique notamment par la méthode unique de segmentation des tests en fonction des phases d'usage de la batterie. Ce découpage facilite l'analyse de sous-motifs remarquables tels que les portions de test à faible courant qui peuvent faire l'objet d'une analyse incrémentale de la capacité.

La documentation des données produites par l'expérience et par DATTES a également fait l'objet d'une attention particulière. Les résultats d'analyses des données expérimentales sont complétés par des métadonnées qui permettent à l'utilisateur de documenter l'origine et les méthodes de production des données (date et heure du test, moyen expérimentaux, etc.). L'ensemble des données produites par DATTES sont également structurées et documentées [7].

Enfin, DATTES facilite la réutilisation des données d'analyse produites. La fonction *dattes_export* permet de convertir les résultats de l'analyse dans différents formats tels que **.csv*, **.json* ou **.m*.

## 3. EXEMPLE D'UTILISATION

Cette section illustre le processus d'analyse avec DATTES en présentant l'analyse d'un jeu de données expérimentales. Les lignes de code utilisées sont partagées dans l'annexe B.1. Les données traitées sont issues d'un test de caractérisation décrit dans de précédents travaux [9, 10, 11].

Les données partagées sont issues du cycleur Biologic. La fonction *dattes_import* permet de convertir ces données brutes dans un format standard, le format XML.

Ensuite, la fonction *dattes_structure* permet d'extraire les grandeurs caractéristiques de la batterie et de structurer le test en fonction des phases d'usage. La figure 1 présente l'évolution de la tension et du courant lors du test de caractérisation. La figure 2 montre que chaque changement de phase d'usage (repos, courant constant, tension constante, etc.) est également identifié par un numéro et une couleur. La définition de ces sous-parties permet à l'utilisateur d'adapter la méthode d'analyse aux différentes parties du test.

Pour ce test de caractérisation cinq parties ont été définies par l'utilisateur. Chacune a fait l'objet d'une analyse distincte.

La première partie du test se déroule des phases 1 à 6. Elle consiste à réaliser plusieurs décharges pour assurer un début de test à un état de charge 0% (en anglais, *State of Charge* ou *SoC*). La seconde partie se déroule des phases 7 à 21. Elle vise à réaliser plusieurs cycles de charge/décharge pour mesurer la capacité. La troisième partie a lieu entre les phases 22 à 182 et consiste en des séquences de pulses de courant à différents états de charge. Cette partie permet de déterminer l'influence de l'état de charge sur la mesure d'impédance. La quatrième partie se déroule durant les phases 183 à 185. Cette portion du test est la caractérisation de la cellule à un faible régime de courant. Enfin, la cinquième partie (phases 186 et 187) est une décharge

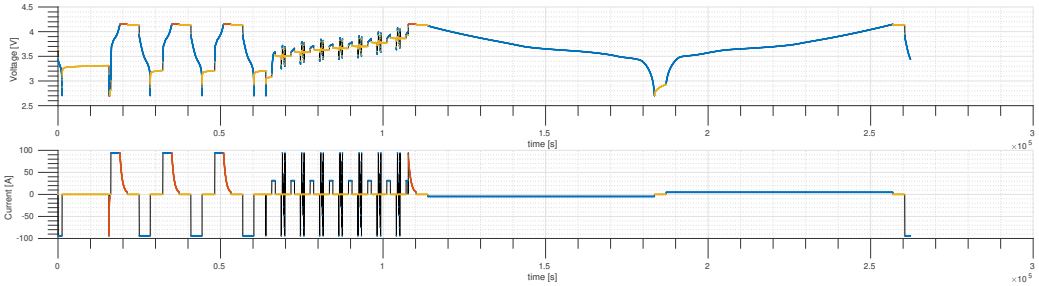


FIG. 1. Identification de modes de fonctionnement avec DATTES. Tension (en haut) et courant (en bas) de la cellule en fonction du temps (en bleu : courant constant ; en rouge : tension constante, en jaune : repos).

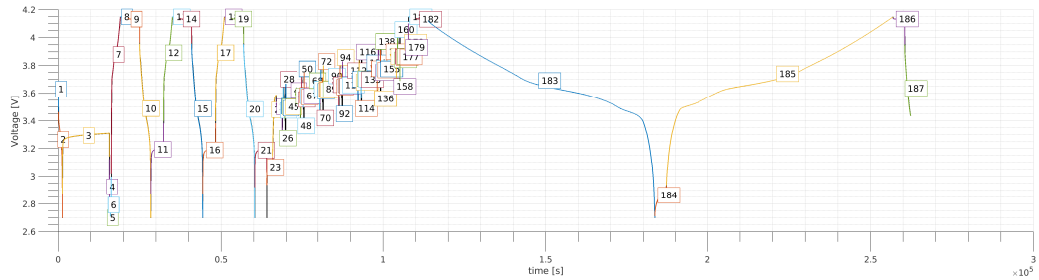


FIG. 2. Découpage du test en phases en fonction du mode de fonctionnement.

partielle pour éviter de stocker la batterie complètement chargée après le test.

La fonction *dattes_configure* permet à l'utilisateur d'adapter la méthodologie d'analyse utilisée pour traiter les différentes sous-parties du test. Le résultat de l'analyse de la capacité est présenté sur la figure 3. Pour obtenir ce résultat, l'utilisateur a défini la plage de tension accessible pour la référence de batterie testée ainsi que sa capacité nominale qui a permis de déterminer le régime de courant (en anglais *C-rate*) utilisé lors du test. À partir de ces informations, DATTES a détecté les phases pertinentes pour une mesure de capacité. Les phases 10, 15 et 20 ont été définies comme pertinentes pour une mesure de capacité en décharge à un régime de courant de 1C tandis que la phase 183 l'a été pour un régime de courant de C/20. De la même manière, DATTES a détecté les phases 7, 12 et 17 comme des phases de mesure de capacité en charge à un régime de courant 1C et à un régime de C/20 pour la phase 185. Pour les phases en charge, des phases à tension constante ont été identifiées immédiatement après la phase à courant constant. DATTES a donc également calculé la capacité en prenant en compte ces phases à tension constante.

DATTES permet également l'analyse de la résistance équivalente sur un pulse avec un courant constant. L'équation 1 montre que la résistance équivalente ($R_{\Delta t}$) est calculée comme le rapport entre la variation de la tension de la cellule et l'amplitude du pulse de courant un certain temps après le début du pulse. La fonction *dattes_configure* permet à l'utilisateur de définir la ou les valeurs de $\Delta t$ de son choix. La figure 4 présente le résultat de l'analyse de la résistance pour un $\Delta t$ égal à 2 et 10 secondes.

$$R_{\Delta t} = \frac{V(t_{pulse} + \Delta t) - V(t_{pulse})}{I_{pulse}} \qquad (1)$$

La figure 2 montre le découpage du test selon la phase d'utilisation de la batterie. Certaines portions du test peuvent être utilisées pour mesurer des résistances. Les phases 4 et 7 sont aussi des exemples de phases où une résistance peut être mesurée bien que ce ne soit pas nécessairement l'intention initiale de l'expérimentateur. Pour résoudre ce problème, la fonction *dattes_configure* permet à l'utilisateur de contraindre la zone de recherche des pulses de courant utilisés pour le calcul de résistance. D'autres restrictions peuvent également être mises en

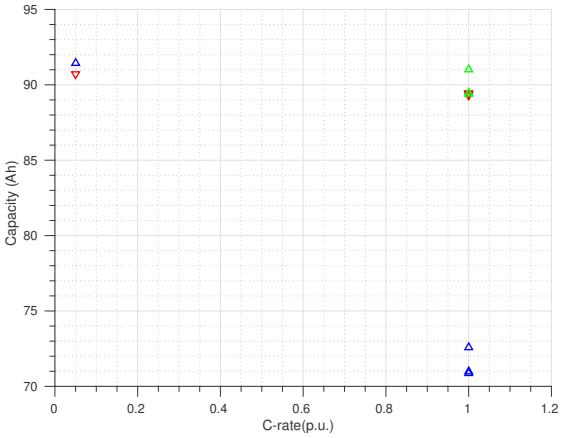


FIG. 3. Capacité (Ah) en fonction du régime de courant (p.u. de la capacité nominale). En rouge, capacité en décharge CC, en bleu capacité en charge CC, en vert capacité en charge CC+CV. Certains points sont confondus.

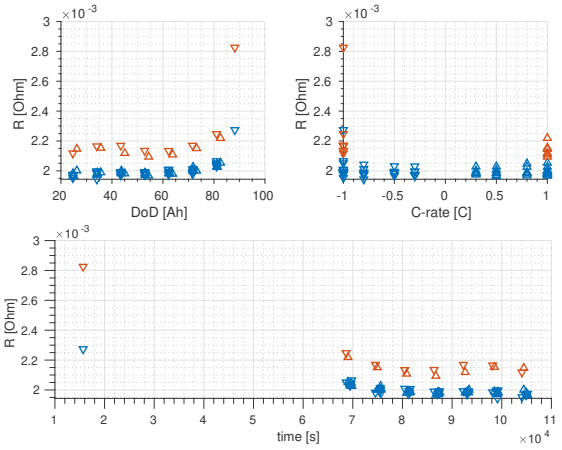


FIG. 4. Tracé de la résistance à 2 (en bleu) et 10 secondes (en rouge) en fonction de la profondeur de décharge (*DoD, Depth of Discharge*, en haut à gauche), du régime de courant (*C-rate*, en haut à droite) et du temps (en bas).

place par exemple sur la durée minimale/maximale des pulses utilisés ou sur la durée minimale du repos qui les précèdent.

Les phases 183 à 185 ont été identifiées comme pertinentes pour la caractérisation du comportement à faible courant de la batterie. DATTES permet d'analyser la tension en circuit ouvert dont le résultat est présenté sur la figure 5. Sur cette figure, un cycle à un régime de courant de 1C a été pris en compte. Grâce à la fonction *dattes_configure*, l'utilisateur peut éventuellement restreindre l'analyse des phases à faible régime de courant en définissant un seuil de courant maximal au dessus duquel les phases ne sont pas prises en compte. Les figures 6 et 7 montrent respectivement les résultats d'analyse incrémentale de la capacité (*ICA*) et d'analyse différentielle de la tension (*DVA*). Pour ce type d'analyse, le type de filtre est un exemple de paramètre pouvant être défini par l'utilisateur [7].

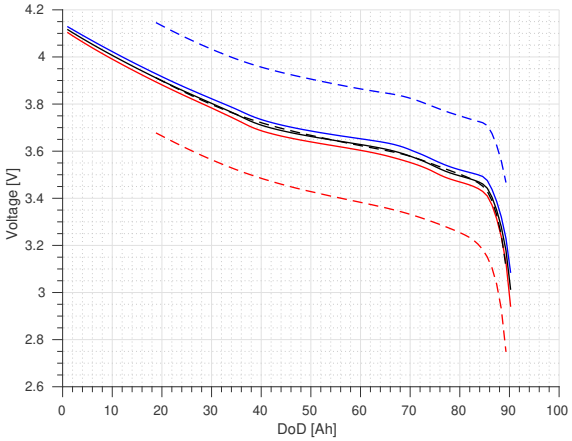


FIG. 5. Diagramme de pseudo OCV à C/20 (ligne continue) et à 1C (ligne discontinue). En rouge, le demi cycle de décharge, en bleu le demi cycle de charge, en noir le pseudo OCV.

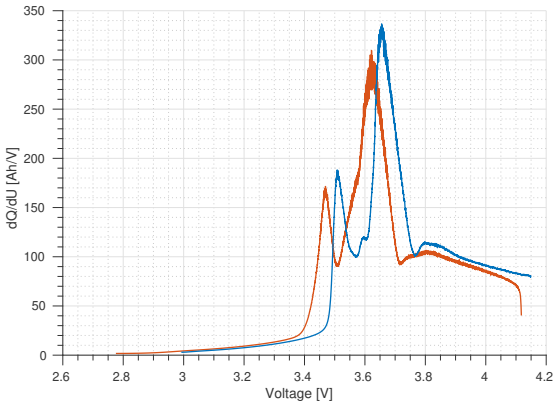


FIG. 6. Courbe ICA (rouge = décharge, bleu = charge).

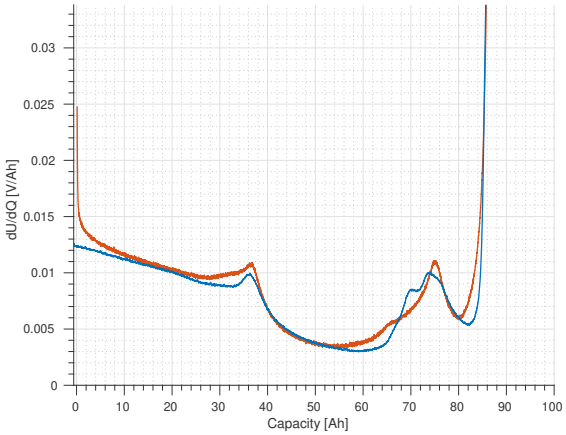


FIG. 7. Courbe DVA (rouge = décharge, bleu = charge).

## 4. CONCLUSION

Le traitement des données expérimentales sur les systèmes de stockage d'énergie est une tâche centrale dans le quotidien des chercheurs. Jusqu'à présent, la plupart des équipes de recherche privilégient leurs propres outils, ce qui limite considérablement la reproductibilité de la recherche. L'absence d'outils ouverts disponibles sur MATLAB , le langage le plus populaire dans le domaine, était un frein majeur à l'utilisation à large échelle d'outils ouverts. Cet article a présenté DATTES, le premier logiciel de traitement des données expérimentales pour les batteries écrit en code MATLAB et partagé sous une licence ouverte. Le logiciel fournit une boite à outils complète et personnalisable pour l'extraction, l'analyse et la visualisation des données expérimentales. Il crée également des passerelles avec d'autres logiciels et outils ouverts. DATTES permet ainsi aux utilisateurs de tirer le meilleur parti de leurs données expérimentales et de s'engager dans une démarche de science ouverte et reproductible.

## 5. RÉFÉRENCES


[1] Simon, Hettrick *Software in research survey (1.0)*, Zenodo, **2018**. https ://doi.org/10.5281/zenodo.1183562

[2] Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl, D., & Wilson, G., *How do scientists develop and use scientific software ?*, 2009 ICSE workshop on software engineering for computational science and engineering, **2009**

[3] Hassini, M., Redondo-Iglesias, E., & Venet, P., *Lithium-ion battery data : From production to prediction.* Batteries, **2023**. (En cours de soumission).

[4] Herring, P. ; Gopal, C.B. ; Aykol, M. ; Montoya, J.H. ; Anapolsky, A. et al. *BEEP : A Python library for Battery Evaluation and Early Prediction*, SoftwareX, **2020**, 11, 100506. doi :10.1016/j.softx.2020.100506

[5] Sulzer, V., Marquis, S. G., Timms, R., Robinson, M., & Chapman, S. J. *Python battery mathematical modelling (PyBaMM)*, Journal of Open Research Software, **2021**, 9(1). doi :10.5334/JORS.309

[6] Murbach, M. D., Gerwe, B., Dawson-Elli, N., & Tsui, L. K. . *impedance. py : A Python package for electrochemical impedance analysis.* Journal of Open Source Software, **2020**, 5(52), 2349. doi :10.21105/joss.02349

[7] Redondo-Iglesias E., Hassini M. *Data Analysis Tools for Tests on Energy Storage*. https://dattes.gitlab.io

[8] Jeanneret B., B. Kabalan, E. Redondo-Iglesias, R. Trigui, E. Vinot *VEHLIB*. https://gitlab.univ-eiffel.fr/eco7/vehlib

[9] Hassini, M., Redondo-Iglesias, E., Venet, P., Gillet, S., & Zitouni, Y. *Second-life batteries in a Mobile Charging Station : Model Based Performance Assessment*. Proceedings of the 35th International Electric Vehicle Symposium and Exhibition (EVS35), Oslo, Norway, 11–15 June **2022**.

[10] Hassini, M., Redondo-Iglesias, E., Venet, P., Gillet, S., & Zitouni, Y. *Second Life Batteries in a Mobile Charging Station : Experimental Performance Assessment.* **2022**.

[11] Hassini, M. ; Redondo-Iglesias, E. ; Venet, P. Second-Life *Batteries Modeling for Performance Tracking in a Mobile Charging Station*. World Electr. Veh. J. **2023**, 14, 94. doi :10.3390/wevj14040094

## A. INFORMATIONS SUR DATTES

TABLEAU 1 – Informations sur DATTES

| | |
|---|---|
| Version actuelle | DATTES 23.05 |
| Lien vers le dépôt du code | https://gitlab.com/dattes/dattes |
| Licence du logiciel | GNU GPL v3 |
| Système de suivi de version | git |
| Langage du code | MATLAB ou GNU Octave |
| Fonctionne sur | Linux, OSX ou Windows. |
| Lien vers la documentation | https://dattes.gitlab.io/ |
| E-mail de contact | dattes@univ-eiffel.fr |

## B. EXEMPLES DE CODE

### *B.1. Code utilisé dans la partie 3*

```
%Before running this script, download data for this demo:
% URL to define....
close all
clear all

% Import all biologic tests in src_folder:
% 'v' = verbose
% 'm'= merge files in each folder to single xml file
xml_filelist = dattes_import('./','biologic','vm');

% Take first file in the list:
file_in = xml_filelist{1};

% Structure data:
result = dattes_structure(file_in);

% Configure data (default configuration):
result = dattes_configure(result);

% Analyse data
% 'CRPI' = capacity, resistance, pseudo OCV, ICA
result = dattes_analyse(result,'CRPI');

% Plot results:
% 'xpc' = profiles, phases, configuration
% 'CRPI' = capacity, resistance, pseudo OCV, ICA
dattes_plot(result,'xpcCRPI');

%Save results in mat file:
dattes_save(result);
```

### *B.2. Exemples de code pour l'import de données (dattes_import)*

```
% 1. Chercher et importer des fichiers bitrode
dattes_import(source_dir,'bitrode',options, dest_dir);

% 2. Chercher et importer des fichiers biologic
dattes_import(source_dir,'biologic',options, dest_dir);

% 3. Chercher et importer des fichiers arbin (*.CSV)
dattes_import(source_dir,'arbin_csv',options, dest_dir);

% 4. Chercher et importer des fichiers arbin (*.XLS)
dattes_import(source_dir,'arbin_xls',options, dest_dir);

% 5. Chercher et importer des fichiers neware
dattes_import(source_dir,'neware',options, dest_dir);

% 6. Chercher et importer des fichiers digatron
dattes_import(source_dir,'digatron',options, dest_dir);
```

### *B.3. Exemples de code pour l'analyse de données (dattes_analyse)*

```
% 1. Calcul de capacité
dattes_analyse(mat_file,'C');

% 2. Calcul de résistance
dattes_analyse(mat_file,'R');

% 3. Identification d'impédance
dattes_analyse(mat_file,'Z');

% 4. Identification d'OCV par points
dattes_analyse(mat_file,'O');

% 5. Identification d'OCV par pseudo-OCV
dattes_analyse(mat_file,'P');

% 6. Analyse incrémentale de la capacité
dattes_analyse(mat_file,'I');
```

### *B.4. Exemples de code pour la visualisation (dattes_plot)*

```
% 1. visualisation des profils
dattes_plot(mat_file,'x')

% 2. visualisation du SoC et du DoD
dattes_plot(mat_file,'S')

% 3. visualisation des phases
dattes_plot(mat_file,'p')

% 4. visualisation de la configuration
dattes_plot(mat_file,'c')

% 5. visualisations de l'analyse
% 5.1. Capacité
dattes_plot(mat_file,'C')

% 5.2. Résistance
dattes_plot(mat_file,'R')

% 5.3. Impédance
dattes_plot(mat_file,'Z')

% 5.4. OCV par points
dattes_plot(mat_file,'O')

% 5.5. pseudo OCV
dattes_plot(mat_file,'P')

% 5.6. ICA/DVA
dattes_plot(mat_file,'I')
```

### *B.5. Exemples de code pour l'export de données (dattes_export)*

```
% 1. export du résultat en .json
dattes_export(result,'Aj'); % 'A'=all, 'j' = json

% 2. export des profils en .csv
dattes_export(result,'Pc'); % 'P'=profiles, 'c' = csv

% 3. export des profils en .json
dattes_export(result,'Pj'); % 'P'=profiles, 'j' = json

% 4. export des phases en .csv
dattes_export(result,'pc'); % 'p'=phases, 'c' = csv

% 5. export des metadonnées en .json
dattes_export(result,'Mj'); % 'M'=metadata, 'j' = json

% 6. export des metadonnées comme script
dattes_export(result,'Ms'); % 'M'=metadata, 's' = *.m

% 7. export de la configuration comme script
dattes_export(result,'Cs'); % 'C'=configuration, 's' = *.m
```