Modélisation des microréseaux : quelles architectures de code pour quelles fonctions ?

Pierre HAESSIG[1], Elsy EL SAYEGH[1,2], Evelise DE GODOY ANTUNES[3], Nabil SADOU[1]
[1]IETR, CentraleSupélec, Rennes, [2]EDF R&D, Palaiseau, [3]LAPLACE, Toulouse

RÉSUMÉ – Le dimensionnement des systèmes énergétiques tels que les microréseaux constitue un problème d'optimisation qui nécessite un programme informatique pour obtenir une solution numérique. Plus le problème de dimensionnement est scientifiquement riche (ex. : prise en compte des incertitudes), plus le programme pour le résoudre est informatiquement complexe. Cette complexité peut mettre en péril la validité et la maintenabilité d'un tel programme. La conception d'un outil de dimensionnement de microréseau est donc un (méta)-problème en soi. Cet article propose des pistes de conceptions logicielles modulaires qui permettent de traiter à la fois un problème de dimensionnement "de base" et plusieurs variantes scientifiques importantes. Cette approche qui vise à réutiliser un maximum de briques de base devrait permettre d'obtenir des programmes de meilleure qualité. Ces propositions s'accompagnent de la publication des premières versions de la famille d'outils de simulation et d'optimisation open source Microgrids.X.

Mots-clés – Microréseau, dimensionnement, conception logicielle

1. INTRODUCTION

Les microréseaux, systèmes énergétiques autonomes, permettent l'électrification des petites communautés éloignées des grands réseaux continentaux. Le dimensionnement optimal de leurs composants (moyens de production intermittents ou contrôlables, moyens de stockage…) est un problème d'optimisation où la performance dépend de l'interaction systémique des composants plutôt que de leurs seules caractéristiques individuelles. Pour cette raison, la résolution du problème de dimensionnement n'est pas analytique, mais utilise des *outils numériques* (depuis au moins les années 1990 avec HOMER [1] ou Hybrid2 [2]). Les recherches sur le sujet nécessitent donc le développement de programmes/codes de calcul. Le processus de développement, avec certaines spécificités liées aux logiciels scientifiques [3] (cité par [4]), vise à produire un programme doté des propriétés désirables suivantes :

- **correction** : produire le résultat spécifié pour un problème à traiter donné
- **facilité d'usage** : produire des résultats avec simplicité (relativement à complexité du problème)
- **flexibilité/richesse fonctionnelle** : produire des résultats pour des variantes du problème
- **durabilité/maintenabilité** : produire des résultats au fil des années malgré les changements d'environnement (OS, versions des dépendances) et de personnes
- **rapidité d'exécution** : produire le résultat "assez vite" pour être utilisable et/ou pertinent

En sus de la vitesse d'exécution, il y a également l'enjeu important de la *rapidité de développement*, mais qui est attaché au processus de développement plutôt qu'au programme qu'il produit. Remarquons que les enjeux de flexibilité et de durabilité seront, peut-être, plus désirables dans un contexte de recherche en équipe et sur le temps long. Ils expriment, de façon plus générale, un besoin *d'évolutivité* qui est particulièrement fort pour les codes scientifiques (par rapport aux programmes informatiques en général) : pouvoir ajouter des fonctionnalités tout en préservant l'existant. En effet, la spécification du résultat attendu est souvent vague au début d'un projet de recherche [3].

Point important, tous ces enjeux (sauf sans doute celui de la vitesse d'exécution) bénéficient de trois attributs désirables de tout code informatique : *lisibilité, réutilisabilité* et *testabilité* et ces attributs sont des effets secondaires positifs d'une **conception modulaire** de l'architecture d'un programme [5].

On pourrait croire, vu le nombre d'outils de dimensionnement déjà créés, que le sujet du développement de ces outils est dépassé. En effet, après HOMER [1] et Hybrid2 [2], sont apparus plus récemment (i)HOGA [6] puis ODYSSEY [7] et PREMO [8]. Cependant, comme analysé par Guinot dans sa thèse qui justifiait la création d'ODYSSEY, vu le nombre de variantes potentielles du problème à traiter, "l'élaboration d'un outil unique semble extrêmement difficile à envisager" [7, Ch2]. De plus, aucun de ces outils n'est open source, ce qui pose des problèmes généraux de science ouverte [9], mais surtout cela rend plus difficiles les *contributions méthodologiques* à la résolution du problème de dimensionnement des microréseaux. En effet, ces contributions nécessitent souvent de modifier la formulation du problème, ce qui implique de modifier un outil logiciel malheureusement inaccessible. Il existe donc toujours un besoin de concevoir de nouveaux outils de dimensionnement, en particulier pour répondre aux problématiques scientifiques d'actualité telles que la prise en compte des incertitudes [10], l'impact environnemental sur cycle de vie [11, 12] ou les systèmes multi-énergies [13].

Dans cet article, nous étudions donc la question suivante : *Comment architecturer un programme de simulation énergétique de microréseau ?* Dans ce (méta)-problème de conception logicielle, au-delà des enjeux généraux de la programmation scientifique (cf. listes de "bonnes pratiques" [4, 5]), nous nous concentrons sur l'étude des *liens entre choix d'architectures dans une optique de modularité et prise en compte des enjeux scientifiques* précités. Cette analyse s'appuie sur notre expérience en recherche et en enseignement sur le sujet. En particulier, nous avons récemment publié les premières versions du code Microgrids.X [14], une famille d'outils dans la lignée d'HOMER, mais open source, utilisé en recherche [15] et en enseignement (formation ingénieur niveau Master 1 et 2 sur l'optimisation des systèmes énergétiques, où l'enjeu de la modularité est également important dans un but pédagogique).

2. DIMENSIONNEMENT DES MICRORÉSEAUX : PROBLÈME DE BASE ET VARIATIONS

Avant d'aborder l'architecture des outils de dimensionnement de microréseaux, nous définissons ce que nous appelons le "problème de base" ainsi que ses variantes qui vont justifier le besoin d'architectures modulaires présentées dans la partie 3.

2.1. Dimensionnement par approche "boite noire"

Le problème de dimensionnement de microréseau consiste à trouver un dimensionnement satisfaisant pour les différents composants d'un tel système (puissance nominale ou capacité énergétique des moyens de production et de stockage). Le ca-

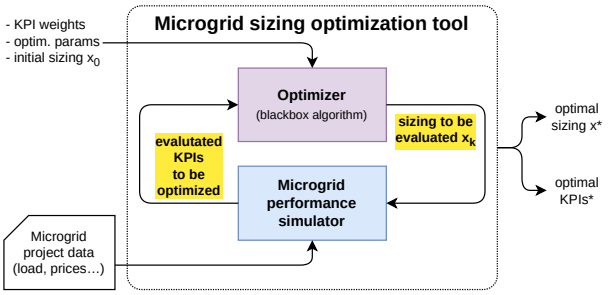

FIG. 1. Architecture typique d'un outil de dimensionnement optimal de microréseau basé sur une approche "optimisation boite noire". ("KPIs" : Key Performance Indicators, qui sont de nature technico-économique).

ractère satisfaisant est mesuré via une ou des fonctions objectifs à optimiser et des contraintes à satisfaire qui expriment des besoins technico-économiques (coût total sur cycle de vie, qualité de service, etc.).

Nous nous plaçons dans un contexte d'étude de *prédimensionnement*, située avant une étude techniquement plus détaillée, qui doit orienter une *prise de décision d'investissement*. Ainsi, nous utilisons une *modélisation énergétique*, c'est-à-dire une évaluation des échanges d'énergie entre composants sur des échelles de temps allant de quelques minutes (pas de temps) à quelques décennies (horizon économique du projet).

De plus, nous nous plaçons dans l'approche "optimisation boite noire" illustrée Fig. 1. Elle est utilisée par HOMER [1], les autres outils précités [2, 6, 7, 8] et notre outil Microgrids.X. Dans cette approche, un algorithme d'optimisation fait appel à un simulateur chargé d'évaluer la performance d'un dimensionnement donné ("modèle direct"). Il est écrit dans un langage impératif (Python, Matlab, Julia, C++, etc.). L'optimiseur (souvent de type génétique, mais pas obligatoirement [15]) étant une routine générique prise dans une bibliothèque, nous n'en traitons pas ici. Les *questions d'architecture de code portent donc sur le simulateur*.

Précisons qu'il existe une démarche alternative à l'optimisation boite noire. C'est l'approche "Mathematical Programming" utilisée par exemple par DER-CAM [16], FINE [17] ou OMEGAlpes [18]. Elle est basée sur une modélisation par équations acausales et une résolution par un solver d'optimisation (souvent LP ou MILP [1]). Cette approche a ses avantages, comme la grande flexibilité dans la structure des systèmes énergétiques modélisables (cf. discussion ci-après), mais elle a aussi ses défauts. En particulier, la performance opérationnelle est surestimée à cause d'une gestion d'énergie intrinsèquement anticipative [19]. Vu que les langages de développement sous-jacents sont très différents (langage de description d'équations comme AMPL, GAMS, PyOMO ou JuMP), les enjeux de modularité du code sont différents nous n'en traiterons pas davantage ici.

2.2. *Problème de base et variations*

Dans le contexte de modélisation énergétique évoqué ci-avant, nous définissons un "problème de base" de dimensionnement de microréseau. Vis-à-vis de chacun des attributs de ce cas simple, nous définissons des extensions qui constituent des verrous scientifiques d'actualité [13, Ch 1].

1. **Structure de microréseau simple** : monoénergie (électrique) avec 1 charge, n sources de production fatales, 1 générateur contrôlable, 1 moyen de stockage. Extensions :
   — Système multi-énergies (thermique, hydrogène)
   — n générateurs contrôlables et/ou moyens de stockage
   — Plusieurs types de charges (critique/non critique), flexibilité (charge déplaçable, effacement, etc. souvent lié à l'aspect multi-énergies)
2. **Gestion d'énergie simple** basée sur des règles de priorité : alimentation de la charge prioritairement par les sources renouvelables (ou fatales) puis le stockage, puis le générateur (version simplifiée de la stratégie *load-following* utilisée dans HOMER [1, 20]). Extensions :
   — Gestion d'énergie basée sur des règles plus complexes et paramétrables [21] (ex. : stratégie *cycle-charging* utilisée également dans HOMER [1, 20])
   — Gestion d'énergie basée sur de l'optimisation (ex. Model Predictive Control [22]).
3. **Optimisation technico-économique figée** (ex. dans HOMER, minimisation du Net Present Cost (NPC) sous contrainte d'un taux d'énergie non servie supérieur à un seuil désiré) Extensions :
   — Libre choix du problème d'optimisation parmi les indices de performance mono-impact technico-économiques calculés
   — Calcul multi-impact d'Analyse sur Cycle de Vie (ACV). Exemples de catégories d'impacts environnementaux en ACV : changement climatique, acidification, épuisement des ressources naturelles.
4. **Entrées déterministes** : séries temporelles de consommation et météo et paramètres technico-économiques (prix des composants, etc.) supposés connus avec certitude. Extension :
   — Entrées incertaines (intervalles, distributions de probabilité, processus aléatoires, etc.)
5. **Simulation opérationnelle sur une seule année représentative** (et donc extrapolation sur la durée du projet de la performance technico-économique supposée constante). Extension :
   — Simulation opérationnelle pluriannuelle qui prend en compte l'évolution des données
6. **Dimensionnement fixe** pour toute la durée du projet. Extension :
   — Possibilité de redimensionner les composants en cours de projet (ex. : ajout de panneaux solaires)

Il faut préciser que certaines des extensions mentionnées sont interdépendantes. Par exemple, la structure du microréseau est liée à la gestion d'énergie (cf. partie 3.3). De même, le redimensionnement est lié à une évolution pluriannuelle (cf. partie 3.5)

2.3. *Architecture de base du simulateur Microgrids.X*

Nous détaillons ici l'architecture de base de notre simulateur Microgrids.X [14], inspiré d'HOMER [1], qui permet de résoudre le problème dimensionnement de base expliqué partie 2.2 et ouvre la voie à certaines extensions expliquées dans la partie suivante. Notons que Microgrids.X est en fait une *famille* de codes, partageant une architecture et une nomenclature similaires, mais implémentés dans différents langages : Julia (Microgrids.jl), Matlab/Octave (Microgrids.m), Python (Microgrids.py) ainsi qu'une démo en ligne (Microgrids.web).

Comme expliquée partie 2.1, dans l'approche de dimensionnement par "optimisation boite noire", toute la complexité de modélisation se trouve dans le simulateur chargé d'évaluer la performance d'un dimensionnement donné. Celui-ci se compose de deux blocs principaux (Fig. 2) :

— le **simulateur opérationnel** modélise les flux d'énergie entre composants au pas de temps de l'ordre d'une heure, sur *une seule année représentative*. Ce calcul nécessite de choisir une loi de gestion de l'énergie, actuellement basée sur des règles simples. Les trajectoires opérationnelles sont agrégées en statistiques annuelles pour alimenter le simulateur économique (ex. : volume de fuel consommé, énergie non servie à la charge).
— le **simulateur économique** évalue des indicateurs de coût global du projet sur sa durée de vie (NPC, LCOE : Levelized Cost of Energy) qui incluent l'investissement initial, la maintenance et remplacement des composants, avec

1. (MI)LP : (Mixed Integer) Linear Programming

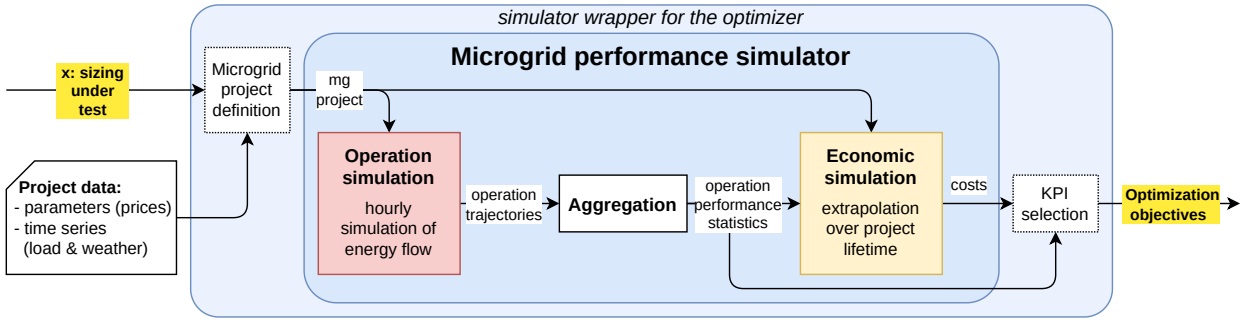

FIG. 2. Architecture du simulateur Microgrids.X [14], inspiré d'HOMER [1]. Le cœur du simulateur reçoit une structure de données descriptive du projet. Il peut fonctionner indépendamment de la boucle d'optimisation (Fig. 1), mais peut s'y connecter via une couche d'adaptation légère ("wrapper") qui fusionne le vecteur de paramètres à optimiser x dans la structure descriptive. Cette couche calcule en sortie la ou les fonctions objectifs et contraintes à optimiser.

effet d'un taux d'actualisation. Ce calcul fait implicitement une *extrapolation* des statistiques de performance opérationnelle sur la durée de vie du projet, en supposant qu'elles sont constantes au fil des années.

C'est le bloc opérationnel qui concentre le plus de complexité (dispatch d'énergie) et accapare le plus grand temps de calcul. Remarquons que ce découpage repose sur l'hypothèse du problème de base que l'on ne simule qu'une année représentative (et l'on voit dans l'étude comparative de Guinot [7, Ch2 §3.2.2] que c'est une hypothèse partagée par beaucoup d'outils). Nous discutons partie 3.5 comment dépasser ce cadre.

3. ARCHITECTURE DU SIMULATEUR FACE AUX ENJEUX ET AUX VARIANTES DU PROBLÈME

Nous étudions à présent dans quelle mesure l'architecture du simulateur de base (Fig. 2) permet de répondre à la fois aux enjeux généraux exprimés en introduction et surtout aux enjeux scientifiques (les extensions du problème de base détaillées partie 2.2). Il s'agit en particulier de voir comment une conception modulaire permet de réutiliser au mieux les éléments de l'architecture de base, gage d'une meilleure évolutivité et qualité du code.

3.1. Enjeu de rapidité d'exécution

Paradoxalement, nous traitons en premier l'enjeu moins prioritaire de la performance d'exécution du code. En effet, la recherche de performance peut malheureusement entrer en contradiction avec une conception modulaire et contraindre certains choix. À côté de cela, nous allons voir qu'il est utile de prendre en compte l'enjeu de performance en même temps que certaines variantes du problème à résoudre.

Effet du langage de programmation Tout d'abord, comme analysé par Guinot en 2013 [7, Ch3 §2], le choix du langage de programmation influe sur la rapidité. Son choix du C++ pour ODYSSEY (comme Dufo-López pour (i)HOGA [6]) est motivé par le caractère *compilé* de langage, mais les langages *interprétés* (Python, Matlab/Octave) sont plus populaires en programmation scientifique, car plus flexibles. Cette dichotomie entre compilé et interprété s'est cependant amoindrie depuis, car la décennie 2010 a vu apparaître des outils de compilation de code Python (Cython [23], Pythran[24]) ou bien la compilation à la volée (dite "JIT : Just-in-Time" [25]) avec Numba [26] pour Python et avec, de façon native, le langage Julia [27]. Matlab dispose aussi d'une compilation JIT renouvelée en 2015 [28], potentiellement très rapide (selon nos mesures sur Microgrids.m), mais d'utilisation un peu incertaine car non documentée.

Ces différentes technologies JIT ont pour point commun de reposer très fortement sur le *typage des variables* (entier, flottant, etc.) qui, dans des langages dynamiques comme Python, Matlab ou Julia, doit être inféré ce qui n'est pas toujours facile. De notre expérience avec ces trois langages, la simulation de microréseau n'est pas toujours facile à accélérer avec la compilation JIT pour une raison de *structure de données*. En effet, la description d'un projet de microréseau nécessite beaucoup [2] de paramètres (rectangle biseauté Fig. 2).

Pour répondre à l'enjeu de modularité et de lisibilité, nous avons regroupé ces paramètres par composant dans des structures de données hiérarchisées hétérogènes (`struct` Matlab et Julia, dictionnaires, classes). Cependant, on sort alors du "cas d'école" d'application du JIT où les fonctions à accélérer ne prennent en entrée que quelques variables numériques (scalaires ou tableaux). Par exemple, Numba ne supporte qu'expérimentalement et avec des restrictions les classes et dictionnaires Python. Nous ne l'avons donc pas encore essayé et notre simulateur Python est nettement plus lent que ceux en Matlab et Julia [3] (facteur ~100). En Julia, nous avons dû utiliser une sémantique assez avancée ("Parametric Composite Types") pour obtenir un simulateur à la fois rapide et flexible (pour le calcul numérique exact du gradient par différentiation automatique [15]).

Parallélisation Un deuxième enjeu de rapidité d'exécution est lié à la *parallélisation* des calculs. Il prend de l'importance depuis la généralisation des processeurs multi-coeurs.

Dans la boucle d'optimisation de l'approche boite noire (Fig. 1), la parallélisation la plus naturelle semble être l'évaluation simultanée de plusieurs dimensionnements. Cependant, cela nécessite 1) un algorithme d'optimisation itératif qui travaillent par "paquets" de solutions et 2) que l'interface (API) d'un tel solver permette la parallélisation. Pour l'aspect algorithmique, ce sont typiquement les optimiseurs à population (génétique, essaim particulaire, etc.). Pour l'API, les bibliothèques dédiées à l'optimisation globale permettent généralement bien la parallélisation (ex. : Matlab Global Optimization Toolbox [29] ou PyGMO pour Python). À l'inverse, dans les bibliothèques d'optimisation "généraliste" c'est plus rare (bibliothèque scipy.optimize : parallélisation possible seulement avec `differential_evolution`, bibliothèque NLopt [30] : aucune parallélisation possible malgré quelques algorithmes parallélisables comme le multi-start).

Vu que le support de la parallélisation côté optimiseur est imparfait, il semble intéressant d'étudier comment paralléliser les calculs à *l'intérieur* du simulateur. Cela nous semble difficile et peu intéressant pour le simulateur de base (Fig. 2) qui s'exécute en moins d'1 ms, mais très pertinent lorsqu'on souhaite prendre en compte des incertitudes (approche Monte Carlo §3.4) ou une simulation pluriannuelle (§3.5).

2. 35 paramètres pour un microréseau simple générateur-solaire-batterie
3. notre simulateur Julia, le plus rapide, évalue en 0.2 ms un microréseau sur 1 année de référence à pas d'1 heure (ordinateur portable Intel Core i7-1165G7)

3.2. *Flexibilité des critères d'optimisation et évaluation multi-impacts*

L'optimalité du dimensionnement que doit fournir l'outil est mesurée par différents critères technico-économiques qui vont s'insérer dans une ou des fonctions objectifs et des contraintes. La flexibilité d'usage de l'outil pousse à rendre personnalisable ces fonctions. De ce point de vue, le problème d'optimisation résolu par HOMER est fixé (minimisation du Net Present Cost sous contrainte de charge non servie suffisamment faible) et plus ouvert dans iHOGA [31], mais toujours basée sur une liste prédéfinie de quelques indicateurs. Notre d'architecture de simulateur (Fig. 2) permet un peu plus de flexibilité grâce à l'utilisation d'une fonction "wrapper" (enveloppe bleu clair). Le coeur du simulateur (bleu plus foncé) produit beaucoup d'indicateurs de performance (KPIs), regroupés dans des structures de données hiérarchisées dans lesquels la fonction wrapper, qui appelle le simulateur, peut piocher. Cette approche, plus flexible, exige par contre que l'utilisateur code ce wrapper, c'est-à-dire une fonction de quelques lignes, pour laquelle un exemple est fourni. Cette approche s'implémente également plus facilement avec un langage interprété qu'avec un langage compilé (où la fonction wrapper devra être un remplacé par exemple par un mécanisme d'options comme dans l'interface d'iHOGA [31, §3.1.2 Optimization tab]). Nous recommandons aussi cette séparation entre coeur de simulateur et wrapper dans un contexte d'enseignement, car cela facilite pour les étudiants l'utilisation de différents optimiseurs, par exemple mono- et multi-objectifs, dans un but pédagogique de comparaison.

Notre proposition a néanmoins ses limites. Tout d'abord, si un indicateur de performance n'est pas calculé dans le cœur du simulateur, il faudra bien l'y ajouter. Ensuite, la prise en compte des multiples impacts sur cycle de vie, qui prend de l'importance en génie électrique [11, 12], est également un point délicat pour l'architecture du simulateur. En effet, les méthodologies d'ACV définissent différentes catégories d'impacts environnementaux : changement climatique, acidification, épuisement des ressources, etc.[32, Ch5]). Cependant, la liste des impacts pris en compte peut changer d'une étude à l'autre. Cela implique que, dans les données d'entrées du simulateur, chaque composant doit être doté d'une liste de facteurs d'impact de taille non définie statiquement. Ensuite, ces données doivent être propagées dans le simulateur pour calculer les impacts totaux du microréseau. Nous n'avons pas de proposition pour implémenter cela pour l'instant. Seule certitude, il y a un intérêt à faire le calcul d'ACV en un seul appel au simulateur pour profiter du fait que la simulation opérationnelle est commune à tous les calculs d'impact.

3.3. *Structure de microréseau complexe et gestion d'énergie*

L'étude des systèmes multi-énergies (hydrogène, thermique, etc. plutôt que "simplement" électrique) ou des structures multi-noeuds est un enjeu scientifique important [13]. Même si l'architecture de simulateur proposée (Fig. 2) ne présage pas de la complexité de la structure du microréseau, notre simulateur Microgrids.X ne modélise pour l'instant qu'une structure électrique simple ("problème de base" §2.2). Le manque de flexibilité de l'outil se situe dans :

1. la structure de données qui décrit le projet
2. la loi de gestion d'énergie

La première condition pour permettre une modélisation de système multi-énergies est d'imaginer une structure de données adaptée, donc plus complexe. Voir par exemple la discussion de Radet sur la conception de son outil `Genesys.jl` [13, Ch6].

Le point le plus bloquant se situe sur la loi de gestion d'énergie qui est pour l'instant codée "en dur" dans le bloc de simulation opérationnelle et qui ne gère qu'une répartition des flux d'énergie basée sur des règles de priorité simple inspirées du *load-following* d'HOMER [1, 20], pour un seul moyen de stockage et un seul générateur contrôlable. Il est bien sûr imaginable de supporter des cas plus complexes, mais la difficulté est plus profonde : si l'utilisateur est autorisé à décrire un microréseau de structure complexe (multi-énergies, multi-noeuds, etc.), il faut qu'il puisse décrire une loi de gestion adaptée.

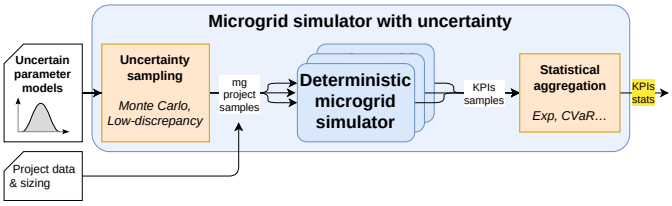

FIG. 3. Simulateur de microréseau avec entrées incertaines. L'implémentation se fait par des appels multiples au simulateur de base déterministe (Fig. 2) sur des échantillons des valeurs incertaines. Les échantillons d'indicateurs de performance (KPIs) ainsi produits sont ensuite agrégés statistiquement.

Il est envisageable de modifier le simulateur opérationnel pour qu'il prenne en entrée une fonction de dispatch fournie par l'utilisateur (via une référence/pointeur/handle de fonction). Les langages interprétés permettent cela sans recompilation. Cependant, il faut alors définir la signature, c'est-à-dire les entrées et sorties, de cette fonction. Doit-elle être générique ou dépendante de la structure du microréseau ? En tout cas, un tel mécanisme fournirait une flexibilité également bienvenue pour implémenter des lois gestion plus complexes y compris pour des systèmes mono énergie (ex. règles paramétrables [21] ou stratégie *cycle-charging* d'HOMER [1, 20]). Nous n'avons donc pas à ce jour de réponse satisfaisante pour répondre à cet enjeu.

Remarquons que l'approche "Mathematical Programming" (§2.1) est de ce point de vue avantagée, car il n'y a pas de loi de gestion à implémenter et réadapter à chaque cas. En effet, la gestion d'énergie est réalisée *implicitement*, via la résolution du problème global de dimensionnement dont les variables d'échanges d'énergie à chaque instant sont des degrés de liberté.

3.4. *Projet de microréseau avec entrées incertaines*

La prise en compte des incertitudes est une autre des problématiques importantes pour le dimensionnement des microréseaux. Nous nous focalisons ici sur l'incertitude sur les paramètres dont dépend le dimensionnement durant la phase de prédimensionnement, c'est-à-dire bien avant le démarrage du projet (ex. : prix des composants et du carburant, consommation). Les incertitudes peuvent être prises en compte dans une perspective *d'analyse de sensibilité* d'un dimensionnement donné [33] ou bien pour trouver un dimensionnement robuste à l'incertain [34].

Dans tous les deux cas, dans une approche dite Monte Carlo, il s'agit de lancer le simulateur de multiples fois sur un échantillon de valeurs des paramètres incertains. Il est aisé de réutiliser le simulateur de base (qui est déterministe) pour prendre en compte l'incertain via une composition emboitée. Le simulateur Monte Carlo ainsi obtenu (Fig. 3), prend en entrée une description probabiliste des paramètres incertains et contient trois blocs :

1. **Échantillonneur** des paramètres incertains qui génère des réalisations des paramètres incertains à partir de leur description probabiliste
2. **Simulateur déterministe**, *réutilisé sans modification*, à appeler pour chaque échantillon des paramètres
3. **Agrégateur statistique** qui va estimer des quantités déterministes. Ex. : Espérance, Conditional Value at Risk (CVaR) [35][13, Ch4]

Via cette architecture, il n'y a, du point de vue de l'optimiseur, plus de différence fonctionnelle entre le simulateur déterministe et le simulateur Monte Carlo : ils deviennent interchangeables.

En effet, grâce au bloc agrégateur, ces deux simulateurs produisent des quantités déterministes à optimiser.

Comme annoncé partie 3.1, il y a un intérêt à paralléliser les calculs à l'intérieur du simulateur. Pour cela, les appels répétés au simulateur déterministe sont de parfaits candidats. En résumé, la prise en compte des incertitudes est facile à prendre en compte via une conception modulaire. Par contre, prendre en compte les incertitudes augmente forcément le temps de simulation et donc d'optimisation, même si la parallélisation peut atténuer cela en partie.

3.5. Projet de microréseau évoluant au fil des années

La prise en compte de l'évolution d'un microréseau à une échelle pluriannuelle est également un enjeu scientifique important. Nous y distinguons deux types que l'on peut bien sûr combiner :

1. **Données pluriannuelles** : évolution pluriannuelle des données du problème (prix des ressources, consommation, météo, etc.)
2. **Redimensionnement** : le dimensionnement peut évoluer au cours du projet grâce à des réinvestissements en cours de projet (redimensionnement et/ou ajout de nouveaux composants)

Ces deux extensions du problème de base (§2.2) nécessitent de dépasser l'hypothèse habituelle des outils classiques qui ne simulent qu'une année opérationnelle. Notons qu'HOMER propose justement un module optionnel "Multi-year" et iHOGA (version "PRO+") propose une option "multiperiod" qui couvrent l'évolution des entrées (croissance de la charge, baisse de performance des panneaux solaires, etc.), *mais pas le réinvestissement*.

Données pluriannuelles : Pour prendre en charge ce modèle tout en nous appuyant sur les éléments de l'architecture de base (Fig. 2), nous proposons d'étudier deux variantes a priori envisageables Fig. 4. Remarquons d'abord que dans les deux versions intervient une modification inévitable : il faut un nouveau modèle économique qui accepte une *série de statistiques annuelles* au lieu des statistiques d'une année représentative à extrapoler.

Ensuite, pour comprendre la différence entre les deux propositions, il faut savoir que, dans la variante de base (Fig. 2), le simulateur économique, vu qu'il extrapole la vie opérationnelle, est en charge de calculer, approximativement, le planning de remplacement des composants usés (batterie, générateur…) pour en déterminer le coût. Ce calcul n'a en effet pas de sens au sein d'un simulateur opérationnel travaillant sur une unique année représentative. Par contre, dans modèle pluriannuel, la place du calcul de remplacement est ouverte. Dans l'architecture du bas, qui peut sembler plus naturelle, on allonge la durée de la simulation opérationnelle pour couvrir les 10–30 ans d'horizon du projet. Cela permet de prendre en charge un planning de remplacement exact [4]. Même si on intègre finalement la gestion du remplacement dans le bloc opérationnel, il ne suffit de toute façon pas de modifier la valeur d'un paramètre pour allonger la simulation au-delà d'un an. En effet, vu que le calcul économique prend en compte un taux d'actualisation, il est nécessaire de collecter des bilans opérationnels annuels. Ainsi, le bloc de "opérationnel allongé" du bas de la Fig. 4 diverge nettement de la version de base "1-year", ce qui empêche de le réutiliser.

Nous proposons en alternative l'architecture du haut qui utilise une série d'appels au bloc d'opération "1-year" inchangé par rapport à Fig. 2 (parfaite réutilisation), dans lequel l'agrégation de statistiques annuelles est naturel. Par conséquent, nous proposons de maintenir un calcul des remplacements dans le bloc économique, même s'il est plus approximatif (car basé sur

4. avec un décompte à l'heure près de l'instant de fin de vie d'un composant, même si cette précision semble déraisonnable vu l'incertitude entourant la durée de vie d'un composant

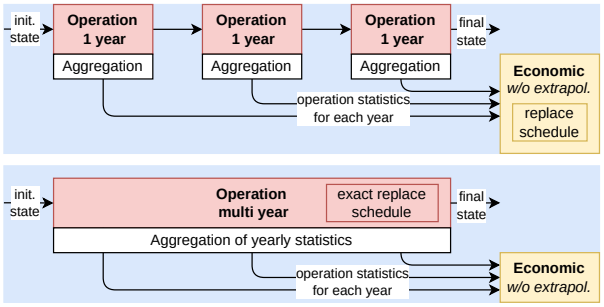

FIG. 4. Simulateur de microréseau pluriannuel : deux alternatives d'architectures envisageables. En haut : appels séquentiels à la simulation opérationnelle annuelle, inchangée par rapport à l'architecture de base Fig. 2. En bas : simulation opérationnelle pluriannuelle d'un seul trait.

les bilans annuels : la date de remplacement peut bouger de quelques semaines).

Pour finir, cette variante du haut de la Fig. 4 ouvre la voie à d'autres sous-variantes, très intéressantes du point de vue de la rapidité d'exécution, au prix de quelques approximations :

— **Parallélisation des simulations opérationnelles**, si *on néglige le chainage des états* final et initial entre années successives. On peut par exemple réinitialiser l'état de charge des batteries à zéro à chaque 1er janvier (approximation pessimiste).

— **Simulation pluriannuelle parcellaire** : ne simuler l'opération qu'une année sur N (par exemple) et interpoler les statistiques opérationnelles sur les années manquantes. Gain de temps de calcul d'un facteur N.

Remarquons que le premier point n'est qu'une extension au cas pluriannuel d'une hypothèse déjà faite sur le cas de base, vu qu'il faut bien initialiser les états de charge à une certaine valeur au démarrage de la simulation d'une année de référence. Ce n'est sans doute pas un problème pour un microréseau avec du stockage de faible autonomie (quelques heures), mais c'est une question ouverte en présence d'un large stockage saisonnier.

La deuxième proposition n'est pas, à notre connaissance, proposée par HOMER ou iHOGA, pour qui l'option "pluriannuelle" implique nécessairement un temps de calcul 20 à 30 fois plus long (le nombre d'années du projet). Cela semble pourtant un compromis intéressant, qui reste bien sûr à quantifier, entre les choix binaires de simuler une seule année de référence versus simuler exactement toutes les années.

Redimensionnement La deuxième cause d'évolution d'un microréseau est due non pas à la variation non contrôlée des données du problème, mais au choix volontaire de changer progressivement le système par redimensionnement et/ou ajout de composants. Notons qu'un lien existe entre ces deux idées : si l'on souhaite redimensionner, c'est sans doute pour *réagir à des changements exogènes*, par exemple augmenter les capacités de production pour réagir à une augmentation de la charge.

Pour prendre en charge le redimensionnement, il serait bien sûr possible de créer un nouveau simulateur ad hoc. Cependant, nous défendons l'idée qu'il est possible, au prix de quelques limitations et approximations, de réutiliser à l'identique le modèle de base dans une composition à peine plus complexe. Nous supposons pour ce faire que les éventuels redimensionnements se produisent à *quelques moments fixés* (via des paramètres) de la vie du projet, typiquement 1 ou 2. Appelons $T_{resize} \in]0, T_{proj}[$ le paramètre donnant l'année du redimensionnement, supposé unique pour la simplicité de l'explication (avec T_{proj} la durée globale du projet de microréseau). Ce cadre s'oppose à une modélisation, sans doute plus réaliste mais plus complexe à implémenter, où les redimensionnements se font au cours du projet lors d'évènements à date non fixée en réaction à des évènements (ex. : à la fin de vie d'un générateur, le remplacer par un autre

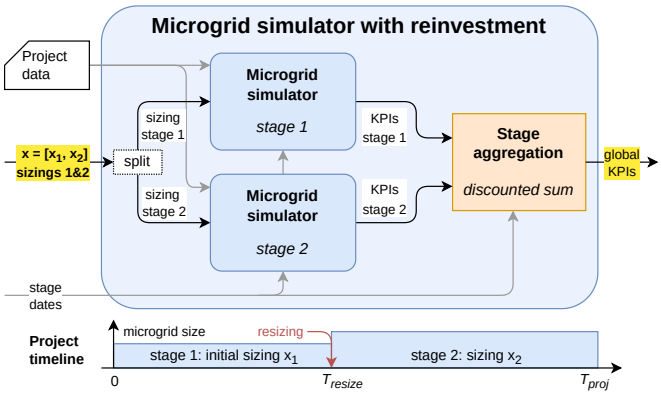

FIG. 5. Simulateur de microréseau avec un redimensionnement en cours de projet. L'implémentation se fait par deux appels au simulateur de base (Fig. 2) suivis d'une agrégation temporelle des indicateurs de performance (KPIs).

de taille différente). Nous proposons la modélisation Fig. 5 où le coût global du projet sur l'horizon T_{proj} se calcule par addition des coûts d'un sous-projet 1 sur la période $0 \rightarrow T_{resize}$ et d'un sous-projet 2 sur la période $T_{resize} \rightarrow T_{proj}$. Le coût global du projet (Net Present Cost) est alors :

$$NPC = NPC_1 + \frac{1}{(1+i)^{T_{resize}}} \times NPC_2 \qquad (1)$$

où i est le taux d'actualisation. Les coûts des sous-projets sont calculés par deux appels au simulateur de base, avec leur dimensionnement et leur durée respective (T_{resize} et $T_{proj} - T_{resize}$). Le facteur d'amortissement permet de convertir le coût présent du sous-projet 2 en coût présent du projet global. Une telle formule peut se généraliser sur tous les indicateurs de performances (KPIs Fig. 2) qui sont additifs, ou au moins décomposables (ex. : opérateur $\max$).

Un point technique est à noter dans cette approche : le calcul du NPC (au moins dans HOMER, iHOGA et Microgrids.X) inclut, comme coût négatif, la valeur résiduelle ("salvage value") de chaque composant à la fin du projet. Autrement dit, le modèle de la Fig. 5 suppose que le microréseau est virtuellement revendu à la date T_{resize} puis racheté à neuf immédiatement après. Enfin, ces nouveaux composants sont revendus à la date T_{proj} (pour une durée d'usage raccourcie $T_{proj} - T_{resize}$). Ce n'est bien sûr pas comme ainsi que le projet sera géré sur le terrain, mais la prise en compte des valeurs résiduelles permet de convertir l'investissement initial en un investissement fractionné en deux dont le total actualisé est supposé identique.

Cependant, nous avons étudié l'équivalence mathématique entre un calcul de NPC classique sans réinvestissement et un calcul avec redimensionnement à l'identique (1) et nous avons montré que la formule de valeur résiduelle habituelle (utilisée dans HOMER) ne donne pas l'équivalence, sauf lorsque le taux d'actualisation est nul ($i = 0$). Avec actualisation, l'écart de NPC peut aller jusqu'à 13%. Nous proposons dans un rapport dédié [36] une *formule alternative de valeur résiduelle* qui rétablit l'équivalence des NPC.

Relevons pour conclure que l'approche proposée, où les composants sont virtuellement revendus puis rachetés à neuf, a certaines limites. Par exemple on ne peut plus modéliser de vieillissement progressif (alors que c'est un des buts de l'approche pluriannuelle Fig. 4). Par contre, les autres variations long terme des données d'entrée (consommation, prix) se combinent naturellement avec notre modèle de réinvestissement. Dernier point, les simulations des différents sous-projets de notre proposition Fig. 5 peuvent être parallélisées si on néglige le couplage des états (comme discuté pour l'approche Monte Carlo §3.4).

4. CONCLUSIONS

Nous proposons des architectures modulaires de simulateurs de microréseaux dans un contexte d'optimisation boite noire et en lien avec notre logiciel de dimensionnement open source multi langage Microgrids.X [14]. Ces architectures permettent d'étudier des variantes du problème de base qui sont des enjeux scientifiques importants : prise en compte des incertitudes, des évolutions pluriannuelle et des réinvestissement en cours de projet. Par contre, nous n'avons pas, à ce stade, de réponse satisfaisante pour modéliser des microréseaux à la structure complexe (multi-énergies).

Nous avons montré comment la prise en compte de ces variantes peut se faire via des *compositions* qui réutilisent au maximum les blocs du simulateur de base. Cette réutilisation de modules est gage de pérennité et qualité du code. Elle se fait parfois au prix de quelques approximations que nous jugeons acceptables dans une phase de prédimensionnement.

En perspective, même si nous n'en avons pas traité ici, il est possible de combiner les propositions des Fig. 3 et 5 pour traiter du dimensionnement face à l'incertain dans une approche Multi-stage Stochastic Programming tel que [10, 19].

5. REMERCIEMENTS

La première version Julia de l'outil (https://github.com/Microgrids-X/Microgrids.jl) a été développée lors du stage d'Evelise de Godoy Antunes. Elle était financée en partie par le Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, par le Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq) et par le projet "Accélérer le dimensionnement des systèmes énergétiques avec la différentiation automatique" from GdR SEEDS (CNRS, France) en collaboration avec Benoît Delinchant du G2Elab.

6. RÉFÉRENCES

[1] T. Lambert, P. Gilman, and P. Lilienthal, "Micropower system modeling with HOMER," in *Integration of Alternative Sources of Energy*, F. A. Farret and M. G. Simões, Eds. John Wiley & Sons, Dec. 2005.

[2] J. Manwell, A. Rogers, G. Hayman, C. Avelar, J. McGowan, U. Abdulwahid, and K. Wu, "Hybrid2 - A Hybrid System Simulation Model : Theory Manual," University of Massachusetts, Tech. Rep., 2006.

[3] J. Segal, "Models of scientific software development," in *First International Workshop on Software Engineering for Computational Scienceand Engineering, Leipzig, Germany*, 2008. [Online]. Available : https://se4science.org/workshops/secse08/Papers/Segal.pdf

[4] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson, "Best Practices for Scientific Computing," *PLoS Biology*, vol. 12, no. 1, p. e1001745, 01 2014.

[5] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal, "Good enough practices in scientific computing," *PLOS Computational Biology*, vol. 13, no. 6, pp. 1–20, 06 2017.

[6] R. Dufo-López and J. L. Bernal-Agustín, "Design and control strategies of PV-diesel systems using genetic algorithms," *Solar Energy*, vol. 79, no. 1, pp. 33–46, 2005.

[7] B. Guinot, "Evaluation multicritère des technologies de stockage couplées aux énergies renouvelables : conception et réalisation de la plateforme de simulation odyssey pour l'optimisation du dimensionnement et de la gestion énergétique," Ph.D. dissertation, nNT : 2013GRENI027. [Online]. Available : https://tel.archives-ouvertes.fr/tel-00934515/

[8] C. H. P. Moraes, S. Nasr, B. Jacquet, A. El-Akoum, and P. Duclos, "Design to cost approach applied to isolated microgrids," in *CIRED 2021 - The 26th International Conference and Exhibition on Electricity Distribution*. Institution of Engineering and Technology.

[9] S. Pfenninger, J. DeCarolis, L. Hirth, S. Quoilin, and I. Staffell, "The importance of open data and software : Is energy research lagging behind ?" *Energy Policy*, vol. 101, pp. 211–215, feb 2017.

[10] D. Fioriti, R. Giglioli, D. Poli, G. Lutzemberger, A. Micangeli, R. Del Citto, I. Perez-Arriaga, and P. Duenas-Martinez, "Stochastic sizing of iso-

lated rural mini-grids, including effects of fuel procurement and operational strategies," *Electric Power Systems Research*, vol. 160, pp. 419–428, 2018.

[11] C. Jaouen, "Méthodologie de dimensionnement sur cycle de vie d'une distribution en courant continu dans le bâtiment : applications aux câbles et convertisseurs statiques DC/DC," Ph.D. dissertation, ENS Cachan, 2012. [Online]. Available : https://www.theses.fr/2012DENS0037

[12] Y. Krim, M. Sechilariu, F. Locment, and A. Alchami, "Global Cost and Carbon Impact Assessment Methodology for Electric Vehicles' PV-Powered Charging Station," *Applied Sciences*, vol. 12, no. 9, p. 4115, apr 2022.

[13] H. Radet, "Integrated design methods for distributed multi-energy systems under uncertainties," Ph.D. dissertation, Institut National Polytechnique de Toulouse (Toulouse INP), 2022. [Online]. Available : https://www.theses.fr/2022INPT0023

[14] P. Haessig, "Operational and economic simulation of Microgrid projects," 2022. [Online]. Available : https://pierreh.eu/Microgrids-X/

[15] E. de Godoy Antunes, P. Haessig, C. Wang, and R. Chouhy Leborgne, "Optimal Microgrid Sizing using Gradient-based Algorithms with Automatic Differentiation," in *ISGT Europe 2022, Novi Sad, Serbia*, 2022. [Online]. Available : https://hal.archives-ouvertes.fr/hal-03370004

[16] A. S. Siddiqui, C. Marnay, J. L. Edwards, R. Firestone, S. Ghosh, and M. Stadler, "Effects of carbon tax on microgrid combined heat and power adoption," *Journal of Energy Engineering*, vol. 131, no. 1, pp. 2–25, apr 2005.

[17] L. Welder, D. Ryberg, L. Kotzur, T. Grube, M. Robinius, and D. Stolten, "Spatio-temporal optimization of a future energy system for power-to-hydrogen applications in germany," *Energy*, vol. 158, pp. 1130–1149, sep 2018.

[18] S. Hodencq, M. Brugeron, J. Fitó, L. Morriet, B. Delinchant, and F. Wurtz, "OMEGAlpes, an Open-Source Optimisation Model Generation Tool to Support Energy Stakeholders at District Scale," *Energies*, vol. 14, no. 18, p. 5928, sep 2021.

[19] E. El Sayegh, N. Sadou, P. Haessig, S. Nasr, J. Bruschi, B. Jacquet, A. El Akoum, and H. Guéguen, "Towards avoiding microgrid design failures arising from an unrealistic operating strategy : an anticipative White Box model versus a responsive Black Box model," in *Jeunes Chercheurs en Génie Électrique (JCGE22), Le Croisic*, 2022. [Online]. Available : https://hal.archives-ouvertes.fr/hal-03702973

[20] C. D. Barley and C. B. Winn, "Optimal dispatch strategy in remote hybrid power systems," *Solar Energy*, vol. 58, no. 4, pp. 165–179, 1996.

[21] P. Haessig, J. J. Prince Agbodjan, R. Bourdais, and H. Guéguen, "Solar home 2020 : enrichissement du benchmark open source de gestion d'énergie avec entrées incertaines," in *SGE 2021, Nantes, France*, Jul. 2021. [Online]. Available : https://hal.archives-ouvertes.fr/hal-03366746

[22] B. Li, R. Roche, and A. Miraoui, "Microgrid sizing with combined evolutionary algorithm and milp unit commitment," *Applied Energy*, vol. 188, pp. 547–562, 2017.

[23] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython : The Best of Both Worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, mar 2011.

[24] S. Guelton, "Pythran : Crossing the Python Frontier," *Computing in Science & Engineering*, vol. 20, no. 2, pp. 83–89, mar 2018.

[25] J. Aycock, "A brief history of just-in-time," *ACM Computing Surveys*, vol. 35, no. 2, pp. 97–113, jun 2003.

[26] S. K. Lam, A. Pitrou, and S. Seibert, "Numba : a LLVM-based Python JIT compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC (LLVM'15)*. ACM, nov 2015.

[27] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia : A Fresh Approach to Numerical Computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, jan 2017.

[28] L. Shure, "Run Code Faster With the New MATLAB Execution Engine. Loren on the Art of MATLAB," 2016, accessed 2023-05-11. [Online]. Available : https://blogs.mathworks.com/loren/2016/02/12/run-code-faster-with-the-new-matlab-execution-engine/

[29] MathWorks, "How Solvers Compute in Parallel. Matlab Global Optimization Toolbox documentation," 2023, accessed 2023-05-11. [Online]. Available : https://fr.mathworks.com/help/gads/how-solvers-compute-in-parallel.html

[30] S. G. Johnson and J. Schueller, "NLopt : Nonlinear optimization library," Astrophysics Source Code Library, record ascl :2111.004, p. ascl :2111.004, 2021. [Online]. Available : https://ui.adsabs.harvard.edu/abs/2021ascl.soft11004J

[31] R. Dufo-López, "iHOGA version 3.4 User's manual," Universidad de Zaragoza, Tech. Rep., 2023. [Online]. Available : https://ihoga.unizar.es/Desc/iHOGA_User_manual.pdf

[32] O. Jolliet, M. Saade-Sbeih, S. Shaked, A. Jolliet, and P. Crettaz, *Environmental Life Cycle Assessment*, 1st ed. CRC Press, 2015.

[33] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Global sensitivity analysis. The Primer*. John Wiley & Sons, Ltd, 2008. [Online]. Available : http://www.andreasaltelli.eu/resources

[34] E. El Sayegh, P. Haessig, N. Sadou, J. Bruschi, B. Jacquet, S. Nasr, and H. Guéguen, "Assessing the impact of uncertainties on the techno-economic performance of microgrids," in *27th International Conference on Electricity Distribution (CIRED 2023), Rome*, 2023, paper n° 10762.

[35] A. Shapiro and A. Philpott, "A Tutorial on Stochastic Programming," 2007. [Online]. Available : https://sites.gatech.edu/alexander-shapiro/publications/

[36] P. Haessig, "Economic consistency of salvage value definitions," Tech. Rep. hal-04097092, 2023. [Online]. Available : https://hal.science/hal-04097092