Université Abdelmalek Essaadi

Faculté des Sciences et Techniques
Tanger

2019/2020

Projet de Fin d'Études

Master Sciences et Techniques

Option : Génie Énergétique

Titre :

Estimation des paramètres des cellules photovoltaïques

Présenté par :

Youssef Kharchouf

Date de soutenance : Juin 2020

Devant le Jury :

Nom et Prénom	*Établissement*	*Qualité*
		Président
		Examinateur
		Encadrant Externe
Dr. Adil Chahboun	**FST de Tanger**	***Encadrant de la FST***

PFE effectué au :
(Laboratoire ou Société ou Organisme)

LOGO du Laboratoire ou de la Société ou de l'organisme, ...

Résumé

Les cellules solaires photovoltaïques sont généralement modélisées avec un circuit électrique comprenant un certain nombre de composants localisés. Les paramètres de ces composants déterminent la précision de ces modèles mais ne sont généralement pas fournis par les fabricants. Cependant il est possible d'estimer ces paramètres avec la caractéristique I-V de la cellule ou du module PV en se basant sur des techniques numériques ou analytiques. Les méthodes évolutionnaires tel que l'évolution différentielle prennent le problème d'un point de vue d'optimisation mathématique dont la fonction objectif à minimiser étant l'erreur entre la caractéristique calculée du modèle et les données expérimentales. Dans ce travail nous analysons la performance de l'évolution différentielle appliquée aux modèles simple et double diode avec la fonction W de Lambert. De plus, on propose une stratégie métaheuristique pour déterminer les paramètres de l'évolution différentielle pour optimiser le temps de convergence.

Abstract

Solar PV cells are generally represented by a lumped-element model with a set number of components which are represented by their parameters. These parameters are not readily available in manufacturer data-sheets despite being crucial to the accuracy of the model. A possible approach is to estimate these parameters using the Current-Voltage characteristic of a cell using numerical or analytic techniques. Metaheuristics such as Differential Evolution take the approach of an optimization problem with the objective function to be minimized being the error with experimental Current-Voltage data. This work is an analysis of the performance of Differential Evolution applied to the single and double diode models with the Lambert W function on various solar cell technologies. A further metaheuristic strategy to determine the parameters of differential evolution is proposed using a higher-order differential evolution in order to speed convergence. A tool implementing these methods has been developed using the Python programming language.

Nomenclature

CR	Taux de croisement	
D	Dimensions de l'espace de recherche	
E_g	Énergie de gap	eV
F	Facteur de mutation	
I_0	Courant de Saturation	µA
I_D	Courant de la diode	A
I_{PV}	Photocourant	A
N_P	Taille de la population	
R_p	Résistance shunt	Ω
R_s	Résistance série	Ω
a	Facteur d'idéalité	
k	Constante de Boltzmann	$1.3806503 \times 10^{-23}$ J K^{-1}
q	Charge Élémentaire	$1.6021764 \times 10^{-19}$ C
T	Température de la cellule/du module	K
$\vec{V}_{base}$	Vecteur de base	
$\vec{T}_i$	Vecteur d'essai	
$\vec{M}_i$	Vecteur mutant	
V_t	Voltage Thermique	V
ABC	Colonies d'abeilles artificielles (*Artificial Bee Colony Optimization*)	
CIABC	Chaotic Improved Artificial Bee Colony	
CWOA	Chaotic Whale Optimization Algorithm	
ED	Évolution Differentielle (*Differential Evolution*)	
ED3P	Évolution différentielle à trois points	
ELPSO	Enhanced Leader Particle Swarm Optimization	
GA	Algorithme Génétique	
GSA	Gravitational Search Algorithm	
PSO	Optimisation par Essaims Particulaires (*Particle Swarm Optimization*)	
RMSE	Racine de l'erreur quadratique moyenne (*Root Mean Squared Error*)	
SA	Recuit Simulé (*Simulated Annealing*)	
SQ	Formalisme Shockely-Queisser	

Table des matières

Introduction Générale

L'épuisement des énergies fossiles et leur nature non-durable a conduit à la croissance rapide des énergies renouvelables comme sources alternatives d'énergie. L'énergie solaire est l'une de ces sources renouvelables les plus répandues et a prouvé son utilité dans plusieurs domaines d'application grâce à sa nature quasiment inexhaustible. Les technologies photovoltaïques en particulier ont fait l'objet intense d'étude de la part de la communauté scientifique depuis la première cellule en silicium cristallin à 6 % de Chapin et al. [1] en 1954. Ceci a entraîné une croissance considérable de l'industrie photovoltaïque. En fait, la capacité de production en PV a dépassé les 150 GW [2] en 2019 (figure 1), ce qui correspond une croissance d'environ 100 GW depuis 2014. Les décennies de recherche depuis la cellule de Chapin et al. visent généralement l'amélioration des cellules solaires sur trois axes : *(i)* en explorant les différentes architectures possibles des cellules, *(ii)* en développant des matériaux qui permettent d'augmenter l'efficacité de conversion de l'énergie solaire incidente ou *(iii)* en réduisant le coût du processus de production. La première génération des technologies PV se basait sur des wafers de silicium comme matériau actif de conversion. La deuxième génération remplace le silicium avec des semi-conducteurs à couches minces, qui, parmi d'autres avantages, réduisent considérablement les besoins en matières premières. Les troisièmes et futures générations comprennent les cellules organiques, Perovskite et les Quantums Dots etc. et présentent des pistes d'investigations pour réduire davantage les coûts de production et augmenter l'efficacité de conversion.

Le principe de fonctionnement des cellules photovoltaïques a été discuté en profondeur dans la littérature [3, 4, 5]. Les photons à énergie supérieure au bandgap du matériau semi-conducteur sont absorbés en transférant leur énergie aux électrons dans la bande de valence, ce qui leur permet de passer vers la bande de conduction en laissant un "trou" derrière. Le rôle du champ électrique présent dans la zone de déplétion des jonctions P-N est de séparer ces paires électron-trou et leur permettre de circuler à travers une charge extérieure. Toutefois, il y aurait toujours un plafond sur ce mécanisme de conversion "lumière $\rightarrow$ électricité" pour les cellules mono-jonction selon la limite Schockley-Queisser (SQ) [6]. Le modèle SQ postule que tous les photons à énergie supérieure au gap E_g sont absorbés et que la totalité des recombinaisons qui surviennent sont strictement l'inverse du processus d'absorption, c'est-à-dire des recombinaisons radiatives qui réémettent des photons.
Cependant, la performance d'une cellule réelle sera toujours inférieure à la limite SQ (figure 2). Les mécanismes idéaux postulés par le formalisme de SQ sont affectés dans la pratique par plusieurs phénomènes imprévus qui parviennent des propriétés des matériaux utilisés. À titre d'exemple, la présence des défauts dans un semi-conducteur entraîne des recombinaisons non-radiatives où l'énergie des électrons est libérée en phonons dans la structure cristalline et non pas en photon, ainsi que la recombinaison Auger où un autre électron reçoit cette énergie. Les difficultés qui limitent les technologies PV nécessitent une meilleure compréhension des phénomènes physiques complexes impliqués, et l'étude de la performance des cellules par rapport au modèle SQ donne des indices sur la nature des gains en performance qu'une technologie particulière a le potentiel de réaliser.

La modélisation des cellules solaires s'avère nécessaire par conséquent. Elle est essentielle pour la conception, l'analyse et l'estimation de la performance des cellules solaires. Elle permet d'effectuer l'analyse des propriétés électriques pendant leur phase de développement, l'émulation des systèmes PV pour la prédiction des rendements énergétiques [7], le contrôle de qualité des cellules en phase de production et l'étude des effets de dégradation pendant la phase d'opération [8, 9]. Bien qu'il y ait des méthodes de simulation des phénomènes physiques dans la cellule comme celles qui reposent sur la dynamique moléculaire, la théorie fonctionnelle de la densité ou tout simplement des méthodes

numériques appliquées aux équations des semi-conducteurs (équation de Poisson et équations de continuité), etc. Dans ce travail nous nous intéressons à l'approche des circuits équivalents à éléments localisés qui modélisent le comportement I-V de la cellule.

La modélisation d'éléments localisés permet de décrire le comportement des systèmes physiques dispersés spatialement avec un ensemble d'éléments localisés dont chacun représente un phénomène physique particulier. L'analogie électrique dans les problèmes de transferts de chaleur est un exemple de cette méthode où une résistance localisée dans le modèle pourrait représenter la résistance thermique spatialement distribuée selon l'épaisseur de la paroi concernée. Dans le cas des cellules PV, chaque élément du circuit équivalent représente aussi un phénomène physique spécifique dans la cellule (e.g. une résistance en série modélise les pertes ohmiques causées par la résistance intrinsèque au semi-conducteur utilisé). Puisque la caractéristique I-V dépend entièrement des paramètres associés aux éléments localisés du circuit. Il s'agirait donc d'essayer de minimiser l'erreur entre la courbe caractéristique du modèle et celle de la cellule réelle en jouant sur les valeurs des paramètres. Pour ce faire, il existe deux approches principales. La première est l'approche analytique. Elle consiste à utiliser les données sur les points clés de la courbe caractéristique (tension circuit ouvert, courant court-circuit, la pente de la courbe en ces points, ou encore le point de puissance maximale etc) et à effectuer certaines simplifications pour concevoir des formules approximatives, mais rapide à calculer. Toutefois, les simplifications effectuées peuvent conduire à des résultats imprécis ou non physiques (e.g. résistance négative). De plus, le fait que cette approche utilise les données des quelques points seulement la rend vulnérable aux bruits de mesures. La deuxième approche est l'extraction numérique. On se retrouve avec un problème d'optimisation, dont la fonction objectif à extrémiser (minimiser) est l'erreur entre le modèle et la courbe expérimentale de la cellule réelle. L'algorithme utilisé pour l'optimisation en soi pourrait être déterministe comme dans le cas des méthodes de Newton-Raphson, Levenberg–Marquardt etc. ou des algorithmes stochastiques/métaheuristique comme les techniques évolutionnaires. Ces dernières comprennent l'algorithme génétique (GA) , l'Optimisation par Essaims Particulaires (PSO), Recuit Simulé (SA) etc. Les méthodes déterministes imposent généralement des critères de convexité, différentiabilité (et par conséquent continuité) de la fonction objectif. Bien qu'elles soient efficaces en termes d'optimisation locale avec l'accès aux6 données de gradient, elles sont très susceptibles à se piéger dans des extremums locaux, ce qui limite leurs capacités d'optimisation globale. Par contre les métaheuristiques n'ont pas d'exigences sur la fonction objectif, et le choix des conditions initiales appropriées les rend plus robustes envers les pièges d'extremums locaux.

Dans ce travail, après avoir discuté le concept de modélisation par circuits électriques des cellules PV, nous présenterons la technique de l'évolution différentielle proposée par Storn et Price en 1995 [10]. Comme les autres techniques évolutionnaires, elle consiste à générer une population de solutions sur laquelle on effectue des opérations de mutation et de croisement pour sélectionner les solutions qui constitueront la génération suivante. Ensuite nous allons synthétiser l'ED avec la modélisation par circuits électriques en utilisant la fonction W de Lambert. Nous analysons la performance de l'ED en la comparant avec les autres techniques disponibles dans la littérature avant de conclure en proposant une stratégie pour mieux sélectionner les paramètres de l'ED et optimiser la convergence et le temps de calcul.

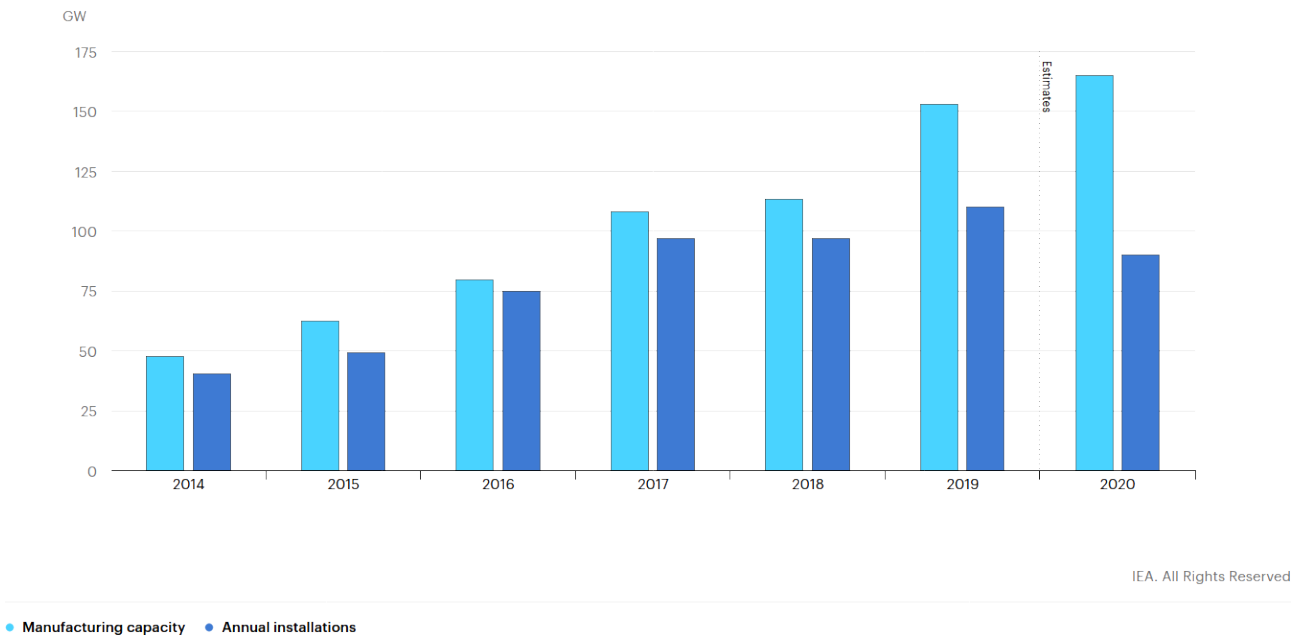


FIGURE 1 – Fabrication et demande des modules solaires photovoltaïques, 2014-2020. Source : IEA analysis based on Paula Mints (2020), The Solar Flare, SVP Market Research, San Francisco, CA [2]

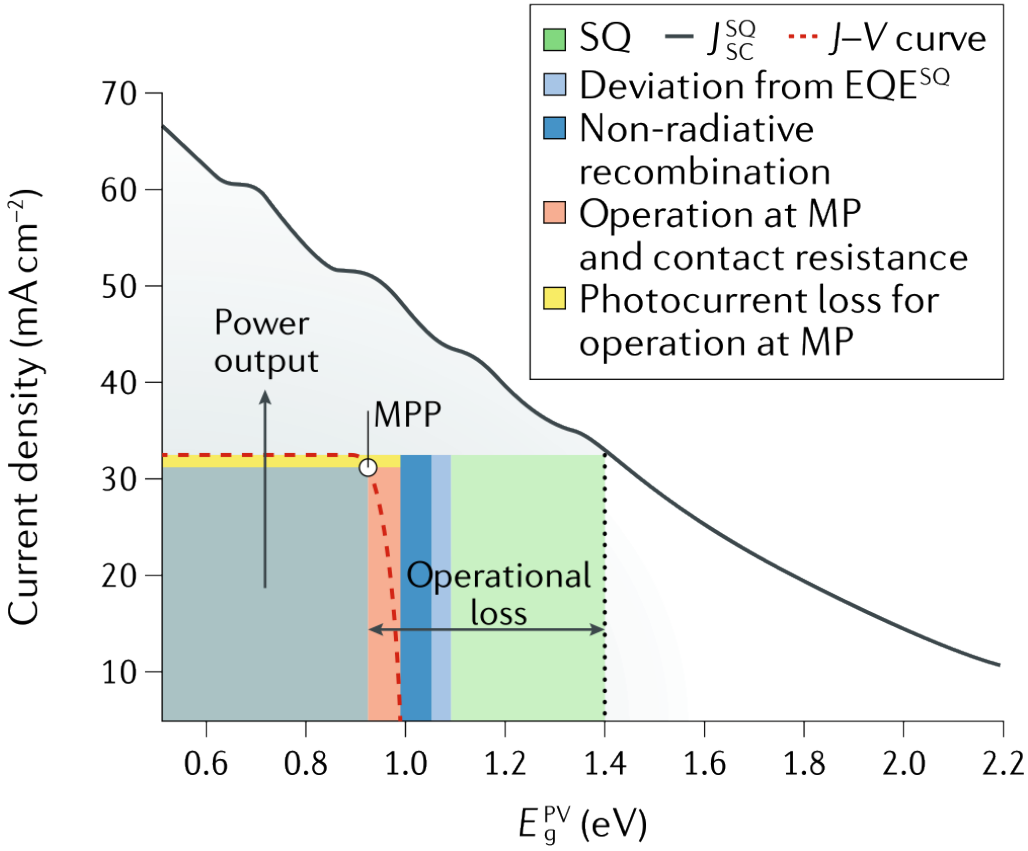


FIGURE 2 – Différence entre le formalisme de SQ et une cellule réelle. Le photo-courant maximal CC à la limite de Schockley-Queisser est tracé en fonction du gap (E_g^{PV}). La ligne pointillée en rouge indique la caractéristique J-V de la cellule [11]

Chapitre 1

Modèles à circuits électriques pour les cellules PV

1.1 Introduction

Dans ce chapitre, nous parlerons des circuit électriques à éléments localisés qui sont utilisés pour la modélisation des cellules PV. Plus précisément, on discutera les modèles simple et double diode, les rôles et l'influence de chaque paramètre sur le modèle et finalement les techniques utilisées dans la littérature pour l'identification et l'estimation de ces paramètres.

1.2 Généralités

Fondamentalement, les cellules PV se composent de deux couches de semi-conducteurs à dopages différents, avec la jonction de ces deux couches étant exposée à la lumière incidente. En effet, les électrons dans la bande de conduction sont capables d'être transférés vers la bande de valence tant que l'énergie du photon incident $E = h\nu$ est supérieure à la largeur de la bande interdite $E_g = E_c - E_v$. Toute l'architecture d'une cellule quelconque vise à profiter le plus que possible de la différence de tension engendrée par les excitons séparés par le champ électrique dans la zone de déplétion (ou zone de charge d'espace) (figure 1.1). Par exemple, les contacts supérieurs sont des oxydes transparents conductifs (dit couche fenêtre) pour laisser passer le plus grand nombre de photons possible vers la couche absorbante de la cellule. Pour quelques cellule en couche mince, le contact arrière se compose d'une couche réfléchissante (ZnO et Ag ou Al) qui renvoie la lumière vers la couche absorbante pour minimiser davantage les pertes en lumière transmise [12].

En fait, le comportement électrique d'un telle cellule en l'absence de la lumière est identique à celui d'une diode PN classique dont la caractéristique est décrite par l'équation de Shockley [14] (équation 1.1). Dans cette formule, le I_0 est le courant de saturation de la diode, q la charge élémentaire e^-, a le facteur d'idéalité, k la constante de Boltzmann et T la température de la cellule. Parfois on

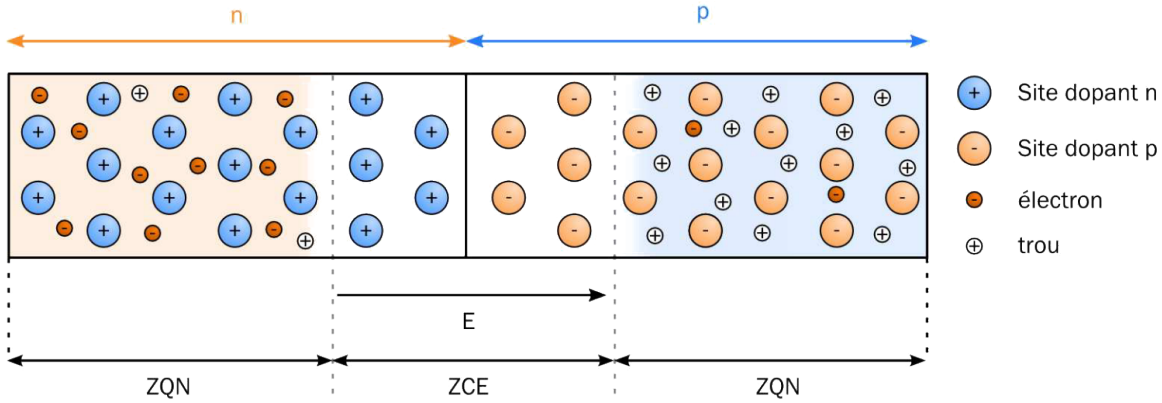
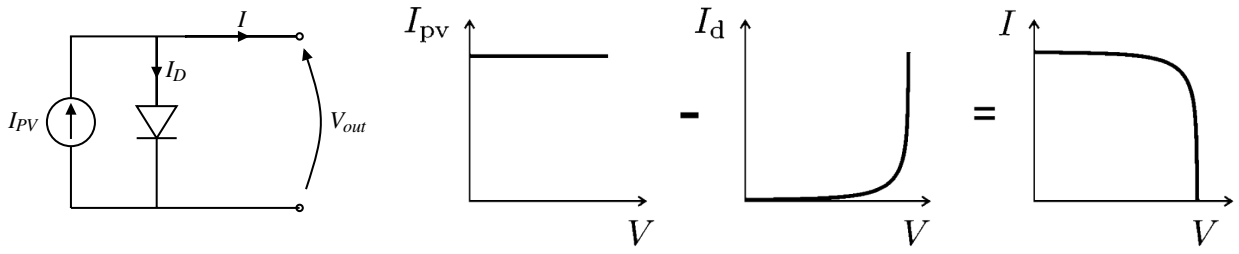


FIGURE 1.1 – Schéma d'une jonction PN représentant : la zone de charge d'espace (ZCE), les zones quasi-neutres (ZQN), les différents porteurs de charge, les sites dopants et le champ électrique E - Roger, 2013 [13].



(a) Modèle d'une cellule idéale (b) Le courant I est la superposition de I_{PV} et de I_D [16]

FIGURE 1.2 – Modèle idéal d'une cellule PV

utilise la notion de *voltage thermique* : $V_t = \frac{kT}{q}$.

$$I_D = I_0 \left[e^{\left(\frac{qV}{akT}\right)} - 1 \right] \tag{1.1}$$

Bien que ces modèles d'éléments localisé sont efficaces et précis pour les cellules de première génération et celles en couches minces, il existe aujourd'hui plusieurs technologies émergentes qui n'utilisent pas le champs électrique de la zone de déplétion pour la séparation des charges, et utilisent d'autre mécanismes et sources de champ électrique. La promesse des matériaux ferroélectriques par exemple, provient de leur champ électrique intrinsèque dont la divergence positive de la polarisation $\nabla \cdot \mathbf{P} \neq \mathbf{0}$ crée un déséquilibre de charges aux parois des domaines à polarisation différentes [15]. Pour ce genre de technologies sans jonctions et par conséquent, sans comportement similaire aux diodes, il est toujours difficile de concevoir un modèle de diode pour simuler le comportement complexe de ces cellules.

Avec la présence de la lumière, les photons assez énergétiques permettent la création des paires électron-trous qui, à leur tour, créent une différence de potentiel à travers la jonction. Une partie de ces porteurs de charge se recombine, mais l'autre se diffuse à travers les contacts de la cellule, ce qui donne naissance à un *photo-courant* I_{PV} dont la valeur dépend de l'intensité de la lumière incidente. En ajoutant le photo-courant I_{PV} à l'équation de Shockley 1.2, on retrouve un modèle élémentaire qui décrit une cellule idéale, composé d'une source de courant connectée à une diode en parallèle (figure 1.2.a). Évidemment ce modèle est complètement décrit par trois paramètres : *(i)* Le photo-courant I_{PV}, *(ii)* le facteur d'idéalité de la diode a et *(iii)* son courant de saturation I_0.

$$I = I_{PV} - I_0 \left[e^{\left(\frac{qV}{akT}\right)} - 1 \right] \tag{1.2}$$

On voit dans la figure 1.2.b que la courbe caractéristique connue de la cellule se forme par une translation verticale par I_{PV} d'une part, et de la forme exponentielle de la diode de l'autre. La courbure dans la région du point de puissance maximale est déterminée par le facteur d'idéalité de la diode.

1.3 Le modèle simple diode

Le modèle idéal précèdent est utile pour éclaircir le concept de la modélisation par circuits d'éléments localisés. Par contre, dans la pratique, et pour plus de précision, il est nécessaire de considérer les pertes ohmique de la cellule dans le matériau semi-conducteur utilisé et dans les contacts des électrodes. La manière la plus directe de modéliser ces phénomènes est de tout simplement ajouter une résistance en série R_s au circuit idéal. Ceci nous laisse avec la relation modifiée 1.3 du modèle dit *"simple diode-R_s"* qui a besoin de 4 paramètres pour une description complète : I_{PV}, a, I_0 et R_s.
L'une des limites de ce modèle provient de son imprécision lorsque la cellule subit des variations non-négligeables de température. L'insertion d'une résistance shunt permet d'améliorer la sensibilité du modèle aux variations de température et en même temps de considérer tout courant de fuite pouvant traverser la jonction [17]. On se retrouve cette fois avec les 5 paramètres du modèle dit *"simple diode-R_p"* ou tout simplement *"simple diode"* : I_0, I_{PV}, a, R_s et R_p. La relation entre ces paramètres

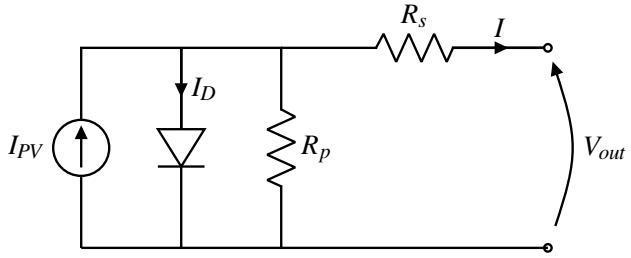


FIGURE 1.3 – Modèle Simple Diode

est décrite par l'équation 1.4. Ce modèle offre un compromis entre la simplicité et la précision et par conséquent est très largement testé et utilisé dans la littérature [18]. Le diagramme du circuit de ce modèle est présenté dans la figure 1.3.

$$I = I_{PV} - I_0 \left[e^{\left(\frac{q(V+IR_s)}{akT} \right)} - 1 \right] \tag{1.3}$$

$$I = I_{PV} - I_0 \left[e^{\left(\frac{q(V+IR_s)}{akT} \right)} - 1 \right] - \frac{V + IR_s}{R_p} \tag{1.4}$$

1.4 Modèle double diode

Jusque là on a graduellement amélioré la performance et la précision des modèles de cellules PV en considérant de plus en plus de phénomènes physique qui se produisent dans la cellule (pertes ohmiques, effets des variations de température et courants de fuites). Un phénomène potentiellement influent qui n'est pas considéré par le modèle simple diode est celui des recombinaisons. Comme son nom l'indique, le modèle *"double diode"* (figure 1.4 n'est que le modèle simple diode auquel on ajoute une autre diode en shunt pour mieux modéliser les effets de recombinaisons surtout aux conditions d'illumination faible [17].
Il est évident que l'addition d'une autre diode complique considérablement le modèle et on se retrouve avec deux paramètres supplémentaires (I_{PV}, I_{01}, I_{02}, $a1$, $a2$, R_s et R_p). Toutefois, dans les cas où la complexité des calculs associés n'est pas contraignante, la précision de se modèle est supérieure. L'équation 1.5 décrit la relation entre les 7 paramètres du modèle.

$$I = I_{PV} - I_{01} \left[e^{\left(\frac{q(V+IR_s)}{a_1 kT} \right)} - 1 \right] - I_{02} \left[e^{\left(\frac{q(V+IR_s)}{a_2 kT} \right)} - 1 \right] - \frac{V + IR_s}{R_p} \tag{1.5}$$

1.5 Influence des paramètres

Chacun des paramètres affecte le comportement de la cellule et la forme de la courbe caractéristique différemment. Tout d'abord, il faut noter que depuis l'équation 1.4, au point court-circuit, on

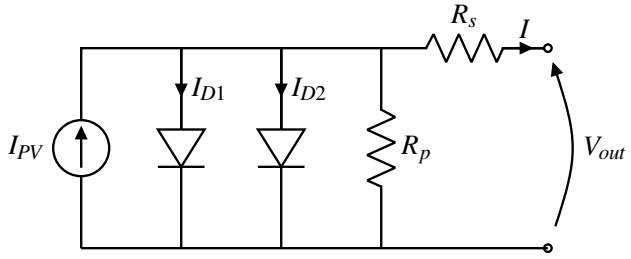
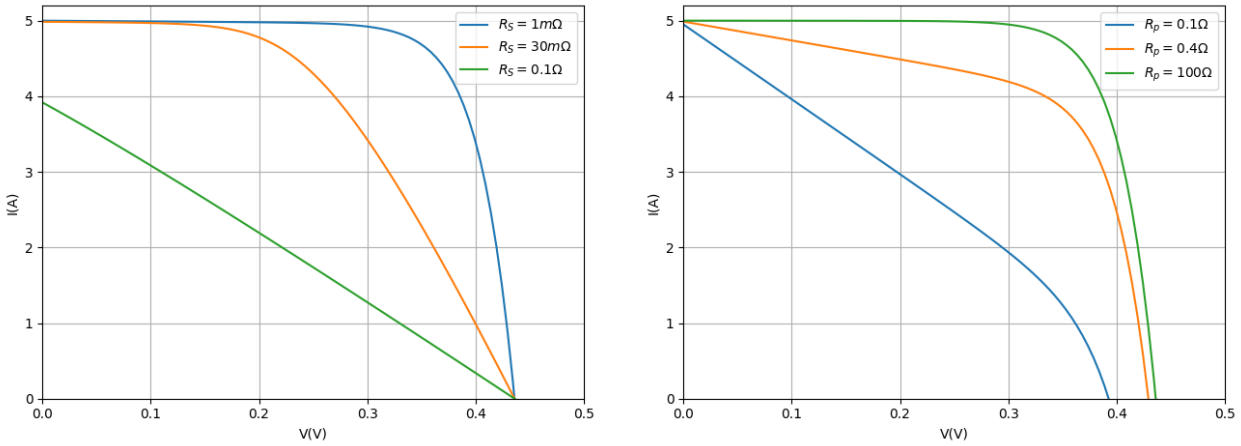


FIGURE 1.4 – Le modèle double diode

peut négliger le courant de saturation I_0 devant le photo-courant I_{PV} qui est supérieur de plusieurs ordres de grandeur. Cette simplification mène au fait que le photo-courant détermine le courant court-circuit de la diode et $I_{PV} \approx I_{CC}$. C'est en fait une simplification que plusieurs méthodes analytiques emploient [19, 16]. En ce qui concerne la résistance série R_s, l'augmenter engendre une chute de tension entre la jonction et la sortie de la cellule. On voit dans la figure 1.5 la courbe se rapprocher de plus en plus du comportement d'une résistance simple et que des valeurs très grande de R_s entraînent même une diminution légère de I_{SC}.

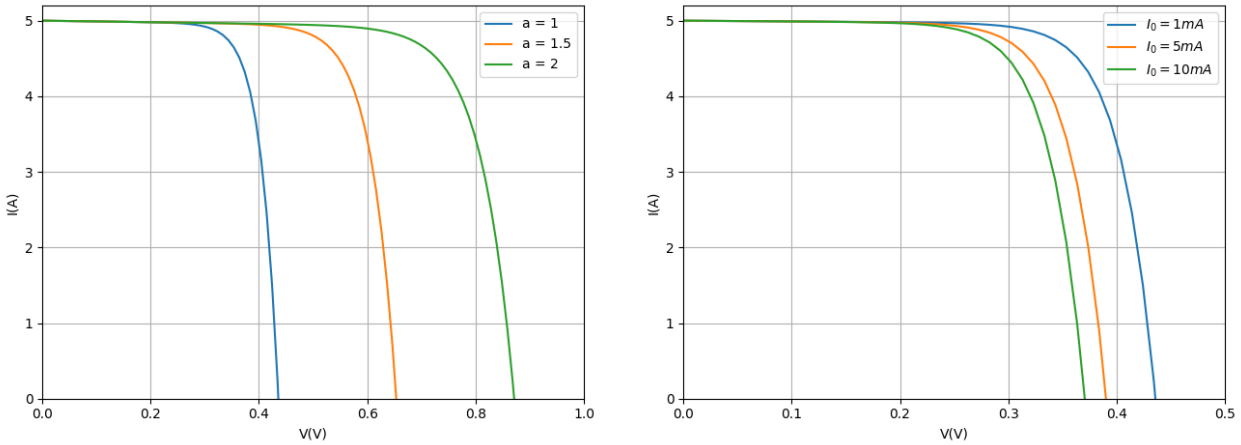
Pour la résistance shunt R_p, sa diminution conduit à un détournement d'un partie de plus en plus importante du courant sortant de la jonction, à travers la résistance en parallèle. Le courant total étant constant, le courant de sortie se retrouve de plus en plus réduit. Et comme dans le cas de la résistance série, une cellule avec beaucoup de pertes shunt à un comportement de plus en plus résistif (figure 1.5.b).

L'équation de Shockley (équation 1.1) est en fait retrouvée en négligeant toute recombinaison des porteurs de charges dans la zone de déplétion. Dans les diodes et les transistors réels, on constate une différence entre les courbes I-V et le modèle de Shockley [14]. Effectivement, le facteur d'idéalité modélise cette déviation du cas idéal. Une diode idéale est marquée par un facteur d'idéalité $a = 1$ alors qu'une diode où les évènements de recombinaison dominent dans la zone de déplétion est marquée par $a = 2$.



(a) Influence de R_s (b) Influence de R_p

FIGURE 1.5 – Influences de résistances du modèle sur la courbe caractéristique



(a) Influence du facteur d'idéalité (b) Influence du courant de saturation

FIGURE 1.6 – Influence des paramètres de la diode sur la courbe I-V

1.6 État de l'art d'estimation des paramètres

Les paramètres (soit les valeurs des éléments localisés des circuits) représentent une description complète du modèle et une bonne estimation de leurs valeurs est essentielle avant leur application. On utilise souvent les données des data-sheets offertes par les fabricants des cellules et modules PV qui couvrent les points clés de la caractéristique I-V (court-circuit, circuit ouvert et puissance maximale). Les méthodes analytiques tendent à utiliser ces informations en plus de quelques expressions comme la pente de la courbe aux points clés pour déterminer les résistances en shunt R_p et en série R_s. On trouve souvent aussi des simplifications comme le fait de considérer que seuls les I_{PV} et I_0 sont significativement influents sur la caractéristique [20]. Dans le modèle simple diode contenant les 5 paramètres I_{PV}, I_0, a, R_s et R_p, il est très courant de tout simplement négliger I_0 devant I_{PV} au point court circuit, et par conséquent considérer que $I_{PV} = I_{CC}$ (La valeur de I_0 est ensuite extraite des conditions circuit ouvert) [16, 20, 19]. En ce qui concerne a, R_s et R_p, on a besoin de davantage d'équations. Généralement, les chercheurs utilisent soit les dérivées de la caractéristique aux points clés, soit des expressions analytiques des coefficients de température le liant avec les paramètres.
Le problème étant de nature non-linéaire, les techniques d'optimisation avec des capabilités de recherche globales dans l'espace de recherche s'offrent comme alternative. La précision de ces techniques dépends évidemment de la fonction objectif considérée, des conditions initiales et de la nature de l'algorithme lui-même [21, 22, 23]. Les techniques de calcul souple et les algorithmes évolutionnistes ont susciter beaucoup d'intérêt récemment dans la littérature dans le but d'estimer les paramètres des cellules PV. Des techniques de réseaux de neurones artificiels [24, 25, 26], logique floue [27, 28, 29], algorithme génétique [30, 31, 32], optimisation par essaims particulaires (Particle Swarm Optimization) [33, 34] et évolution différentielle [23, 35, 36] ont été utilisées. Par contre, à cause de leurs nature stochastique, elles ne sont pas utilisées par les simulateurs PV qui sont contraints par des critères durs de cohérence et de temps de calcul. Ceci dit, elles sont très utiles lorsqu'il y a un besoin de précision sur les paramètres pour servir à l'optimisation du processus de fabrication ou pour l'étude de dégradation des cellules [37, 24].

1.7 Conclusion

Après avoir présenté les modèles simple et double diode, leur nature non-linéaire et le nombre relativement limité des paramètres (5 et 7 respectivement), semble suggérer les techniques d'optimisation comme méthodes d'extraction. Cependant, on n'a pas de raison à s'attendre que la topologie des fonctions objectif dans les espaces de recherche concernés soient même continues, voir différentiables. Dans le chapitre suivant, nous parlerons de l'une de ces techniques d'optimisation qui ne présuppose ni continuité, ni différentiabilité et est relativement récente : l'évolution différentielle.

Chapitre 2

Évolution Différentielle

2.1 Introduction

L'Évolution Différentielle (ED) est un algorithme évolutionnaire développée par Storn et Price [10] en 1995. Il est versatile et relativement simple à implémenter et utiliser, ce qui en fait un outil essentiel dans toute boite à outils d'optimisation. Comme toutes les techniques évolutionnaires, le principe de l'évolution différentielle repose sur la génération d'une population de N_P solutions (ou "vecteurs") qui permettent d'évaluer une fonction objectif à des points initiaux distribués aléatoirement dans un espace de recherche borné selon l'utilisateur. Ces points sont "perturbés" dans les générations successives de la population pour essayer de trouver des solutions extrémisant la fonction objectif. L'une des caractéristiques qui font la particularité de tout algorithme évolutionnaire est l'opération utilisée pour effectuer cette perturbation. Dans le cas de l'évolution différentielle, on perturbe une solution avec la différence de deux autres vecteurs de la population, multipliée par un facteur F, c'est l'opération dite de "mutation" comme est présenté dans la figure 2.1. Ce nouveau vecteur subit une opération de croisement avec le vecteur initial pour produire le vecteur d'essai qui est comparé avec le vecteur de même indice dans la population. Ceci est refait jusqu'à ce que tous les vecteurs de la population soient comparés avec un vecteur d'essai (soit N_P fois), créant la génération suivante. L'algorithme continue à créer de plus en plus de générations jusqu'à ce qu'un critère d'arrêt est satisfait. Souvent c'est un nombre de générations maximal ou une valeur de tolérance sur la fonction objectif.

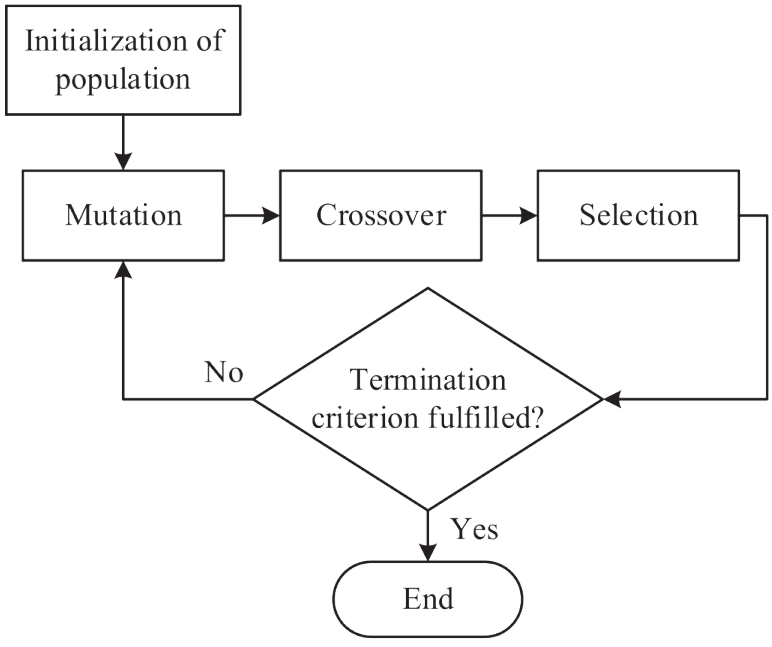


FIGURE 2.1 – Organigramme des étapes de l'Évolution Différentielle [38]

2.2 Description de l'algorithme

2.2.1 Initialisation

La convergence des techniques d'optimisation non-linéaires est toujours conditionnée par un "bon" choix des conditions initiales. Dans le cas de l'évolution différentielle, il s'agit de l'ensemble des vecteurs constituants la population initiale. Si un vecteur solution quelconque se constitue de D paramètres, notre espace de recherche est dit à D dimensions, ce qui fait que notre population initiale se compose de N_P vecteurs à D éléments. Mais pour pouvoir effectuer l'initialisation de la population, les bornes de l'espace de recherche doivent être spécifiées. Chacun des D paramètres doit avoir une borne supérieure et inférieure, ce qui fait un total de $2 \times D$ valeurs pour spécifier complètement les limites de l'espace. Reste le mécanisme utilisé pour effectivement générer les vecteurs dans cet espace borné. Pour couvrir entièrement et uniformément cet espace, il faut, pour chaque paramètre de chaque vecteur, générer aléatoirement et uniformément une valeur comprise dans la fourchette déterminée par un générateur de nombres aléatoires. En considérant que l'indice j associé à un vecteur $\vec{V}$ désigne le $j-$ème paramètre, on peut accomplir ceci avec la formule 2.1.

$$V_j = V_{\min,j} + \text{rand}[0,1](V_{\max,j} - V_{\min,j}) \tag{2.1}$$

On suppose avoir accès à une fonction rand$[0,1]$, qui joue le rôle du générateur de nombres aléatoires uniformes et que $0 \leq \text{rand}[0,1] < 1$. $V_{\min,j}$ et $V_{\max,j}$ sont les bornes inférieures et supérieures du $j-$ème paramètres, respectivement. Figure 2.2 montre un exemple d'une population initialisée dans un espace de recherche 2 dimensionnel. Les contours sont les *"isolignes"* de la fonction objectif, le minimum global doit être à l'intérieur de la fourchette spécifiée.

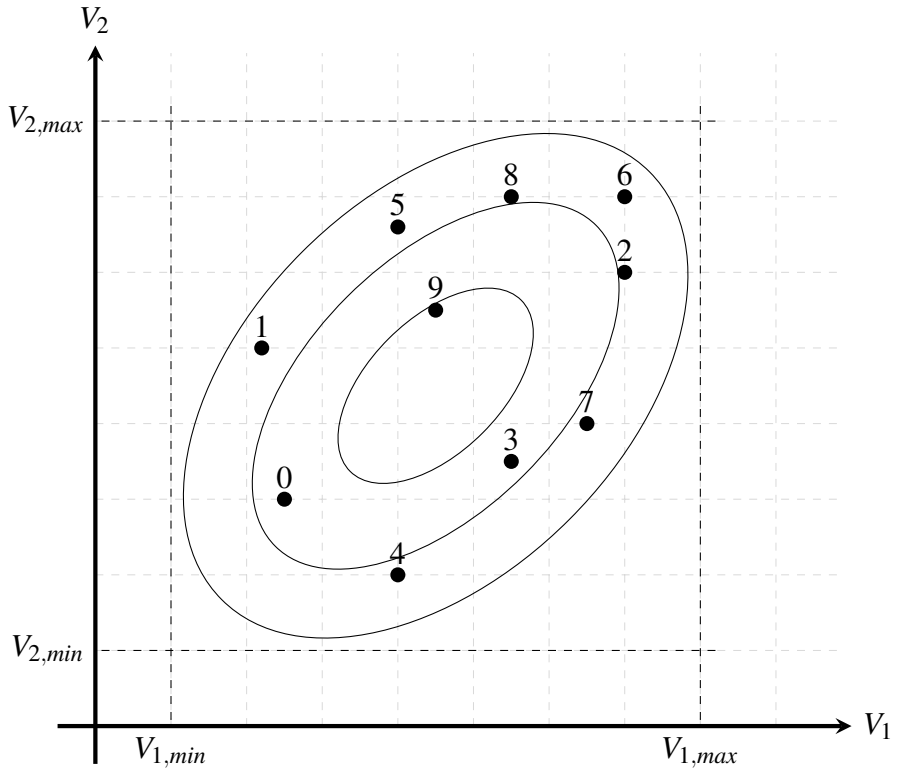


FIGURE 2.2 – Exemple d'une population initialisée dans un espace de recherche à deux dimensions. Dans ce cas $D = 2$, $N_P = 10$ et l'indice de génération $g = 0$ puisqu'il s'agit de la population initiale

2.2.2 Mutation

Après l'initialisation de la population, l'ED modifie, croise et recombine la population pour produire des vecteurs d'essai (un nombre N_P de ces vecteurs) qui seront comparés avec les vecteurs cibles qui leurs correspondent dans la population. La *Mutation* dans l'ED est en fait une "mutation différentielle". Elle consiste à ajouter la différence de deux vecteurs choisis aléatoirement de la population, multipliée par un facteur de pondération ou "Facteur de Mutation", à un troisième vecteur de base distinct. Cette opération produit un vecteur mutant M selon l'équation 2.2. L'indice i indique le i-ème vecteur de la population, et gen la génération où l'on est. Il n'y a pas de limite dure sur le Facteur de Mutation F, mais les valeurs supérieures à 1 sont rarement considérées. Dans notre cas, on considère que $F \in [0,1]$. Il existe plusieurs stratégies pour choisir le $\vec{V}_{base}$, le seul prérequis étant qu'il soit distinct du vecteur cible. Dans ce travail nous utiliserons exclusivement le choix du vecteur ayant la meilleure qualité ou valeur de "fitness" calculée par la fonction objectif. Les vecteurs de la différence $\vec{V}_{r1,gen}$ et $V_{r2,gen}$ sont choisis aléatoirement de la population pour chaque vecteur mutant et ne doivent qu'être distincts l'un de l'autre et des vecteurs cible $\vec{V}_{i,gen}$ et de base $\vec{V}_{base,gen}$.

$$\vec{M}_{i,gen} = \vec{V}_{base,gen} + F(\vec{V}_{r1,gen} - \vec{V}_{r2,gen}) \tag{2.2}$$

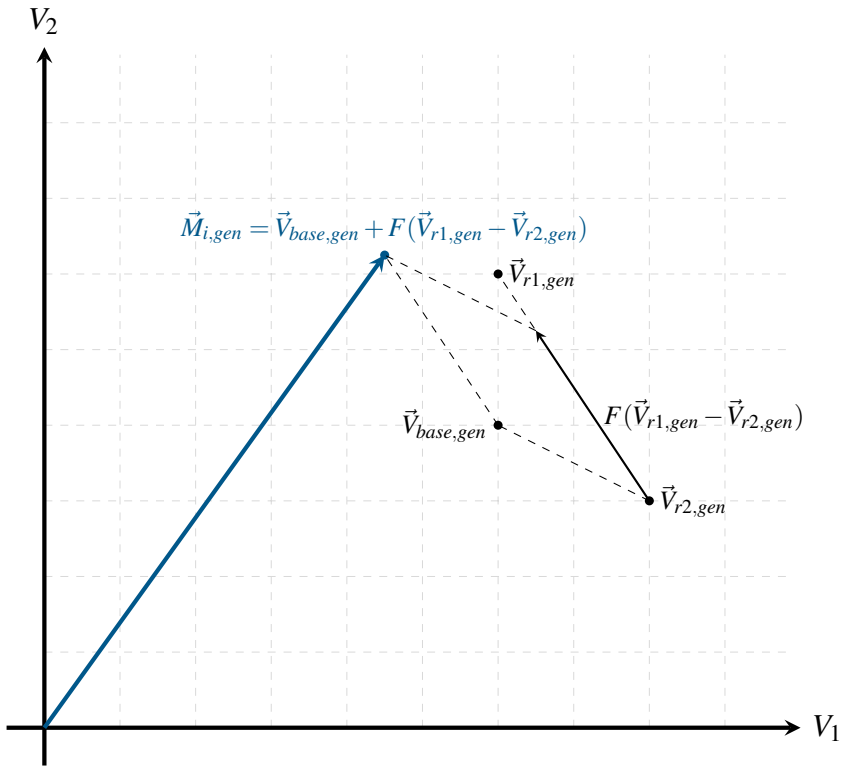


FIGURE 2.3 – L'opération de mutation différentielle ajoute $F(\vec{V}_{r1,gen} - \vec{V}_{r2,gen})$ au vecteur de base $\vec{V}_{base,gen}$ pour produire un mutant $\vec{M}_{i,gen}$

2.2.3 Croisement

La opération de mutation est suivie d'un croisement ou *recombinaison discrète*. Elle consiste à construire le vecteur d'essai $\vec{T}_{i,gen}$ à partir des éléments (i.e. paramètres) de deux vecteurs différents selon une probabilité spécifiée. Chaque élément j du vecteur d'essai est choisi selon la formule 2.3.

$$T_{j,i,gen} = \begin{cases} M_{j,i,gen} & \text{si rand}[0,1] \leq CR \text{ ou } j = j_{rand} \\ V_{j,i,gen} & \text{sinon} \end{cases} \tag{2.3}$$

$CR \in [0,1]$ détermine la probabilité que le paramètre provienne du vecteur mutant tant que rand$[0,1]$ est effectivement un générateur aléatoire uniforme. Le taux de croisement CR est l'un des paramètres spécifiés par l'utilisateur au début. Un nombre aléatoire j_{rand} est aussi choisi tel que $0 \leq j_{rand} < D$ pour s'assurer que le nouveau vecteur d'essai ne duplique pas complètement le vecteur cible $\vec{V}_{i,gen}$.

2.2.4 Sélection

Si le vecteur d'essai $\vec{T}_{i,gen}$ est évalué à une valeur inférieure par la fonction objectif f au vecteur cible $\vec{V}_{i,gen}$, il le remplace dans la génération suivante (i.e. $gen+1$). Sinon le vecteur cible survit à la sélection et retient sa place dans la génération suivante (équation 2.4). Quand la nouvelle population est complète, le processus de mutation, croisement et sélection est renouvelé jusqu'à ce qu'un critère d'arrêt est vérifié (e.g. un nombre maximal de générations gen_{max})

$$\vec{V}_{i,gen+1} = \begin{cases} \vec{T}_{i,gen} & \text{si } f(\vec{T}_{i,gen}) \leq f(\vec{V}_{i,gen}) \\ \vec{V}_{i,gen} & \text{sinon} \end{cases} \tag{2.4}$$

2.3 Remarques

On vient de citer toutes les étapes de l'ED "classique" comme elle a été présenté par Storn et Price 2005 [39]. Cependant il existe plusieurs variations ou "stratégies" de l'ED selon la manière avec laquelle on choisi le vecteur de base avant la mutation, le nombre de différences pondérées ajoutées et la méthode de croisement. Nous avons opté à choisir le vecteur à meilleure valeur de fitness comme vecteur de base, auquel on ajoute une seule différence de vecteurs pondérée par F, et finalement un croisement binomial. Le mot technique pour cette stratégie est "DE/best/1/bin" dont une demonstration en pseudocode est dans l'algorithme 1 dans la section suivante.

Il faut aussi noter que cet algorithme comme il est, n'est contraint nul part à ne générer que des solutions comprises dans les limites initiales de l'espace de recherche. Ceci présente en fait un avantage de l'ED puisque ça permet d'explorer les zones au-delà des limites de la population initiale pour potentiellement trouver un minimum global qu'on est pas toujours assurés qu'il soit dans la zone initiale. Cependant, on peut tomber sur des solutions non "physiques" et qui ne présentent aucun intérêt pour notre application. Storn et Price citent plusieurs techniques pour résoudre ce problème, mais dans notre cas on effectue un test sur chaque vecteur mutant. Si un de ses paramètres se retrouve à l'extérieur des limites spécifiées de l'espace de recherche, on le pénalise en imposant une valeur de fitness assez large pour éliminer les chances que ce vecteur pourra survivre vers la génération suivante.

Dans notre application de l'ED sur le problème d'identification des paramètres de modèles à diodes, nous prenons l'approche d'un problème d'optimisation. En effet, un vecteur solution se constitue de tous les paramètres nécessaires pour une description complète du modèle équivalent, c'est-à-dire un espace de recherche à $D = 5$ dimensions dans le cas du modèle simple diode (R_P), et à D = 7 dimensions dans le cas du modèle double diode. La fonction objectif à minimiser devrait être une mesure de la différence entre la courbe I-V du modèle et celle de la cellule experimentale. La méthode la plus répandue pour quantifier cette différence est la racine de l'erreur quadratique moyenne ou *Root Mean Squared Error*. L'erreur d'un vecteur $\vec{V}_{sol}$ est donnée par l'équation 2.5 où l'indice i indique un point dans la courbe caractéristique I-V.

$$f(\vec{V}_{sol}) = RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(I_{i,exp} - I_{i,cal}(\vec{V}_{sol}))^2} \tag{2.5}$$

2.4 Pseudo-code

Une implémentation de l'Évolution Différentielle en pseudo-code est la suivante :

Algorithme 1 : Stratégie DE/best/1/bin

$F \leftarrow$ Facteur de mutation;
$CR \leftarrow$ Taux de croisement;
$N_P \leftarrow$ Taille de la population;
$gen_{max} \leftarrow$ Nombre de générations;
$D \leftarrow$ Nombre de paramètres;
$gen = 0$;
Initialize population $P_0 = [V_0, V_1, .., V_{N_P}]$;
for $i = 0$ *to* N_P **do**
    $V_{min} = [V_{1,min}, V_{2,min}, .., V_{D,min}]$;
    $V_{max} = [V_{1,max}, V_{2,max}, .., V_{D,max}]$;
    $V_i = V_{min} + \text{rand}[0,1](V_{max} - V_{min})$;
end for
while $gen < gen_{max}$ **do**
    for $j = 0$ *to* N_P **do**
        $V_{j,gen} = [V_{1,j,gen}, V_{2,j,gen}, .., V_{D,j,gen}]$;
        Choisir le vecteur de base et deux vecteurs aléatoires V_{r1} et $V_{r2} \in P_{gen}$;
        $V_{base} = V \in P_{gen} \mid \forall (K \in P_{gen}), \quad f(V) \leq f(K)$;
        Mutation :
        $M_{j,gen} = V_{base} + F(V_{r1} - V_{r2})$;
        Croisement :
        $T_{j,gen} = [T_{1,j,gen}, T_{2,j,gen}, .., T_{D,j,gen}]$;
        if $rand[0,1] < CR$ *or* $j = j_{rand}$ **then**
            $T_{i,j,gen} = M_{i,j,gen}$;
        else
            $T_{i,j,gen} = V_{i,j,gen}$;
        end if
        Selection :
        if $f(T_{j,gen}) < f(V_{j,gen})$ **then**
            $V_{j,gen+1} = T_{j,gen}$;
        else
            $V_{j,gen+1} = V_{j,gen}$;
        end if
    end for
    $gen = gen + 1$;
end while

2.5 Conclusion

Maintenant que nous avons décrit l'évolution différentielle tel qu'elle a été décrite par Storn et Price, nous avons opté à utiliser la stratégie DE/best/1/bin et l'erreur quadratique moyenne comme fonction objectif. Dans ce qui suit, il va falloir rendre l'évolution différentielle compatible avec les modèles à diodes cité dans le chapitre 1. Pratiquement, il faudrait concevoir une méthode pour retrouver la caractéristique IV associée à chaque vecteur solution pour pouvoir calculer son erreur quadratique moyenne par rapport aux données expérimentales. Ceci va ensuite permettre à l'évolution différentielle d'effectuer ses opérations de sélection selon la fonction objectif.

Chapitre 3

Évolution Différentielle sur les circuits simple et double diodes

3.1 Introduction

Maintenant que nous avons abordé les modèles à diodes et l'Évolution Différentielle indépendemment l'un de l'autre, dans ce chapitre nous allons démontrer la manière pratique d'appliquer l'ED sur les modèles pour estimer les valeurs des paramètres. Nous allons présenter l'outil qu'on a développé pendant ce projet pour utiliser cette méthode avec un stratégie métaheuristique supplémentaire pour optimiser le choix des facteur de mutation et taux de croisement de l'ED. En ce qui concerne l'implémentation pratique de cette technique, on utilise le langage de programmation *Python* qui nous permet de faire appel a quelques bibliothèques scientifiques pour fournir les outils mathématiques et statistiques requis par la méthode. Dans cette partie, nous allons fournir des petits extraits de code pertinents à la discussion qui montrent comment utiliser notre outil. Pour des raisons de clarté et breveté, ce ne sont pas des extraits complètement fidèles au code utilisé en réalité. Le code complet et non modifié est disponible comme annexe à la fin de ce document.

3.2 Fonction W de Lambert

Les méthodes évolutionnaires dépendent d'une fonction objectif qu'il faut minimiser pour sélectionner les meilleures solutions dans une population. Dans notre cas, la fonction objectif est la *RMSE* et elle quantifie la différence entre la courbe caractéristique du modèle et les données expérimentales. Cependant, chaque fois qu'un vecteur solution $\vec{V}$ (formules 3.1) est généré, il faut pouvoir recréer la courbe I-V associée pour permettre à la fonction objectif de calculer la RMSE.

$$\vec{V}_{\text{simple diode}} = \begin{bmatrix} R_s \\ R_p \\ a \\ I_0 \\ I_{PV} \end{bmatrix}, \quad \vec{V}_{\text{double diode}} = \begin{bmatrix} R_s \\ R_p \\ a_1 \\ a_2 \\ I_{01} \\ I_{02} \\ I_{PV} \end{bmatrix} \tag{3.1}$$

Spécifiquement, il faut exercer un pression de sélection sur la population qui va nous permettre de nous débarrasser des solutions non-physiques et donc sans intérêt pour nous. Concrètement, tout vecteur solution comprennent des valeurs de résistance série R_s ou parallèle R_p négatives est pénalisé avec une RMSE arbitrairement large. C'est en fait le rôle des lignes 4-6 dans l'implementation de la fonction objectif en python :

```python
# Cette fonction prend un vecteur solution et les points IV expérimentaux comme arguments
def objf(vecteur, exp_v, exp_i):
    # Pénalisons le vecteur si les valeurs sont non-physiques
    rs, rp = vecteur[0], vecteur[1]
    if rs < 0 or rp < 0:
        return 100 # Valeur fitness large
    ical = i_from_vect(vector, exp_v) # Une fonction donnant la caractéristique IV de "vector"
    erreur = ical - exp_i
    return np.sqrt(np.mean(erreur ** 2)) # RMSE
```

Les équations de modèles simple et double diode (équations 1.4, 1.5, respectivement) sont transcendantes, il est donc impossible d'extraire directement le courant à partir de la tension et des paramètres (Le courant I figure simultanément dans le premier membre et dans l'exponentiel du second). Ainsi, il n'est pas trivial de remplir la fonction de `i_from_vect(vector, exp_vol)` qui est censée calculer le courant à partir d'un vecteur solution et d'un ensemble de voltages. Pour résoudre ce problème, plusieurs méthodes on été utilisées initialement avec des approches d'approximation analytique ou itérative [40, 41, 42]. Ces méthodes sont approximatives mais permettent de trouver la solution explicitement avec des fonctions élémentaires (Développement Taylor par exemple). Dans notre cas, on fait recours à la méthode de Jain et Kapoor (2004) [43, 44] qui utilise la fonction W de Lambert pour une solution analytique exacte de ces équations.

3.2.1 Définition

La "*fonction W*" de Lambert est définie comme l'inverse de la fonction $w \to f(w) = we^w$, où $w = W_k(z) \mid z \in \mathbb{C}$. La fonction f n'étant pas surjective, la fonction $W_k(z)$ est donc *multivaluée* et comprends plusieurs branches indexées par k (W_0 est choisie comme branche principale). Si $x \in \mathbb{R}$ et $-1/e \leq x < 0$, il existe deux valeurs réelles possible de $W(x)$ (figure 3.1).

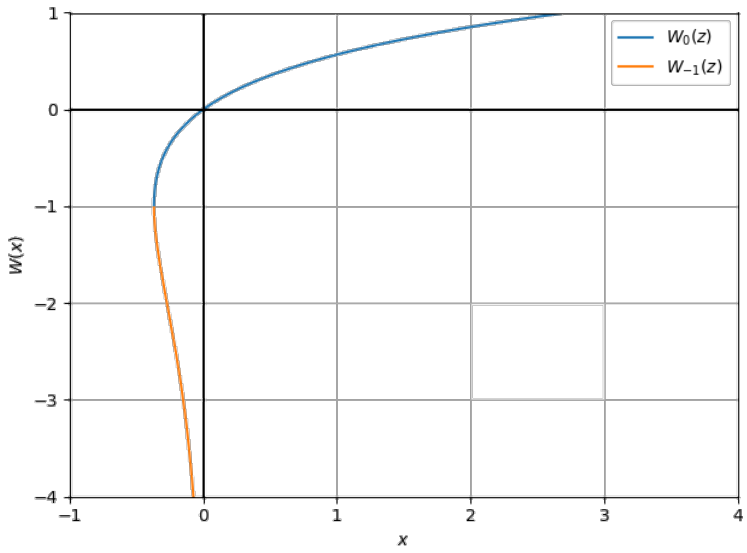


FIGURE 3.1 – Les deux branches réelles de $W(x)$ lorsque x est réel

3.2.2 Évaluation de la fonction W

Le fait qu'il n'existe pas de fonctions mathématiques élémentaires donnant explicitement $W(z)$ est remédié par l'existence de plusieurs *algorithmes de recherche des zéros* permettant le calcul des valeurs de n'importe quelle branche de la fonction W. Dans notre cas spécifique, la bibliothèque scientifique *scipy* de Python fournit une fonction W implémentée par l'itération de Halley qui est un

exemple d'une méthode de classe "Householder" [1]. La méthode de Halley a été appliquée à la fonction W par Corless et al. [45] donnant le schéma itératif suivant :

$$w_{j+1} = w_j - \frac{w_j e^{w_j} - z}{e^{w_j}(w_j+1) - \frac{(w_j+2)(w_j e^{w_j}-z)}{2w_j+2}} \tag{3.2}$$

Un exemple basique de l'utilisation de cette méthode en Python est le suivant :

```
import numpy as np
from scipy.special import lambertw # scipy fournit la fonction W

z = np.linspace(-1/np.e, 3, 1000) # evaluons W entre -1/e et 3
w0 = lambertw(z, 0) # choisir la branche principale
```

3.2.3 Résolution des modèles simple et double diode par la fonction W

Depuis la définition de la fonction W, la solution d'une équation $xe^x = a$ est $x = W(a)$. En effectuent des manipulations algébriques élémentaires sur le modèle simple diode (equation 1.4), Jain et Kapoor ont montré que l'expression explicite du courant en fonction des paramètres et de la tension est :

$$I = \frac{R_{sh}(I_0 + I_{PV}) - V}{R_s + R_{sh}} - \frac{W\left(\frac{R_s I_0 R_{sh}}{aV_{th}(R_s+R_{sh})} e^{\left(\frac{R_{sh}(R_s I_{PV} + R_s I_0 + V)}{aV_{th}(R_s+R_{sh})}\right)}\right) aV_{th}}{R_s} \tag{3.3}$$

On remarque bien que le second membre ne contient nul part un terme de courant I et que le courant est donné entièrement en fonction des paramètres du modèle et du voltage. Il faut noter aussi que le terme de la fonction W est sous risque d'un dépassement et de retourner des valeurs infinies. Pour des raisons de stabilité numérique on utilise la notion de *Conductance Shunt* : $C_{sh} = \frac{1}{R_{sh}}$ car cette résistance prend souvent de valeurs $\gg 1$ ce qui entraîne un risque de divergence des calculs [2]. Avec la substitution de la conductance shunt, si $R_{sh} \to \infty$ alors $C_{sh} \to 0$ ce qui assure la stabilité du calcul numérique.

Le modèle double diode (équation 1.5) contient un terme exponentiel pour chaque diode. De la même manière que Jain et Kapoor on retrouve explicitement l'expression du courant avec deux termes de la fonction W (equation 3.4).

$$\begin{aligned} I = & \frac{R_{sh}(I_{01} + I_{02} + I_{PV}) - V}{R_s + R_{sh}} \\ & - \frac{a_1}{2R_s} W\left(\frac{R_s R_{sh}(I_{01}+I_{02})}{a_1(R_s+R_{sh})} e^{\left(\frac{R_{sh}(R_s I_{PV} + R_s I_{01} + R_s I_{02} + V)}{a_1(R_s+R_{sh})}\right)}\right) \\ & - \frac{a_2}{2R_s} W\left(\frac{R_s R_{sh}(I_{01}+I_{02})}{a_1(R_s+R_{sh})} e^{\left(\frac{R_{sh}(R_s I_{PV} + R_s I_{01} + R_s I_{02} + V)}{a_2(R_s+R_{sh})}\right)}\right) \end{aligned} \tag{3.4}$$

3.3 Utilisation de l'outil *DEPV*

Pour appliquer l'Évolution Différentielle avec succès sur les modèles à diodes, il faut savoir choisir les bonnes valeurs de paramètres de contrôle CR, F et N_P. Il n'existe pas de règle stricte mais Storn et Price [39] donnent quelques indications. En ce qui concerne le facteur de mutation F, les valeurs $F \geq 1$ ne sont pas fiables et souvent convergent très lentement par rapport aux $F < 1$. Cependant, Zaharie (2002) [46] constate une borne inférieure de $F > 0.4$. Puisque l'opération de *sélection* tend

1. La méthode de Newton est un exemple d'une méthode de Householder
2. On trouve souvent des valeurs larges de résistance shunt, ce qui explique l'existence dans la littérature de plusieurs modèles utilisant la simplification $R_{sh} = +\infty$

à réduire la diversité dans la population, le rôle de la *mutation* et de balancer cette pression exercée sur la population et tend à augmenter la diversité. Si F est trop petit, l'ED peut converger de manière prématurée même avec l'absence de la pression sélective. En ce qui concerne le taux de croisement CR, Salomon (1996) [47] a démontré les limites d'un CR trop petit et par conséquent Storn et Price recommandent des valeurs de CR proche de 1. Reste à choisir la taille de la population N_P, généralement $10D \leq N_P \leq 20D$ est recommandé mais dans notre cas on optera à $N_P = 100$.

Dans ce projet, nous avons développé une bibliothèque en Python fournissant une interface permettant de lancer des calculs avec la technique de l'ED sur n'importe quelles cellule, à condition d'avoir accès aux données expérimentales. On fournit la fonction `read_csv()` qui permet à l'utilisateur d'extraire les points expérimentaux de la caractéristique IV (voltage en abscisses et courant en ordonnées) à partir d'un fichier .txt ou .csv. Nous fournissons aussi une classe `DE` qui gère tous les calculs. Il suffit de lui donner les paramètres nécessaires tel que les bornes de l'espace de recherche et le fichier contenant les données expérimentales. Pour plus de contrôle sur les paramètres de l'algorithme, la classe `DE` expose plusieurs variables à travers son "constructeur". Le prototype de la fonction constructrice ou `__init__()` de la classe `DE` est le suivant :

```
  def __init__(self, bounds, ivdata, Ns, Np, temp, popsize=100, maxiter=200, mutf=0.7, crossr=0.8):
      """
      :param bounds: Dictionary of bounds in this form {'rp': [lower, upper], 'rs': [lower, upper] ...}
      :param ivdata: [Voltages, Currents]
      :param Ns: Number of cells in series
      :param Np: Number of cells in parallel
      :param temp: Temperature
      :param popsize: Population size
      :param maxiter: Maximum number of generations
      :param mutf: Mutation Factor F
      :param crossr: Crossover Rate CR
      """
```

Notons les valeurs prise par défaut pour la taille de la population $N_P = 100$, le nombre de générations maximal $Gen_{max} = 200$, le facteur de mutation $F = 0.7$ et le taux de croisement $CR = 0.8$. Si l'utilisateur ne fournit pas explicitement ces paramètres, les valeurs par défaut sont alors utilisées. Un exemple d'exécution de la classe DE sur la cellule est le suivant :

```
# On importe la classe DE et la fonction read_csv()
from objects import DE, read_csv

# Bornes de l'espace de recherche
bornes = {'rp':  [2, 100],
          'rs':  [0, 1],
          'a':   [1, 2],
          'i0':  [1e-07, 1e-04],
          'ipv': [0, 10]}

# Température en K et nombre de cellules en serie et en parallele
T = 33 + 275.15
Ns, Np = 1, 1

# Données expérimentales disponible dans un fichier .csv
exp = read_csv("data/RTC33D1000W.csv")

# On crée un objet DE pour utiliser ces données et effectuer le calcul
RTC = DE(b, exp, Ns, Np, T)

# On lance le calcul
RTC.solve()

# Traçage des graphes et résultats
RTC.plot_fit_hist()
RTC.plot_result(print_params=True)
```

Si on veut utiliser le modèle double diode, il suffit de donner 7 intervalles au lieu de 5 comme premier argument. DEPV reconnaît automatiquement le modèle à utiliser à partir de la taille du dictionnaire `bornes` fournit à la classe `DE` :

```python
bornes = {'rp':  [2, 100],
     'rs':  [0, 1],
     'a1':  [1, 2],
     'a2':  [1, 2],
     'i01': [1e-7, 1e-5],
     'i02': [1e-7, 1e-5],
     'ipv': [0, 5]}

T = 45 + 275.15
Ns, Np = 36, 1
exp = read_csv("data/PWP.csv")
PWP = DE(bornes, exp, Ns, Np, T)
PWP.solve()
PWP.plot_result(print_params=True)
```

3.3.1 Interface graphique

L'interface programmatique offerte par la classe `DE` permet à l'utilisateur d'inclure sa fonctionnalité dans ses propres scripts pour n'importe quelle application. On peut par exemple effectuer des études statistiques sur le comportement et la performance de cette classe en l'exécutant sous d'autres codes. Ceci nous permettra plus tard d'étudier la cohérence des résultats retrouvés par cette méthode et quantifier sa performance et le temps de calcul en fonction des inputs.

Dans le cas où on n'a pas besoin d'accéder à cette fonctionnalité programmatiquement et qu'on veut juste retrouver les résultats immédiatement, il suffit d'utiliser directement l'interface graphique de DEPV. On a toujours besoin de fournir le fichier contenant les données expérimentales, la température, les bornes et les cellules en série et en parallèle (figure 3.2). Les résultats et les graphes sont affichés par l'interface dans la figure 3.3

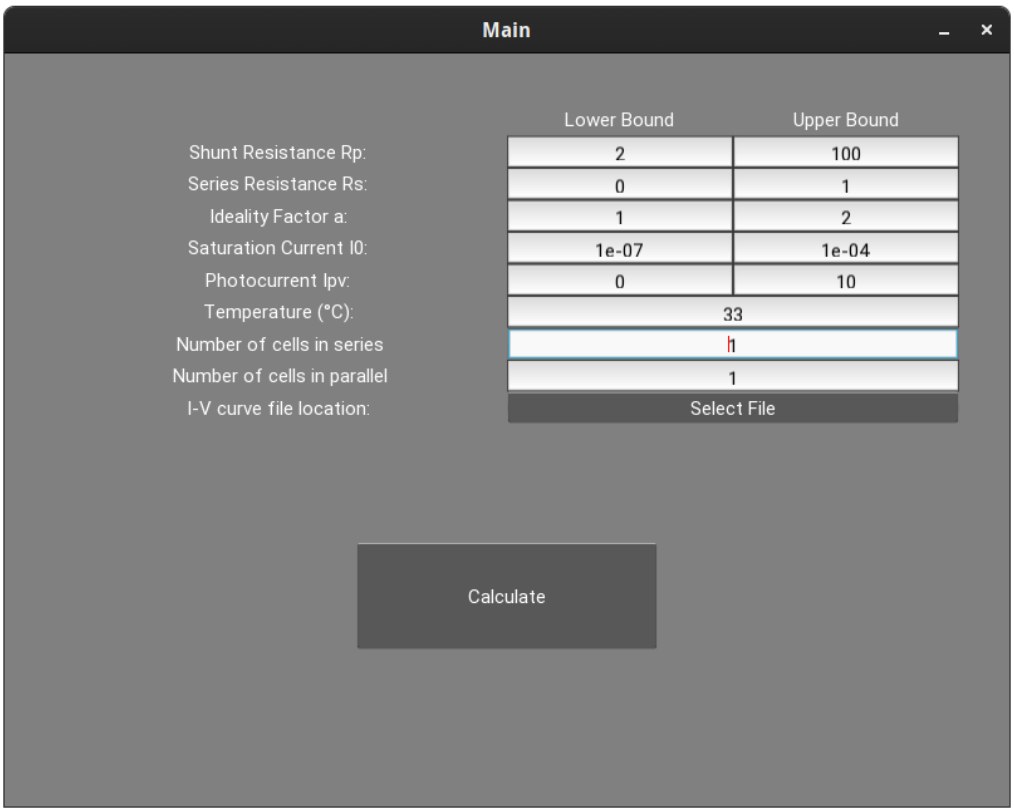


FIGURE 3.2 – Interface principale de l'outil DEPV avec les champs texte pour insérer les bornes, la température, les cellules en séries/parallèle et le fichier .csv contenant les données expérimentales

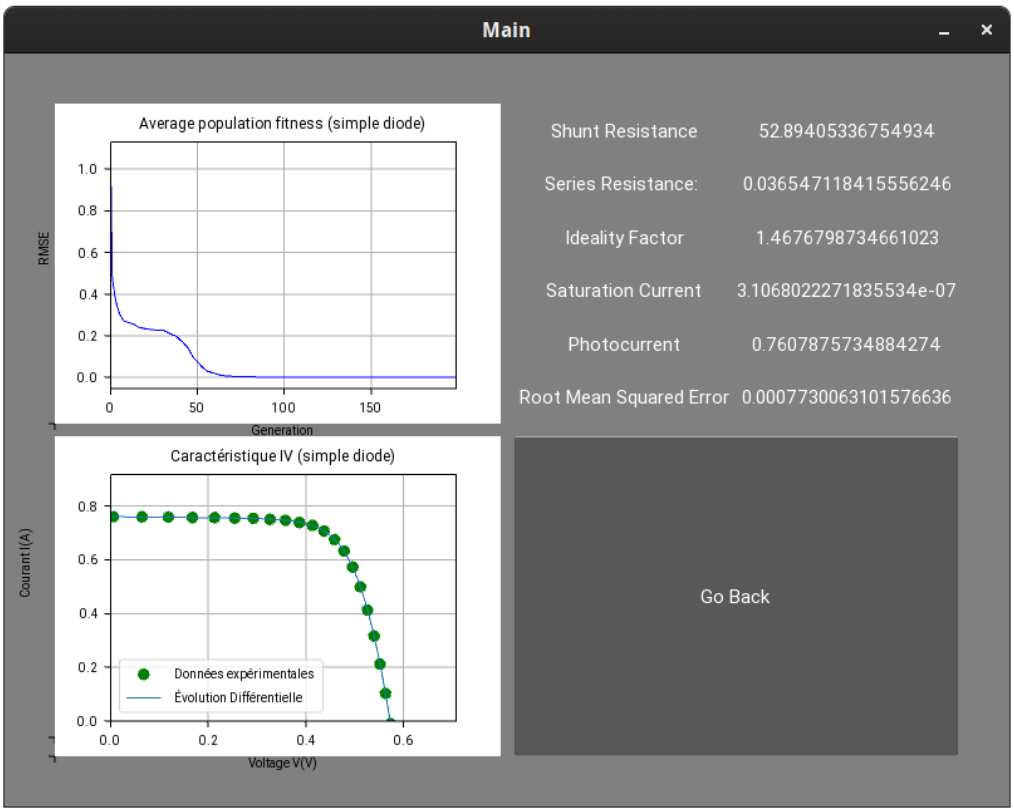


FIGURE 3.3 – Graphes et résultats affichés par DEPV pour la cellule RTC France 57 mm

3.4 Stratégie métaheuristique

Comme on l'a déjà mentionné, la sélection des paramètres de l'ED, notamment le Facteur de Mutation F et le taux de croisement CR, reste assez arbitraire et empirique malgré son influence déterminante sur la convergence de la technique. Bien que Storn et Price [39] recommandent des valeurs de F proche de 1 et que les limites déterminées par Zaharie et al. [46] sont souvent correctes, il existe des cas atypiques comme l'estimation des états fondamentaux des agrégats $Si_x - H$ par Chakraborti et al. (2001) [48]. Dans leur cas, l'ED a réussi à minimiser l'énergie des agrégats atomiques avec des facteurs de mutation entre 0.0001 et 0.4.

On propose donc une technique de méta-optimisation permettant de déterminer ces paramètres sans faire recours à l'arbitraire/l'empirique. Cette solution consiste à utiliser l'ED à un niveau plus haut pour déterminer les paramètres à utiliser pour l'ED usuelle pour l'estimation des paramètres comme on a décrit dans le chapitre précèdent. Le nouvel espace de recherche est donc à deux dimensions seulement (F et CR) et par conséquent n'a besoin que d'une population de $10 \times D = 20$ vecteurs solutions. En effet, chaque vecteur solution représente une instance entière d'Évolution Différentielle avec des F et CR différents. La fonction objectif est simplement la valeur fitness du meilleur résultat obtenu avec un F et CR spécifique. Il est vrai que cette technique apparaît très coûteuse en termes de calcul, puisque dans ce cas, chaque vecteur solution ne pourra être évalué qu'après sa propre instance d'ED ait convergé. Dans l'ED "standard" décrite dans le chapitre 2, une instance effectue $N_P \times Gen_{max} = 100 \times 200 = 20000$ évaluations de la fonction objectif. Avec cette ED "d'ordre supérieur", le nombre d'évaluations sera $N_P^{\text{higher order}} \times Gen_{max}^{\text{higher order}} \times 20000$. Ce qui correspond à 2×10^6 évaluations pour $N_P^{\text{higher order}} = 20$ et $Gen_{max}^{\text{higher order}} = 50$. En revanche, une seule exécution de cette technique au préalable et nécessaire. Les valeurs obtenues avec cette technique peuvent par la suite être réutilisées plusieurs fois avec l'ED standard et seulement les 20000 évaluations. Le nature coûteuse de cette technique est donc moins problématique puisque elle n'est utile que dans les cas où les ressources en capacité de calcul et en temps ne sont pas déterminantes. Par ailleurs, l'utilisation de

cette technique permet de choisir les paramètres optimaux et donc à accélérer la convergence pour les cas où la rapidité des calculs est effectivement essentielle (Estimation des paramètres en temps réel par exemple). Le pseudo-code de cette technique est présenté dans l'algorithme 2. Notons que dans la ligne 26, l'évaluation de la fonction objectif $f(V_{j,gen})$ contient une instance entière de l'algorithme 1 avec le facteur de mutation et taux de croisement fournis par le vecteur $V_{j,gen}$.

Algorithme 2 : Stratégie métaheuristique pour déterminer F et CR

1 $CR^{\text{higher order}} \leftarrow$ Taux de croisement;
2 $D^{\text{higher order}} \leftarrow$ Dimensions de l'espace de recherche (2 dans ce cas);
3 $N_P^{\text{higher order}} \leftarrow$ Nombre de population $(10 \times D)$;
4 $gen = 0$;
5 Initialisation de la population $P_0 = [V_0, V_1, .., V_{N_P}]$;
6 **for** $i = 0$ *to* N_P **do**
7 $\quad V_{min} = [V_{F,min}, V_{CR,min}]$;
8 $\quad V_{max} = [V_{F,max}, V_{CR,max}]$;
9 $\quad V_i = V_{min} + \text{rand}[0,1](V_{max} - V_{min})$;
10 **end for**
11 **while** $gen < gen_{max}$ **do**
12 $\quad$ **for** $j = 0$ *to* N_P **do**
13 $\quad\quad V_{j,gen} = [V_{F,j,gen}, V_{CR,j,gen}]$;
14 $\quad\quad$ Choisir le vecteur de base et deux vecteurs aléatoires V_{r1} et $V_{r2} \in P_{gen}$;
15 $\quad\quad V_{base} = V \in P_{gen} \mid \forall (K \in P_{gen}), \quad f(V) \leq f(K)$;
16 $\quad\quad$ Mutation :
17 $\quad\quad M_{j,gen} = V_{base} + F^{\text{higher order}}(V_{r1} - V_{r2})$;
18 $\quad\quad$ Croisement :
19 $\quad\quad T_{j,gen} = [T_{F,j,gen}, T_{CR,j,gen}]$;
20 $\quad\quad$ **if** $rand[0,1] < CR^{higher\ order}$ *or* $j = j_{rand}$ **then**
21 $\quad\quad\quad T_{i,j,gen} = M_{i,j,gen}$;
22 $\quad\quad$ **else**
23 $\quad\quad\quad T_{i,j,gen} = V_{i,j,gen}$;
24 $\quad\quad$ **end if**
25 $\quad\quad$ Sélection :
26 $\quad\quad$ **if** $f(T_{j,gen}) < f(V_{j,gen})$ **then**
27 $\quad\quad\quad V_{j,gen+1} = T_{j,gen}$;
28 $\quad\quad$ **else**
29 $\quad\quad\quad V_{j,gen+1} = V_{j,gen}$;
30 $\quad\quad$ **end if**
31 $\quad$ **end for**
32 $\quad gen = gen + 1$;
33 **end while**

3.5 Conclusion

Dans ce chapitre nous avons synthétisé l'ED avec les modèles à diodes à l'aide de la fonction W de Lambert. Dans ce qui suit nous allons montrer les résultats de cette technique et une étude comparative de cette méthode avec les autres techniques de l'état de l'art. La stratégie métaheuristique qu'on vient de présenter sera aussi comparée avec l'ED standard pour essayer de quantifier les gains en performance de la méthode.

Chapitre 4

Résultats et analyse

4.1 Introduction

Dans ce chapitre final nous allons analyser et discuter les résultats de l'application de l'ED. Les cas d'études considérés sont une cellule très largement étudiée dans la littérature (R.T.C France 57 mm) et des module PV en mono-Si (Schutten Solar STM6-40/36) et poly-Si (Photowatt-PWP 201). On effectue aussi une analyse sur la stabilité et cohérence de la méthode avant de conclure avec une comparaison entre la méthode standard et la stratégie métaheuristique proposée à la fin du troisième chapitre. Tous les calculs ont été effectués et les résultats obtenus avec *Python 3.8.3* (implementation standard *CPython*) et un processeur *Intel i5-7200U* avec une fréquence d'horloge maximale de 3.1GHz sous le système d'exploitation *Arch Linux x86_64* Kernel version *5.7.2 arch1-1*.

4.2 Cas d'études et résultats

4.2.1 Cas 1 : Cellule 57-mm de R.T.C France

Ce premier cas d'étude concerne la cellule en silicium de R.T.C France avec un diamètre de 57 mm qui a été très largement étudiée dans la littérature. Sa courbe caractéristique a été mesurée dans des conditions de température $T = 33\,^{\circ}\mathrm{C}$ et irradiance solaire $1000\,\mathrm{W\,m^{-2}}$ et comprend 26 points expérimentaux dont 20 est dans le premier quadrant (figure 4.1). Le tableau 4.1 montre le bornes de l'espace de recherche et figure 4.2 prouve que la caractéristique expérimentale et calculée par l'ED sont graphiquement quasi-identiques. Figure 4.2.b montre l'évolution de la moyenne des valeurs de fitness des vecteurs évalués par la fonction objectif dans chaque génération consécutive. On constate que dès la $50^{\text{ème}}$ génération, l'ED a pratiquement déjà convergé.

Le tableau 4.2 présente une comparaison entre l'ED et d'autre méthodes simple diode appliquées à la cellule R.T.C France. On constate que les valeurs retrouvées par l'ED sont assez proches de celles des autres travaux. En effet, l'ED parvient à une erreur RMSE de 7.7692×10^{-4} qui est supérieure aux autres techniques similaire comme les essaims particulaires [49], l'algorithme des colonies d'abeilles artificielles [50] et l'ED à trois points [38]. Les résultats les moins précis sont ceux de la méthode de Newton à moindres carrés [21] et l'algorithme génétique [50].

En ce qui concerne le modèle double diode, les paramètres sont contraints dans les limites dans le tableau 4.3. Les résultats de l'ED en double diode sont comparés avec ceux de quelques autres méthode dans le tableau 4.4.

TABLEAU 4.1 – Bornes utilisées de l'espace de recherche pour le modèle simple diode pour le cellule R.T.C France

Paramètre	R_s	R_p	a	I_0	I_{PV}
Borne supérieure	1	100	2	1×10^{-6}	10
Borne inférieure	0	2	1	1×10^{-7}	0

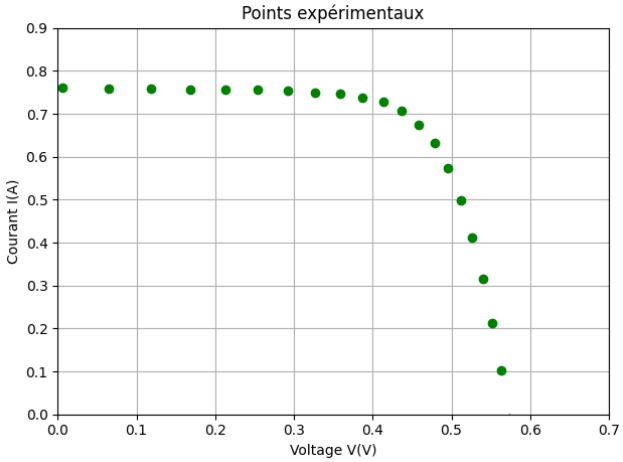
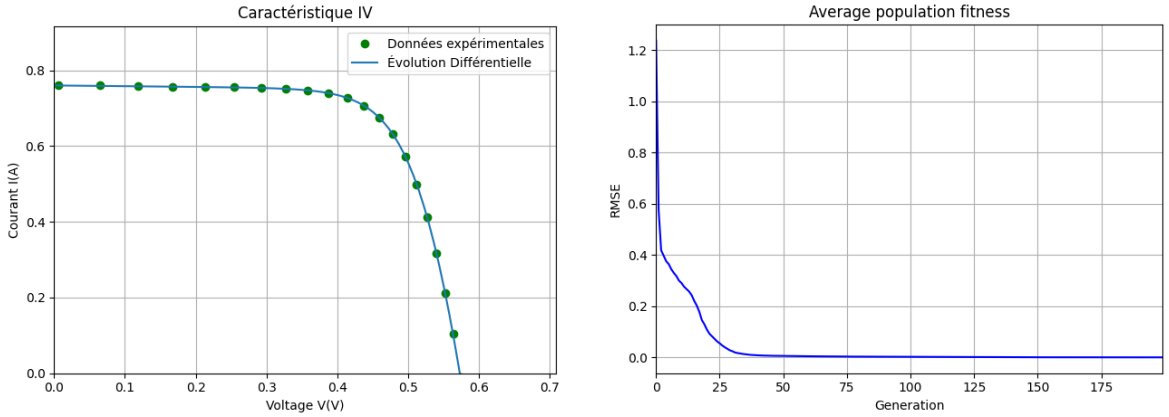


FIGURE 4.1 – Données expérimentales de la caractéristique IV de la cellule R.T.C France mesurées à 33 °C



(a) Comparaison entre la courbe expérimentale et la caractéristique calculée.

(b) Évolution de la valeur moyenne de fitness de chaque génération

FIGURE 4.2 – Résultats de l'ED appliquée sur la cellule R.T.C France 57 mm.

TABLEAU 4.2 – Comparaison de l'ED avec d'autres méthodes utilisant le modèle simple diode appliquées à la cellule R.T.C France 57 mm dans la littérature

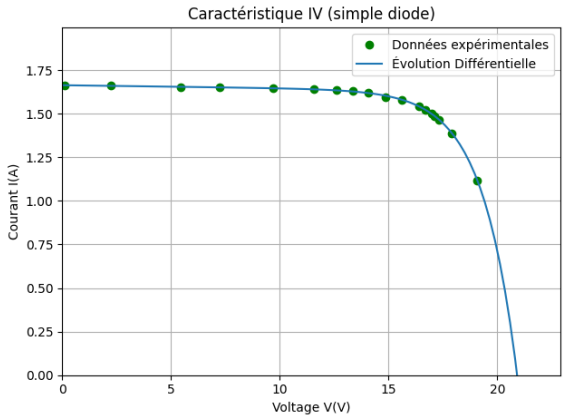
Paramètres	Référence	R_s (Ω)	R_p(Ω)	a	I_0 (μA)	I_{PV} (A)	$RMSE$
ED (simple diode)		0.0363	54.1134	1.4709	0.3209	0.7607	7.7692×10^{-4}
ED3P	[38]	0.0363	54.1924	1.4798	0.3191	0.7607	8.1291×10^{-4}
PSO	[49]	0.0363	53.8550	1.4816	0.3245	0.7607	9.8606×10^{-4}
ABC	[50]	0.0364	53.6433	1.4817	0.3251	0.7608	9.8620×10^{-4}
Newton	[21]	0.0364	53.7634	1.4837	0.3223	0.7608	9.70×10^{-3}
GA	[50]	0.0299	42.3729	1.5751	0.8087	0.7619	1.90×10^{-2}

TABLEAU 4.3 – Bornes de l'espace de recherche à 7 dimensions pour la cellule R.T.C France 57mm

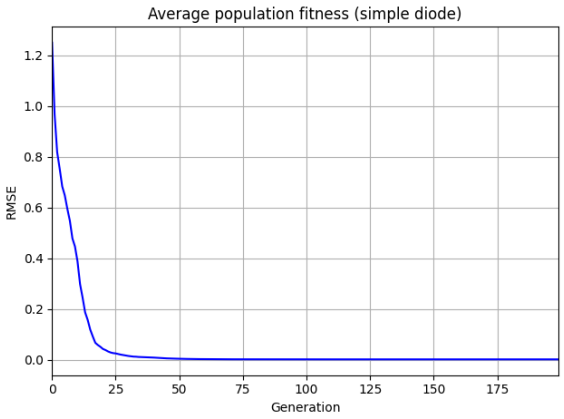
Paramètre	R_s	R_p	a_1	a_2	I_{01}	I_{02}	I_{PV}
Borne supérieure	1	100	2	2	1×10^{-4}	1×10^{-4}	10
Borne inférieure	0	2	1	1	1×10^{-7}	1×10^{-7}	0

TABLEAU 4.4 – Comparaison de l'ED avec d'autres méthodes dans la littérature utilisant le modèle double diode sur la cellule R.T.C France 57mm

Paramètres	Référence	R_s (Ω)	R_p(Ω)	a_1	a_2	I_{01} (A)	I_{02} (A)	I_{PV} (A)	*RMSE*
ED (double diode)		0.02061	51.9345	1.87579	1.43602	4.2322×10^{-7}	1.8726×10^{-7}	0.76055	7.63×10^{-4}
ABC	[50]	0.0364	53.7804	1.4495	1.4885	4.07×10^{-8}	2.874×10^{-7}	0.7608	9.861×10^{-4}
PSO	[51]	0.05861	18.2106	1.000 12	1.00091	2.8601×10^{-10}	1×10^{-12}	0.7633	8.1646×10^{-3}
GSA	[52]	0.029 14	51.116	1.6087	1.628 89	6.60621×10^{-7}	4.55149×10^{-7}	0.768 86	5.91958×10^{-3}



(a) Correspondence de l'ED aux données expérimentales. La distribution non-optimale des points n'entrave pas la convergence.



(b) Évolution de la valeur moyenne de fitness de chaque génération. L'ED est convergeante dès la 50ème itération.

FIGURE 4.3 – Résultats de l'ED appliquée sur au module Schutten Solar STM6-40/36

4.2.2 Cas 2 : Module monocristallin Schutten Solar STM6-40/36

Nous nous concernons dans ce deuxième cas d'étude du module monocristallin Schutten Solar STM6-40/36 composé de 36 cellules (156mm × 156mm) en série. Les données expérimentales ont été prises à une température de 51°C. Une comparaison des résultats de l'ED avec d'autres méthodes est présentée dans le tableau 4.5. La correspondance de la caractéristique calculée par l'ED aux points experimentaux est démontrée graphiquement dans la figure 4.3. Malgré la distribution irrégulière des points, l'ED arrive a produire une solution précise. Sa précision et de même ordre de grandeur que ED3P [38] mais elle est supérieure aux techniques des colonies des abeilles artificielles (ABC) [50], de sa version améliorée par Oliva et al. (CIABC) [53] et Chaotic Whale Optimization Algorithm (CWOA) [54].

Les résultats de l'ED avec le modèle double diode sur le module STM6-40/36 sont présenté dans le tableau 4.7 avec deux autres techniques : La colonie d'abeilles artificielle et les essaims particulaires. Le tableau 4.6 montre les bornes utilisées comme limites de l'espace de recherche. Notons la similarité des qualités des résultats du modèle simple et double diode.

TABLEAU 4.5 – Comparaison de l'ED (simple diode) avec d'autres méthodes dans la littérature sur le module STM6-40/36

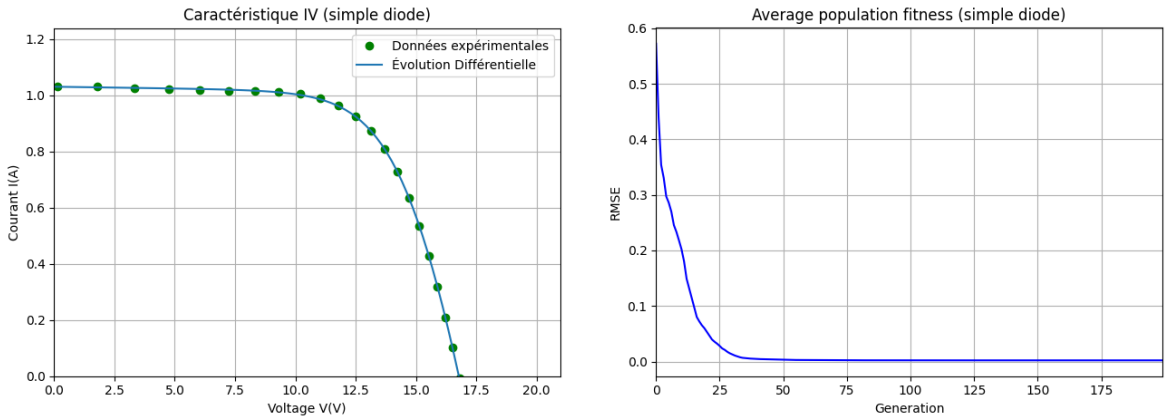
Paramètres	Référence	R_s (mΩ)	R_p(Ω)	a	I_0 (µA)	I_{PV} (A)	*RMSE*
ED (simple diode)		0.2801	16.5854	1.5571	2.8049	1.6633	1.7721×10^{-3}
ED3P	[38]	0.4186	16.7328	1.5656	2.7698	1.6632	1.7740×10^{-3}
ABC	[50]	4.99	15.206	1.4866	1.5	1.6644	1.838×10^{-3}
CIABC	[53]	4.4	15.617	1.4976	1.6642	1.6760	1.819×10^{-3}
CWOA	[54]	5	15.4	1.5	1.6338	1.7	1.800×10^{-3}

TABLEAU 4.6 – Limites de l'espace de recherche pour l'ED double diode sur le module photovoltaïque STM6-40/36

Paramètre	R_s	R_p	a_1	a_2	I_{01}	I_{02}	I_{PV}
Borne Supérieure	1	100	2	2	1×10^{-4}	1×10^{-4}	10
Borne inférieure	0	2	1	1	0	0	0

TABLEAU 4.7 – Comparaison de l'ED (double diode) avec d'autres méthodes sur le module photovoltaïque STM6-40/36

Paramètres	Référence	R_s (Ω)	R_p(Ω)	a_1	a_2	I_{01} (A)	I_{02} (A)	I_{PV} (A)	*RMSE*
ED (double diode)		0.0177	16.7050	1.9049	1.52461	1.006×10^{-6}	2.9858×10^{-6}	1.6633	1.7724×10^{-3}
ELPSO	[55]	0.0138	16.8580	1.8706	1.16648	1.670×10^{-8}	6.21092×10^{-6}	1.6648	1.8307×10^{-3}
ABC	[55]	0.03434	26.0613	1.9851	1.4687**	8.938×10^{-6}	1×10^{-12}	1.66347	2.0538×10^{-3}



(a) Caractéristique expérimentale et calculée du module polycristallin Photowatt-PWP 201.

(b) Courbe de convergence de L'ED. Notons la convergence rapide (avant < 100 itérations)

FIGURE 4.4 – Résultats de l'ED appliquée sur au module Photowatt PWP-201

TABLEAU 4.8 – Comparaison de l'ED avec d'autres méthodes simple diode dans la littérature sur le module Photowatt-PWP 201

Paramètres	Référence	R_s (mΩ)	R_p(Ω)	a	I_0 (μA)	I_{PV} (A)	*RMSE*
ED (simple diode)		0.0343	22.8238	1.3139	2.6380	1.0314	2.0529×10^{-3}
ED3P	[38]	0.0347	19.3720	1.3002	2.1247	1.0335	2.4227×10^{-3}
ISCE	[56]	0.0333	27.2772	1.3512	3.4823	1.0305	2.4251×10^{-3}
CIABC	[57]	0.0333	27.2772	1.3512	3.4822	1.0305	2.425×10^{-3}
Newton	[21]	0.0335	15.2625	1.3458	3.2876	1.0318	5.6010×10^{-1}

TABLEAU 4.9 – Comparaison de l'ED avec d'autres méthodes double diode sur le module photovoltaïque Photowatt-PWP 201

Paramètres	Référence	R_s (Ω)	R_p(Ω)	a_1	a_2	I_{01} (A)	I_{02} (A)	I_{PV} (A)	*RMSE*
ED (double diode)		0.8604	19.2098	1.2165	1.5326	1.6785×10^{-7}	2.6073×10^{-6}	1.03193	1.50208×10^{-3}
TVACPSO	[51]	1.2356	22.8236	1.3210	2.7778	2.6381×10^{-6}	1×10^{-12}	1.03143	2.0530×10^{-3}

4.2.3 Cas 3 : Module polycristallin Photowatt-PWP 201

Le 3$^{\text{ème}}$ cas concerne le module polycristallin Photowatt-PWP 201 composé de 36 cellules en série. Les données expérimentales ont été prise dans des conditions d'irradiation de 1000W m^{-2} et une température de $T = 45°$C. Le tableau 4.8 compare les résultats obtenus par Évolution Différentielle contre d'autres méthodes dans la littérature. Les valeurs des paramètres simple diode retrouvée par l'ED sont très similaires aux autres, mais sont plus précises en termes de RMSE. Figure 4.4 montre la courbe caractéristique calculée (a) et la courbe de convergence de l'ED (b). Les bornes de l'espace de recherche sont identiques à celle de la cellule R.T.C France (Tableau 4.1) sauf pour le courant de saturation qu'on limite à $1 \times 10^{-7} \leq I_0 \leq 1 \times 10^{-5}$. Le modèle double diode est plus précis que le modèle simple diode avec l'ED, mais ce dernier reste plus précis avec l'ED que les autres techniques utilisant le modèle simple diode.

4.2.4 Remarques

Le modèle double diode contient deux termes exponentiels nécessitant deux évaluations de la fonction W de Lambert pour chaque point expérimental, ce qui est relativement coûteux d'un point de vue de calcul numérique. Par ailleurs, puisque tous les paramètres sont traités indépendamment des autres, l'espace de recherche est effectivement à 7 dimensions. Puisque la qualité des résultats des modèles simple et double diode est quasi-identique dans le cas de la cellule R.T.C France (Tableaux 4.2 et 4.4 respectivement) ainsi que pour le module photovoltaïque monocristallin Schutten Solar STM6-40/36 (Tableaux 4.5 et 4.7), on constate que le modèle simple diode et très adéquat en terme de précision et supérieur en terme d'efficacité et rapidité de calcul.

4.3 Analyse et cohérence de l'ED

La performance supérieure démontrée par l'ED par rapport aux autres algorithmes provient probablement de ses capacités à la *recherche globale* et l'auto-adaptation permise par les vecteurs de différences qui permettent au début d'explorer tout l'espace de recherche, et se raffinent progressivement pour optimiser la recherche locale dans la région du minimum global. L'existence d'une multitude de minimums locaux est démontrée dans la figure 4.5 où on a projeté l'espace de recherche 5-dimensionnel sur 2 dimensions au voisinage du minimum global. On fait varier le facteur d'idéalité a et la résistance série R_s dont le modèle est très sensible aux variations. Les trois autres paramètres R_p, I_0 et I_{PV} sont fixés sur les valeurs du minimum global retrouvé par l'ED comme dans le tableau 4.2.

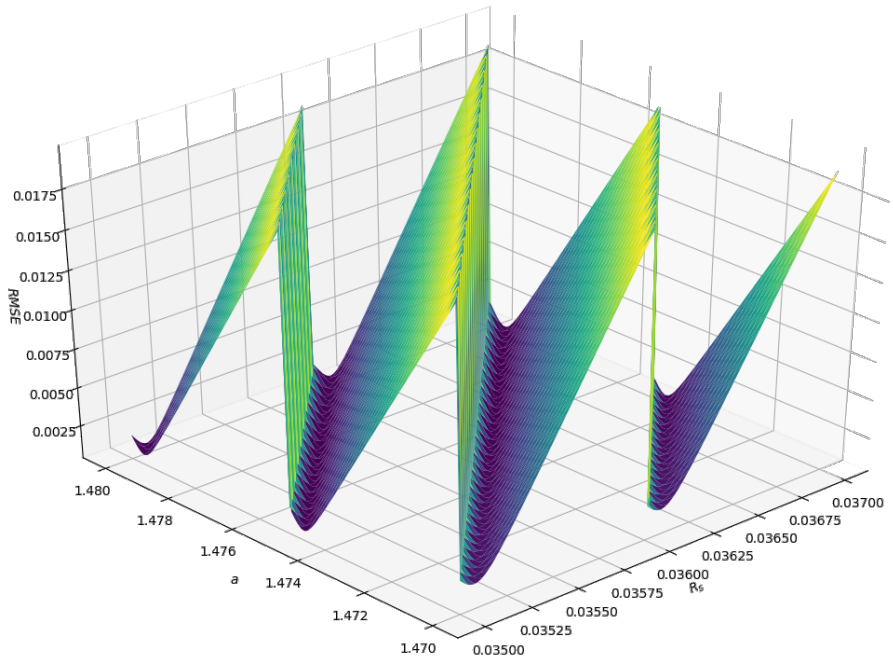
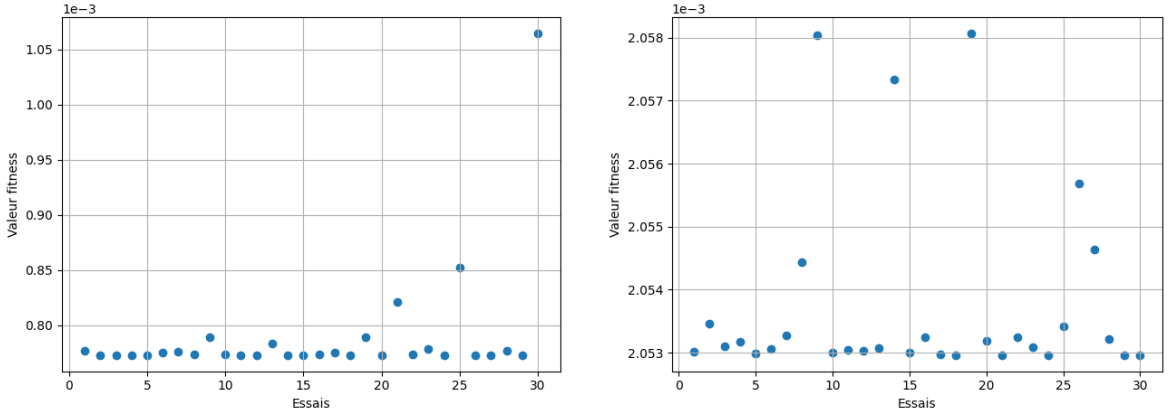


FIGURE 4.5 – Le voisinage du minimum global retrouvé par l'ED selon le facteur d'idéalité et la résistance en série. Notons l'existence des "vallées" qui comprennent potentiellement plusieurs minimums locaux

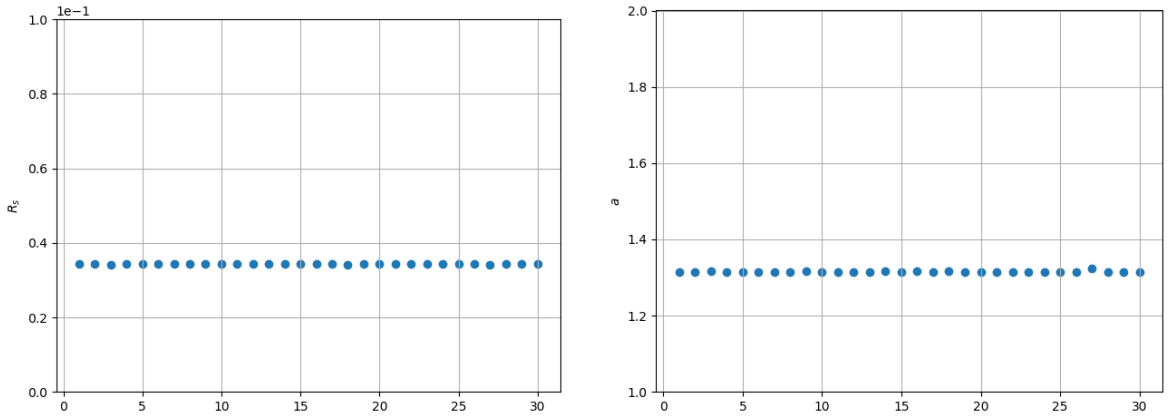
La nature stochastique de l'Évolution Différentielle fait qu'elle donne des résultats différents après chaque essai. Ceci impose une analyse de cohérence de la méthode pour estimer la fiabilité de l'ED pendant plusieurs essais consécutifs. Figure 4.6 montre la RMSE de la solution finale dans 30 essais indépendants de l'ED. Tous les points sont localisés dans une région très concentrée de l'espace de recherche ce qui indique que l'ED parvient effectivement à localiser le minimum global. La cohérence des différents essais concernant la R_s et le a du module Photowatt-PWP 201 (Figure 4.7) et les écart-types des paramètres montrés sur le tableau 4.10 confirment ceci puisque ils peuvent être interprétés comme un indice de "stabilité" de l'algorithme qui quantifie sa capacité a reproduire les mêmes résultats. Tous les écart-types sont de l'ordre de 10^{-3} ou moins, donc une solution précise est garantie dans n'importe quel essai.



(a) Cellule R.T.C France 57 mm

(b) Module Photowatt-PWP 201

FIGURE 4.6 – Les RMSEs obtenues lors de 30 essais indépendants



(a) Résistance série R_s (b) Facteur d'idéalité a

FIGURE 4.7 – Résultats obtenus pour le facteur d'idéalité et la résistance série du module Potowatt-PWP 201 pendant 30 essais indépendants

TABLEAU 4.10 – Valeurs Moyennes de quelques paramètres influents de la cellule R.T.C France 57 mm et les écart-types associés des 30 essais

Paramètre	Valeur Moyenne	Écart-Type
$RMSE$	7.8925×10^{-4}	5.3670×10^{-5}
R_s	3.6355×10^{-2}	3.8936×10^{-4}
a	1.4722	5.7539×10^{-3}
I_0	3.2655×10^{-7}	3.4071×10^{-8}

4.4 Analyse de la stratégie métaheuristique

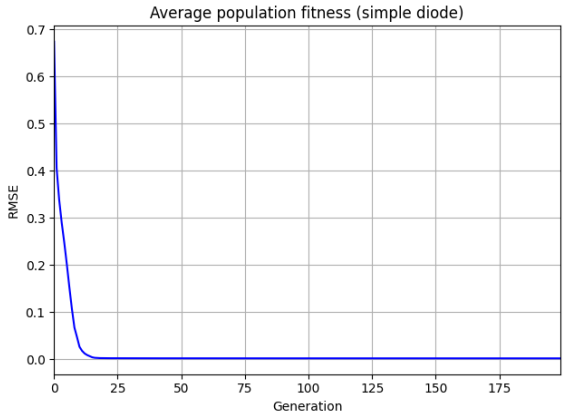
Pour analyser la stratégie métaheuristique proposée dans le dernier chapitre, nous allons prendre le module polycristallin Photowatt-PWP 201 et la cellule R.T.C France 57mm. Conformément aux recommandations de Storn et Price, nous avons choisi comme valeurs standard du facteur de mutation et du taux de croisement 0.7 et 0.8, respectivement. En fait les paramètres retrouvés dans le cas d'étude 3 avec les courbes caractéristique et de convergence, ont tous été retrouvé par l'ED configuré avec ces valeurs par défaut. Pour appliquer la métaheuristique, un nombre de population relativement faible est adéquat puisque notre espace de recherche est 2-dimensionnel : $N_P^{\text{higher order}} = 10 \times D = 20$. On limite aussi le nombre maximal de générations à $Gen_{max}^{\text{higher order}} = 50$. Pratiquement, la fonction objectif est donc la valeur fitness *à la* $50^{ème}$ *itération*. Le fait que l'ED risque de ne pas avoir déjà convergé ne nous pose pas de problèmes, puisque notre but est de retrouver les valeurs de F et CR permettant la convergence la plus rapide [1] que possible. Les paramètres de la métaheuristique elle-même sont $F^{\text{higher order}} = 0.8$ et $CR^{\text{higher order}} = 0.8$. Les résultats de cette stratégies sont :

$$F = 0.55, \quad CR = 0.94 \quad \text{Module Photowatt-PWP 201} \tag{4.1}$$

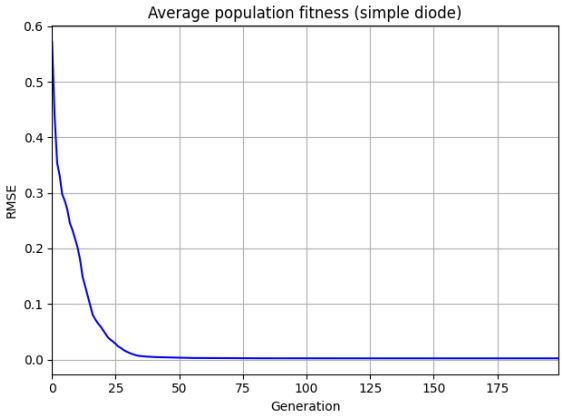
$$F = 0.55, \quad CR = 0.88 \quad \text{Cellule R.T.C France 57mm} \tag{4.2}$$

Les figures 4.8 et 4.9 comparent les courbes de convergences entre l'ED avec et sans stratégie métaheuristique. Malgré le fait que le facteur de mutation retrouvé est relativement relativement inférieur aux recommandations de Storn et Price, il est toujours compris dans les limites de Zaharie et al. [46]. Notons que la rapidité de convergence est sensiblement supérieure dans les figures 4.8 et 4.9, l'ED a déjà convergé à l'itération 25, alors que l'ED standard a besoin de plus de deux fois plus d'itérations [2]. En terme de temps de calcul, on teste dans les mêmes conditions les deux méthodes

1. Convergence rapide mais non-prematurée
2. Itérations et générations sont équivalentes dans ce cas

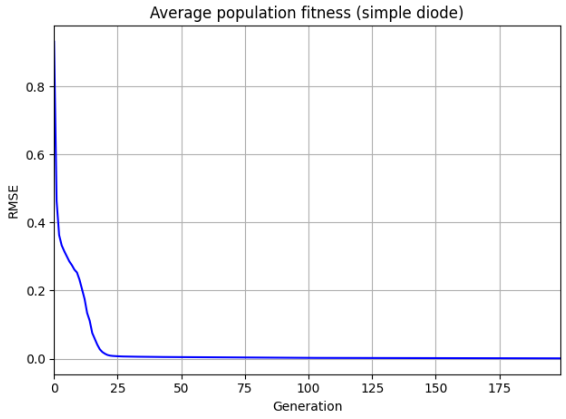


(a) Convergence de l'ED configurée avec les parametres retrouvée par la métaheuristique

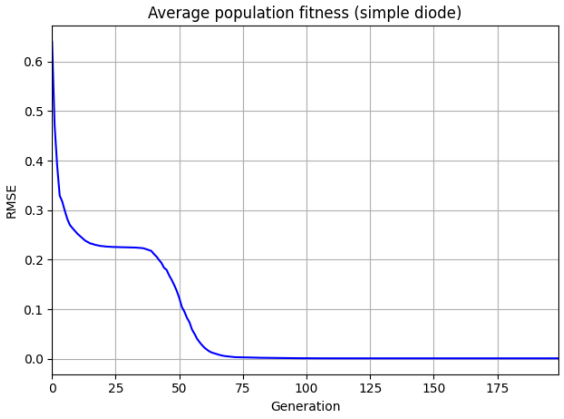


(b) Convergence de l'ED avec les paramètres standard

FIGURE 4.8 – Comparaison de l'ED avec et sans métaheuristique sur le module Photowatt-PWP 201



(a) Convergence de l'ED configurée avec les parametres retrouvée par la métaheuristique



(b) Convergence de l'ED avec les paramètres standard

FIGURE 4.9 – Comparaison de l'ED avec et sans métaheuristique sur la cellule R.T.C France 57 mm

en les "chronométrant" avec script Python. L'ED standard s'exécute au bout d'un temps moyen de 1.68535s sur 30 essais indépendants et un écart-type de 4.751×10^{-4}. Pour l'ED avec métaheuristique, le temps moyen d'exécution est 0.82392s, ce qui correspond à une diminution de 51% du temps de calcul. Il n'y a pas de compromis envers la stabilité et la cohérence puisque l'écart-type des RMSEs est 4.0162×10^{-4}.

4.5 Conclusion

Après avoir étudié la performance de l'ED sur la cellule et les deux modules photovoltaïques, on peut donc conclure que l'ED avec la fonction W de Lambert est plus précise que les autres techniques disponibles dans la littérature. La mutations différentielle apparaît très adaptée aux genres d'espaces de recherche présentés par les courbes caractéristique des cellules solaires. On a aussi utilisé une évolution différentielle à un ordre supérieur pour déterminer les paramètres optimaux des ED pour les modèles à diodes, ce qui a amélioré la convergence and réduit le temps de calcul.

Conclusion générale

Dans ce travail, nous avons étudié une approche permettant de modéliser le comportement électrique des cellules solaires photovoltaïques en estimant les paramètres des modèles d'éléments localisés à diodes. Les modèles simple et double diodes ont été présenté avec le rôle et l'influence de chaque paramètre sur la caractéristique IV du modèle. On a ensuite proposé de formuler l'extraction des paramètres en problème d'optimisation mathématique en se basant sur l'évolution différentielle qui est une technique évolutionnaire relativement récente. La fonction W de Lambert a été utilisée pour permette la reconstruction des caractéristique I-V à partir des paramètres du modèle et donc de calculer l'erreur quadratique moyenne qu'on utilise comme fonction objectif guidant la sélection des solutions de l'évolution différentielle. Nous avons aussi proposé une stratégie pour déterminer les paramètres de contrôle optimaux de l'évolution différentielle en l'utilisant à un ordre supérieur.

Un outil informatique a été développé sous le langage de programmation Python et ses bibliothèques scientifiques. Son interface programmatique a été utilisée pour appliquer cette technique sur une cellule photovoltaïque très largement étudiée dans la littérature (R.T.C France 57mm) et deux modules photovoltaïques : le monocristallin Schutten Solar STM6-40/36 et polycristallin Photowatt-PWP 201. On a pu constater que l'évolution différentielle avec la fonction W Lambert est plus précises que les autres techniques évolutionnaires similaires de l'état de l'art. En raison de la nature stochastique de cette méthode, on a dû effectuer une analyse de stabilité et de cohérence qui a montré que l'évolution différentielle converge systématiquement vers le minimum global. De plus, la stratégie métaheuristique qu'on a proposé a permit de réduire considérablement le temps de calcul en retrouvant les paramètres de contrôle optimaux pour une convergence encore plus rapide nécessitant quelques dizaines d'itérations seulement.

Les circuit électriques à diodes parviennent à modéliser les cellules photovoltaïques grâce à leurs jonctions PN sur lesquelles se base toute le principe de conversion d'énergie solaire en énergie électrique. Cependant, les technologies émergentes en photovoltaïques commencent à explorer d'autres méthodes de conversion parfois même contournant le besoin d'une jonction PN. La modélisation par circuit à diode est bien évidemment inadéquate pour ce genre de cellules, elles nécessiterons des modèles plus spécialisés, tenant en compte le mécanisme principal de séparation des charges, des propriétés opto-électroniques des matériaux utilisés et de tout un ensemble de phénomènes physiques affectant le comportement électrique de la cellule. Les travaux de recherches futurs dans ce domaine devraient évaluer la viabilité des circuits à éléments localisés pour ces nouvelles technologies. L'application des techniques évolutionnaires dépend du fait que le nombre de paramètres à optimiser soit limité, puisque l'optimisation sur des espaces de recherche avec un nombre considérable de dimensions est toujours très coûteuse en temps et en capacité de calcul, ce qui rends la convergence vers le minimum global beaucoup plus difficile et moins garantie.

Annexe A

Évolution différentielle avec Python

Tout le code est disponible en ligne. Pour le telecharger, il suffit d'executer `git clone https://github.com/nightowl97/DEPV.git`.

```python
import csv
import matplotlib.pyplot as plt
import numpy as np
import scipy.constants as constants
from scipy.special import lambertw


# Reads experimental IV data from a csv file
def read_csv(filename):
    with open(filename) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        next(csv_reader)  # Ignore the header
        data = [(float(row[0]), float(row[1])) for row in csv_reader]
        voltages, currents = np.array([row[0] for row in data]), np.array([row[1] for row in data])
        return voltages, currents


class DE:

    def __init__(self, bounds, ivdata, Ns, Np, temp, popsize=100, maxiter=200, mutf=0.7, crossr=0.8):
        """
        :param bounds: Dictionary of bounds in this form {'rp': [lower, upper], 'rs': [lower, upper] ...}
        :param ivdata: [Voltages, Currents]
        :param Ns: Number of cells in series
        :param Np: Number of cells in parallel
        :param temp: Temperature
        :param popsize: Population size
        :param maxiter: Maximum number of generations
        :param mutf: Mutation Factor F
        :param crossr: Crossover Rate CR
        """
        self.bounds = bounds  #
        self.popsize = popsize
        self.maxiter = maxiter
        self.mutf = mutf
        self.crossr = crossr
        self.dim = len(bounds)  # Dimensions of the search space
        self.populations = self.maxiter * [self.popsize * [None]]  # Initial populations
        self.fitnesses = self.maxiter * [self.popsize * [100]]  # Initial fitnesses
        self.ivdata = ivdata
        self.Ns = Ns
        self.Np = Np
        self.final_res = (None, None)  # Final result containing (vector, fitness)
        self.temp = temp

    # Main differential evolution algorithm
    def solve(self):
        vth = constants.Boltzmann * self.temp / constants.elementary_charge
        # Initial uniform distribution
        self.populations[0] = np.random.uniform([i[0] for i in self.bounds.values()],
                                                [j[1] for j in self.bounds.values()], size=(self.popsize,
                                                ↪  self.dim))
        for gen in range(self.maxiter):
            # Initial fitness
            self.fitnesses[gen] = [DE.objf(vec, self.ivdata, vth, self.Ns, self.Np) for vec in
            ↪  self.populations[gen]]

            fittest_index = int(np.argmin(self.fitnesses[gen]))
            fittest = self.populations[gen][fittest_index]
```

```
58
59              # Iterate over population
60              for i in range(self.popsize):
61                  # Generate donor vector from 2 random vectors and the fittest
62                  indices = [ind for ind in range(self.popsize) if ind != i and ind != fittest_index]
63                  i1, i2 = np.random.choice(indices, 2, replace=False)
64
65                  # DE mutation
66                  mutant = fittest + self.mutf * (self.populations[gen][i1] - self.populations[gen][i2])
67
68                  # Crossover
69                  crosspoints = np.random.rand(self.dim) <= self.crossr
70                  if not any(crosspoints):
71                      crosspoints[np.random.randint(0, self.dim)] = True
72                  trial = np.where(crosspoints, mutant, self.populations[gen][i])
73
74                  # Penalty
75                  for j, key in enumerate(self.bounds.keys()):
76                      lower, upper = self.bounds[key]
77                      # Solution vectors need to be [rp, rs, a, i0, iph]
78                      if not lower <= trial[j]:
79                          trial[j] = lower + np.random.rand() * (upper - lower)
80                      if not upper >= trial[j]:
81                          trial[j] = upper - np.random.rand() * (upper - lower)
82
83                  # Selection
84                  trialf = self.objf(trial, self.ivdata, vth, self.Ns, self.Np)
85
86                  if gen + 1 < self.maxiter:
87                      if trialf < self.fitnesses[gen][i]:
88                          #  Select for next generation
89                          self.populations[gen + 1][i] = trial
90                          self.fitnesses[gen + 1][i] = trialf
91                      else:
92                          self.populations[gen + 1][i] = self.populations[gen][i]
93
94          # setting the final result
95          fittest_index = int(np.argmin(self.fitnesses[-1]))
96          result = self.populations[-1][fittest_index]
97          result_fitness = self.fitnesses[-1][fittest_index]
98          assert result_fitness == np.min(self.fitnesses[-1][fittest_index])
99          self.final_res = result, result_fitness
100
101     @staticmethod
102     def objf(vector, exp_data, vth, Ns, Np):
103         # If vector is none, max fitness
104         if vector is None:
105             return 100
106         voltages = exp_data[0]
107         ical = DE.i_from_v(vector, voltages, vth, Ns, Np)
108         delta = ical - exp_data[1]  # compare currents for each voltage value
109         # Fitness penalty (J = 100) for unphysical values of Rs and Rp
110         if vector[0] < 0 or vector[1] < 0:
111             return 100
112         return np.sqrt(np.mean(delta ** 2))
113
114     @staticmethod
115     def i_from_v(vector, v, vth, Ns, Np):
116         # Takes a solution vector of 5/7 params and a voltage to calculate the current
117         if vector is not None:
118             if len(vector) != 5 and len(vector) != 7:
119                 print("V:{}".format(vector))
120                 print("Vector has {} parameters! needs 5 or 7.\n Exiting..".format(len(vector)))
121                 exit(0)
122
123         output_is_scalar = np.isscalar(v)
124
125         # For numerical stability: Gsh=1/Rsh, so Rsh=np.inf ==> Gsh=0
126         conductance_shunt = 1. / (vector[0] * (Ns / Np))
127         #  Single diode
128         if len(vector) == 5:
129             resistance_shunt, resistance_series, n, saturation_current, photocurrent = vector
130
131             Gsh, Rs, a, V, I0, IL = np.broadcast_arrays(conductance_shunt, resistance_series * (Ns / Np),
132                                                         n * Ns * vth, v, saturation_current * Np,
133                                                         photocurrent * Np)
134
135             # Initialize output I (V might not be float64)
```

```
        current = np.full_like(V, np.nan, dtype=np.float64)  # noqa: E741, N806

        # Determine indices where 0 < Rs requires implicit model solution
        idx_p = 0. < Rs

        # Determine indices where 0 = Rs allows explicit model solution
        idx_z = 0. == Rs

        # Explicit solutions where Rs=0
        if np.any(idx_z):
            current[idx_z] = IL[idx_z] - I0[idx_z] * np.expm1(V[idx_z] / a[idx_z]) - \
                      Gsh[idx_z] * V[idx_z]

        # possibility of overflow
        if np.any(idx_p):
            # LambertW argument
            argW = Rs[idx_p] * I0[idx_p] / (a[idx_p] * (Rs[idx_p] * Gsh[idx_p] + 1.)) * \
                   np.exp((Rs[idx_p] * (IL[idx_p] + I0[idx_p]) + V[idx_p]) /
                          (a[idx_p] * (Rs[idx_p] * Gsh[idx_p] + 1.)))

            # lambertw typically returns complex value with zero imaginary part
            # may overflow to np.inf
            lambertwterm = lambertw(argW).real

            # Jain and Kapoor, 2004
            current[idx_p] = (IL[idx_p] + I0[idx_p] - V[idx_p] * Gsh[idx_p]) / \
                             (Rs[idx_p] * Gsh[idx_p] + 1.) - (a[idx_p] / Rs[idx_p]) * lambertwterm

        if output_is_scalar:
            return current.item()
        else:
            return current

    # Double diode
    if len(vector) == 7:
        resistance_shunt, resistance_series, n1, n2, saturation_current1, saturation_current2, photocurrent =
        ↪  vector
        Gsh, Rs, a1, a2, V, I01, I02, IL = np.broadcast_arrays(conductance_shunt, resistance_series,
                                                               n1 * Ns * vth, n2 * Ns * vth, v,
                                                               saturation_current1 * Np, saturation_current2
                                                               ↪  * Np,
                                                               photocurrent * Np)
        # Initialize output I (V might not be float64)
        current = np.full_like(V, np.nan, dtype=np.float64)
        # Determine indices where 0 < Rs requires implicit model solution
        idx_p = 0. < Rs

        # Determine indices where 0 = Rs allows explicit model solution
        idx_z = 0. == Rs

        # Explicit solutions where Rs=0
        if np.any(idx_z):
            current[idx_z] = IL[idx_z] - I01[idx_z] * np.expm1(V[idx_z] / a1[idx_z]) - \
                             I02[idx_z] * np.expm1(V[idx_z] / a2[idx_z]) - \
                             Gsh[idx_z] * V[idx_z]

        # Only compute using LambertW if there are cases with Rs>0
        # Does NOT handle possibility of overflow, github issue 298
        if np.any(idx_p):
            # LambertW arguments, cannot be float128, may overflow to np.inf
            argw1 = (Rs[idx_p] * (I01[idx_p] + I02[idx_p]) / (a1[idx_p] * (Gsh[idx_p] * Rs[idx_p] + 1))) * \
                    np.exp((Rs[idx_p] * (IL[idx_p] + I01[idx_p] + I02[idx_p]) + V[idx_p]) /
                           (a1[idx_p] * (Gsh[idx_p] * Rs[idx_p] + 1))
            )
            argw2 = (Rs[idx_p] * (I01[idx_p] + I02[idx_p]) / (a2[idx_p] * (Gsh[idx_p] * Rs[idx_p] + 1))) * \
                    np.exp((Rs[idx_p] * (IL[idx_p] + I01[idx_p] + I02[idx_p]) + V[idx_p]) /
                           (a2[idx_p] * (Gsh[idx_p] * Rs[idx_p] + 1))
            )
            # lambertw typically returns complex value with zero imaginary part
            # may overflow to np.inf
            lambertwterm1 = lambertw(argw1).real
            lambertwterm2 = lambertw(argw2).real

            # Eqn. 2 in Jain and Kapoor, 2004
            #  I = -V/(Rs + Rsh) - (a/Rs)*lambertwterm + Rsh*(IL + I0)/(Rs + Rsh)
            # Recast in terms of Gsh=1/Rsh for better numerical stability.
            # Eqn. 2 in Jain and Kapoor, 2004
            #  I = -V/(Rs + Rsh) - (a/Rs)*lambertwterm + Rsh*(IL + I0)/(Rs + Rsh)
```

```
                # Recast in terms of Gsh=1/Rsh for better numerical stability.
                current[idx_p] = (IL[idx_p] + I01[idx_p] + I02[idx_p] - (V[idx_p] * Gsh[idx_p])) / \
                                 (Gsh[idx_p] * Rs[idx_p] + 1) - (a1[idx_p] / (2 * Rs[idx_p])) * lambertwterm1 - \
                                 (a2[idx_p] / (2 * Rs[idx_p])) * lambertwterm2

            if output_is_scalar:
                return current.item()
            else:
                return current

    # PLOTTING
    # Plots a given vector solution
    def plot_solution(self, vector):
        vth = constants.Boltzmann * self.temp / constants.elementary_charge
        # Plot experimental points
        f, gr = plt.subplots()
        voltages, currents = self.ivdata
        xlim, ylim = max(voltages), max(currents)
        gr.plot(voltages, currents, 'go', label="Données expérimentales")

        # Calculate given solution adding 20% voltage axis length
        v = np.linspace(0, xlim + (.2 * xlim), 100)
        calc_current = DE.i_from_v(vector, v, vth, self.Ns, self.Np)

        # print("Solution:")
        # d = [print(str(i) + "\n") for i in zip(v, calc_current)]

        # Plotting (evil muuuhahahahaa)
        if self.dim == 7:
            gr.title.set_text("Caractéristique IV (double diode)")
        else:
            gr.title.set_text("Caractéristique IV (simple diode)")
        gr.plot(v, calc_current, label="Évolution Différentielle")
        gr.set_xlabel("Voltage V(V)")
        gr.set_ylabel("Courant I(A)")
        gr.grid()
        gr.legend()
        gr.set_xlim((0, xlim + (.2 * xlim)))
        gr.set_ylim((0, ylim + (.2 * ylim)))
        plt.show()
        return f, gr

    def plot_fit_hist(self):
        assert self.maxiter == len(self.fitnesses)
        averages = [np.average(generation) for generation in self.fitnesses]
        # print("FITNESS HISTORY:")
        # [print(average) for average in averages]
        f, gr = plt.subplots()
        if self.dim == 7:
            gr.title.set_text("Average population fitness (double diode)")
        else:
            gr.title.set_text("Average population fitness (simple diode)")
        gr.plot(averages, 'b')
        gr.grid()
        gr.set_xlabel("Generation")
        gr.set_ylabel("RMSE")
        gr.set_xlim((0, self.maxiter - 1))
        plt.show()
        return f, gr

    # Plots the best solution vector
    def plot_result(self, print_params=False):
        # plots the best solution
        vth = constants.Boltzmann * self.temp / constants.elementary_charge
        result, result_fitness = self.final_res
        graph = self.plot_solution(result)
        if print_params:
            if self.dim == 5:
                print("Rsh = {}\nRs = {}\na = {}\nI0 = {}\nIpv = {}".format(*result))
            elif self.dim == 7:
                print("Rsh = {}\nRs = {}\na1 = {}\na2 = {}\nI01 = {}\nI02 = {}\nIpv = {}".format(*result))
            else:
                raise ValueError("Search space dimensionality must be either 5 for single or 7 for double
                ↪  diodes")
            print("Root Mean Squared Error:\t{0:.5E}".format(result_fitness))
        return graph
```

Annexe B

Code de l'interface graphique

main.py :

```
import os
import matplotlib
from kivy.app import App
from kivy.lang import Builder
from plyer import filechooser
from kivy.config import Config
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.properties import ObjectProperty
from objects import DE, read_csv
import matplotlib.pyplot as plt
# from kivy.garden.matplotlib.backend_kivyagg import FigureCanvas

matplotlib.use('module://kivy.garden.matplotlib.backend_kivy')
Config.set('kivy', 'exit_on_escape', '1')


class ParamWindow(Screen):
    rplower = ObjectProperty(None)
    rpupper = ObjectProperty(None)
    alower = ObjectProperty(None)
    aupper = ObjectProperty(None)
    i0lower = ObjectProperty(None)
    i0upper = ObjectProperty(None)
    ipvlower = ObjectProperty(None)
    ipvupper = ObjectProperty(None)
    rslower = ObjectProperty(None)
    rsupper = ObjectProperty(None)
    tempc = ObjectProperty(None)
    series = ObjectProperty(None)
    parallel = ObjectProperty(None)
    path = ""

    def browse_files(self):
        try:
            self.path = filechooser.open_file(path=os.getcwd(), title="Pick a CSV file..",
                                              filters=[("Comma-separated Values", "*.csv")])[0]
        except TypeError:
            self.path = ""

    def calculate(self, root_manager):
        # Construct bounds dict
        bounds = {'rp': [float(self.rplower.text), float(self.rpupper.text)],
                  'rs': [float(self.rslower.text), float(self.rsupper.text)],
                  'a':   [float(self.alower.text), float(self.aupper.text)],
                  'i0':  [float(self.i0lower.text), float(self.i0upper.text)],
                  'ipv': [float(self.ipvlower.text), float(self.ipvupper.text)]}
        Ns, Np = int(self.series.text), int(self.parallel.text)

        temperature = float(self.tempc.text) + 275.15
        algo = DE(bounds, [*read_csv(self.path)], Ns, Np, temperature)
        algo.solve()
        f1, ax1 = algo.plot_fit_hist()
        canvas1 = f1.canvas
        f2, ax2 = algo.plot_result()
        canvas2 = f2.canvas
        root_manager.results_scrn.graphid1.add_widget(canvas1)
        root_manager.results_scrn.graphid2.add_widget(canvas2)

        sol_vec, fitness = algo.final_res
```

```
        root_manager.results_scrn.set_results(*sol_vec, fitness)


class ResultsWindow(Screen):

    rp_label = ObjectProperty(None)
    rs_label = ObjectProperty(None)
    a_label = ObjectProperty(None)
    i0_label = ObjectProperty(None)
    ipv_label = ObjectProperty(None)
    rmse_label = ObjectProperty(None)
    graphid1 = ObjectProperty(None)
    graphid2 = ObjectProperty(None)

    def set_results(self, rp, rs, a, i0, ipv, rmse):
        self.rp_label.text = str(rp)
        self.rs_label.text = str(rs)
        self.a_label.text = str(a)
        self.i0_label.text = str(i0)
        self.ipv_label.text = str(ipv)
        self.rmse_label.text = str(rmse)

    def go_back(self):
        # clear box layouts
        self.graphid1.clear_widgets()
        self.graphid2.clear_widgets()


class WindowManager(ScreenManager):
    # results_scrn = ObjectProperty(None)
    pass


kv = Builder.load_file("main.kv")


class MainApp(App):
    def build(self):
        return kv


if __name__ == "__main__":
    MainApp().run()
```

main.kv :

```
# Filename: main.kv

WindowManager:
    id: window_manager
    results_scrn: results_screen
    ParamWindow:
        id: param_screen
    ResultsWindow:
        id: results_screen

<ParamWindow>:
    name: "params"
    rplower: rplower
    rpupper: rpupper
    rslower: rslower
    rsupper: rsupper
    alower: alower
    aupper: aupper
    i0lower: i0lower
    i0upper: i0upper
    ipvlower: ipvlower
    ipvupper: ipvupper
    tempc: tempc
    series: series
    parallel: parallel

    canvas.before:
        Color:
            rgba: 0.5, 0.5, 0.5, 1
        Rectangle:
```

```
            pos: self.pos
            size: self.size

    GridLayout:
        cols:1
        padding: 40
        spacing: 10
        size: root.width, root.height

        GridLayout:
            cols: 2
            Label:
            GridLayout:
                cols: 2

                Label:
                    text: "Lower Bound"

                Label:
                    text: "Upper Bound"

            Label:
                text: "Shunt Resistance Rp: "

            GridLayout:
                cols: 2
                TextInput:
                    multinline:False
                    text: "2"
                    halign: "center"
                    id: rplower
                TextInput:
                    multinline:False
                    text: "100"
                    halign: "center"
                    id: rpupper

            Label:
                text: "Series Resistance Rs: "

            GridLayout:
                cols: 2
                TextInput:
                    multinline:False
                    text: "0"
                    halign: "center"
                    id: rslower
                TextInput:
                    multinline:False
                    text: "1"
                    halign: "center"
                    id: rsupper

            Label:
                text: "Ideality Factor a: "

            GridLayout:
                cols: 2
                TextInput:
                    multinline:False
                    text: "1"
                    halign: "center"
                    id: alower
                TextInput:
                    multinline:False
                    text: "2"
                    halign: "center"
                    id: aupper

            Label:
                text: "Saturation Current I0: "

            GridLayout:
                cols: 2
                TextInput:
                    multinline:False
                    text: "1e-07"
                    halign: "center"
```

```
                    id: i0lower
                TextInput:
                    multinline:False
                    text: "1e-04"
                    halign: "center"
                    id: i0upper

            Label:
                text: "Photocurrent Ipv: "

            GridLayout:
                cols: 2
                TextInput:
                    multinline:False
                    text: "0"
                    halign: "center"
                    id: ipvlower
                TextInput:
                    multinline:False
                    text: "10"
                    halign: "center"
                    id: ipvupper

            Label:
                text: "Temperature (°C): "
            TextInput:
                multinline:False
                text: "33"
                halign: "center"
                id: tempc

            Label:
                text: "Number of cells in series"

            TextInput:
                multinline:False
                text: "1"
                halign: "center"
                id: series

            Label:
                text: "Number of cells in parallel"

            TextInput:
                multinline:False
                text: "1"
                halign: "center"
                id: parallel

            Label:
                text: "I-V curve file location: "

            Button:
                text: "Select File"
                on_release: root.browse_files()

        GridLayout:
            cols:3

            Label:
            Label:
            Label:
            Label:

            Button:
                text:"Calculate"
                on_release:
                    root.manager.current = "results"
                    root.manager.transition.direction = "left"
                    root.calculate(root.manager)

            Label:
            Label:
            Label:
            Label:

<ResultsWindow>:
    name: "results"
```

```
    graphid1: graphid1
    graphid2: graphid2
    rp_label: rp_label
    rs_label: rs_label
    a_label: a_label
    i0_label: i0_label
    ipv_label: ipv_label
    rmse_label: rmse_label

    canvas.before:
        Color:
            rgba: 0.5, 0.5, 0.5, 1
        Rectangle:
            pos: self.pos
            size: self.size


    GridLayout:
        cols: 2
        id: main_results_grid
        padding: 40
        spacing: 10

        BoxLayout:
            id: graphid1

        GridLayout:
            cols: 2
            id: result_grid

            Label:
                text: "Shunt Resistance"
            Label:
                id: rp_label
                text: "0"

            Label:
                text: "Series Resistance: "
            Label:
                id: rs_label
                text: "0"

            Label:
                text: "Ideality Factor"
            Label:
                id: a_label
                text: "1"

            Label:
                text: "Saturation Current"
            Label:
                id: i0_label
                text: "0"

            Label:
                text: "Photocurrent"
            Label:
                id: ipv_label
                text: "0"

            Label:
                text: "Root Mean Squared Error"
            Label:
                id: rmse_label
                text: "0"

        BoxLayout:
            id: graphid2

        Button:
            size_hint: 0.2, 0.1
            pos_hint: {'center_x': 0.5, 'bottom': 0.99}
            text: "Go Back"
            on_release:
                root.manager.current = "params"
                root.manager.transition.direction = "right"
                root.go_back()
```

Annexe C

Code de la stratégie métaheuristique

```
from objects import DE, read_csv
import numpy as np
import matplotlib.pyplot as plt

# Example data
b = {'rp':  [2, 100],
     'rs':  [0, 1],
     'a':   [1, 2],
     'i0':  [1e-07, 1e-04],
     'ipv': [0, 10]}
expdata = read_csv("data/RTC33D1000W.csv")
T = 33 + 275.15

iternum = 50
popsize = 20
F = 0.8
CR = 0.8

populations = np.full((iternum, popsize, 2), None, dtype=np.float64)
fitnesses = np.full((iternum, popsize), 100, dtype=np.float64)
fitness_hist = []
avg_fit_hist = []


def get_final_res(vec):
    de_obj = DE(b, expdata, 1, 1, T, popsize=50, maxiter=50,
                mutf=vec[0], crossr=vec[1])
    de_obj.solve()
    return de_obj.final_res[1]


# initial population
populations[0] = np.random.uniform([0, 0], [1, 1], size=(popsize, 2))
fitnesses[0] = [get_final_res(vec) for vec in populations[0]]


for gen in range(iternum):
    print(gen)

    fittest_index = np.argmin(fitnesses[gen])
    fittest = populations[gen, fittest_index]

    fitness_hist.append(fitnesses[gen, fittest_index])
    avg_fit_hist.append(np.average(fitnesses[gen]))

    # iterate over pop
    for i in range(popsize):
        # Generate donor vector from 2 random vectors and the fittest
        indices = [ind for ind in range(popsize) if ind != i and ind != fittest_index]
        i1, i2 = np.random.choice(indices, 2, replace=False)

        # mutation
        mutant = fittest + F * (populations[gen, i1] - populations[gen, i2])
        # crossover
        crosspoints = np.random.rand(2) <= CR
        if not any(crosspoints):
            crosspoints[np.random.randint(0, 2)] = True
        trial = np.where(crosspoints, mutant, populations[gen, i])

        # penalty
```

```
        if trial[0] <= 0 or trial[0] > 1:
            trial[0] = np.random.uniform(low=0.5, high=1)
        if trial[1] <= 0 or trial[1] > 1:
            trial[1] = np.random.uniform(low=0.5, high=1)

        # selection
        f = get_final_res(trial)

        if gen + 1 < iternum:
            if f < fitnesses[gen, i]:
                populations[gen + 1, i] = trial
                fitnesses[gen + 1, i] = f
            else:
                populations[gen + 1, i] = populations[gen, i]
                fitnesses[gen + 1, i] = fitnesses[gen, i]


# final res
fittest_index = np.argmin(fitnesses[-1])
result = populations[-1][fittest_index]
result_fitness = fitnesses[-1][fittest_index]

print(result)
print(result_fitness)

plt.plot(avg_fit_hist, 'b')
plt.xlabel("Generation")
plt.ylabel("RMSE")
plt.ylim([0, 0.1])
plt.xlim([0, 50])
plt.show()
```

Annexe D

Code de l'analyse de cohérence

```
from objects import DE, read_csv
import numpy as np
import matplotlib.pyplot as plt


b = {'rp':  [2, 100],
     'rs':  [0, 1],
     'a':   [1, 2],
     'i0':  [1e-07, 1e-04],
     'ipv': [0, 10]
     }

T = 45 + 275.15
history = np.full(30, None, dtype=np.float64)
rs_hist = np.full(30, None, dtype=np.float64)
a_hist = np.full(30, None, dtype=np.float64)
i0_hist = np.full(30, None, dtype=np.float64)


for i in range(30):
    algo = DE(b, [*read_csv("data/PWP")], 36, 1, T)
    algo.solve()
    history[i] = algo.final_res[1]
    rs_hist[i] = algo.final_res[0][1]
    a_hist[i] = algo.final_res[0][2]
    i0_hist[i] = algo.final_res[0][3]

plt.scatter(range(1, 31), history)
plt.xlabel("Essais")
plt.ylabel("Valeur fitness")
plt.grid()
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.show()

plt.scatter(range(1, 31), rs_hist)
plt.ylabel(r"$R_s$")
plt.ylim([0, 0.1])
plt.grid()
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.show()

plt.scatter(range(1, 31), a_hist)
plt.ylabel(r"$a$")
plt.ylim([1, 2])
plt.grid()
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.show()

plt.scatter(range(1, 31), i0_hist)
plt.ylabel(r"$I_0$")
plt.grid()
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.show()

print(np.mean(history))
print(np.mean(rs_hist))
print(np.mean(a_hist))
print(np.mean(i0_hist))

print("----")
```

```
print(np.std(history))
print(np.std(rs_hist))
print(np.std(a_hist))
print(np.std(i0_hist))
```

```
from objects import DE, read_csv
import datetime
import numpy as np
import matplotlib.pyplot as plt

rmse_history = np.full(30, None, dtype=np.float64)
temp_history = np.full(30, None, dtype=np.float64)

for i in range(30):
    print(i)
    begin_time = datetime.datetime.now()
    b = {'rp':     [2, 100],
         'rs':     [0, 1],
         'a':      [1, 2],
         'i0':     [1e-7, 1e-5],
         'ipv':    [0, 5]
         }
    T = 45 + 275.15
    algo = DE(b, [*read_csv("data/PWP")], 36, 1, T, mutf=0.55, crossr=0.94, maxiter=25)
    algo.solve()

    rmse_history[i] = algo.final_res[1]
    temp_history[i] = (datetime.datetime.now() - begin_time).total_seconds()

plt.scatter(range(1, 31), rmse_history)
plt.xlabel("Essais")
plt.xlim([0, 30])
plt.ylabel("RMSE")
plt.grid()
plt.ticklabel_format(style='sci', axis='y', scilimits=(0, 0))
plt.show()

plt.scatter(range(1, 31), temp_history)
plt.xlabel("Essais")
plt.xlim([0, 30])
plt.ylim([0, 5])
plt.ylabel("seconds")
plt.grid()
plt.ticklabel_format(style='sci', axis='y', scilimits=(0, 0))
plt.show()

print(np.mean(temp_history))

print(np.std(rmse_history))
```

Annexe E

Données expérimentales

TABLEAU E.1 – Cellule R.T.C France 57 mm

Voltage V	*Current* A
−0.2057	0.764
−0.1291	0.762
−0.0588	0.7605
0.0057	0.7605
0.0646	0.76
0.1185	0.759
0.1678	0.757
0.2132	0.757
0.2545	0.7555
0.2924	0.754
0.3269	0.7505
0.3585	0.7465
0.3873	0.7385
0.4137	0.728
0.4373	0.7065
0.459	0.6755
0.4784	0.632
0.496	0.573
0.5119	0.499
0.5265	0.413
0.5398	0.3165
0.5521	0.212
0.5633	0.1035
0.5736	−0.01
0.5833	−0.123
0.59	−0.21

TABLEAU E.2 – Module Schutten Solar STM6-40/36

Voltage V	*Current* A
0.118	1.663
2.237	1.661
5.434	1.653
7.26	1.65
9.68	1.645
11.59	1.64
12.6	1.636
13.37	1.629
14.09	1.619
14.88	1.597
15.59	1.581
16.4	1.542
16.71	1.524
16.98	1.5
17.13	1.485
17.32	1.465
17.91	1.388
19.08	1.118

TABLEAU E.3 – Module Photowatt-PWP 201

Voltage V	*Current* A
0.1248	1.0315
1.8093	1.03
3.3511	1.026
4.7622	1.022
6.0538	1.018
7.2364	1.0155
8.3189	1.014
9.3097	1.01
10.2163	1.0035
11.0449	0.988
11.8018	0.963
12.4929	0.9255
13.1231	0.8725
13.6983	0.8075
14.2221	0.7265
14.6995	0.6345
15.1346	0.5345
15.5311	0.4275
15.8929	0.3185
16.2229	0.2085
16.5241	0.101
16.7987	−0.008
17.0499	−0.111
17.2793	−0.209
17.4885	−0.303

Bibliographie

[1] D. M. Chapin, C. S. Fuller, and G. L. Pearson, "A new silicon p-n junction photocell for converting solar radiation into electrical power [3]," 1954.

[2] "Renewable Energy Market Update," tech. rep., IEA, 2020.

[3] L. Fraas and L. Partain, *Solar Cells and their Applications : Second Edition*. 2010.

[4] S. M. Sze, *Semiconductor Devices : Physics and Technology*. 2006.

[5] S. R. Wenham, M. A. Green, M. E. Watt, R. P. Corkish, and A. B. Sproul, *Applied photovoltaics, third edition*. 2013.

[6] W. Shockley and H. J. Queisser, "Detailed balance limit of efficiency of p-n junction solar cells," *Journal of Applied Physics*, 1961.

[7] J. P. Ram, H. Manghani, D. S. Pillai, T. S. Babu, M. Miyatake, and N. Rajasekar, "Analysis on solar PV emulators : A review," 2018.

[8] K. L. Kennerud, "Analysis of Performance Degradation in CDS Solar Cells," *IEEE Transactions on Aerospace and Electronic Systems*, 1969.

[9] W. J. Jamil, H. Abdul Rahman, S. Shaari, and Z. Salam, "Performance degradation of photovoltaic power system : Review on mitigation methods," 2017.

[10] R. Storn and K. Price, "Differential evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces," tech. rep., 1995.

[11] P. K. Nayak, S. Mahesh, H. J. Snaith, and D. Cahen, "Photovoltaic solar cell technologies : analysing the state of the art," *Nature Reviews Materials*, 2019.

[12] K. L. Chopra, P. D. Paulson, and V. Dutta, "Thin-film solar cells : an overview," *Progress in Photovoltaics : Research and Applications*, 2004.

[13] C. Roger, *Developpement de cellules photovoltaïques à base de CIGS sur substrats métalliques*. PhD thesis, Université de Grenoble, 2013.

[14] W. Shockley, "The Theory of p-n Junctions in Semiconductors and p-n Junction Transistors," *Bell System Technical Journal*, 1949.

[15] H. Huang, "Ferroelectric photovoltaics," *Nature photonics 4*, pp. 134–135, 2010.

[16] M. G. Villalva, J. R. Gazoli, and E. R. Filho, "Comprehensive approach to modeling and simulation of photovoltaic arrays," *IEEE Transactions on Power Electronics*, 2009.

[17] V. J. Chin, Z. Salam, and K. Ishaque, "Cell modelling and model parameters estimation techniques for photovoltaic simulator application : A review," 2015.

[18] C. Carrero, J. Amador, and S. Arnaltes, "A single procedure for helping PV designers to select silicon PV modules and evaluate the loss resistances," *Renewable Energy*, 2007.

[19] H.-L. Tsai, C.-S. Tu, and Y.-J. Su, "Development of Generalized Photovoltaic Model Using MATLAB/SIMULINK," *Lecture Notes in Engineering and Computer Science*, vol. 2173, no. 1, pp. 846–851, 2008.

[20] G. Ciulla, V. Lo Brano, V. Di Dio, and G. Cipriani, "A comparison of different one-diode models for the representation of I-V characteristic of a PV cell," 2014.

[21] T. Easwarakhanthan, J. Bottin, I. Bouhouch, and C. Boutrit, "Nonlinear Minimization Algorithm for Determining the Solar Cell Parameters with Microcomputers," *International Journal of Solar Energy*, 1986.

[22] K. M. El-Naggar, M. R. AlRashidi, M. F. AlHajri, and A. K. Al-Othman, "Simulated Annealing algorithm for photovoltaic parameters identification," *Solar Energy*, 2012.

[23] W. T. Da Costa, J. F. Fardin, D. S. Simonetti, and L. D. V. Neto, "Identification of photovoltaic model parameters by differential evolution," in *Proceedings of the IEEE International Conference on Industrial Technology*, 2010.

[24] M. Balzani and A. Reatti, "Neural network based model of a PV array for the optimum performance of PV system," in *2005 PhD Research in Microelectronics and Electronics - Proceedings of the Conference*, 2005.

[25] L. Zhang and Y. F. Bai, "Genetic algorithm-trained radial basis function neural networks for modelling photovoltaic panels," *Engineering Applications of Artificial Intelligence*, 2005.

[26] E. Karatepe, M. Boztepe, and M. Colak, "Neural network based solar cell model," *Energy Conversion and Management*, 2006.

[27] M. T. Elhagry, A. A. Elkousy, M. B. Saleh, T. F. Elshatter, and E. M. Abou-Elzahab, "Fuzzy modeling of photovoltaic panel equivalent circuit," in *Midwest Symposium on Circuits and Systems*, 1997.

[28] T. Bendib, F. Djeffal, D. Arar, and M. Meguellati, "Fuzzy-logic-based approach for organic solar cell parameters extraction," in *Lecture Notes in Engineering and Computer Science*, 2013.

[29] M. AbdulHadi, A. M. Al-Ibrahim, and G. S. Virk, "Neuro-fuzzy-based solar cell model," *IEEE Transactions on Energy Conversion*, 2004.

[30] J. A. Jervase, H. Bourdoucen, and A. Al-Lawati, "Solar cell parameter extraction using genetic algorithms," *Measurement Science and Technology*, 2001.

[31] N. Moldovan, R. Picos, and E. Garcia-Moreno, "Parameter extraction of a solar cell compact model usign genetic algorithms," in *Proceedings of the 2009 Spanish Conference on Electron Devices, CDE'09*, 2009.

[32] M. S. Ismail, M. Moghavvemi, and T. M. Mahlia, "Characterization of PV panel and global optimization of its model parameters using genetic algorithm," *Energy Conversion and Management*, 2013.

[33] M. Ye, X. Wang, and Y. Xu, "Parameter extraction of solar cells using particle swarm optimization," *Journal of Applied Physics*, 2009.

[34] J. J. Soon and K. S. Low, "Photovoltaic model identification using particle swarm optimization with inverse barrier constraint," *IEEE Transactions on Power Electronics*, 2012.

[35] K. Ishaque, Z. Salam, S. Mekhilef, and A. Shamsudin, "Parameter extraction of solar photovoltaic modules using penalty-based differential evolution," *Applied Energy*, 2012.

[36] W. Gong and Z. Cai, "Parameter extraction of solar cell models using repaired adaptive differential evolution," *Solar Energy*, 2013.

[37] T. Ikegami, T. Maezono, F. Nakanishi, Y. Yamagata, and K. Ebihara, "Estimation of equivalent circuit parameters of PV module and its application to optimal operation of PV system," *Solar Energy Materials and Solar Cells*, 2001.

[38] V. J. Chin and Z. Salam, "A New Three-point-based Approach for the Parameter Extraction of Photovoltaic Cells," *Applied Energy*, 2019.

[39] K. Price, R. M. Storn, and J. A. Lampinen, *Differential evolution : a practical approach to global optimization*. Springer Science & Business Media, 2005.

[40] M. Shur, T. A. Fjeldly, and B. J. Moon, "Approximate Analytical Solution of Generalized Diode Equation," *IEEE Transactions on Electron Devices*, 1991.

[41] M. T. Abuelma'Atti, "Improved approximate analytical solution for generalised diode equation," *Electronics Letters*, 1992.

[42] S. K. Datta, K. Mukhopadhyay, S. Bandopadhyay, and H. Saha, "An improved technique for the determination of solar cell parameters," *Solid State Electronics*, 1992.

[43] A. Jain and A. Kapoor, "Exact analytical solutions of the parameters of real solar cells using Lambert W-function," *Solar Energy Materials and Solar Cells*, 2004.

[44] S. xian Lun, S. Wang, G. hong Yang, and T. ting Guo, "A new explicit double-diode modeling method based on Lambert W-function for photovoltaic arrays," *Solar Energy*, 2015.

[45] R. M. Corless, G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth, "On the Lambert W function," *Advances in Computational Mathematics*, 1996.

[46] D. Zaharie, "Critical values for the control parameters of differential evolution algorithms," in *MENDEL 2002, 8th Int. Conf. on Soft Computing*, pp. 62–67, 2002.

[47] R. Salomon, "Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms," *Bio-Systems*, 1996.

[48] N. Chakraborti, K. Misra, P. Bhatt, N. Barman, and R. Prasad, "Tight-binding calculations of Si-H clusters using genetic algorithms and related techniques : Studies using differential evolution," *Journal of Phase Equilibria*, 2001.

[49] N. F. A. Hamid, N. A. Rahim, and J. Selvaraj, "Solar cell parameters identification using hybrid Nelder-Mead and modified particle swarm optimization," *Journal of Renewable and Sustainable Energy*, 2016.

[50] D. Oliva, E. Cuevas, and G. Pajares, "Parameter identification of solar cells using artificial bee colony optimization," *Energy*, 2014.

[51] A. R. Jordehi, "Time varying acceleration coefficients particle swarm optimisation (TVACPSO) : A new optimisation algorithm for estimating parameters of PV cells and modules," *Energy Conversion and Management*, 2016.

[52] A. R. Jordehi, "Gravitational search algorithm with linearly decreasing gravitational constant for parameter estimation of photovoltaic cells," in *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*, 2017.

[53] D. Oliva, A. A. Ewees, M. A. El Aziz, A. E. Hassanien, and M. P. Cisneros, "A chaotic improved artificial bee colony for parameter estimation of photovoltaic cells," *Energies*, 2017.

[54] D. Oliva, M. Abd El Aziz, and A. Ella Hassanien, "Parameter estimation of photovoltaic cells using an improved chaotic whale optimization algorithm," *Applied Energy*, 2017.

[55] A. Rezaee Jordehi, "Enhanced leader particle swarm optimisation (ELPSO) : An efficient algorithm for parameter estimation of photovoltaic (PV) cells and modules," *Solar Energy*, 2018.

[56] X. Gao, Y. Cui, J. Hu, G. Xu, Z. Wang, J. Qu, and H. Wang, "Parameter extraction of solar cell models using improved shuffled complex evolution algorithm," *Energy Conversion and Management*, 2018.

[57] L. Wu, Z. Chen, C. Long, S. Cheng, P. Lin, Y. Chen, and H. Chen, "Parameter extraction of photovoltaic models from measured I-V characteristics curves using a hybrid trust-region reflective algorithm," *Applied Energy*, 2018.