Faisabilité de dimensionnement d'un flyback par optimisation SQP sur des critères dynamiques, avec détection de régime permanent

Lucas AGOBERT, Laurent GERBAUD, Benoît DELINCHANT
Univ. Grenoble Alpes, CNRS, Grenoble INP*, G2Elab, 3800 Grenoble, France

RESUME - Le dimensionnement de systèmes sur critères dynamiques est une problématique importante pour les concepteurs, que l'on peut résoudre en utilisant des algorithmes d'optimisation. Si l'on souhaite utiliser un algorithme d'optimisation avec gradients, l'enjeu est de parvenir à modéliser ces systèmes dynamiques, qui peuvent inclure des évènements, tout en calculant efficacement leurs gradients. Nous proposons pour cela un couplage entre la dérivation de code et le calcul formel, appliqué au système dynamique, au solveur et aux fonctions de post-Processing. Nous proposons d'illustrer ces aspects sur le dimensionnement d'un flyback. Les développements méthodologiques seront implémentés dans l'outil d'optimisation open source en Python : NoLoad[1].

Mots-clés—Dimensionnement, optimisation, SQP, systèmes dynamiques, extraction de caractéristiques

1. INTRODUCTION

En génie électrique, les concepteurs cherchent souvent à dimensionner des systèmes sur des critères dynamiques de manière à obtenir des performances souhaitées. Une façon classique de résoudre ces problèmes inverses est d'utiliser un algorithme d'optimisation d'ordre zéro (sans gradients), couplé à un simulateur vu comme une « boite noire ». Cependant, cela peut être très coûteux en temps de calcul si on augmente le nombre de paramètres de dimensionnement et de contraintes de performances.

Pour traiter plus de contraintes, une alternative est d'utiliser des algorithmes d'ordre un (exploitant les gradients). Cependant, pour obtenir les gradients d'outils numériques, les différences finies ne garantissent pas une bonne précision et peuvent elles aussi, devenir coûteuses en temps de calcul, avec l'augmentation du nombre de paramètres de dimensionnement.

Nous souhaitons montrer que l'on peut calculer efficacement les gradients même sur un problème dynamique complexe. Nous visons à faire un test de faisabilité de ce type d'approche avec le dimensionnement d'un dispositif réel d'électronique de puissance : un flyback.

Notre approche « boite blanche », par opposition à l'approche « boite « noire », ne s'applique pas aux solveurs dynamiques du commerce qui ne sont pas en open-source car nous avons besoin d'accéder au contenu du solveur pour le dériver. De plus, une particularité des systèmes dynamiques en génie électrique est d'inclure des systèmes d'état linéaires ou non, pouvant changer en cours de simulation, et nécessitant souvent de gérer des évènements [1]. Ces derniers sont modélisés en informatique par des branchements conditionnels (« if-then-else » ou « switch-case ») éléments indispensables d'algorithmes que l'on est en mesure de dériver [2]. C'est ce que nous avons mis en œuvre pour le dimensionnement de modèles statiques [3].

Dans cet article, le système dynamique est modélisé par des équations d'état qui changent en cours de simulation, avec une gestion d'évènements (commutation de semi-conducteurs). Nous couplons ce modèle avec un simulateur dynamique que nous avons implémenté de manière adaptée à une dérivation symbolique et pour y intégrer des fonctions de post-processing. Nous utiliserons la dérivation de code par surcharge d'opérateur (librairie JAX[2]) et un algorithme d'optimisation par programmation séquentielle quadratique (SLSQP de SciPy[3]). La partie 2 de cet article sera consacrée à la présentation du modèle du flyback. La partie 3 présentera la méthodologie de résolution du problème d'optimisation, notamment le calcul des gradients. La partie 4 sera dédiée au problème d'optimisation avec 2 sous-parties : la définition du cahier des charges et les résultats obtenus. Une 5ème partie conclura cet article.

2. PRESENTATION DU CAS D'ETUDE

Le système présenté dans cet article est une alimentation à découpage de type flyback. Combinée avec un redresseur AC-DC, elle forme un convertisseur qui permet d'alimenter à partir du réseau monophasé 240V AC, une charge alimentée en continu. Son schéma électrique est présenté sur la figure 1 [4][5].

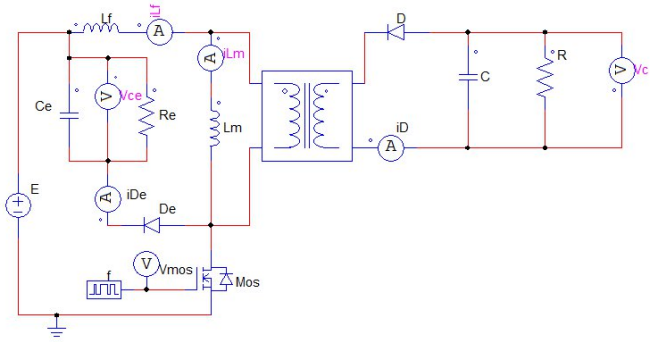

Fig. 1. Schéma électrique du flyback

Dans notre cas, on cherche à faire une simulation dynamique dont les variables d'état sont : *vce* la tension aux bornes de l'écrêteur, *vc* la tension aux bornes de la résistance R

[1] https://gricad-gitlab.univ-grenoble-alpes.fr/design_optimization/NoLoad_v2
[2] https://jax.readthedocs.io/en/latest/
[3] https://pypi.org/project/scipy/

***Institute of Engineering Univ. Grenoble Alpes**

en sortie, *ilm* le courant dans l'inductance magnétisante Lm, et *ilf* le courant dans l'inductance de fuite Lf. Les variables de sortie sont les courants dans les diodes D et De, respectivement nommés id et ide.

Ce système d'EDO présente plusieurs configurations qui peuvent changer suivant la commutation du transistor MOS T (commande C) ou du blocage des diodes D et De (cf figure2).

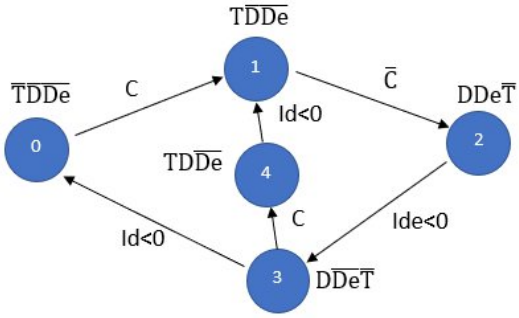

Fig. 2. Graphe des états du flyback

Lors de la boucle d'optimisation, on simulera le système jusqu'à la détection de son régime permanent et on cherchera à extraire des grandeurs des courbes obtenues : le minimum et le maximum sur une période de chaque variable d'état, et les valeurs moyenne et efficace sur une période d'ilm. Ces grandeurs serviront à établir les contraintes de dimensionnement de notre système.

Dans notre cas, plusieurs problèmes se posent quant à la résolution de ce problème d'optimisation avec l'état de l'art :

- Les simulateurs existants (par exemple PSIM, Portunus, Saber, Simplorer, etc.) même s'ils sont pilotables, sont « fermés » aux concepteurs dans le sens où ils ne sont pas dérivables.
- Un algorithme d'optimisation stochastique se limite à des problèmes avec peu de contraintes, sinon les temps de calcul sur CPU explosent.
- Un algorithme déterministe avec gradients permet de traiter des problèmes avec plus de contraintes (comme notre cas). Cependant, le problème du calcul des dérivées se pose, et la principale méthode utilisée est celle des différences finies, qui entraîne des problèmes de précision et de temps de calcul.
- Les simulateurs ne permettent de faire des simulations qu'avec un temps final fixe comme condition d'arrêt. Or, dans notre cas, le temps d'arrêt de notre simulation peut être différent en changeant les valeurs des composants électroniques du système. Nous devons simuler le régime transitoire du système, ce qui engendre du calcul inutile et pose le problème du stockage des données.
- La capacité à extraire des grandeurs d'une simulation dynamique (minimum, maximum, moyenne et valeur efficace d'une variable d'état), ainsi que leurs dérivées respectives par rapport aux entrées d'optimisation P.
- Le flyback est un système dynamique qui peut changer de configuration. Ceci pose la question de la détection des évènements permettant de passer d'une configuration à une autre, tel qu'explicité dans la figure 2.

3. PROBLEME D'OPTIMISATION

3.1. *Boucle d'optimisation*

Comme le problème d'optimisation se fait sur un système dynamique, la boucle d'optimisation est plus complexe qu'elle ne l'aurait été pour un système statique. Elle est représentée sur la figure 2.

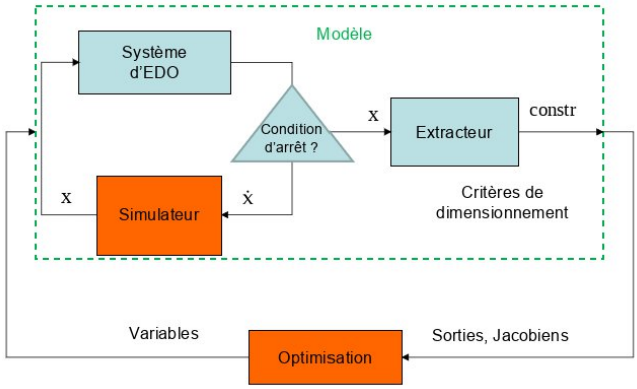

Fig. 3. Schéma représentant la boucle d'optimisation

Elle comporte un modèle qui fournit des contraintes d'optimisation (nommées constr sur la figure 2) ainsi que leurs dérivées par rapport aux entrées d'optimisation (appelées P) à l'algorithme d'optimisation SQP de Scipy. Ce modèle est composé de plusieurs éléments :

- Un système d'EDO décrit par le concepteur pour fournir les dérivées temporelles des variables d'état du système.
- Un simulateur temporel qui résout les systèmes d'EDO pour fournir les valeurs du vecteur d'état au cours de la durée de simulation.
- Une condition d'arrêt de la simulation temporelle autre que le temps final fixe : la détection de régime permanent.
- Un extracteur de caractéristiques qui va calculer à partir de la simulation temporelle les contraintes nécessaires au problème d'optimisation.

Cette procédure existe déjà dans des logiciels d'optimisation. Cependant, la simulation temporelle d'une part et l'extracteur de caractéristiques associé à l'algorithme d'optimisation d'autre part sont réalisés dans des logiciels différents (comme par exemple Portunus pour la simulation, et Cades pour l'optimisation). Les différentes parties de la méthodologie présentée dans cet article sont réalisées dans la même librairie Python NoLOAD, pour avoir un couplage plus fort et améliorer les performances par rapport à l'état de l'art.

Une autre problématique de cette méthodologie est l'obtention des gradients pour chacune des parties du modèle explicitées ci-dessus. La méthode des différences finies ayant parfois des problèmes de précision et en temps de calcul, une autre méthode est utilisée : la différentiation automatique (ou dérivation de code) qui sera présenté dans le paragraphe suivant.

3.2. *La dérivation de code*

La dérivation de code est une méthode qui calcule la dérivée exacte d'une fonction en un point donné [2]. Elle se base sur la composition de la fonction à dériver en des fonctions mathématiques élémentaires (exponentielle, logarithme, trigonométriques) dont la dérivée est connue. La formule de dérivation d'une fonction composée donnée dans l'expression (1) est alors utilisée.

$$\frac{df(g(x))}{dx} = \frac{df}{dg(x)} * \frac{dg(x)}{dx} \quad (1)$$

La dérivation de code existe depuis plusieurs décennies. Il existe depuis quelques années des bibliothèques Python (ou dans d'autres environnements) qui implémentent la dérivation de code en utilisant 2 techniques : la transformation du code source et la surcharge d'opérateurs. La transformation du code source crée un code dérivé de la fonction à dériver, tandis que la surcharge d'opérateurs calcule les dérivées en temps réel, en utilisant les dérivées des fonctions mathématiques élémentaires et des opérateurs (addition, multiplication, ...) programmées dans des classes. Nous choisissons la librairie Python JAX, qui est implémentée avec la surcharge d'opérateurs, car elle comporte la fonction JIT (Just-In-Time) qui permet de rendre plus rapide l'exécution du code. Cela sera très utile dans la boucle d'optimisation car les simulations temporelles à effectuer seront nombreuses. De plus, elle permet au programmeur de définir lui-même la fonction « dérivée » d'une fonction Python, lorsqu'une autre alternative de dérivation plus judicieuse existe.

3.3. Le simulateur d'EDO

Pour résoudre des systèmes d'EDO, il est primordial de créer un algorithme de résolution d'EDO sous la forme d'une fonction Python. Un algorithme Runge-Kutta 45 à pas adaptatif est utilisé, se basant sur le système d'EDO fourni par le concepteur sous forme de l'équation (2). Les coefficients de cet algorithme Runge-Kutta sont les mêmes que ceux utilisés dans la fonction ode45 de Matlab, c'est-à-dire établis par Domand-Price[6].

$$\begin{cases} \frac{dx}{dt} = f(x,t,P) \\ x(t0) = x0 \end{cases} \quad (2)$$

Pour obtenir la dérivée de cet algorithme de résolution d'EDO, la dérivation de code pourrait être directement appliquée. Or, comme expliqué dans [7], l'application de la dérivation de code à ce type d'algorithme produit des difficultés liées à la persistance du graphe de calcul : la dérivation du simulateur d'EDO par rapport au pas de calcul ne fonctionne pas. Nous devons donc définir un code « dérivé » de cet algorithme, pour que la dérivée soit faite uniquement par rapport aux entrées d'optimisation P, et non par rapport à d'autres paramètres comme le pas de calcul ou l'objet classe qui définit notre système. Il en résulterait sinon des indéterminations mathématiques qui rendraient impossible la résolution du problème d'optimisation. Il faut ainsi rechercher une dérivation formelle de cet algorithme.

Ainsi la dérivation d'un simulateur d'EDO peut consister à résoudre le système d'EDO « dérivée » présenté dans l'équation (3) [8], par extension du système d'état.

$$\begin{cases} \frac{d}{dt}\left(\frac{dx}{dP}\right) = \frac{df(x,t,P)}{dP} \\ \frac{dx}{dP}(t0) = \frac{dx0}{dP} \end{cases} \quad (3)$$

Nous définissons donc le code dérivé du simulateur d'EDO comme un autre simulateur d'EDO permettant de résoudre le système présenté en (2).

Lors de l'exécution du problème d'optimisation, les valeurs des variables d'état au cours du temps ne sont pas stockées pour économiser de la mémoire.

3.4. La détection d'évènements pour le changement de configuration

Ce système présente différentes topologies qui induisent chacune un système d'EDO différent. Notre simulateur d'EDO devra être capable de détecter les évènements qui permettent ces changements de configuration. Les évènements peuvent être liés :

- à la commande qui changent de valeur : ces instants de commutations peuvent être connus à l'avance.
- au franchissement de seuil d'un variable d'état (dans notre système, cela correspond aux courants dans les diodes qui s'annulent, ce qui bloquent ces dernières).

Les évènements liés à la commande sont détectés avec la méthode de la prédiction : le pas de temps est adapté pour que la simulation tombe exactement sur un changement d'état.

Les évènements liés au franchissement de seuil sont détectés avec la méthode de la « correction » : lorsqu'une variable d'état dépasse un seuil, une interpolation linéaire est effectuée pour que le vecteur d'état se cale parfaitement sur l'instant temporel correspondant au seuil.

3.5. La condition d'arrêt : détection de régime permanent

Une simulation temporelle du flyback avec des paramètres d'optimisation donnés comporte un régime transitoire et un régime permanent. L'objectif est d'arrêter la simulation lorsque le régime permanent sera atteint, et après avoir simulé le système une période supplémentaire, pour calculer les valeurs moyenne et efficace du courant dans l'inductance magnétisante Lm. Le principe est explicité sur la figure 4.

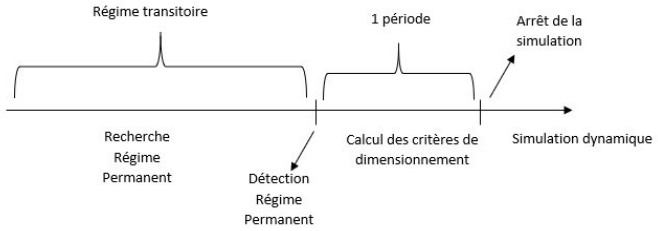

Fig. 4. Principe de la simulation dynamique avec détection de régime permanent Schéma représentant la boucle d'optimisation

Pour détecter le régime permanent, comme ici les valeurs des grandeurs d'état ne sont pas stockées, nous proposons de calculer les valeurs minimale et maximale de chaque grandeur d'état sur deux groupes de même nombre de périodes (choisi arbitrairement) successives. Si tous les minimums et maximums de chaque variable d'état sont les mêmes à une précision près sur les deux groupes de périodes, alors le régime permanent est considéré comme atteint. Le fait de prendre des groupes de périodes, et non des périodes « simples » permet d'éviter de tomber sur des « faux » régimes permanents, où les valeurs observées sont proches d'une période à l'autre mais où le régime est encore transitoire (problème que nous avons rencontré).

3.6. L'extracteur de caractéristiques

Les grandeurs « types » à extraire de la simulation temporelle sur une période T pour une variable d'état sont : la valeur minimale et la valeur maximale qui ne comportent pas d'expression analytique, ainsi que les valeurs moyenne et

efficace qui sont définies respectivement par les expressions présentées dans [9]. Comme aucune donnée n'est stockée en cours de simulation, ces valeurs sont calculées en cours de simulation, sur la dernière période de simulation (figure 4).

Pour déterminer toute valeur minimale (respectivement maximale), nous comparons à chaque pas de simulation toute variable d'état calculée avec sa valeur minimale (respectivement maximale) déterminée à l'itération précédente.

Les valeurs moyenne et efficace s'obtenant analytiquement avec une intégrale, elles se calculent numériquement avec la méthode des trapèzes avec le pas de résolution de l'EDO, comme décrit en [9].

Pour obtenir les dérivées des grandeurs extraites de la simulation par rapport aux paramètres d'optimisation P, suivant les cas, nous utilisons soit la dérivation de code, soit des formules mathématiques si celles-ci sont connues.

Les dérivées des valeurs minimale et maximale par rapport à P sont obtenues avec la dérivation de code car elles ne disposent pas de formule mathématique analytique.

4. PROBLEME D'OPTIMISATION

4.1. Définition du cahier des charges

Les variables d'entrée à dimensionner dans le problème d'optimisation sont essentiellement les valeurs des composants électroniques et les caractéristiques du transformateur (rapport de transformation, nombre de spires au primaire, coefficient de couplage) (cf. tableau 1).

Tableau 1. Liste des variables à optimiser et leurs intervalles de variation

Nom de la variable d'entrée (unité)	**Bornes**
m : ratio du transformateur (/)	[2,3]
n1 : nombre de spires au primaire (/)	[15,150]
Ce : capacitance de l'écrêteur (F)	$[10^{-8},10^{-5}]$
Re : résistance de l'écrêteur (ohm)	$[10^{3},10^{4}]$
C : capacitance du secondaire (F)	$[10^{-4},3*10^{-3}]$
Lm : inductance magnétisante (H)	$[2*10^{-4},1.5*10^{-3}]$
k : coefficient de couplage (/)	[0.8,0.99]

La fonction objective de ce problème d'optimisation consiste à limiter les pertes du transformateur. Parmi les contraintes d'inégalité, l'encombrement du système est limité (volume occupé par le transformateur), et le système doit être à la limite entre les conductions continue et discontinue. Les sorties d'optimisation sont données dans le tableau 2.

Tableau 2. Liste des contraintes à satisfaire et leurs intervalles à respecter

Nom de la contrainte d'inégalité (unité)	**Intervalles**
Fonction objective : Pertes Totales du transformateur (W)	/
Acuivre : surface totale d'enroulement de cuivre (mm^2)	$[0,10^{-3}]$
deltaV : limitation de tension dans l'inductance Lf (V)	[-30, -20]
IDmax : courant maximal dans la diode D	[0,14]
ilmin : courant minimal dans l'inductance Lm (A)	[0.01,0.05]
volumetransfo : volume du transformateur (mm^3)	[0,7630]
rendement : rendement du transformateur (/)	[0.8,1]

4.2. Résultats du problème d'optimisation

Les résultats sont obtenus avec l'algorithme d'optimisation SLSQP (librairie SciPy) avec une précision sur la fonction objectif de 10^{-5}. L'optimisation a convergé après 15 itérations en 18 secondes. Les résultats d'optimisation sont présentés dans le tableau 3 :

Tableau 3. Résultats de l'optimisation

Variable optimisée	**Résultat**	**Contraintes**	**Résultat**
m	2.75	PertesTotales	13.02
n1	98	Acuivre	$2.59*10^{-4}$
Ce	$1.83*10^{-7}$	deltaV	-20
Re	10^{4}	IDmax	6.73
C	10^{-4}	ilmin	0.05
Lm	$2.47*10^{-4}$	volumeTransfo	4369
k	0.80	rendement	0.87

5. CONCLUSIONS

Nous avons montré dans cet article différents aspects de l'optimisation à gradients de systèmes dynamiques. Le premier résultat remarquable est la possibilité d'exploiter les gradients même pour des EDO à évènements. Nous avons également intégré des fonctions avancées à la simulation pour automatiser les itérations d'optimisation (recherche de régime permanent) et ajout de critères dynamiques au dimensionnement (valeur maximale, etc.). Nous avons également montré l'intérêt de coupler diverses techniques de dérivation pour atteindre de bonnes calculatoires : dérivation formelle pour le solveur et les formules de post-Processing, et dérivation de code pour les modèles physiques qui doit pouvoir s'adapter rapidement à n'importe quel autre dispositif à dimensionnement, notamment les convertisseurs DAB.

6. REFERENCES

[1] Breitenecker, Felix. "Development of simulation software-from simple ode modelling to structural dynamic systems." Proceedings of the 22nd European Conference on Modelling and Simulation (ECMS 2008).

[2] Gebremedhin, Assefaw H., et Andrea Walther. « An Introduction to Algorithmic Differentiation ». WIREs Data Mining and Knowledge Discovery, vol. 10, no 1, janvier 2020. DOI.org (Crossref), https://doi.org/10.1002/widm.1334.

[3] Agobert, Lucas, et al. « NoLOAD, Open Software for Optimal Design and Operation using Automatic Differentiation ». OIPE2020 - 16th International Workshop on Optimization and Inverse Problems in Electromagnetism, 2021. HAL Archives Ouvertes, https://hal.archives-ouvertes.fr/hal-03352443.

[4] Ferrieux, Jean-Paul, and François Forest. Alimentation à découpage, convertisseurs à résonance: principes, modélisation, composants. Dunod, 2022.

[5] Larouci, Chérif. Conception et optimisation de convertisseurs statiques pour l'électronique de puissance Application aux structures à absorption sinusoïdale. Institut National Polytechnique de Grenoble - INPG, 13 mai 2002. theses.hal.science, https://theses.hal.science/tel-00491438.

[6] Shampine, Lawrence F. « Some Practical Runge-Kutta Formulas ». Mathematics of Computation, vol. 46, no 173, 1986, p. 135-50. www.ams.org, https://doi.org/10.1090/S0025-5718-1986-0815836-3.

[7] Fischer, Vincent. Composants logiciels pour le dimensionnement en génie électrique : application à la résolution d'équations différentielles. Grenoble INPG, 1 janvier 2004. theses.fr, https://www.theses.fr/2004INPG0092.

[8] Kiehl, Martin. « Sensitivity Analysis of ODEs and DAEs — Theory and Implementation Guide ». Optimization Methods and Software, vol. 10, no 6, janvier 1999, p. 803-21. DOI.org (Crossref), https://doi.org/10.1080/10556789908805742.

[9] Gerbaud, Laurent, et al. « Obtaining the most exact Jacobian of the time modelling of power electronics structures for gradient optimisations ». COMPEL – The international journal for computation and mathematics in electrical and electronical engineering, vol. 41, n° 6, janvier 2022, p. 2096-108. Emerald Insight. https://doi.org/10.1108/COMPEL-10-2021-0398

[10] Press, William H., et al. Numerical recipes. Cambridge University Press, London, England, 1988.