REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE
LA RECHERCHE SCIENTIFIQUE

Département d'Electronique

Projet de Fin d'Etudes

pour l'obtention du diplôme

d'Ingénieur d'Etat en Electronique

Présenté par :

Kherchi Mohamed

Smaïli Mohamed Djamel

Thème

Etude et implémentation d'une commande MPPT neuro-floue sur FPGA

Membres du Jury :

Mr. C.LARBES, Maître de Conférences, ENP	Rapporteur
Mr. M .HADDADI, Professeur, ENP	Président
Mr. M.S.AIT-CHEIKH, Docteur, ENP	Examinateur

2008-2009

Ecole Nationale Polytechnique 10, Avenue Hassen Badi, El-Harrach, ALGER

Dédicace

Je dédie ce travail à :

Ma très chère mère, mon très cher père,

Mes sœurs,

Ainsi qu'à toute ma famille et mes amis.

Smaili Mohamed Diamel

Kherchi Mohamed

ملـخص:

الهدف من هاته الدراسة التي بين أيدينا هو إعداد متحكم أوتوماتيكي غايته تتبع نقطة الاستطاعة القصوى للمولد الكهروضوئي وذلك اعتمادا على تقنية جديدة تندرج ضمن ما يعرف بالذكاء الاصطناعي والتي تعرف بتقنية الشبكات العصبونية المدمجة بالمنطق الغامض وهذا من أجل استغلال أكمل للطاقة التي يمكن أن تنتجها الألواح الشمسية.

هذا العمل يبرز مختلف مراحل إعداد هذا المتحكم المرتكز على التقنية المذكورة آنفا وذلك باستخدام الأدوات التي يوفرها بـــرنامج « Matlab » مع القيام بتجربة المتحكم عن طريق المحاكاة بواسطة برنامج « Simulink » التابع لنفس البرنامج المذكور مؤخرا؛ نردف ذلك بشرح أهم مراحل برمجته على دارة « FPGA » المتميزة بسرعة التنفيذ ومرونة الاستعمال مستخدمين لأجل هذا الغرض لغة البرمجة « VHDL » .

كلمات مفتاحية : كهروضوئي، تتبع نقطة الاستطاعة العظمى، شبكة العصبونات الاصطناعية، المنطق الغامض، شبكة العصبونات بالمنطق الغامض، المراقب العصبوني الغامض، دارة« FPGA » ،لغة البرمجة «VHDL »

Résumé

Ce travail consiste à concevoir un contrôleur pour la poursuite du point de puissance maximale (MPPT) des générateurs photovoltaïques. C'est une commande par l'approche neuro-floue qui est définie comme étant un réseau neuronal multicouches avec des paramètres flous, ou comme un système flou mis en application sous une forme distribuée parallèle. Le contrôleur neuro_flou sert à exploiter au mieux la puissance délivrée par les générateurs photovoltaïques.
Ce mémoire présente le développement d'une telle commande et son implémentation sur un circuit FPGA. Les circuits FPGA sont particulièrement utiles grâce à leurs faible temps de développement, leur puissance de calcul et leur flexibilité.

Mots clés : Photovoltaïque, MPPT, Réseaux de neurones, Logique floue, contrôleur neuro_flou, Circuit FPGA, langage VHDL.

Abstract

In this work, we present a maximum power point tracking (MPPT) controller for photovoltaic generators. Our interest was particularly directed to a new technique which belongs to artificial intelligence. It is based on a neuro-fuzzy approach, which is defined as a multilayer neural network with fuzzy parameters, or as a fuzzy system implemented in a distributed parallel form. The neuro-fuzzy controller is used to exploit effectively the power delivered by solar panels.
This paper presents the design and the implementation of this controller on an FPGA circuit relying on the various steps needed to program this circuit using the development card "Memec Design *Virtex-II*». The use of FPGA circuits has many advantages such as flexibility, fast calculation, and a relative ease for theirs programmation.

Key words: Photovoltaic, MPPT, Artificial neural network, Fuzzy logic, neuro-fuzzy, neuro-fuzzy controller, FPGA circuit, VHDL language.

Table des matières :

Table des figures :

Introduction générale :

Depuis le siècle dernier, la consommation de l'énergie a augmenté d'une manière considérable et nos ressources en pétrole, charbon, ou en gaz ne sont pas éternelles. Et il est aussi de notre devoir de ne pas consommer plus pour ne pas aggraver la pollution.

Une alternative se trouve dans l'utilisation des énergies renouvelables telles que l'énergie hydraulique, éolienne ou solaire. L'énergie émise par le soleil voyage jusqu'à la terre sous forme de rayonnement électromagnétique. Ce rayonnement est semblable aux ondes radioélectriques mais il possède une gamme de fréquences différentes. L'énergie solaire disponible est souvent exprimée en unités d'énergie par temps par unité de surface, par exemple en Watt par mètre carré (W/m^2).

La quantité d'énergie disponible au cours du trajet entre le soleil et l'atmosphère extérieure de la terre est égale à environ 1 367 W/m^2. Une partie de l'énergie solaire est absorbée lorsque le rayonnement traverse l'atmosphère terrestre. Il en résulte qu'au cours d'une journée ensoleillée, la quantité d'énergie solaire disponible à la surface de la terre dans la direction du soleil est généralement de 1 000 W/m^2.

Quel que soit le moment, l'énergie solaire disponible est principalement fonction de la position du soleil et de la nébulosité. Sur une base mensuelle ou annuelle, la quantité d'énergie solaire disponible dépend également de l'emplacement géographique. De plus, l'énergie solaire utilisable dépend de l'énergie solaire disponible, d'autres conditions météorologiques, de la technologie utilisée et de l'application.

L'électricité solaire est vue comme étant une importante énergie renouvelable qui pourra être une alternative aux autres sources classiques d'énergie pour satisfaire les larges besoins en énergie dans le futur. L'électricité solaire trouve tout son avantage dans des applications de petite et moyenne

consommation dans des régions isolées et loin des lignes de distribution électrique.

L'électricité solaire est en train de s'imposer depuis que les modules photovoltaïques sont devenus moins chers avec un rendement acceptable. En parallèle, la technologie des composants semi-conducteurs de grande puissance a nettement progressé par l'introduction de composants très performants du point de vue rendement et puissance de fonctionnement.

Une caractéristique importante des modules photovoltaïques est que la puissance maximale disponible est fournie seulement en un seul point de fonctionnement donné, localisé par une tension et un courant connus, appelé en anglais Maximum Power Point (MPP). L'autre problème est que la position de ce point n'est pas fixe mais se déplace en fonction de l'ensoleillement et de la température. A cause du coût relativement onéreux de ce genre d'énergie on doit extraire le maximum de watts des modules photovoltaïques. Cela nécessite un mécanisme de poursuite (Tracking) du point de puissance maximale appelé « Maximum power point tracking» (MPPT) afin que la puissance maximale soit générée en permanence.

Dans ce mémoire, nous nous intéressons au problème de poursuite du point de puissance maximale d'un générateur photovoltaïque suivant deux concepts : l'approche floue et l'approche neuro-floue .

Pour cela les différentes étapes de conception et de réalisation d'un contrôleur MPPT neuro-flou vont être décrites dans ce mémoire.

L'outil FUZZY de Matlab a été d'abord utilisé pour la conception d'un contrôleur MPPT flou afin de construire une base se données. Cette dernière a été ensuite utilisée pour la conception d'un contrôleur MPPT neuro-flou conjointement avec l'outil ANFIS de Matlab.après des simulations concluantes on est passé à l'étape de réalisation hardware :

- *De contrôleur MPPT neuro-flou sur un circuit FPGA en utilisant la carte de développement* ***Virtex-II V2MB1000 de Memec***

- *D'un étage de puissance consistant en un hacheur dévolteur à base d'un MOSFET.*

Chapitre : 1

Système photovoltaïque

1.1 Introduction :

Un système photovoltaïque est un système d'alimentation électrique, constitué principalement d'un générateur photovoltaïque composé d'un seul ou plusieurs panneaux solaires, d'un ensemble de batteries pour le stockage d'énergie électrique, d'un ou de plusieurs convertisseurs continu-continu pour fournir les tensions d'alimentation adéquates pour les batteries et les charges continues, et un convertisseur continu-alternatif pour l'alimentation des autres appareils à courant alternatif.

Ce chapitre décrit les concepts de base de systèmes photovoltaïques et de production d'électricité grâce à l'effet photovoltaïque. [1]

1.2 L'effet photovoltaïque :

L'énergie photovoltaïque (PV) est la transformation directe de la lumière en électricité. On utilise pour cela une photopile. L'effet photovoltaïque fut observé la première fois en 1839 par le physicien français Edmond Becquerel. Toutefois, ce n'est qu'au cours des années 1950 que les chercheurs de la compagnie Bell Téléphone, aux États-Unis, parvinrent à fabriquer la première photopile, l'élément primaire d'un système photovoltaïque.[2]

1.3 La photopile :

Cette photopile, qu'on appelle aussi cellule solaire ou cellule photovoltaïque, est fabriquée à l'aide de matériaux semi-conducteurs. On peut la représenter comme une diode plate qui est sensible à la lumière.

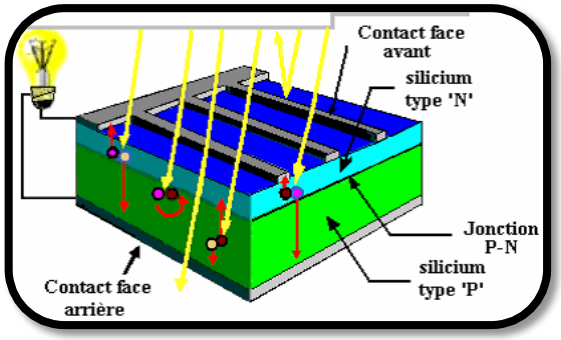


Figure 1.1. Description d'une cellule photovoltaïque

Quand un photon, d'énergie suffisante, heurte un atome sur la partie négative de cette diode, il excite un électron et l'arrache de sa structure moléculaire, créant ainsi un électron libre sur cette partie. Une photopile est fabriquée de manière à ce que cet électron libre ne puisse se recombiner facilement avec un atome à charge positive, avant qu'il n'ait accompli.

Un travail utile en passant dans un circuit extérieur. La cellule photovoltaïque produira de l'électricité à courant continu (CC), et son énergie produite sera fonction principalement de la lumière reçue par la photopile.[2]

1.4 Caractéristiques de la cellule photovoltaïque :

Le schéma du circuit équivalent d'une cellule photovoltaïque qui est largement utilisé dans la littérature [3], [4] est représenté sur la (figure 1.2) :

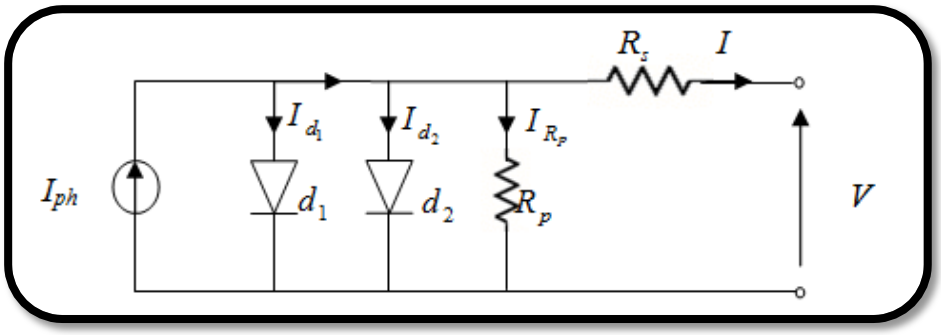


***Figure 1.2 Schéma du modèle équivalent à deux diodes d'une cellule* photovoltaïque.**

Comme le montre le schéma de la (figure 1.2) une photopile comporte en réalité une résistance série *Rs* et une résistance en dérivation ou shunt *Rp*. Ces résistances auront une certaine influence sur la caractéristique I-V de la photopile :

• La résistance série est la résistance interne de la cellule ; elle dépend principalement de la résistance du semi-conducteur utilisé, de la résistance de contact des grilles collectrices et de la résistivité de ces grilles.

• La résistance shunt est due à un courant de fuite au niveau de la jonction ; elle dépend de la façon dont celle-ci a été réalisée.

D'après la (figure 1.2) le modèle mathématique pour la caractéristique courant-tension est donnée par:

$$I = I_{ph} - I_{s1}\left[e^{\frac{q(V+IR_s)}{\eta_1 KT}} - 1\right] - I_{s2}\left[e^{\frac{q(V+IR_s)}{\eta_2 KT}} - 1\right] - \frac{V+IR_s}{R_p} \qquad (1-1)$$

Où :

I et V sont le courant et la tension de sortie de la cellule photovoltaïque.

I_{ph} est le photo-courant produit.

I_{s1} et I_{s2} sont les courants de saturation des diodes.

η_1 et η_2 les facteurs de pureté de la diode.

R_s et R_p sont respectivement la résistance série et la résistance parallèle.

T est la température absolue en Kelvin.

$q(1{,}602 \,.\, 10^{-19}\ C)$ est la charge élémentaire constante.

La constante de Boltzmann k $(1{,}380 \,.\, 10^{-23}\ J/K)$.

Le photo-courant $I_{ph.max}$ est atteint à une insolation maximum, souvent on a

$(I_{ph=} I_{ph.max} * S)$ avec S : pourcentage d'insolation.

Il est évident de l'équation (1.1), que la caractéristique courant-tension dépend fortement de l'insolation et de la température. La dépendance de la température est encore amplifiée par les propriétés du photo-courant I_{ph} et les courants de saturation inverse des diodes qui sont donnés par [3]:

$$I_{ph}(T) = I_{ph}|_{(T=298K)}[\,1 + (T - 298\,.K)(5.10^{-4})] \qquad (1-2)$$

$$I_{s1} = K_1 T^3 e^{-\frac{E_g}{kT}} \qquad (1-3)$$

$$I_{s2} = K_2 T^{\frac{5}{2}} e^{-\frac{E_g}{kT}} \qquad (1-4)$$

Où : E_g est la bande d'énergie du semi-conducteur et :

$$K_1 = 1.2\ A/cm^2.K^3 \qquad (1-5)$$

$$K_2 = 2.9\ 10^5\ A/cm^2.K^{\frac{5}{2}} \qquad (1-6)$$

1.5 Influence de l'ensoleillement et de la température :

Le courant produit par la photopile I_{ph} est pratiquement proportionnel à l'éclairement solaire S. Par contre, la tension V aux bornes de la jonction varie peu car elle est fonction de la différence de potentiel à la jonction N-P du matériau lui-même [5]. La tension de circuit ouvert ne diminuera que légèrement avec l'éclairement. Ceci implique donc que :

• La puissance optimale de la cellule (Pm) est pratiquement proportionnelle à l'éclairement.

• Les points de puissance maximale se situent à peu près à la même tension ; voir (figure 1.3).

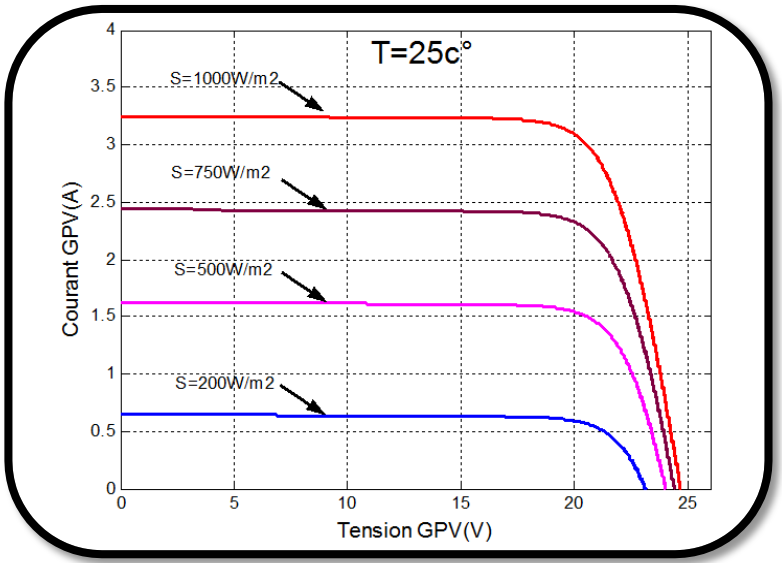


Figure1.3.a Influence de l'ensoleillement sur les courbes I-V.

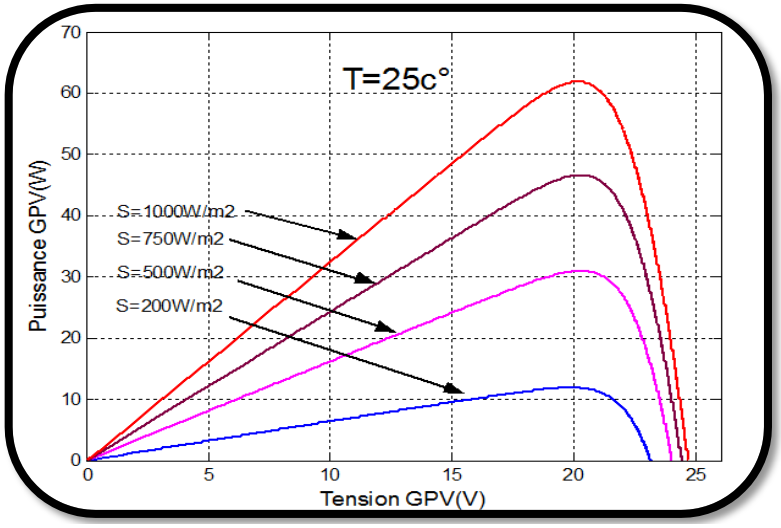


Figure1.3.b Influence de l'ensoleillement sur les courbes P-V.

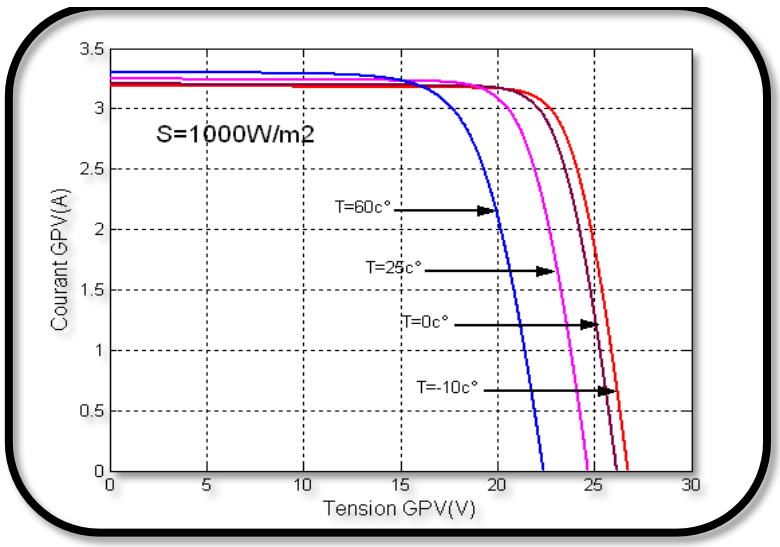


Figure 1.4.a Influence de la température sur les courbes I-V.

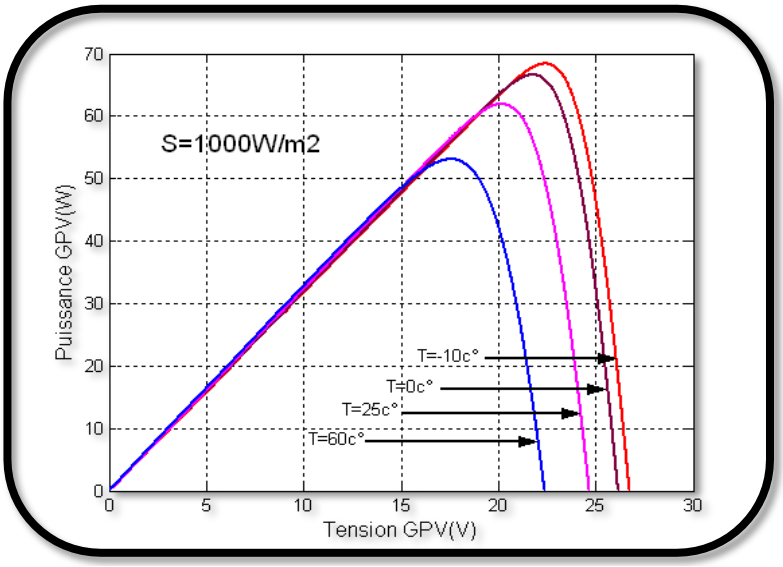


Figure 1.4.b Influence de la température sur les courbes P-V.

L'influence de la température est non négligeable sur la caractéristique courant/tension d'un semi-conducteur ; voir (figure 1.4). Pour une température qui change, on peut voir que la tension change considérablement tandis que le courant demeure constant. [2]

1.6. Le module photovoltaïque :

En associant les cellules PV en série (ajout des tensions de chaque cellule) ou en parallèle (somme des intensités de chaque cellule), on peut constituer un générateur PV selon les besoins des applications visées. Les deux types de regroupement sont en effet possibles et souvent utilisés afin d'obtenir en sortie des valeurs de tension et intensité souhaitées.

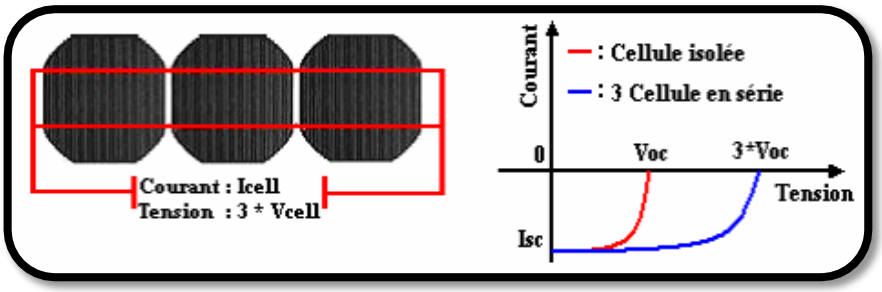


Figure 1.5. Cellules connectées en série avec leur caractéristique courant-tension

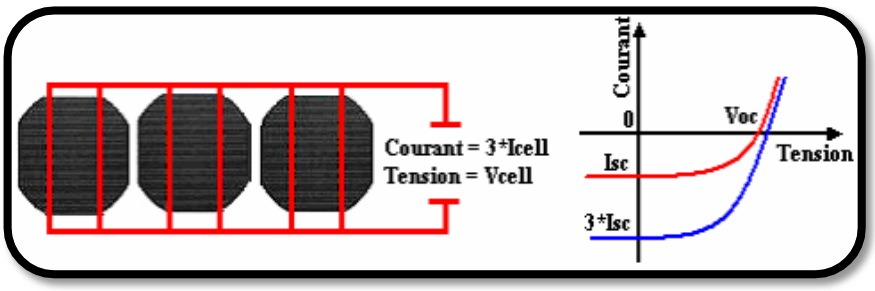


Figure 1.6. Cellules connectées en parallèle avec leur caractéristique courant-tension

La considération du modèle de circuit équivalent (figure 1.2) mène à l'équation pour une rangée photovoltaïque de cellules (généralement appelée un module solaire ou une rangée solaire) avec *Z* cellules photovoltaïques raccordées en série (1-7).

$$I = I_{ph} - I_{s1}\left[e^{\frac{q(V+IZR_s)}{Z\eta_1 KT}} - 1\right] - I_{s2}\left[e^{\frac{q(V+IZR_s)}{Z\eta_2 KT}} - 1\right] - \frac{V + IZR_s}{ZR_p} \qquad (1-7)$$

Ces panneaux peuvent alors être encore arrangés en série ou en parallèle pour réaliser la tension et les valeurs de courant désirés pour le système voulu.

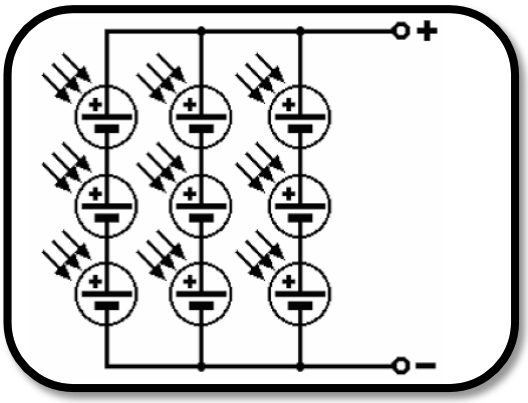


Figure 1.7. Ensemble de cellules PV montées en série-parallèle pour former un module PV

Pour obtenir donc une force électromotrice supérieure à 12 volts il est nécessaire de mettre en série plusieurs cellules de 0,6 volts. Par exemple un panneau fournissant 20 volts à vide est constitué de 36 cellules. Par contre, la mise en parallèle de cellules permet d'obtenir un courant d'intensité plus grande. Le câblage série-parallèle est utilisé pour obtenir un générateur aux caractéristiques souhaitées.

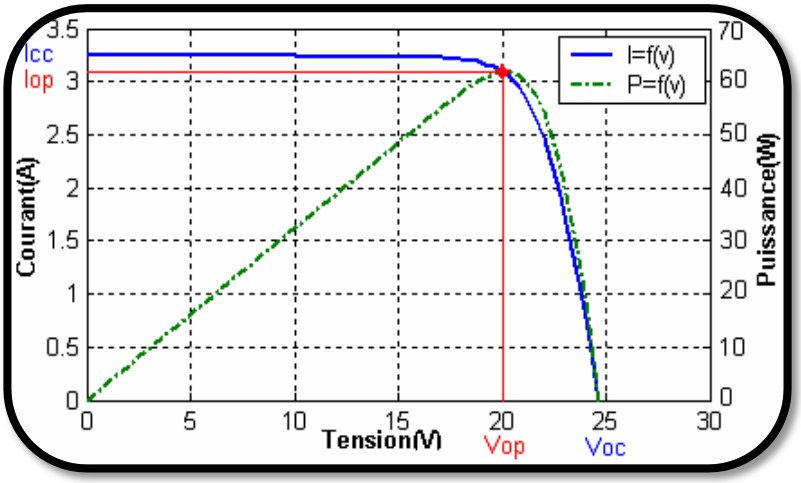


Figure 1.8. Comportement d'un générateur PV décrit par 36 cellules en série

Ainsi, en généralisant à N_n cellules en série et N_m cellules en parallèle, la puissance disponible en sortie du générateur PV ainsi constitué est donnée par :

$$P_{PV} = N_n . V_{PV} . N_m . I_{PV} \qquad (1-8)$$

Avec :

N_n et N_m respectivement, le nombre de cellules en série et le nombre de cellules en parallèle.

V_{PV} et I_{PV} respectivement, la tension et le courant du générateur PV.

Remarque :

La plupart des panneaux PV commercialisés comportent 36 cellules en série.

Si l'on désire avoir un générateur PV ayant un courant de sortie plus intense, il faudra soit faire appel à des cellules PV de grandes surfaces, soit associer plusieurs modules PV de caractéristiques similaires en parallèle.

La première constatation qu'on peut faire est que pour qu'un générateur PV puisse fonctionner de façon optimale, il faut que les $\boldsymbol{Nn}$ et $\boldsymbol{Nm}$ cellules se comportent toutes de façon identique. Pour cela, il faut qu'elles soient issues de la même technologie et plus encore, du même lot de fabrication, Il faut ensuite qu'elles soient soumises aux mêmes conditions de fonctionnement (éclairement, température, vieillissement et inclinaison). Et enfin, il faut que l'ensemble soit connecté à une charge qui permet de faire fonctionner le générateur PV proche de sa puissance optimale correspondant à la somme des $\boldsymbol{Nn\ x\ Nm}$ cellules élémentaires.

En résumé, la puissance du générateur PV sera optimale si chaque cellule fonctionne à sa puissance maximale notée P_{max} . Soit, si elle a à ses bornes une tension V_{op} nommée tension optimale et un courant I_{op} , nommé courant optimal.

Soit :

$$P_{op} = N_n . N_m . P_{max} = N_n . N_m . V_{op} . I_{op} \qquad (1-9)$$

Pour cela, les fabricants ont choisi pour réduire les disfonctionnements de ne pas commercialiser des cellules PV seules mais sous forme de module pré câblé constitué pour la plupart de 36 cellules en série. Chaque référence de module a ses propres caractéristiques électriques qui sont garanties à ±10% selon le lot de fabrication.

En résumé, un générateur PV peut être constitué d'une ou plusieurs cellules en série ou en parallèle mais aussi d'un ou plusieurs modules PV commerciaux.[6]

1.7 Comparaison de différents types de modules PV existants dans le commerce :[6]

:

Fabricant	Modèle	Pmax en (W)	Vopt en (V)	Iopt en (A)	Nombre de cellules	Hauteur en cm	Largeur en cm
BPSolar	BP5170	170	36	4,72	72	159	75,5
BPSolar	BP585	85	18	4,72	36	118,8	50,8
BPSolar	BP120	120	33,7	3,56	72	111,4	99,1
BPSolar	BPMSX60	60	16,8	3,56	36	111	50,2
BPSolar	BPS10	10	16,9	0,59	-	61,6	38,2
BPSolar	BP3160	160	35,1	4,55	72	158,7	79
Photowatt	PWX100	11	17	0,65	36	58	26
Photowatt	PWX1000	105	34,6	3,05	72	133,5	67,3
Photowatt	PWX750	80	17,3	4,6	36	123	55,6
Totalénergie	PW75O	75	-	-	36	-	-

Tableau 1.1 Comparaison de différents types de modules PV de commerce

1.8. Contexte de l'étude sur les systèmes de conversion photovoltaïques (PV) :

Plusieurs études ont montrées qu'un certain nombre d'améliorations pouvaient être faites pour augmenter le rendement et la fiabilité de ce type de systèmes de conversion, dont un grand nombre étaient déjà commercialisés. En effet, si l'on considère le fonctionnement d'un générateur PV à base de cellules en silicium, comme il est décrit dans la première partie de ce chapitre, les premières améliorations à effectuer pour gagner en rendement sont :

1. Améliorer le refroidissement des cellules PV qui permettrait de gagner environ 20% de rendement sur la totalité des capteurs PV par un refroidissement judicieux. En effet, le rendement de conversion photons-électrons décroît rapidement au fur et à mesure que la température interne de la cellule augmente.
2. Améliorer la connectique entre les différents panneaux PV par une architecture appropriée. En effet, une perte de rendement de 20% supplémentaire est à noter sur les systèmes existants entre la puissance qu'ils pourraient théoriquement délivrer et celle effectivement transférée à la charge. Cette perte est communément imputée à l'interface de puissance entre les panneaux et la charge.

On peut remarquer aussi que la puissance de sortie d'un panneau solaire ne dépend pas uniquement de la température et de l'insolation, mais aussi de la tension V de fonctionnement. Le point de la puissance maximale indiqué comme MPP (maximum power point) sur la (figure 1.8) est le point désiré pour le fonctionnement d'une rangée photovoltaïque pour obtenir un rendement maximal en puissance. Les valeurs correspondantes pour la tension et le courant s'appellent respectivement V_{op} et le I_{op}.

Les courbes de puissances représentées respectivement sur les figures 1.3 et 1.4 montrent explicitement que la puissance de sortie d'un module solaire P_{op} est subordonnée directement aux changements, de l'intensité de lumière incidente sur le module solaire, ainsi que la température de l'environnement, en conséquence la tension V_{op} et le courant I_{op} de fonctionnement changent constamment avec le changement de ces variables environnementales.

Pour remédier à ces contraintes, un équipement spécifique, doit être mis entre le module solaire et la charge dont le rôle est l'adaptation des deux équipements (module solaire/charge) pour un meilleur transfert d'énergie vers la charge. Ce dernier peut augmenter d'une manière significative le rendement et cela en ajustant la charge du système de telle manière que la tension V de service soit toujours approximativement égale à la tension V_{op} de fonctionnement optimal.

$$V = V_{op} + \delta \qquad (1-10)$$

δ : Représente la correction introduite par l'équipement qui est aussi petite que possible.

Sachant qu'un écart de 10% de la tension de service V par rapport à la tension optimale V_{op} enregistre une perte de puissance de service P de presque 25% de la puissance optimale P_{op} (figure 1.9).

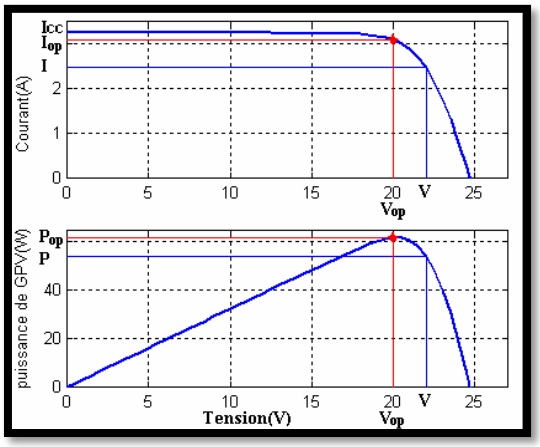


Figure 1.9. Point de puissance maximale (PPM), puissance et courant

Comme on peut remarquer sur la (figure 1.10), une distribution des points de puissances maximales (PPM) sous forme d'un champ de points formants une surface, qui sont représentés en fonction de la variation de la température et de l'ensoleillement. En conséquence de l'analyse de la distribution des points, toute installation photovoltaïque dotée d'un équipement adéquat (contrôleur MPPT) va lui assurer un fonctionnement optimal en un endroit quelconque de la surface énergétique correspondant à un changement donné en température ou en ensoleillement ou bien les deux simultanément.[6]

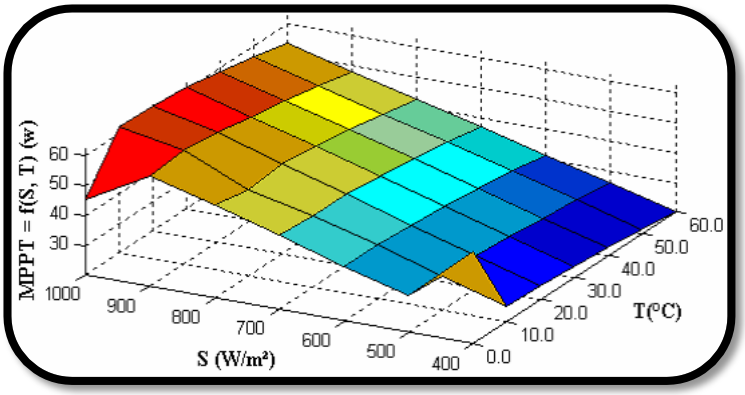


Figure 1.10. Champ de distribution spatiale des points de puissances maximales en fonction des changements climatiques en température et en ensoleillement.

1.9 Les systèmes photovoltaïques avec batterie :

Un système photovoltaïque avec batterie peut être comparé à une charge alimentée par une batterie qui est chargée par un générateur photovoltaïque. Il comprend généralement les composantes de base suivantes :

- un générateur photovoltaïque composé d'un ou de plusieurs modules photovoltaïques pour charger les batteries.
- une batterie pour stocker l'énergie électrique et alimenter les charges à courant continu (DC).
- un convertisseur continu-continu (hacheur) pour fournir la tension d'alimentation adéquate pour les batteries
- un convertisseur continu-alternatif (onduleur) pour alimenter les charges à courant alternatif.
- La (figure.1.11) illustre un système photovoltaïque simple avec batterie.[2]

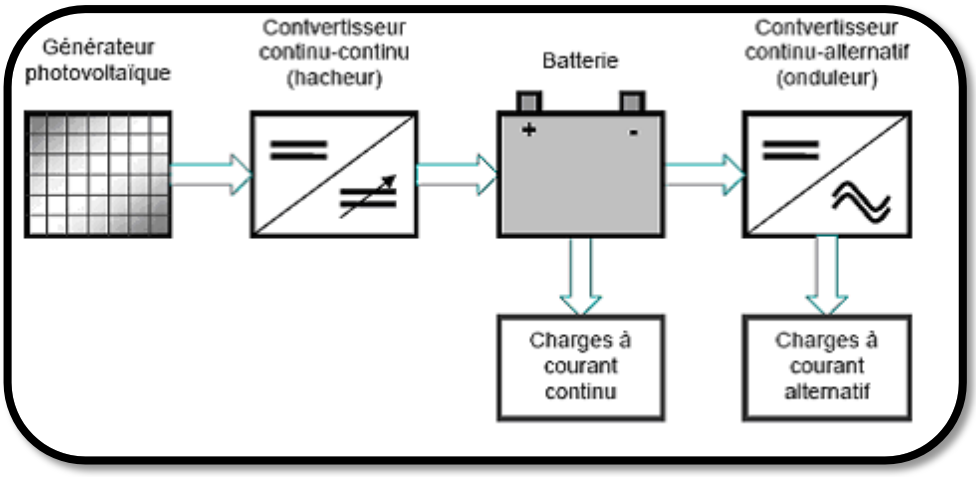


Figure. 1.11 Composantes de base d'un système PV autonome avec batterie

1.10 Convertisseurs continu/continu (DC/DC) :

1.10.1 Introduction :

L'utilisation des convertisseurs DC-DC permet le contrôle de la puissance électrique dans les circuits fonctionnant en courant continu avec une très grande souplesse et un rendement élevé qui dans notre cas va nous permettre de poursuivre le point de fonctionnement optimum. Dans cette partie nous allons voir les principes de fonctionnement des hacheurs qui sont des convertisseurs directs du type continu-continu, dont le rôle primordial est de transformer une puissance d'entrée $P_e = V_e I_e$ en une puissance de sortie $P_s = V_s I_s$ avec ($V_s \neq V_e$) avec un rendement de conversion $\eta = \frac{P_s}{P_e} = \frac{V_s I_s}{V_e I_e}$ très grande (proche de 100 %). En effet, contrairement aux alimentations classiques (linéaires) qui utilisent un transistor qui dissipe $(V_e - V_s)I_s$, les alimentations à découpage mettent en œuvre un commutateur qui travaille en tout ou rien.

Les circuits des hacheurs se composent de condensateurs, d'inductances et de commutateurs. Dans le cas idéal, tous ces dispositifs ne consomment aucune puissance active, c'est la raison pour laquelle on a de bons rendements. Le commutateur est réalisé avec un dispositif semi-conducteur, habituellement un transistor MOSFET ou un IGBT.

Pendant le fonctionnement du hacheur, le transistor sera commuté à une fréquence constante f_s avec un temps de fermeture égal à $(d . T_s)$ et un temps d'ouverture égal a $((1 - d).T_s)$ ou :

- T_s est la période de commutation qui est égale à $1/f_s$.
- d le rapport cyclique du commutateur $(d \in [0 , 1])$. (figure : 1.12)

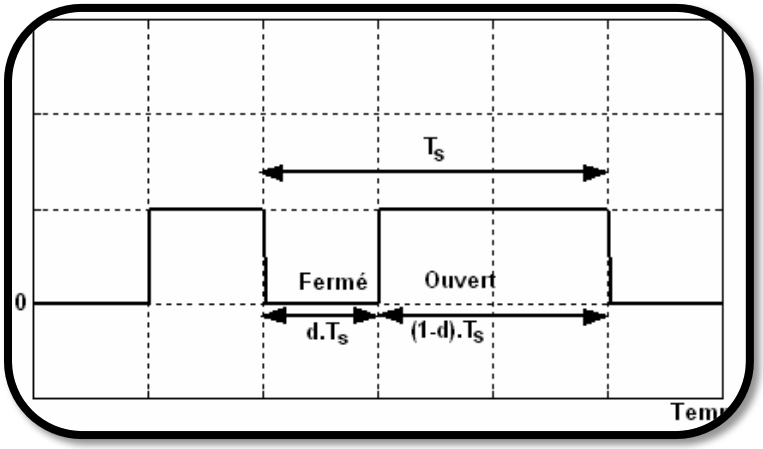


Figure 1.12: Représentation de la sortie d'un commutateur en ouverture/fermeture sur une période.

1.10.2 Type de convertisseurs :

Il y a différents types de convertisseurs DC-DC. Une première distinction est à faire entre le convertisseur à isolement galvanique entre la sortie et l'entrée, et le convertisseur qui présente une borne commune entre l'entrée et la sortie (Notez que dans le cas de l'isolement galvanique le rendement se verra affecté d'au moins 10%, ce qui est inadmissible dans le cas par exemple des applications en énergie photovoltaïque).
Dans les convertisseurs à stockage d'énergie, on distingue :

- Les élévateurs (BOOST) dans lesquels $V_s > V_e$.
- Les abaisseurs (BUCK) dans lesquels $V_s < V_e$.
- Il y a aussi un type (BUCK-BOOST) : mélange entre les deux précédents.
  - **Dans notre système, on va s'intéresser à l'hacheur « Buck » et on donne ici son étude avec un peu de détail :**

1.10.3 Hacheur dévolteur « Buck » :

Le convertisseur dévolteur (figure 1.13) peut être souvent trouvé dans la littérature sous le nom de hacheur Buck ou hacheur série.

L'utilisation du Buck dans les systèmes de conversions photovoltaïques est tout à fait adéquate, étant donné que le générateur photovoltaïque est une source de courant continu, tandis que le récepteur qui est généralement la batterie, a la nature d'une source de tension.

Son application typique est de convertir sa tension d'entrée en une tension de sortie inférieure, où le rapport de conversion $M = {V_s}/{V_e}$ change avec le rapport cyclique d du commutateur.

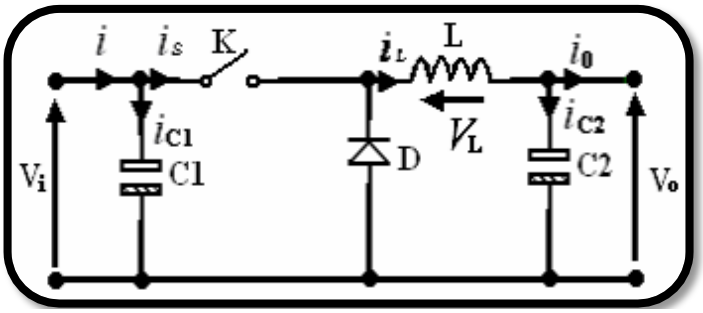


Figure 1.13: Circuit idéal du hacheur série

À l'état conducteur de l'interrupteur ***K***, pendant la durée t_{on} c.-à-d. $t \in [\,0\,,\ d.T_s]$, la diode se bloque et un courant circulera dans la charge à travers l'inductance, cette dernière stocke une énergie :

$$w = \frac{1}{2}\, L\, {I_L}^2 \qquad (1-11)$$

Lors des commutations successives de l'interrupteur ***K***, le courant à l'intérieur de l'inductance aura deux composante, l'une est constante égale à la valeur moyenne du courant sur une période du hachage, tandis que la composante variable est égale à :

- Lorsque ***K*** est fermé :

$$(\Delta I_l)_{on} = \frac{V_e - V_s}{L}\, t_{on} \qquad (1-12)$$

- Lorsque ***K*** est ouverte, la self se décharge dans la sortie pendant le temps t_{off}.

$$(\Delta I_l)_{off} = \frac{V_s}{L}\, t_{off} \qquad (1-13)$$

Ces variations du courant sur un cycle du hachage engendrent des ondulations, donc des harmoniques, mais on peut écrire en régime permanent, pour dire qu'à une exactitude près les ondulations peuvent êtres négligées et le signal peut être rapproché à sa composante constante.

$$\left(I_s = \frac{V_s}{R_l}\right) \Longrightarrow (\Delta I_l)_{on} + (\Delta I_l)_{off} = 0 \qquad (1-14)$$

On obtient alors l'expression fondamentale :

$$V_s = \frac{t_{on}}{t_{on} + t_{off}}\, V_e \qquad (1-15)$$

En appelant T la période de récurrence d'état de $\boldsymbol{K}$, on peut écrire $T = t_{on} + t_{off}$ ce qui permet d'écrire :

$$V_s = \frac{t_{on}}{T} V_e \qquad (1-16)$$

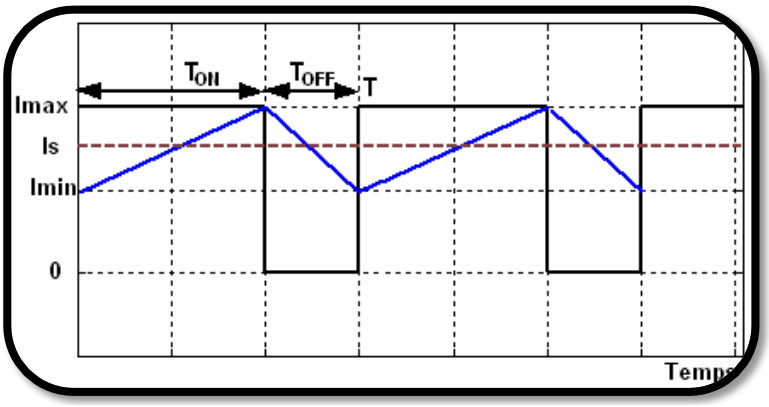


Figure 1.14: Courant dans la self du hacheur série

Ce régime de conduction de courant est appelé le régime continu.

Lorsque : $T > t_{on} + t_{off}$, le régime est appelé conduction discontinue (figure1.13). Dans ce cas le courant de sortie vaut :

$$I_s = \frac{I_M}{2T}\left(t_{on} + t_{off}\right) = \frac{V_e - V_s}{2TL} t_{on}\left(t_{on} + t_{off}\right) = \frac{V_s}{R_L} \qquad (1-17)$$

Ainsi :

$$V_s = \frac{1}{1 + \frac{2LT}{{t_{on}}^2}\frac{I_s}{V_e}} V_e \qquad (1-18)$$

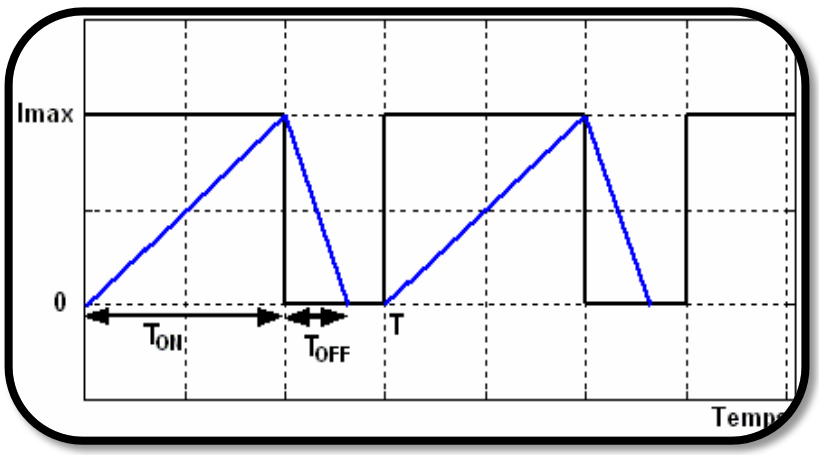


Figure 1.15: Courant en discontinuité dans la self du hacheur série

Savoir le comportement réel de ce convertisseur, nécessite de connaître en détail son modèle mathématique. Pour cela nous devons faire la représentation du circuit équivalent par les deux états du commutateur et de tirer par suite le modèle mathématique reliant les variables d'entrée/sortie. La (figure1.16) montre le schéma du circuit équivalent d'un convertisseur dévolteur avec le commutateur fermé, tandis que la (figure1.17) représente le convertisseur dévolteur avec le commutateur ouvert pendant$(1-d\,)TS$.

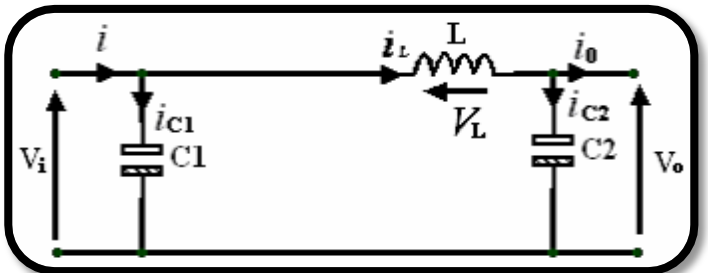


Figure 1.16: Schéma équivalent du hacheur série quand t $\in [\mathbf{0}, \boldsymbol{dT_s}]$

Lorsqu'on applique la loi de Kirchhoff sur le circuit ci-dessus on aura les équations suivantes :

$$i_{c1} = C_1 \frac{dV_i(t)}{dt} = i(t) - i_L(t) \qquad (1-19)$$

$$i_{c2} = C_2 \frac{dV_0(t)}{dt} = i(t) - i_0(t) \qquad (1-20)$$

$$V_L(t) = L\frac{di_L(t)}{dt} = V_i(t) - V_0(t) \qquad (1-21)$$

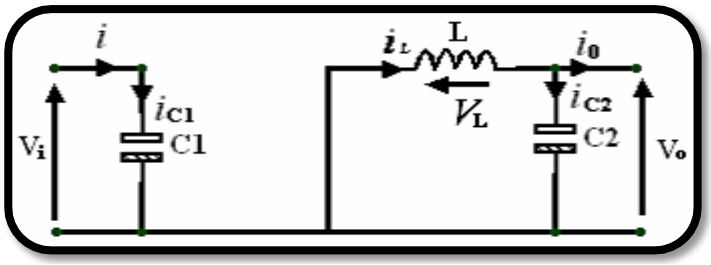


Figure 1.17: Schéma équivalent du hacheur série quand t $\in [\boldsymbol{d.T_s}, \boldsymbol{T_s}]$

Les équations suivantes sont déduites de la figure ci-dessus :

$$i_{C1}(t) = C_1 \frac{d\,V_i(t)}{dt} = i(t) \qquad (1-22)$$

$$i_{C2}(t) = C_2 \frac{dV_0(t)}{dt} = i_L(t) - i_0(t) \qquad (1-23)$$

$$V_L(t) = L\frac{d\,i_L(t)}{dt} = -V_0(t) \qquad (1-24)$$

Les équations de (1-19) à (1-24) sont les équations de base du hacheur dévolteur.

Les valeurs moyennes de la tension et du courant sont données par :

$$V_{l,moy} = < V_l > = \frac{1}{T}\int_0^T V_l(t)\,dt = 0 \qquad (1-25)$$

$$I_{c,moy} = < i_C > = \frac{1}{T}\int_0^T i_C(t)\,dt = 0 \qquad (1-26)$$

1.10.4 Le rapport de conversion :

L'expression de la tension d'inductance peut être facilement dérivée de la forme d'onde sur la (figure1.18).

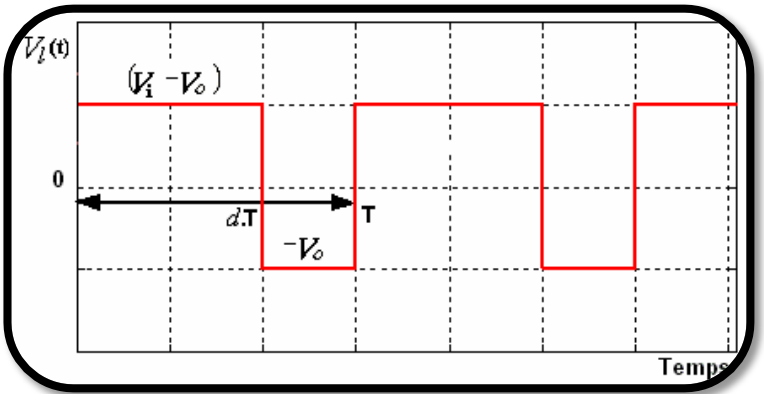


Figure 1.18: La caractéristique de la tension d'inductance V_l (t) pour un hacheur série

La tension moyenne d'inductance est égale à zéro en régime permanent :

$$V_L(t) = d(V_i - V_0) + (1-d)(-V_0) = 0 \qquad (1-27)$$

Les équations de base qui décrivent les caractéristiques du courant continu à l'état d'équilibre d'un hacheur dévolteur sont :

$$d.V_i = V_0 \qquad (1-28)$$

$$I = d.I_L \qquad (1-29)$$

$$I_i = I_0 \qquad (1-30)$$

On déduit donc de l'équation (1.28) que le rapport de conversion d'un hacheur dévolteur est donné par la relation (1.31) en considérant que le convertisseur est idéal (sans perte) :

$$M(d) = \frac{V_0}{V_i} = d \qquad (1-31)$$

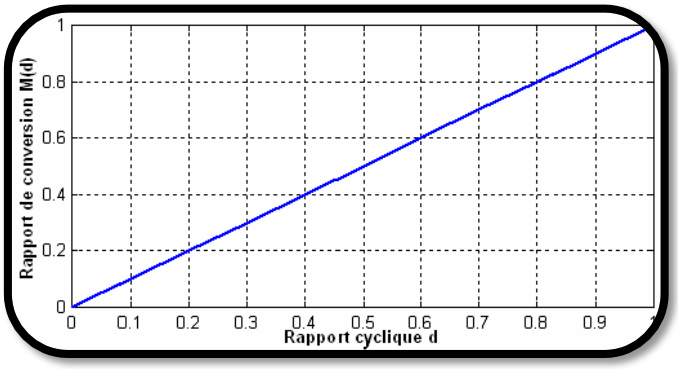


Figure 1.19: Le rapport de conversion M (d) pour un hacheur série

1.11 Modélisation de la batterie :

Le modèle de batterie le plus simple se compose d'un ensemble de force électromotrice en série avec une résistance interne. Mais il se trouve que ce modèle ne peut pas expliquer une autre réalité de la batterie, c'est que lorsque la batterie est mise en circuit ouvert, cette dernière perd sa charge électrique avec le temps.

Lu, de Liu, et de Wu utilisent un modèle qui a été à l'origine développé par Salameh, Casacca, et Lynch dans et comme il est représenté sur la (figure1.23). C'est une amélioration du modèle généralement utilisé de circuit équivalent de Thévenin. Le modèle de Thévenin décrit le comportement primaire d'une batterie correctement, mais n'explique pas les propriétés lentement changeantes d'une batterie telles que l'augmentation et la diminution de la tension de fonctionnement provoquée par le processus de la charge et de la décharge.

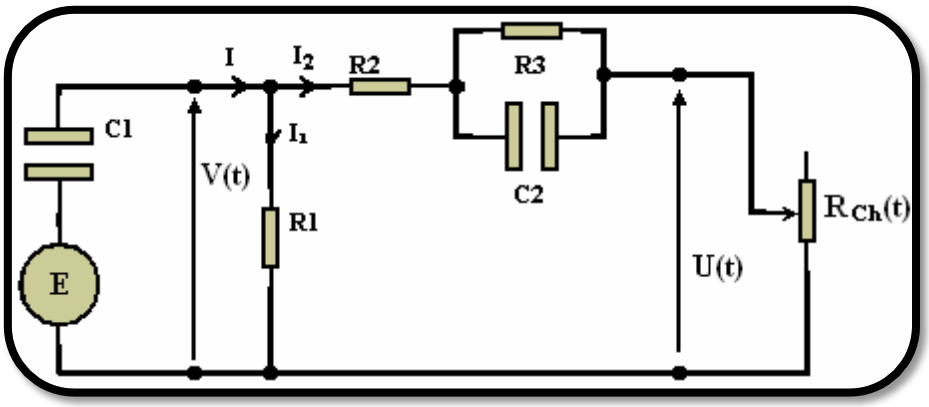


Figure 1.20: Modèle électrique équivalent d'une batterie plomb-acide

Ce nouveau modèle de batterie inclut les composants équivalents pour toutes les caractéristiques de fonctionnement principales d'une batterie plomb-acide : La capacité

électrochimique de la batterie est représentée par le condensateur C1 ayant une tension V(t) à ses bornes qui est donnée par :

$$V(t) = \frac{q(t)}{C_1} + E \qquad (1-32)$$

Avec :

$q(t)$ est la charge électrique du condensateur $C1$ à l'instant t , son énergie est donné par l'équation :

$$w_c = \frac{1}{2}\, C_1 V^2(t) \qquad (1-33)$$

Cependant le comportement de la capacité d'une batterie est autre que le comportement de la capacité d'un simple condensateur.

Cela se concrétise au niveau de l'équation (1.32), à $t = 0 \Rightarrow V_c = \frac{q_0}{C} = 0$, or que celle de la batterie ne vaut pas zéro à son plus bas état de charge, $V(0) = E$ c'est équivalent à un condensateur ayant un niveau minimum de charge, qui est modélisé par une source de tension E en série avec le condensateur $C1$, et qui est équivalent à une énergie minimale $W_{C_{1,min}}$.

$R1$ est une résistance de fuite, c'est à travers elle que s'effectue la décharge d'une batterie en circuit ouvert. $R2$ résistance en bloc d'électrolyte et de plaque et $R3$ résistance de la diffusion d'électrolyte, ce sont des résistances internes de valeurs faibles de quelques Ohms. $U(t)$ est la tension mesurée au niveau des bornes externes de la batterie.

Lorsque la capacité de la batterie $C1$ est entièrement chargée, cet état est représenté par un niveau maximum de son énergie $W_{c,max}$.

$$W_{bat} = W_{C1,max} - W_{C1,min} = \frac{1}{2} C_1 {V_{max}}^2(t) - \frac{1}{2}\, C_1 {V_{min}}^2(t)$$

$$= \frac{1}{2} C_1 \left({V_{max}}^2(t) - {V_{min}}^2(t) \right) \qquad (1-34)$$

L'énergie W_{bat} est donnée en (KWh).

On peut à ce moment exprimer la tension de la batterie $U(t)$ en fonction de la tension de la batterie en circuit ouvert et les autres composants $R2$, $R3$ et $C2$ avec la constante de temps $\tau = R3.C2$.

$$U = U_{oc} + R_3(1 - \exp(\frac{-t}{\tau}))\, I_2 + R_2 I_2 \qquad (1-35)$$

La (figure1.23) peut être mathématiquement exprimée dans le domaine fréquentielle représentant l'impédance équivalente d'entrée d'une batterie plomb-acide.

$$Z(s) = R_2 + R_3(C_2 + R_1)C_1 = R_2 + \frac{R_3}{R_3C_2s + 1} + \frac{R_1}{R_1C_1s + 1} \qquad (1-36)$$

Pour une utilisation dans le modèle mathématique du système, l'équation (1-36) peut être transformée sous la forme suivante :

$$Z(s) = \frac{a_2s^2 + a_1s + a_0}{b_2s^2 + b_1s + b_0} \qquad (1-37)$$

$$a2 = R1R2R3C1C2\,, a1$$
$$= R2R3C2 + R1R2C1 + R1R3C1 + R_1\mathrm{R}_3\mathrm{C}_2 \qquad (1-38)$$

$$a\,0 = R1 + R2 + R3\,, b2 = R1R3C1C2\,, b1 = R1C1 + R3C2\,, b0 = 1. \qquad (1-39)$$

C'est la forme finale du modèle mathématique de la batterie qui sera employé dans les simulations.[6]

1.12 Méthodes de poursuite MPPT :

Afin de faire la poursuite du P.P.M du système photovoltaïque, et comme on l'a déjà vu dans la section 1.10, la tension de sortie du système photovoltaïque est régulé par le convertisseur DC/DC qui lui-même est contrôlé continuellement par son rapport cyclique. Il existe plusieurs méthodes de commande pour la poursuite MPPT.

1.12.1 Philosophie du MPPT :

Presque la plupart des méthodes de poursuite du PPM reposent sur la caractéristique P-V de la cellule PV (voir figure 1.21).

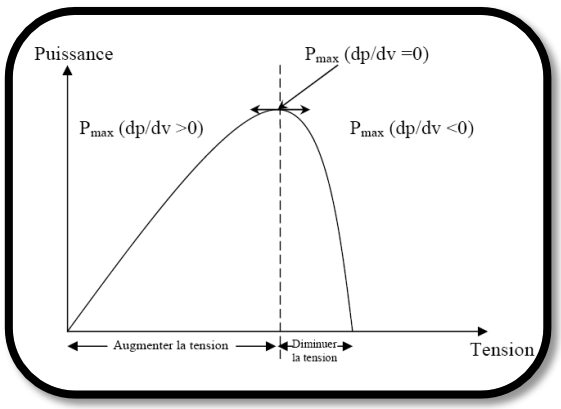


Figure 1.21. Caractéristique P-V d'une cellule PV.

La caractéristique P-V est découpée en deux demi plans :

- Le demi plan gauche ou la dérivée $dp/dv > 0$: Si on est dans cette zone donc on essaye d'augmenter la tension pour aller vers le PPM et cela se fait par la décrémentation du rapport cyclique D ce qui veut dire que :
  $D(k) = D(k-1) - C$ (Avec C : le pas d'incrémentation),
- Le demi-plan droit ou la dérivée $dp/dv < 0$: Si on est dans cette zone donc on essaye de diminuer la tension pour aller vers le PPM et cela se fait par la l'incrémentation du rapport cyclique D ce qui veut dire que :

$$D(k) = D(k-1) + C.$$

Il existe cite plusieurs méthodes qui fonctionnent suivant cette philosophie, par exemple:

- L'Algorithme de contrôle MPPT adaptatif.
- L'Algorithme Perturbation et Observation simple (P and O simple).
- L'Algorithme de l'incrémentation de conductance.
- Méthodes MPPT par contre réaction de tension.

Les deux premières sont expliquées brièvement par la suite.

1.12.2 Algorithme de contrôle MPPT adaptatif :

Le principe de cette méthode a été proposé par A.F. Boeringer .L'algorithme est décrit comme illustré sur la (figure 1.22) :

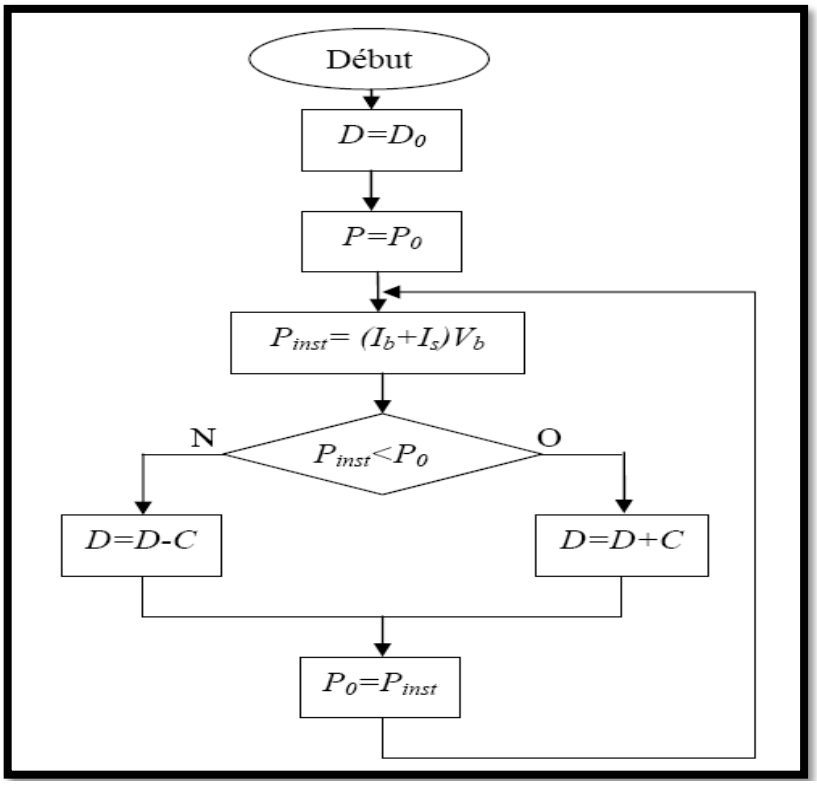


Figure 1.22. Principe classique d'une méthode MPPT.

Comme illustré sur le schéma de la figure (1.22), le système démarre avec un rapport cyclique initial $D0$ et une puissance initiale$P0$, ensuite on mesure la puissance instantanée du générateur PV, et puis on la compare avec$P0$. Si $Pinst$ est inférieure à $P0$ alors D est incrémenté, sinon D est décrémenté. Et enfin $P0$ prend la valeur$Pinst$.

1.12.3 Algorithme Perturbation et Observation simple (P and O simple) :

La figure (1.23) donne l'organigramme de l'algorithme P&O.

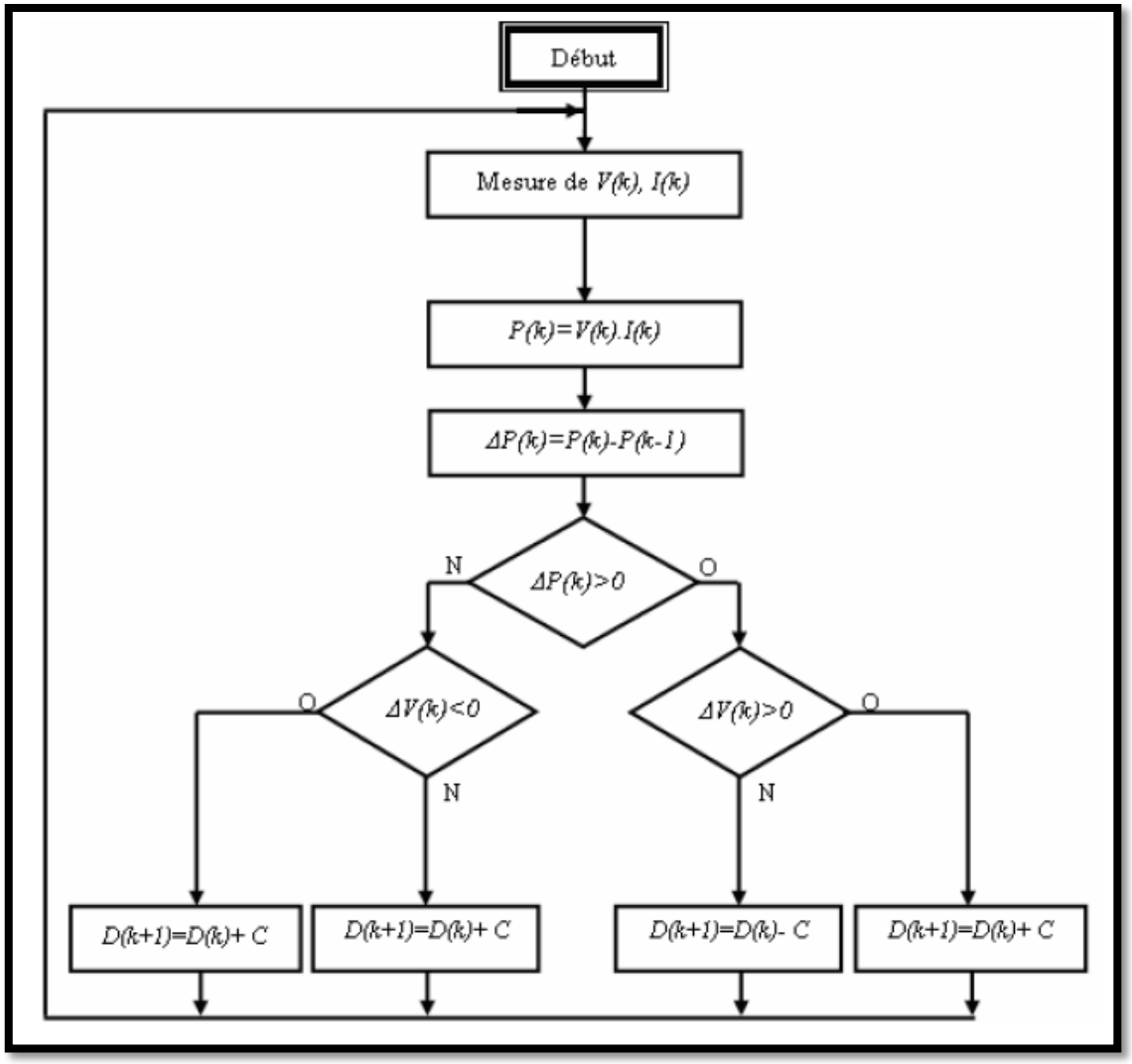


Figure 1.23 Organigramme de l'algorithme Perturbation et Observation (P and O).

L'algorithme P&O fonctionne comme suit :

D'abord on calcule la puissance *P(k)* et on la compare avec la valeur *P(k-1).* Ensuite on effectue un test sur le $\Delta V(k)$ *;* s'il est positif par exemple donc on est dans le plan gauche de la caractéristique P-V ce qui veut dire qu'on doit augmenter la tension, donc on incrémente le rapport cyclique, et ainsi de suite…[2]

1.13 Conclusion :

Dans ce chapitre on a présenté brièvement les systèmes photovoltaïques ainsi que leur réponse aux différentes perturbations. Un générateur PV présente sur sa caractéristique courant-tension un seul point de fonctionnement qui fournit la puissance max, dont il faut poursuivre.

Notre travail sera justement la conception d'un contrôleur qui permet la poursuite du point de puissance maximale, basé sur la logique floue et les réseaux de neurones, une méthode dite neuro-floue.

Chapitre : 2

La logique floue, les réseaux de neurones et les réseaux neuro-flous

2.1 Introduction :

Dans ce chapitre, on va faire une brève étude théorique sur la logique floue, les réseaux de neurones et leur combinaison ; à savoir, les réseaux neuro-flous, nous allons voir aussi comment les utiliser dans la commande afin de les appliquer dans notre cas particulier, qui est la poursuite du point de puissance maximale des panneaux solaires.

2.2 Le caractère flou et les règles linguistiques *:[7]*

Il y a une imprécision inhérente dans notre langage lorsqu'on décrit des phénomènes qui n'ont pas de frontières bien déterminées, comme (le lait est chaud, cet homme est grand etc.…).Les ensembles flous sont des objets mathématique pour modéliser cette imprécision.
Notre principal souci est de manipuler, représenter, faire des déductions depuis ces déclarations imprécises. La théorie des ensembles flous nous permet d'avoir un outil mathématique pour un raisonnement approximatif lorsque les informations disponibles sont incertaines, imprécises, incomplètes, ou vagues .Avec l'utilisation du concept du ***« degrés d'appartenance »*** pour donner une définition mathématique pour les ensembles flous, on augmente le nombre de circonstances encourus dans le raisonnement humain qui peut être soumis à une investigation scientifique.

L'homme fait beaucoup de choses qu'on peut classer dans le domaine du « ***contrôle »,*** des exemples comme : conduire un vélo, frapper une balle de tennis, etc.… Comment fait-on de telles choses ?on n'a pas de mesures précises, pas d'équations différentielles, qui nous disent comment contrôler nos mouvements, mais les humains peuvent devenir très habiles et accomplir des tâches très compliquées. Une explication est, que nous apprenons à travers les

expériences, nôtre bon sens, et par apprentissage, en appliquant un nombre indéfini de règles de la forme ***« si...alors… »*** :

« S'il fera froid demain alors, je vais porter des vêtements chauds. »

L'utilisation de ce type de règles est l'idée de base derrière le contrôle flou. Les variables linguistiques comme **rapide**, **lent**, **grand**, **petit**, sont traduit en ensembles flous ; la version mathématique des règles ***« si…alors… »*** est formée par la combinaison de ces ensembles flous.

2.3 Les ensembles flous dans le contrôle :

Il est facile d'exprimer les règles avec des mots, mais la notion du flou comme illustrée dans les règles ci-dessus, à besoin d'être représenter d'une manière mathématique pour être utiliser dans la théorie du contrôle. Comment fait-on cela ?

La modélisation mathématique des concepts flous a été faite pour la première fois par le professeur **Lotfi Zadeh** en **1965**, pour décrire mathématiquement la classe des objets qui ne possèdent pas des critères exacts d'appartenance. Son affirmation est, que le ***« sens »*** est une affaire de degrés.

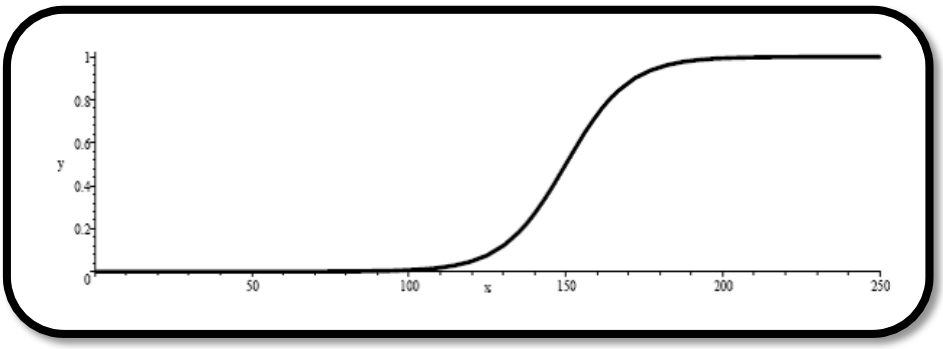


Figure 2.1 : « grandeur » de la hauteur en centimètre

Zadeh a donné des exemples, « la classe des hommes grands» et, « la classe de tout les enfants malades » ; la notion de **grand**, peut être illustrée par un graphe comme celui de la figure 2.1. L'axe des x représente la hauteur en centimètre, et l'axe des y représente le degré dans une échelle allant de 0 à 1, de la grandeur attribuée à cette hauteur. Avant de définir mathématiquement un ensemble flou, on verra les sous-ensembles ordinaires d'une manière spéciale, qui nous permettra d'étendre la notion de sous-ensemble au sous-ensemble flou. Un sous-ensemble ordinaire A d'un ensemble X peut être identifié par la fonction : $X \rightarrow \{0, 1\}$, autrement dit de X vers l'ensemble de 2 éléments $\{0, 1\}$.

$$A(x) = \begin{cases} 0 \; si \; x \notin X \\ 1 \; si \; x \in X \end{cases} \qquad (2-1)$$

Cette fonction s'appelle « ***fonction caractéristique*** » de l'ensemble flou A. Si A est l'ensemble des nombres réels supérieurs ou égaux à 10 et inférieurs ou égaux à 40, l'ensemble A sera défini comme la figure 2.2.

Au contraire, les éléments des sous-ensembles flous d'un ensemble ont des degrés d'appartenance variant de 0 à 1.

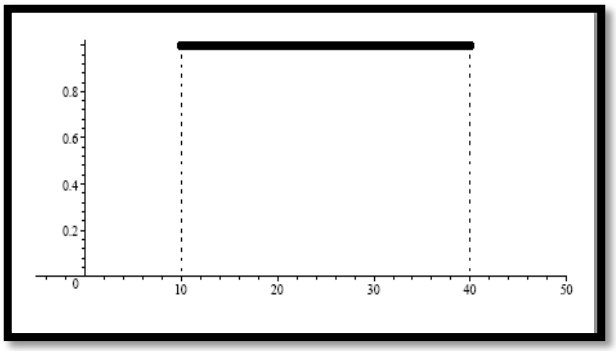


Figure 2.2 A = [10, 40]

2.3.1 Définition d'un ensemble flou :

Un sous-ensemble flou A d'un ensemble X est une fonction $A : X \rightarrow [0,1]$, autrement dit de X vers l'intervalle [0,1].

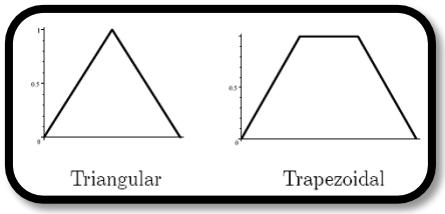
La valeur de $A(x)$ est interprétée comme le degré d'appartenance de x dans A. Cette fonction est parfois appelée **fonction d'appartenance** de A. Dans le cas spéciale où la fonction prend seulement la valeur 0 ou 1, A est appelé un sous-ensemble ordinaire de X et sa fonction d'appartenance coïncide avec sa fonction caractéristique. L'ensemble X est appelé parfois « ***univers de discours*** » .On ne fera pas de différenciation notationnelle entre le sous-ensemble flou A et sa fonction d'appartenance $A : X \rightarrow [0,1]$.

Le support de la fonction $A : X \rightarrow [0,1]$ est l'ensemble :

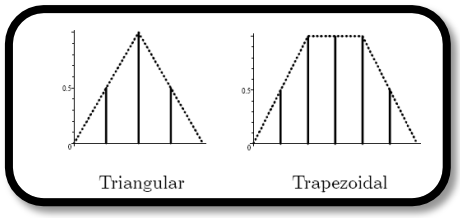
$$supp(A) = \{x \in X \mid A(x) \neq 0\} \qquad (2-2).$$

Ayant une collection de sous-ensembles flous de nombres réels, on peut assumer qu'ils partagent le même univers de discours en prenant X la réunion des supports des ces ensembles flous.Si deux ensembles flous reposent sur le même support et prennent les même valeurs sur ce support, en générale, on n'a pas besoin de faire la différence entre eux.

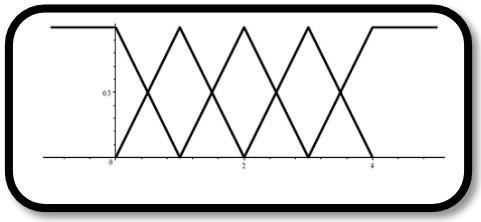
Un ensemble flou est normale s'il existe un $x \in X$ tel que $A(x) = 1$.Un sous-ensemble $A : \mathbb{R} \rightarrow [0,1]$ est convexe si: $x \leq y \leq z$, implique $f(y) \geq f(x) \wedge f(z)$. un ensemble flou $A : \mathbb{R} \rightarrow [0,1]$ avec un support fini qui est à la fois **convexe** et **normale** est appelé un « ***nombre flou*** ». Puisque les données sont généralement numériques, l'univers du discours est un intervalle de nombres réels, ou en pratique un ensemble fini de nombres réels. Les fonctions d'appartenance les plus utilisées sont gaussiennes, sigmoïdes, triangulaire, z-fonction, s-fonction, trapézoïdales figure2.3.

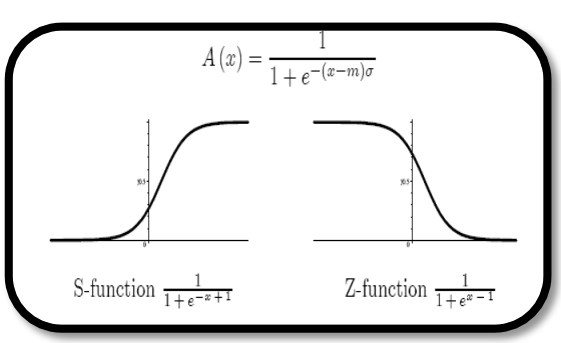


Pour le domaine numérique :



On peut représenter plusieurs fonctions sur un même graphe :





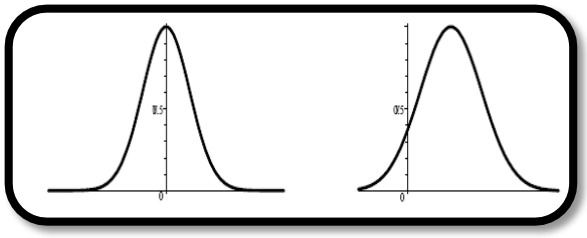


Figure2.3 Les fonctions d'appartenance les plus utilisées

2.4 La combinaison des ensembles flous :

Lorsqu'on travaille avec une collection d'ensembles flous, on a besoin de moyens utiles pour les combiner. Ces moyens doivent coïncider avec les méthodes ordinaires lorsque les ensembles deviennent ordinaires. En d'autres termes, les méthodes de combinaisons des ensembles flous doivent généraliser les méthodes pour les ensembles ordinaires. Ces opérateurs de connections sont appelées ***« opérateurs d'agrégation ».***

Ces opérateurs sont utilisés dans la théorie du contrôle flou pour modéliser des systèmes réels, donc ils doivent non seulement obéir à certains axiomes, mais ils doivent aussi convenir au comportement réel des systèmes ; la rapidité du calcul, est aussi un critère très important.

2.4.1 Minimum, maximum, et le complément :

Les sous ensembles ordinaires d'un ensemble X sont souvent combinés ou niés via l'intersection (**ET**), l'union(**OU**), et le complément (**NON**) :

$$\text{ET} : A \cap B = \{x \in X : x \in A \; et \; x \in B\} \qquad (2-3)$$

$$\text{OU} : A \cup B = \{x \in X : x \in A \; ou \; x \in B\} \qquad (2-4)$$

$$\text{NON} : X - A = \{x \in X : x \notin A\} \qquad (2-5)$$

Ces opérations sont déterminées par leurs fonctions caractéristiques. Les fonctions caractéristiques pour, l'intersection, l'union, et le complément sont :

$$(A \cap B)(x) = \begin{cases} 1 \; si \; x \in A \; et \; x \in B \\ 0 si \; x \notin A \; ou \; x \notin B \end{cases} \qquad (2-6)$$

$$(A \cup B)(x) = \begin{cases} 1 \; si \; x \in A \; ou \; x \in B \\ 0 si \; x \notin A \; et \; x \notin B \end{cases} \qquad (2-7)$$

$$(X - A)(x) = \begin{cases} 1 \; si \; x \notin A \\ 0 si \; x \in A \end{cases} \qquad (2-8)$$

Pour ces fonctions caractéristiques représentant des ensembles ordinaires, les équations suivantes sont satisfaites :

$$(A \cap B)(x) = A(x) \wedge B(x) =^{ou} A(x)B(x) =^{ou} \max\{A(x) + B(x) - 1,0\} \qquad (2-9)$$

$$(A \cup B)(x) = A(x) \vee B(x) =^{ou} A(x) + B(x) - A(x)B(x) =^{ou} \min\{A(x) + B(x), 1\} \qquad (2-10)$$

$$(X - A)(x) = 1 - A(x) =^{ou} (1 - A(x)^a)^{\frac{1}{a}} =^{ou} \frac{1-x}{1+(a-1)x} \; pour \; a > 0 \qquad (2-11)$$

Chacune des équations ci-dessus mène à la généralisation du **ET**, **OU**, et le **NON** pour les sous ensembles flous. Les opérateurs flous classiques les plus utilisés sont:

$$\text{ET : } (A \wedge B)(x) = A(x) \wedge B(x) \quad (minimum) \qquad (2-12)$$

$$\text{OU :} (A \vee B)(x) = A(x) \vee B(x) \quad (maximum) \qquad (2-13)$$

$$\text{Non :} (\neg A)(x) = 1 - A(x) \quad (complément) \qquad (2-14)$$

Où $A(x) \wedge B(x) = \min\{A(x), B(x)\}$ et, $A(x) \vee B(x) = max\{A(x), B(x)\}$.

Les fonctions d'appartenances pour ces ensembles flous seront notées par $A \wedge B$ pour le minimum, $A \vee B$ pour le maximum, et A' ou $\neg A$ pour le complément.

Exemple :

Supposer que, $= [0,1]$, $A(x) = x^2$, et B(x)=$sin\pi x$. Les ensembles $A \wedge B, A \vee B, et\ B'$ sont illustrés ci-dessous :

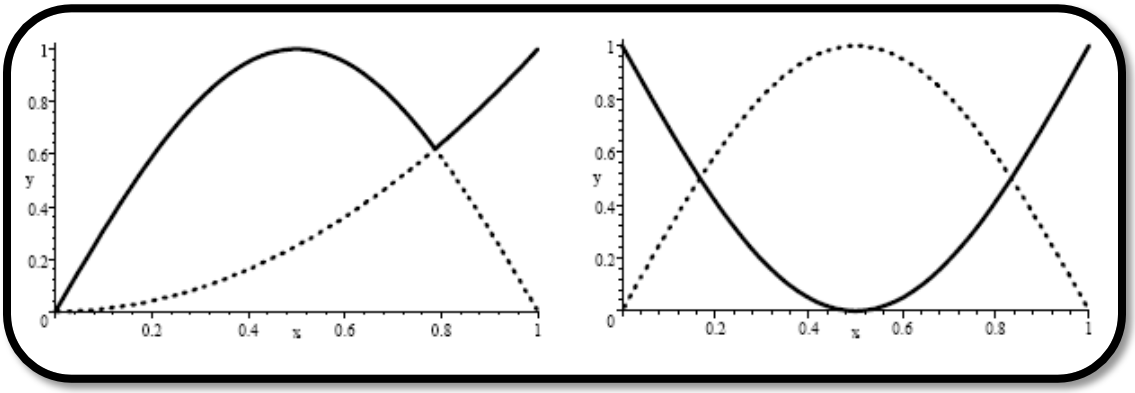


Figure 2.4 Exemple d'agrégation des ensembles flous

A ∧ B(pointillé), A ∨ B(Continu) **B(pointillé), B' (continu)**

Il existe une relation importante entre ces trois opérations, connu par : les lois de **De-Morgan.**

$$(\mathrm{A} \vee \mathrm{B})' = \mathrm{A}' \wedge \mathrm{B}' \text{ et } (\mathrm{A} \wedge \mathrm{B})' = \mathrm{A}' \vee \mathrm{B}' \qquad (2-15)$$

On dit que chacune de ces « ***opérations binaire*** » est duale par rapport à l'autre, relativement au complément.

2.4.2 T-norme, T-conorme, négation :

Malgré que le minimum, le maximum et le complément sont les opérateurs les plus utilisés pour le **ET**, le **OU** et la **NEGATION**, il existe d'autres possibilités pour les ensembles flous. Quelques unes d'entre elles sont mentionnées dans les sections suivantes.

2.4.2.1 Norme triangulaire :

Il existe des situations qui font appel à plusieurs définition du **ET**, et la plus part du temps, elles utilisent une T-norme.

La définition générale d'une T-norme est comme suit :

Définition :

Une T-norme est une relation binaire : $\circ\ [0,1] \times [0,1] \rightarrow [0,1]$ qui satisfait pour tout $x, y, z \in [0,1]$

$$1.\ x \circ y = y \circ x \qquad (2-16)$$

$$2.\ x \circ (y \circ z) = (x \circ y) \circ z \qquad (2-17)$$

$$3.\ x \circ 1 = 1 \circ x = x \qquad (2-18)$$

$$4.\ y \leq z \Rightarrow x \circ y \leq x \circ z \qquad (2-19)$$

Le terme ***« opération binaire »,*** veut dire simplement qu'une T-norme est une fonction de deux variables. La T-norme la plus utilisée est l'opérateur ***« min »***, et c'est aussi la plus grande des t-normes cela veut dire :

$$x \circ y \ \leq x \wedge y \qquad (2-20)$$

, pour n'importe quelle T-norme $\circ$ .La T-norme la plus petite est appelée « ***drastique*** », et, elle est discontinue.

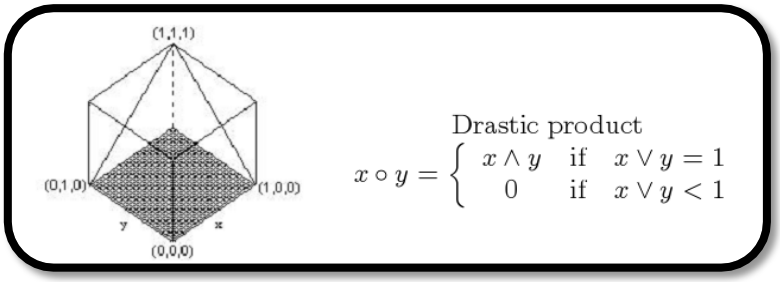


Figure 2.5 La T-norme la plus petite

Les opérateurs les plus utilisés de T-normes autres que le « min » ; sont la T-norme ***« produit »*** ou, le « ***produit algébrique*** ».

$$x \circ y = xy \qquad (2-21)$$

Et aussi la « ***lukazeiwics t-norme » :***

$$x \circ y = max\{x + y - 1,0\} = (x + y - 1) \vee 0 \qquad (2-22)$$

Ces trois T-normes (min, lukazeiwics, le produit algébrique) sont présentés sur la figure suivante :

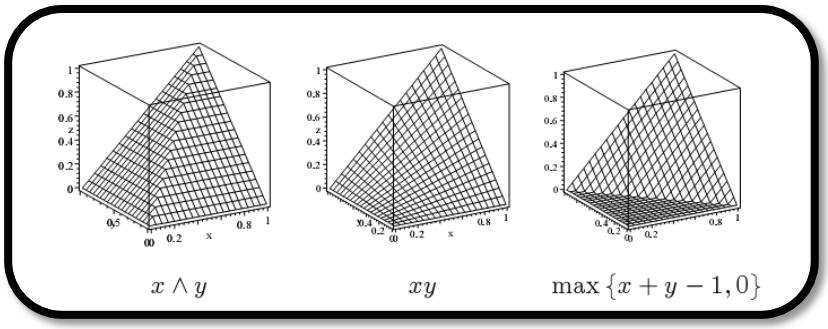


Figure2.6 Les trois T-normes (min, lukazeiwics, le produit algébrique)

Une T-norme qui satisfait $x \circ x = x$ pour tout x est appelée ***idempotente.*** Une T-norme continu qui satisfait $x \circ x < x$ pour tout x sauf, 0 et 1, est appelée ***archimédienne*** .Toutes les T-norme continus autres que le minimum sont archimédienne. Les trois exemples cités ci-dessus sont basiques, dans le sens qu'ils constituent une base pour trouver toutes les autres t-normes continu. Pour une T-norme archimédienne qui satisfait $x \circ x = 0$ seulement lorsque x = 0, est appelée ***stricte***. La T-norme ***produit*** est le prototype d'une T-norme stricte. Les autres T-norme sont appelée ***nilpotentes.*** La T-norme de Lukasiewicz est le prototype d'une T-norme nilpotente.

2.4.2.2 La T-conorme :

La généralisation pour le **OU** logique est la **T-conorme.**

Définition

Une t-conorme est une relation binaire : $*\,[0,1] \times [0,1] \rightarrow [0,1]$ qui satisfait pour $x, y, z \in [0,1]$:

$$1.\ x * y = y * x \qquad (2-23)$$

$$2.\ x * (y * z) = (x * y) * z \qquad (2-24)$$

$$3.\ x * 0 = 0 * x = x \qquad (2-25)$$

$$4.\ y \leq z \Rightarrow x * y \leq x * z \qquad (2-26)$$

Maximum $x \vee y$ est la plus petite T-conorme (et la plus utilisée). La plus grande T-conorm est appelée **somme drastique,** et elle n'est pas continue.

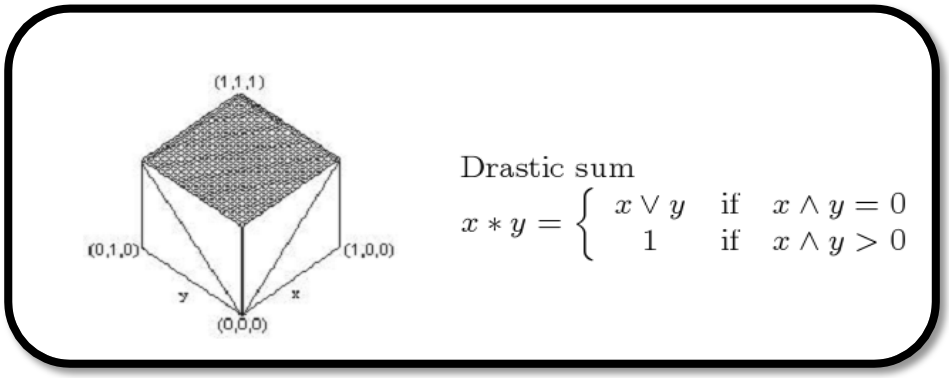


Figure 2.7 La T-conorme la plus grande

En plus du maximum, il existe d'autre t-conorme comme la « ***somme algébrique*** »:

$$x * y = x + y - xy \qquad (2-27)$$

Et la « ***Lukasiewicz »*** t-conorme :

$$x * y = (x + y) \wedge 1 \qquad (2-28)$$

Ces trois T-conormes sont représentées sur la figure suivante :

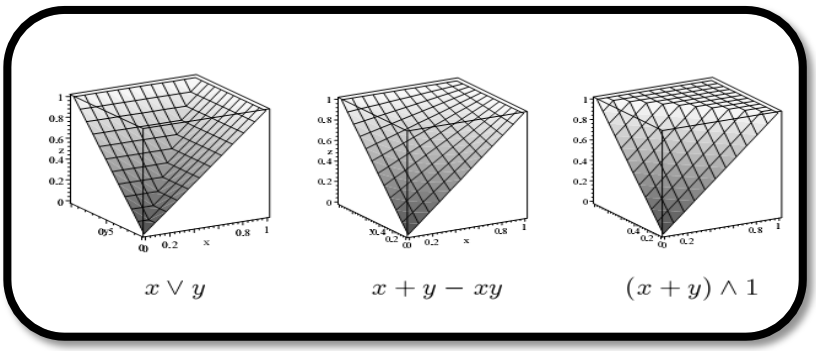


Figure2.8 Les trois T-conormes (max, la somme algébrique, lukazeiwics t-conorme)

Ces trois t-conormes forment une base dans le sens ou n'importe quelle t-conorme continu peut être obtenue par leur combinaison.

Une **t-conorme** qui satisfait $x * x = x$ pour tout x s'appelle ***idempotente.*** La t-conorme maximum est idempotente, et c'est la seule. Une t-conorme continue qui satisfait pour tout x , sauf 0 et 1, $* x > x$, est ***archimédienne***. Toutes les t-conormes qu'on va considérer autre que le maximum sont archimédiennes. Une t-conorme continu pour laquelle $* x = 1$, seulement pour x = 1, est appelée ***stricte***. La t-conorme ***somme algébrique*** est le prototype d'une t-conorme stricte. Les t-conormes archimédiennes qui ne sont pas strictes sont ***nilpotentes***. La Lukasiewicz t-conorme est un prototype pour une t-conorme nilpotente.

2.4.2.3 Négations :

L'opérateur logique $x' = \textbf{non}\, x$, satisfait $1' = 0$ et $0' = 1$, $x \leq y \Longrightarrow x' \geq y'$. Cette opération sera une négation ***forte*** si elle satisfait $(x')' = x$. Une telle opération est appelée une ***involution*** ou une ***dualité***. Les négations fortes sont caractérisées comme des fonctions continues et strictement décroissantes.

$\eta: [0,1] \rightarrow [0,1]$: qui satisfait :

$$\eta\big(\eta(x)\big) = x \qquad (2-29)$$

$$\eta(1) = 0 \qquad (2-30)$$

$$\eta(0) = 1 \qquad (2-31)$$

Le mot ***négation*** est souvent employé pour dire une négation forte .Ici il y a quelques exemples :

$$Standard: x' = 1 - x \qquad (2-32)$$

$$Sugeno: x' = \frac{1-x}{1+ax}, a > -1. \qquad (2-33)$$

$$Yager: x' = (1 - x^a)^{\frac{1}{a}}, a > 0. \qquad (2-34)$$

$$Exponentielle: x' = e^{\frac{a}{\ln(x)}}, a > 0. \qquad (2-35)$$

$$Logarithmique: x' = log_a(a + a^x + 1), a > 0, a \neq 1. \qquad (2-36)$$

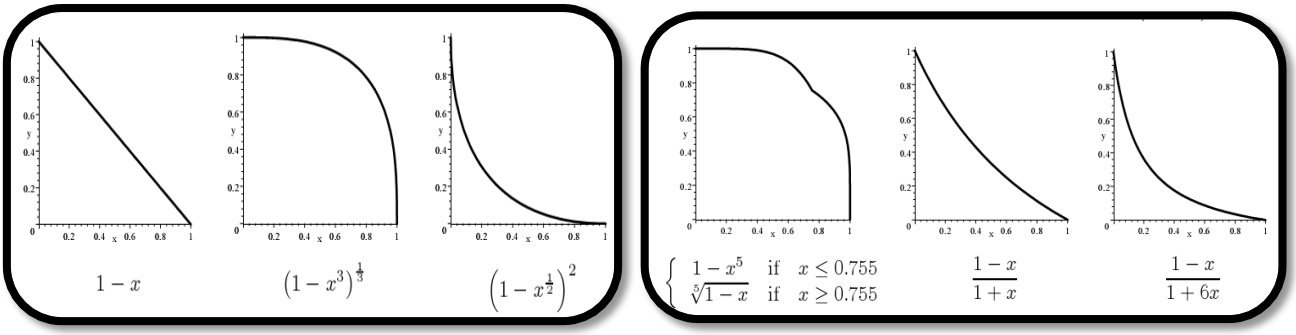


Figure 2.9 Quelques représentations pour les négations fortes

Ces graphiques sont tous symétriques par rapport a la ligne $y = x$, c'est parce que $\eta(\eta(x)) = x$ qui implique que pour chaque point $(x, \eta(x))$, le point $\big(\eta(x), \eta(\eta(x))\big) = (\eta(x), x)$ appartient aussi au graphe. Les deux derniers exemples sont des négations de Sugeno, des fonctions de la forme $x' = \frac{1-x}{1+\lambda x}, \lambda > -1$.

Une T-norme nilpotente a naturellement une négation forte associée avec elle. C'est le résiduel défini par :

$$\eta(x) = \bigvee\{y : x \vartriangle y = 0\} \qquad (2-37)$$

Par exemple $1 - x$ est le résiduel pour le produit de Lukasiewicz.

2.4.3 Le système de de-Morgan :

Si « $'$ » est une négation forte alors :

$$(x \vee y)' = x' \wedge y' \qquad (2-38)$$

$$(x \wedge y)' = x' \vee y' \qquad (2-39)$$

Sont toute les deux valables. Elles sont appelées les lois de de-Morgan. Elles resteront valables ou pas pour d'autres t-normes et t-conormes. Lorsqu'elles le sont, on dit que $\circ$ est dual de * via la négation $'$, le triplet $\circ, *, '$ est le système de de-Morgan sur l'intervalle unité. Par exemple $\wedge, \vee$, et $1 - x$ est un système de de-Morgan.

2.4.4 Les opérateurs de la moyenne:

Dans le sens le plus général, ces opérateurs sont des opérateurs d'agrégations qui produisent un résultat qui se situe entre le minimum et le maximum. Ils représentent un certain type de compromis, et sont d'une importance fondamentale pour la prise de décision. Ce sont des opérateurs non associatifs en général.

Définition :

Un opérateur de moyenne est une opération binaire continue : $\oplus : [0,1] \times [0,1] \longrightarrow [0,1]$

Qui satisfait :

$$1.\ x \oplus y = y \oplus x \qquad (2-40)$$

$$2.\ x \oplus x = x \qquad (2-41)$$

$$3.\ x \leq y\ et\ u \leq v \implies x \oplus u \leq y \oplus v \qquad (2-42)$$

Pour un opérateur de moyenne $\oplus$ sur [0,1], c'est toujours vrai que :

$$x \wedge y \leq x \oplus y \leq x \vee y \qquad (2-43)$$

En d'autres mots, la moyenne de x et y se trouve entre x et y. Pour le voir, observer juste que

$$x \wedge y = (x \wedge y) \oplus (x \wedge y) \leq x \oplus y \leq (x \vee y) \oplus (x \vee y) = (x \vee y) \qquad (2-44)$$

, qui découle des propriétés, idempotente et monotone. Les t-normes et t-conormes sont des opérateurs de moyenne .La moyenne arithmétique $\frac{x+y}{2}$ est bien sur, un opérateur de moyenne ; ainsi que la moyenne géométrique$\sqrt{xy}$, et la moyenne harmonique$xy/(x+y)$.

Toutes les opérations de moyenne peuvent être étendues en opérations sur des ensembles flous via :

$$(A \oplus B)(x) = A(x) \oplus B(x) \qquad (2-45)$$

La moyenne quasi-arithmétique est une généralisation directe de la moyenne arithmétique.

Définition :

La moyenne **quasi-arithmétique** est un opérateur de moyenne, strictement monotone et continu, satisfaisant :

$$(x \oplus y) \oplus (z \oplus w) = (x \oplus z) \oplus (y \oplus w) \qquad (2-46)$$

N'importe qu'elle moyenne quasi-arithmétique$\oplus$, peut être écrite sous la forme :

$$x \oplus y = f^{-1}\left(\frac{f(x)+f(y)}{2}\right) \qquad (2-47)$$

Où f est une fonction strictement croissante et continue, $f: [0,1] \to [0,1]$.

, tel que $f(0) = 0\ et\ f(1) = 1$.

Définition :

L'opérateur **et** logique $\widehat{\oplus}$ de Werner est donné par :

$$x \,\widehat{\oplus}\, y = \gamma(x \wedge y) + (1-\gamma)\frac{x+y}{2} \qquad (2-48)$$

Et l'opérateur **ou** logique $\widecheck{\oplus}$ de Werner est donné par :

$$x \,\widecheck{\oplus}\, y = \gamma(x \vee y) + (1-\gamma)\frac{x+y}{2} \qquad (2-49)$$

Pour un certain $\gamma \in [0,1]$.

Un opérateur, suggéré par Zimmermann et Zysno[10], qui est plus générale dans le sens ou la compensation entre l'intersection et l'union est exprimé par un paramètre γ , est appelé le et-compensatif.

Ce n'est pas un opérateur de moyenne dans le sens de notre définition, du moment qu'il n'est pas idempotent. Par contre, il joue le même rôle dans la prise de décision.

Définition :

Le et-compensatif est :

$$\left(\prod_{i=1}^{n} x_i\right)^{1-\gamma} \left(1-\prod_{i=1}^{n}(1-x_i)\right)^{\gamma} \qquad (2-50)$$

Pour un certain. $\gamma \in [0,1]$.

Cet opérateur est une combinaison entre le produit et la somme algébriques .Le paramètre indique où est localisé l'opérateur actuel entre les deux thèses, donnant le produit si $\gamma = 0$,et la somme algébrique lorsque $\gamma = 1$.

Lorsqu'on désire adapter les variations d'amplitudes des objets individuellement, on peut utiliser les **moyennes pondérées généralisées** pour la moyenne de $x1, x2, \dots, xn$ pondérés par le vecteur $w1, w2, \dots, wn$ avec $\sum_{i=1}^{n} w_i = 1$, comme défini par la formule

$$\left(\sum_{i=1}^{n} w_i {x_i}^a\right)^{\frac{1}{a}} \qquad (2-51)$$

Une technique d'agrégation, de Yager, utilise des opérateurs de moyenne pondérés et ordonnés (OWA :weighted averaging operators), qui donnent le plus grand poids pour les objets qui ont la plus grande amplitude.[11]

Définition :

Un opérateur OWA est un opérateur de dimension n, avec un vecteur associé $W = (w1, \dots, wn)$ satisfaisant $w_i \geq 0$ et $\sum_{i=1}^{n} w_i = 1$, est :

$$F_w : \mathbb{R}^n \to \mathbb{R} : (x_1, \dots, x_n) \to \sum_{j=1}^{n} x_{\sigma(i)} w_i \qquad (2-52)$$

Où $(a_{\sigma(1)}, \dots, a_{\sigma(n)})$est le réarrangement des coordonnes de $(x1, \dots, xn)$ pour que $x_{\sigma(1)} \geq \cdots \geq x_{\sigma(n)}$. Le réarrangement des coordonnés en une séquence ordonnée est une part cruciale de cette définition.

2.5 Combinaison des règles floues :

Les règles utilisées dans un système se présentent sous la forme « si x est A alors y est B », ou A et B sont des ensembles flous, avec x est dans le domaine de , et y dans celui de B. C'est comme une implication tel que : « A implique B ». Dans les ensembles flous, il existe beaucoup de généralisations pour les implications classiques.

Le raisonnement appliqué dans la logique floue est décrit comme :

Prémisse 1 : $si\ x\ est\ A'$

Prémisse 2 : $si\ \ x\ est\ A\ alors\ y\ est\ B$

Conclusion : $y\ est\ B'$

Ou A, A', B, B' sont des ensembles flous représentant des concepts flous. Le calcul de B' peut être fait à l'aide d'une règle basique d'inférence appelée ***règle de calcul d'inférence***, c'est-à-dire $B' = R * A'$ où R est une relation floue représentant une implication ou une proposition conditionnelle floue « ***prémisse 2*** ».Ce schéma d'inférence est parfois décrit comme un problème ***d'interpolation***. L'interpolation est au cœur de l'utilité des systèmes basés sur les règles floues, car elle permet d'employer un nombre relativement faible de règles floues pour caractériser une relation complexe entre deux variables ou plus.
Un grand nombre de formules a été proposé pour cette implication, la plus commune est la ***conjonction compositionnelle.***

$$R(u, v) = A(u) \wedge B(v) \qquad (2-53)$$

B' est définit comme suit :

$$B'(v) = (R \circ A')(u) = \bigvee_u \left(A'(u) \wedge A(u) \wedge B(v)\right) \qquad (2-54)$$

On va décrire quatre mécanismes d'inférence et d'agrégation appelés **Mamdani**, **Larsen**, **Takagi-sugéno-kang**, et **Tsukamoto** couramment utilisés pour interpréter une collection de règles :

Si x est Ai alors y est Bi, $\boldsymbol{i = 1, 2, \dots, n}$.

Dans les exemples qui suivent, on va utiliser les même quatre ensembles flous, A_1, A_2, B_1, B_2 pour illustrer la combinaison des règles flous :

$$A_1(x) = \begin{cases} x & si\ 0 \le s \le 1 \\ 2 - x & si\ 1 \le x \le 2 \\ 0 & ailleur \end{cases} \qquad (2-55)$$

$$A_2(x) = \begin{cases} x - 1 & si\ 1 \le x \le 2 \\ 3 - x & si\ 2 \le x \le 3 \\ 0 & ailleur \end{cases} \qquad (2-56)$$

$$B_1(y) = \begin{cases} \frac{1}{8}y & si\ 0 \le y \le 8 \\ -\frac{1}{4}y + 3 & si\ 8 \le y \le 12 \\ 0 & ailleur \end{cases} \qquad (2-57)$$

$$B_2(y) = \begin{cases} \frac{1}{6}y - \frac{2}{3} & si\ 4 \le y \le 10 \\ -\frac{1}{5}y + 3 & si\ 10 \le y \le 15 \\ 0 & ailleur \end{cases} \qquad (2-58)$$

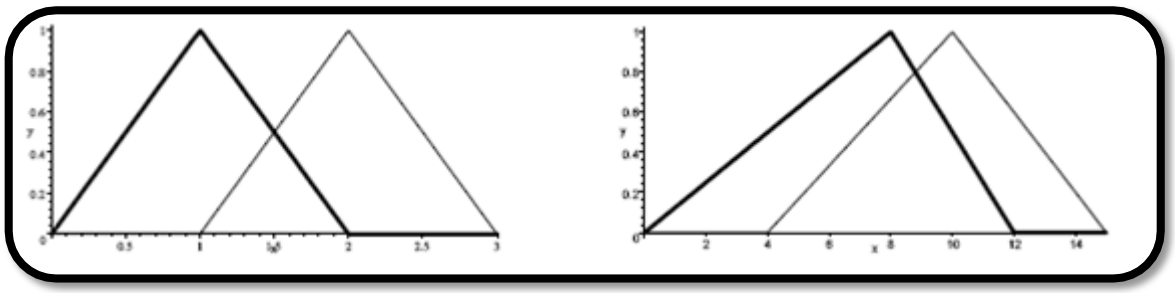


Figure : 2.10 {A1 et A2} {B1 et B2}

En premier, on verra le produit d'ensembles flous, car cela va grandement simplifier la présentation des règles floues pour les modèles de Mamdani et Larsen.

2.5.1 Produit d'ensembles flous :

Le produit d'ensembles ordinaires $X1, X2, \dots \dots, Xn$ est l'ensemble des n-uplet :

$$X_1 \times X_2 \times \dots \times X_n = \prod_{i=1}^{n} X_i = \{(x_1, x_2, \dots, x_n) |\ x_i \in X_i\} \qquad (2-59)$$

Le produit d'ensembles flous :

$A_i: X_i \to [0,1], i = 1, \dots, n$ c'est l'ensemble fou : $A: \prod_{i=1}^{n} X_i \to [0,1]$ définit par :

$$A(x_1, x_2, \dots, x_n) = A_1(x_1) \wedge \dots \wedge A_n(x_n) \qquad (2-60)$$

Ayant les règles :

R_i : Si A_{i1} et A_{i2} et …et A_{ik} alors B_i, $i = 1, 2, .., n$.

Où $Aij : Xj \to [0,1]$ et $Bi : Y \to [0,1]$, prenant le « **ET**» comme le minimum, on peut interpréter ces règles comme :

R_i : Si A_i alors B_i, $i = 1, 2, \dots, n$

Où $A_i = \prod_{j=1}^{n} A_{ij} : \prod_{j=1}^{n} X_j \to [0,1]$, donc dans une situation comme celle-ci, on peut toujours assumer qu'on a juste un ensemble flou A_i pour chaque i, avec le domaine $X = \prod_{j=1}^{n} X_j$ un produit ordinaire d'ensembles .Tout les modèles qu'on va décrire permettent cette simplification. Une règle R_i est dite Qu'elle « ***tire*** » en **x** si $A_i(\mathbf{x}) \neq 0$, en d'autres termes, si **x** est dans le support de A_i.

2.5.2 Le modèle de Mamdani :

Ayant des règles comme, « **si x est** A_i **alors** y **est** B_i », $= 1, \dots\dots, n$, où $\mathbf{x} = (x_1, x_2, \dots, x_n)$, elles sont combinés par le modèle de Mamdani comme suit :

$$R(\mathbf{x}, y) = \bigvee_{i=1}^{n} \big(A_i(\mathbf{x}) \wedge B_i(y)\big) \qquad (2-61)$$

Pou chaque k-uplet $\mathbf{x} = (x_1, x_2, \dots, x_k)$ ceci donne un ensemble flou $R_{\mathbf{x}}$ définit par :

$$R_{\mathbf{x}}(y) = \bigvee_{i=1}^{n} (A_i(\mathbf{x}) \wedge B_i(y)\,) \qquad (2-62)$$

Noter que pour l'ensemble des règles :

***R_i* : Si A_{i1} et A_{i2} et …et A_{ik} alors B_i, $i = 1, 2, .., n$.**

Ceci donne :

$$R_{\mathbf{x}}(y) = R(x_1, x_2, \dots, x_k, y) = \bigvee_{i=1}^{n} (\,A_{i1}(x_1) \wedge \dots \wedge A_{ik}(x_k) \wedge\ B_i(y)). \qquad (2-63)$$

Dans la commande floue, le nombre : $A_{i1}(x_1) \wedge \ldots \wedge A_{ik}(x_k)$ est appelé la « ***force*** » de la règle R_i pour l'entrée **x**. L'ensemble flou $R_{i,\mathbf{x}}$ (y) = A_i(x) $\wedge B_i\,(y)$ est appelé la sortie de contrôle de la règle R_i pour l'entrée **x**, et l'ensemble flou $R_{\mathbf{x}}(y)$ est **la sortie de contrôle agrégée** pour l'entrée **x**.

Exemple :

Prenant les ensembles flous A_i, B_i définis dans les équations (2 -55 /56/57/58) :

En $x = 1.25$, les règles **si x est A_i alors y est B_i**, i=1.2, produit l'ensemble flou :

$$R_{1.25}(y)\begin{cases} \left(\frac{3}{4} \wedge \frac{1}{8}y\right) & si\ 0 \le y \le 4 \\ \left(\frac{3}{4} \wedge \left(\frac{1}{8}y\right)\right) \vee \left(\frac{1}{4} \wedge \left(\frac{1}{6}y - \frac{2}{3}\right)\right) & si\ 4 \le y \le 8 \\ \left(\frac{3}{4} \wedge \left(-\frac{1}{4}y + 3\right)\right) \vee \left(\frac{1}{4} \wedge \left(\frac{1}{6}y - \frac{2}{3}\right)\right) & si\ 8 \le y \le 10 \\ \left(\frac{3}{4} \wedge \left(-\frac{1}{4}y + 3\right)\right) \vee \left(\frac{1}{4} \wedge \left(\frac{-1}{5}y + 3\right)\right) & si\ 10 \le y \le 12 \\ \left(\frac{1}{4} \wedge \left(-\frac{1}{5}y + 3\right)\right) & si\ 12 \le y \le 15 \\ 0 & \text{ailleur} \end{cases} \qquad (2-64)$$

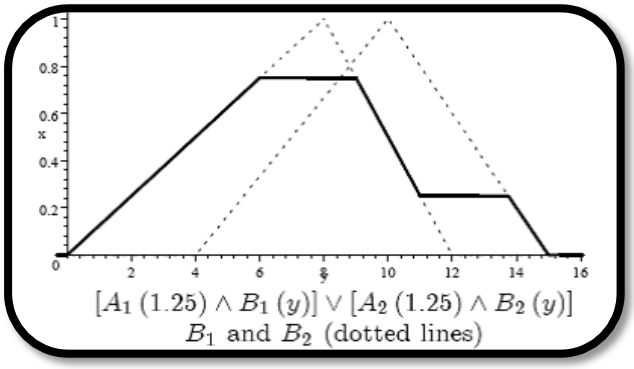


Figure2.11 l'ensemble flou $R_{1.25}(y)$ avec le modèle de Mamdani

2.5.3 Le modèle de Larsen

Ayant des règles **si…alors**, $i = 1 \ldots n$, elles sont combinées par le modèle de Larsen comme suit :

$$R(\mathbf{x}, y) = \bigvee_{i=1}^{n} \left(A_i(\mathbf{x}).B_i(y)\right) \qquad (2-65)$$

Où «.» indique la multiplication. Pour chaque k-uplet $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ ceci donne un ensemble flou :

$$R_{\mathbf{x}}(y) = \bigvee_{i=1}^{n} A_i(\mathbf{x}).B_i(y) \qquad (2-66)$$

Noter que pour l'ensemble des règles

Ri : Si A_{i1} et A_{i2} et ... et A_{ik} alors B_i, $i = 1, 2, \ldots, n$.

Ceci donne :

$R_{\mathbf{x}}(y) = R(x_1, x_2, \ldots, x_k, y) = \vee_{i=1}^{n}\big(A_{i1}(x_1) \wedge A_{i2}(x_2) \wedge \ldots \wedge A_{ik}(x_k)\big).B_i(y)$

Exemple : prenons les ensembles flous A_i et B_i défini précédemment :

En x=1.25, la règle **si x est** A_i **alors y est** B_i **, i=1,2** produit l'ensemble flou :

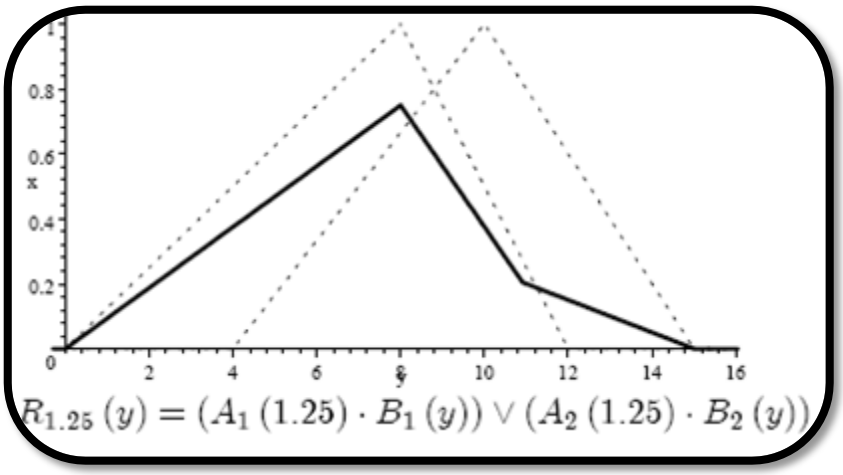


Figure2.12 l'ensemble flou $\mathbf{R_{1.25}(y)}$ avec le modèle de Larsen

2.5.4 Le modèle de Takagi-Sugeno-Kang (TSK)

Pour le modèle TSK, les règles sont données sous la forme :

***Ri* : Si x_1 est A_{i1} et x_2 est A_{i2} et ... et x_k est A_{i1k} alors $f_i(x_1, x_2, \ldots, x_k)$, $i = 1, 2, \ldots, n$**

Ou :

***Ri* : Si $\mathbf{x}_i$ est A_i alors f_i (x), $i = 1, 2, \ldots, n$.**

Où f_1 , f_2 ... f_n sont des fonctions $X = X_1 \times X_2 \times \cdots \times X_k \to R$ et $A_i = \wedge_{j=1}^{k} A_{ij}$

Ces règles sont combinées pour avoir la fonction :

$$R(\mathbf{x}) = \frac{A_1(\mathbf{x})f_1(\mathbf{x}) + A_2(\mathbf{x})f_2(\mathbf{x}) + \cdots + A_n(\mathbf{x})f_n(\mathbf{x})}{A_1(\mathbf{x}) + A_2(\mathbf{x}) + \cdots + A_n(\mathbf{x})} \qquad (2-67)$$

Donc, ce modèle produit une fonction à valeur réelle.

Exemple : prenons les ensembles flous A_i et les fonctions :

$$f_1(x) = 2 + x \qquad f_2(x) = 1 + x$$

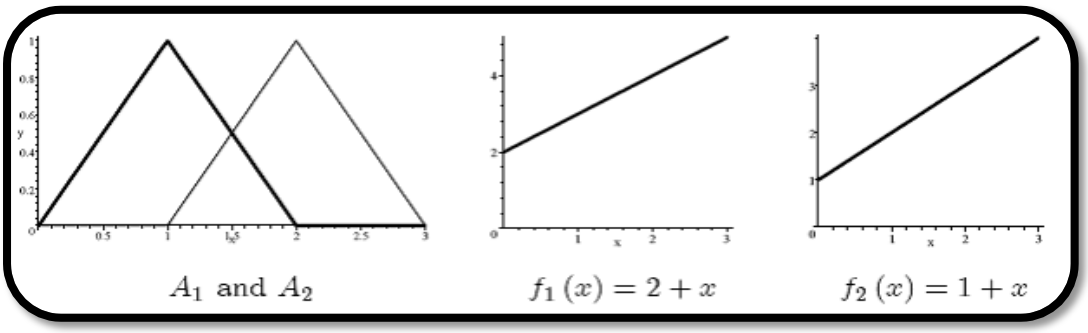


Figure 2.13 Présentation des deux ensembles flous A1 et A2 et les deux fonctions f1 et f2

Alors les règles:

R_i: Si x_i est A_i alors f_i (x), $i = 1, 2$

Produisent la fonction :

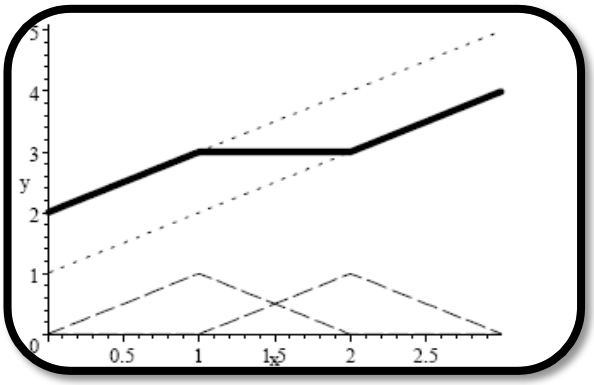


Figure2.14 Résultat de la combinaison des règles Ri : i=1 ,2 avec le modèle TSK

A1 and A2 (lignes en trait interrompu) f1 and f2 (lignes en pointillée)

$$R(\mathbf{x}) = \frac{A_1(\mathbf{x})f_1(\mathbf{x}) + A_2(\mathbf{x})f_2(\mathbf{x})}{A_1(\mathbf{x}) + A_2(\mathbf{x})} \qquad (2-68)$$

2.5.5 Le modèle de Tsukamoto :

Ayant des règles du type **si x est A_i alors y est C_i, $i = 1, \dots, n$**, avec C_i tous monotones (strictement croissante ou décroissante), le modèle de Tsukamoto produit la fonction

$$y = \frac{\sum_{i=1}^{n} C_i^{-1} A_i(x)}{\sum_{i=1}^{n} A_i(x)} \qquad (2-69)$$

La monotonie des ensembles flous Ci est nécessaire pour calculer les fonctions inverses des Ci.

Exemple : prenons deux ensembles flous A_1 et A_2, et les ensembles flous C_1 et C_2 et leurs inverses.

$$C_1(y) = \frac{y}{2}$$
$$C_2(y) = \sqrt{y} \qquad (2-70)$$
$$C_1^{-1}(z) = 2z$$
$$C_2^{-1}(z) = z^2$$

Les règles **si x est A_i alors y est C_i, $i = 1, 2$**, produisent la fonction :

$$R(x) = \frac{\sum_{i=1}^{2} C_i^{-1} A_i(x)}{\sum_{i=1}^{2} A_i(x)} = \begin{cases} 2 & si\ 0 \le x \le 1 \\ 5 - 4x + x^2 & si\ 1 \le x \le 2 \\ 3 - x & si\ 2 \le x \le 3 \end{cases} \qquad (2-71)$$

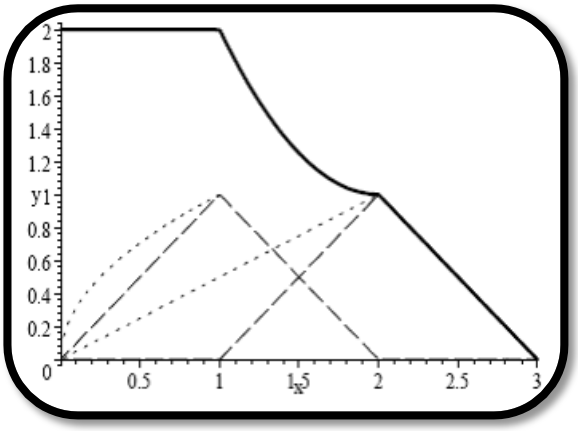


Figure2.15 Résultat de la combinaison des règles Ri : i=1 ,2 avec le modèle de Tsukamoto A1 and A2 (lignes en trait interrompu) C1 and C2 (lignes en pointillée)

2.6 Défuzzification :

La plupart des méthodes pour combiner des règles floues produisent des ensembles flous. Dans la théorie de la commande, une sortie numérique est généralement voulue. Ceci requière certains processus de défuzzification, produisant un nombre qui reflète au mieux l'ensemble flou. Il existe de nombreuses méthodes de défuzzification, on ne citera ici que quelques unes d'entre elles.

On les exposera toutes sur l'ensemble de sortie obtenu par la méthode de Mamdani dans l'exemple (*) de la partie (2.5.2).

En parlant d'une manière générale, il y a deux types de techniques de défuzzification, moments composés et le maximum composé. « Composé » reflète le fait que les valeurs obtenues par la combinaison de plusieurs ensembles flous. Les techniques des moments

composés utilisent le premier moment d'inertie, le maximum composé extrait la valeur pour laquelle l'ensemble flou atteint son maximum.

2.6.1 La méthode du centre de gravité :

La méthode du centre de la zone, ou le centre de gravité, calcule le centre de gravité de la région sous la courbe définie par l'ensemble flou. Si C est l'ensemble flou en question et C est intégrable, alors la valeur défuzzifiée de C par cette méthode est

$$z_0 = \frac{\int_a^b zC(z)dz}{\int_a^b C(z)dz} \qquad (2-72)$$

Où $[a, b]$ est un intervalle contenant le support de C.Si le support de C est fini, le calcul est

$$z_0 = \frac{\sum_{j=1}^{n} z_j C(z_j)}{\sum_{j=1}^{n} C(z_j)} \qquad (2-73)$$

L'ensemble de sortie obtenu dans l'exemple (*) produit la valeur suivante.

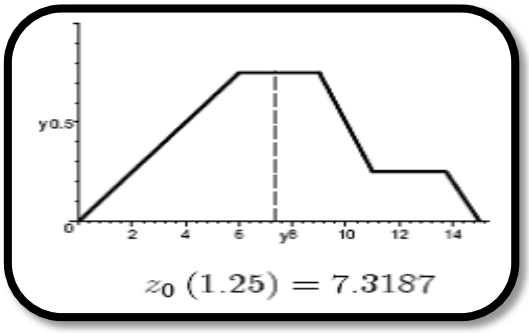


Figure2.16 Illustration la méthode de centre de gravité

La défuzzification par le centre de la zone est la technique la plus répandue.
La valeur défuzzifiée tend à se déplacer doucement en réactions aux petits changements et elle est relativement simple à calculer.

2.6.2 La méthode du centre haut de la zone :

Cette méthode de défuzzification ignore les valeurs de l'ensemble flou en dessous d'un certain $\propto$, en suite on utilise la méthode du centre de gravité sur la courbe restante. Pour $\propto$ = 0.5, l'ensemble de sortie obtenu dans l'exemple (*) produit la valeur z0(1.25) = 7.06.

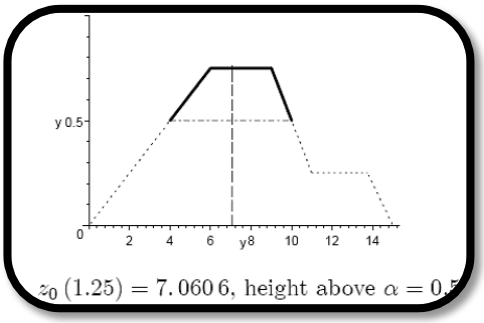


Figure 2.17 Illustration de la méthode du centre haut de la zone

2.6.3 La méthode du critère Max :

Cette méthode choisit une valeur arbitraire de l'ensemble des valeurs dans le domaine pour lequel l'ensemble flou atteint son maximum. L'ensemble de sortie obtenu dans l'exemple (*) produit une valeur $6 \leq z_0(1.25) \leq 9$

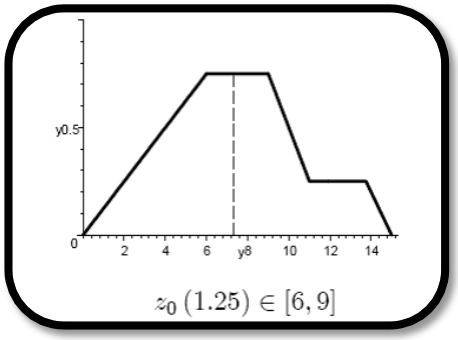


Figure 2.18 Illustration de la méthode du critère Max

Un ensemble flou peut être considéré comme une **distribution de possibilité**, de la variable **Z**, on peut prendre z0 la valeur pour laquelle le degré de possibilité est le plus grand. C'est ça la méthode de défuzzification par le critère maximum.

2.6.4 La méthode du premier maximum :

Cette méthode prend la plus petite valeur dans le domaine pour lequel l'ensemble flou atteint son maximum. La sortie obtenue dans l'exemple (*) produit la valeur $z_0(1.25) = 6$.

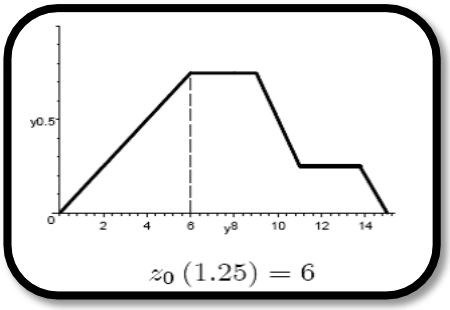


Figure 2.19 Illustration de la méthode du critère Max

2.6.5 La méthode du milieu du maximum :

Cette méthode prend la moyenne de la plus petite valeur et la plus grande valeur dans le domaine où l'ensemble flou atteint son maximum. La sortie obtenue dans l'exemple (*) produit la valeur $z_0(1.25) = 7.5 = \frac{6+9}{2}$

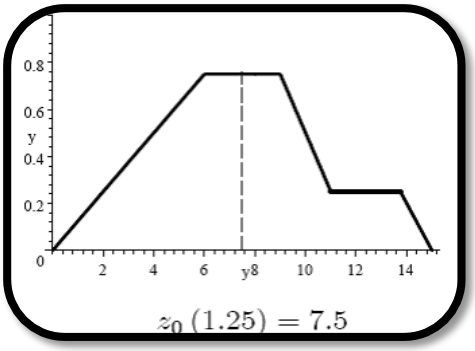


Figure 2.20 Illustration de la méthode du milieu du maximum

Cette méthode parait délicate, surtout si l'ensemble flou se présente sous la forme suivante :

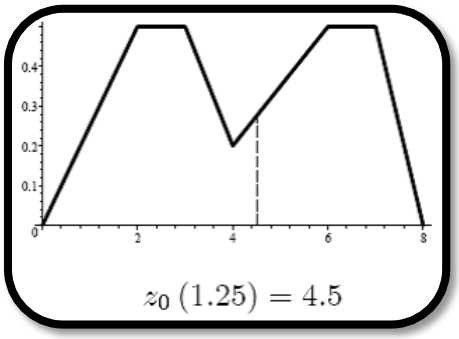


Figure 2.21 Illustration d'un cas délicat de la méthode du milieu du

2.7 La commande floue :

Les contrôleurs basés sur la logique floue exécutent les mêmes taches que les contrôleurs conventionnels, mais ils gèrent des problèmes en utilisant des modèles mathématique flous, au lieu de modèle mathématiques utilisant des équations différentielles. C'est particulièrement utile pour les systèmes dont le modèle mathématique est non linéaire ou bien tout simplement non accessible. L'implémentation de la commande floue ressemble aux lois de contrôle qu'emploient les humains. En créant des machines qui imitent les humains nous ouvrons un nouveau chemin dans la conception des contrôleurs pour des systèmes ayant des modèles mathématiques difficiles à obtenir. Les nouvelles techniques de conception sont basées sur des informations d'un type plus général que les équations différentielles décrivant la dynamique des systèmes.

2.7.1 Exemple un contrôleur flou d'un pendule inversé :

Avec de la pratique, les humains peuvent contrôler un pendule inversé seulement avec leurs sens au lieu d'utiliser des modèles mathématiques compliqués. Pour le pendule inversé, il est possible de concevoir un contrôleur avec succès sans connaitre ou utilisé la dynamique du système. L'avantage est qu'on peut toujours utiliser cette technique dans le cas ou la dynamique du système n'est pas connue.

Pour équilibrer un pendule droit, on sait que la force de contrôle $u(t)$ doit être choisie en accord avec l'amplitude de la variable d'entrée $\theta(t)$ et $\theta'(t)$ qui mesurent l'angle et la vitesse angulaire. La relation entre ces variables est linguistique, beaucoup plus souple que les équations différentielles. C'est probablement ce qui se passe dans le cerveau d'un homme qui calcule les informations qualitativement. Un homme choisit $u(t)$ par le bon sens avec des règles comme :***'si …alors'***, par exemple ***'si $\theta(t)$ est très petite et $\theta'(t)$ est aussi petite alors u est petite'***, ou bien ***'si le pendule est dans une position stable, alors on n'applique aucune force'.***

En prenant en compte toutes les possibilités, le pendule inversé pourra être contrôlé avec succès.

Maintenant, pour pouvoir créer une stratégie de commande automatique imitant l'habileté humaine, il est nécessaire d'être capable de traduire les règles ***'si…alors'*** en formes mathématiques souples pour la machine. Regardons ces règles ***'si...alors..'***, on se pose des questions comme :

- Le 'si…alors' est-il un opérateur d'implication en logique ?
- Comment quelqu'un peut modéliser des adjectifs comme ***'petit'***,**'moyen'** et **'large'** ?
- Ayant un nombre fini de règles **'si...alors'**, comment peut-on manipuler toutes les entrée numériques possibles qui peuvent être mesurés par les capteurs, pour produire l'action de commande ?

Les réponses à toutes ces questions entre dans la théorie de la logique floue. Le terme « ***commande floue »*** se réfère à la science qui conçoit les contrôleurs flous basées sur les mathématiques de la logique floue.

Maintenant, on verra de plus prés la conception d'un contrôleur flou pour un pendule inversé. Les entrées sont la paire (θ, θ'). Soit l'espace d'entrée$X \times Y$, où X et Y sont des intervalles représentant les degrés pour θ et degrés par seconde pour θ'. La sortie ou l'espace de commande pour u est un intervalle représentant des newtons. Les variables linguistiques sont modélisées comme des sous-ensembles flous des espaces X, Y et U par la spécification de leurs fonctions d'appartenances. Par exemple, ces variables linguistiques peuvent

correspondre à grand négatif (NG), moyen négatif (NM) petit négatif (NP), petit positif(PP), petit moyen(PM), et grand positif(PG).

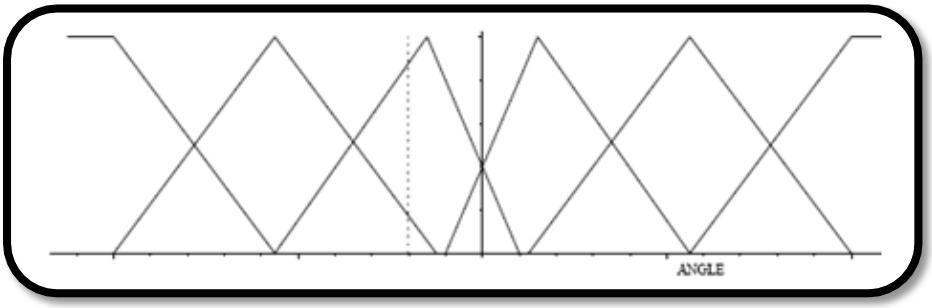


Figure 2.22: Les fonctions d'appartenance pour la position angulaire du pendule

A_1 = NG, A_2=NM, A_3= NP, A4= PP, A_5=PM, A6=PG

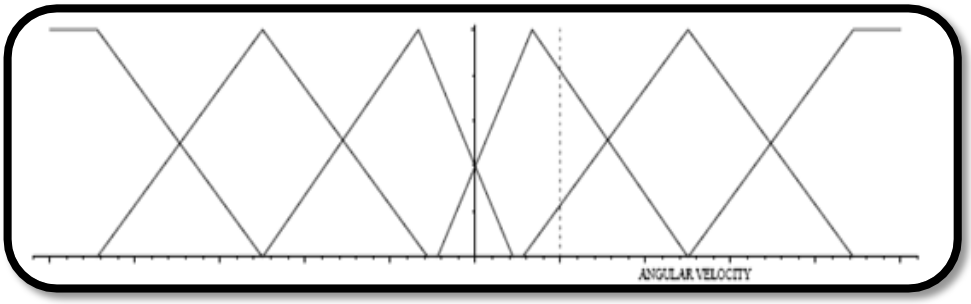


Figure 2.23: Les fonctions d'appartenance pour la vitesse angulaire du pendule

B_1 = NG, B_2=NM, B_3= NP, B4= PP, B_5=PM, B6=PG

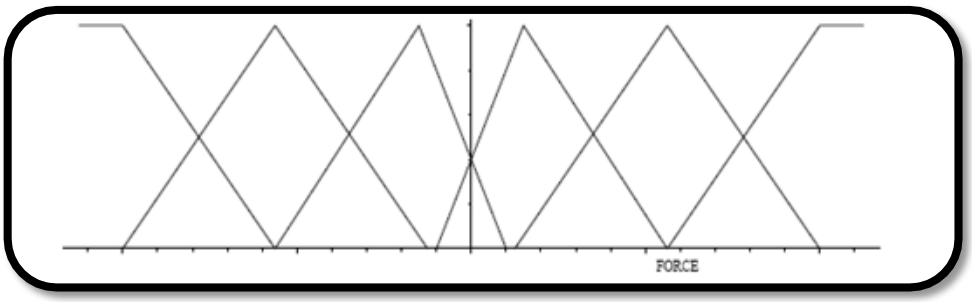


Figure 2.24: Les fonctions d'appartenance pour la force appliquée au pendule

C_1 = NG, C_2=NM, C_3= NP, C4= PP, C_5=PM, C6=PG

Le choix particulier pour une fonction d'appartenance est quelque peu arbitraire. Un des ajustements fait lors du test de la commande du system est d'essayer différentes fonctions d'appartenance, en changeant les paramètres de celle-ci .Ce processus est appelé le **tuning**.

Les règles sont alors de la forme :

1. **Si** l'angle est négatif moyen et la vitesse est moyenne positive **alors** la force appliquée est petite positive.
2. **Si** l'angle est négatif moyen et la vitesse est moyenne positive **alors** la force appliquée est petite positive.
3. Si l'angle est négatif petit et la vitesse est positive moyenne **alors** la force est petite positive.

Et ainsi de suite. Avec six fonctions d'appartenance pour chaque X et Y, on peut avoir 36 règles de la forme

$$\boldsymbol{Rj : si\ \theta\ est\ Aj\ \ et\ \theta'\ est\ Bj\ \ \ alors\ u\ est\ Cj}$$

Où $Aj, Bj, et\ Cj$ sont des sous-ensembles de X, Yet U.

Ci-dessous le tableau (2.1) qui résume les 36 règles dans la forme compacte :

$\theta/_{\theta'}$	NG	NM	NP	PP	PM	PG
NG	NG	NG	NG	NM	NP	PP
NM	NG	NG	NM	NP	PP	PP
NP	NG	NM	NP	PP	PP	PM
PP	NM	NP	NP	PP	PM	PG
PM	NP	NP	PP	PM	PG	PG
PG	NP	PP	PM	PG	PG	PG

Tableau 2.1 Les règles floues pour la commande de pendule inversé

Les entrées (θ, θ') pour chaque règle Rj aura comme conséquence un sous-ensemble flou de U.

Ce sous-ensemble flou est souvent prit comme le minimum :

$$\varphi_j(u) = min\{A_j(\theta), B_j(\theta'), C_j(u)\} \qquad (2-74)$$

La fusion des règles, par l'intermédiaire du maximum, produit un sous-ensemble flou de U représentant une action de commande :

$$\psi(u) = \max\{\varphi_j(u): j = 1,2 \dots, k\} \qquad (2-75)$$

Si, par exemple, les mesures sont θ = −8◦ and θ' = 2◦/ s alors les ensembles flous représentés ci-dessus donnent les valeurs :

$$A_2(-8) = 0.17$$
$$A_3(-8) = 0.88$$
$$B_4(2) = 0.6$$
$$B_5(2) = 0.82$$

Et tous les autres Ai et Bi sont nulle en ce point. Donc les seules règles convenables pour cette entrée sont ces quatre car les autres donnent zéro avec cette entrée.

θ/θ'	NM	NP
PP	NP	NP
PM	NP	PP

Tableau 2.2 les seules règles convenables pour $\theta = -8^\circ$ and $\theta' = 2^\circ/s$

En combinant ces règles on aura l'ensemble flou :

$$\psi(u) = [A_2(-8) \wedge B_4(2) \wedge C_3(u)] \vee [A_2(-8) \wedge B_5(2) \wedge C_3(u)] \vee [A_3(-8) \wedge B_4(2) \wedge C_3(u)] \vee [A_3(-8) \wedge B_5(2) \wedge C_4(u)]$$
$$= [0.17 \wedge 0.6 \wedge C_3(u)] \vee [0.17 \wedge 0.82 \wedge C_3(u)] \vee [0.88 \wedge 0.6 \wedge C_3(u)](\vee [0.88 \wedge 0.82 \wedge C_4(u)] = [0.6 \wedge C_3(u)] \vee [0.82 \wedge C_4(u)] \quad (2-76)$$

Et la sortie finale est le sous-ensemble de U suivant :

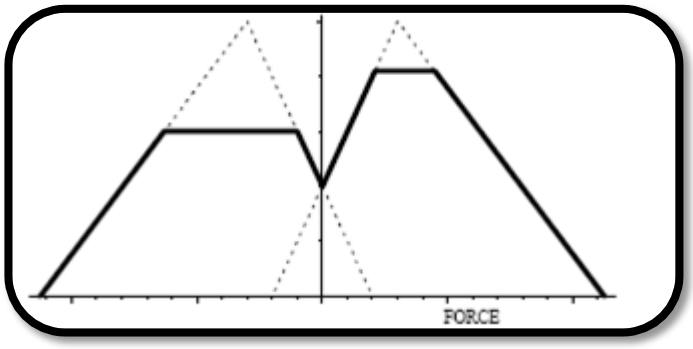


Figure 2.25 Le sous-ensemble produit après la combinaison des quatre règles pour $\theta = -8^\circ$ and $\theta' = 2^\circ/s$

$$\boldsymbol{C_3 = NP \quad C_4 = PP \quad \psi(u) = [0.6 \wedge C_3(u)] \vee [0.82 \wedge C_4(u)]}$$

Pour produire l'action de commande actuelle pour l'entrée (θ, θ') on a besoin de défuzzifier le sous-ensemble $\Psi(u)$. L'action de commande dépend de la technique de

défuzzification appliquée. Un choix naturel est la méthode du **centre de gravité,** explicitement, la commande actuelle est prise comme :

$$u^* = \frac{\int u\psi(u)du}{\int \psi(u)du} \qquad (2-77)$$

La méthodologie décrite ci-dessus avec un tuning suffisant, mènera à une commande réussie du pendule. Les 36 règles possibles déterminées par la table 2.1 sont plus que suffisantes. T.Yamakawa [2] a pu contrôler un pendule inversé sur un chariot avec seulement 7 règles. Il a utilisé un ensemble flou additionnel ***'approximativement zéro'(AZ),*** pour chaque variable linguistique.les sept règles sont les suivantes :

	θ	**θ'**	**u**
1	AZ	AZ	AZ
2	PP	PP	PP
3	PM	AZ	PM
4	PP	NP	AZ
5	NM	AZ	NM
6	NP	NP	NP
7	NP	PP	AZ

Tableau 2.3 Les règles floues optimisées pour la commande de pendule inversé

Dans la pratique, un petit nombre de règles fonctionne souvent aussi bien qu'un grand nombre de règle. Travaillant avec un petit nombre de règle a un grand avantage en terme de cout, d'efficacité, et de vitesse de calcule.

2.7.2 Principales approches à la commande floue :

La connaissance essentielle au sujet de la dynamique d'un système peut être présentée sous forme de base de règles linguistiques. À fin d'effectuer une synthèse de commande, il est nécessaire de modéliser l'information linguistique aussi bien qu'un processus d'inférence, un

processus d'agrégation, et probablement un processus de défuzzification. L'utilisation de la logique floue pour transformer la connaissance linguistique en lois de commande réelles mène au domaine de la commande floue. L'idée principale de la commande floue est de formuler des algorithmes pour des lois de commande utilisant des règles logiques. C'est pourquoi nous appelons parfois cette méthodologie « commande par la logique floue ».

Les règles sont obtenues soit par des experts (contrôleurs humains) ou produites par des observations numériques. Quand les experts humains expérimentés dans la commande d'un système donné sont disponibles, les ingénieurs d'études peuvent les interviewer au sujet de leurs stratégies de commande puis exprimer ces informations comme collection de règles de commande ***« si… alors… ».*** Même lorsque les relations d'entrée-sortie dans un système complexe ne sont pas connues avec précision, le comportement du procédé de commande fournit une source précieuse d'informations pour suggérer des stratégies de commande et extraire des règles floues. C'est-à-dire, on peut appliquer des signaux numériques à l'entrée d'un système de contrôle et prendre des observations sur les sorties ; ensuite les utiliser comme moyens d'obtenir des règles de commande linguistique ***« si… alors… ».***

Il est clair qu'il y a une flexibilité dans le choix des fonctions d'appartenance, des règles, des opérations de logique floue, et des procédures de défuzzification lors de la conception des contrôleurs flous.

Noter que les systèmes flous, ou les modèles flous, sont actuellement des modèles mathématiques, ils sont des descriptions mathématiques des relations entre les variables, utilisant des fonctions d'appartenance des ensembles flous. Ces modèles mathématiques sont flexibles, mais ils ne sont pas « flous » dans le sens exact du terme. Les fonctions d'appartenances, sont des fonctions mathématiques précises.

Juste comme dans le cas de la commande standard, où l'existence des lois de commande devrait être garantie avant la marche à suivre pour obtenir l'algorithme de commande, nous devrions nous demander si une base de règles linguistiques est suffisante pour dériver une loi de commande réussie. La base des règles est une description de la façon dont nous devrions commander le système. Chaque règle est locale en nature, chaque règle nous indique que nous devrions commander le système dans une certaine petite région de l'espace d'entrée X. Puisque nous allons employer cette information pour dériver une loi de commande globale ϕ, ces petites régions spécifiques dont les règles devraient couvrir tous les points de X d'une

certaine manière. En d'autres termes, les fonctions d'appartenances définies sur X devraient former une partition floue de X.-

La méthodologie de la commande floue consiste à choisir et à employer :

1. une collection de règles qui décrivent la stratégie de commande.

2. Des fonctions d'appartenances pour les étiquettes linguistiques dans les règles.

3. Des liaisons logiques pour des relations flous, et :

4. une méthode de défuzzification.

À la fin, la loi de commande dérivée est la réalisation d'une fonction de X àU, où U est l'espace des variables de commande. La stratégie ci-dessus pour dériver des lois de commande est basée légèrement sur le bon sens. Cependant, ses applications réussies peuvent être expliquées théoriquement dans le cadre de la théorie d'approximation des fonctions. Les systèmes flous satisfont une propriété universelle d'approximation, ça signifie qu'il y a un système qui fera la tâche que vous voulez accomplir. Cependant, ceci n'indique pas comment installer ou accorder un tel système flou. De telles justifications mathématiques fournissent seulement une certaine assurance que, si vous travaillez assez dur, il est possible de concevoir des contrôleurs flous réussis.

2.7.2.1 Méthodes de Mamdani et de Larsen

La première application de la logique floue a été développée par E.H. Mamdani en 1974. Il a conçu un système de contrôle flou expérimental pour une combinaison de machine à vapeur et de chaudière en synthétisant un ensemble de règles linguistiques de commande obtenues par des opérateurs humains expérimentés. La méthode populaire connue aujourd'hui comme la méthode de Mamdani est très semblable à sa conception originale.

Pour les méthodes de Mamdani et de Larsen, une base typique de règles est de la forme :

$$\boldsymbol{Rj : Si\ x\ est\ Aj\ alors\ u\ est\ Bj, j = 1, 2, \dots, r}$$

$$avec\ \mathbf{x} = (x_1, \dots\ x_n) \in X\ et\ u \in U, et\ A_j(\mathbf{x}) = min_{i=1\dots n}\{A_{ji}(x_i)\}\ pour:$$

$$A_j = A_{1j} \times A_{2j} \times A_{3j} \times \dots \times A_{nj} : X = X_1 \times X_2 \times \dots \times X_n \rightarrow [0, 1]$$

$$B_j : U \rightarrow [0, 1]$$

Comment pouvons nous interpréter la règle « $\boldsymbol{Rj : Si\ Aj\ (x) alors\ Bj\ (u)}$ » ? Une vue commune est que chaque règle Rj représente une relation floue sur $X \times U$. En d'autres

termes, quand l'entrée est x, alors le degré auquel une valeur $u \in U$ est compatible avec la signification de Rj est :

$$C_j(x,u) = T\left(A_j(x), B_j(u)\right) \qquad (2-78).$$

Où T est une certaine t-norme. Ainsi, Cj est un sous-ensemble flou de $X \times U$. Pour une entrée donnée X, Cj induit un sous-ensemble flou de U, à savoir :

$$C_j^x(x) : u \rightarrow C_j(x,u), u \in U, j = 1,2 \ldots, N \qquad (2-79)$$

Ainsi, la sortie de chaque règle Rj est un sous-ensemble flou de U. La méthode de Mamdani emploie le minimum pour la t-norme :

$$C_j^x(u) = C_j(x,u) = A_j(x) \wedge B_j(u) \qquad (2-80)$$

Et la méthode de Larsen emploie le produit pour la t-norme

$$C_j^x(u) = C_j(x,u) = A_j(x).B_j(u) \qquad (2-81)$$

Après avoir traduit chaque règle Rj en C_j^x, la prochaine tâche est d'assembler toutes les règles ensemble. Nous faisons face au problème suivant : étant donné N sous-ensembles flouC_1^x ... C_N^x de$U$, nous devons les combiner pour produire un résultat global. D'un point de vue sémantique, bien sur, c'est une question de la façon dont les règles sont reliées. D'un point de vue mathématique, nous voulons former un sous-ensemble flou de U avec C_j^x j = 1,…, N. Dans les méthodes de Mamdani et de Larsen, ceci est fait en prenant le maximum. Dans la méthode de Mamdani, pour chaque $x \in X = X1 \cdots \times Xn$, ceci donne le sous-ensemble flou :

$$C^x(u) = C(u|x) = max_{j=1\ldots N}\left(min_{i=1\ldots n}\{A_{ij}(x_i), B_j(u)\}\right) \qquad (2-81)$$

L'équation (2-81) s'appelle la synthèse de Mamdani.

Dans la méthode de Larsen, pour chaque x∈ X, ceci donne le sous-ensemble flou de U.

$$C^x(u) = C(u|x) = max_{j=1\ldots N}\left((min_{i=1\ldots n}\{A_{ij}(x_i)\}.B_j(u))\right) \qquad (2-82)$$

L'équation (2-82) s'appelle la synthèse de Larsen.

La sortie globale est, pour chaque x, un sous-ensemble flou C^x de U. Quelle est la signification de la C^x ? Quand l'entrée x est donnée, chaque action de commande u est compatible avec le degré $C^x(u)$.

Nous avons besoin d'un seul résultat numérique $u^* = u^*(x)$ pour la loi de commande, et nous devons obtenir ceci du sous-ensemble flou C^x. En d'autres termes, nous devons defuzzifier le sous-ensemble flou C^x.

Dans les approches de Mamdani et de Larsen, nous avons défuzzifié la C^x en employant le procédé du centre de gravité, à savoir

$$u^* = \frac{\int u\psi(u)du}{\int \psi(u)du} \qquad (2-83)$$

Naturellement, il y a différentes manières de défuzzifier C^x.

La complexité, et la vitesse du calcul sera affectée par la méthode de défuzzification.

La réponse à comment doit-on choisir une méthode de défuzzification est selon la performance du contrôleur voulue.

2.8 *Réseaux de neurones pour la commande :*

Dans ce qui suit, nous présenterons les dispositifs de calcul connus sous le nom de réseaux de neurones qui sont employés pour résoudre une série de problèmes, y compris des aspects de la commande des systèmes dynamiques complexes. Avec la commande standard, nous comptons sur un modèle mathématique ; avec la commande floue, nous employons un ensemble de règles. Quand les informations disponibles au sujet du comportement d'un système sont principalement des données numériques, la méthodologie des réseaux de neurones est très utile. Vous verrez plus tard, quelques problèmes sont mieux résolus avec une combinaison des approches floues et neurales. Nous avons déterminé ici les idées mathématiques de base utilisées dans les réseaux de neurones pour la commande. Bien que les réseaux de neurones aient des applications dans beaucoup de domaines, nous essayerons d'adresser seulement les parties de la théorie des réseaux de neurones qui s'appliquent directement dans la théorie de commande. Nous aborderons des questions telles que « c'est quoi des réseaux de neurones ? », « Pourquoi nous avons besoin des réseaux de neurones ? », Et « comment pouvons-nous utiliser les réseaux de neurones pour résoudre des problèmes ? ».

2.9 Définition d'un réseau de neurones :[7]

Il est bien connu que les systèmes biologiques puissent effectuer des tâches complexes sans recours aux opérations quantitatives explicites. En particulier, les organismes biologiques sont capables d'apprendre graduellement avec le temps. Ces possibilités

d'apprentissage reflètent la capacité des neurones biologiques d'apprendre par l'exposition au stimulus externe et de généraliser. De telles propriétés des systèmes nerveux les rendent attrayants comme modèles de calcul qui peuvent être conçus pour traiter des données complexes. Par exemple, les possibilités d'apprentissage des organismes biologiques par des exemples suggèrent des possibilités pour l'apprentissage de la machine.

Les réseaux de neurones, ou plus spécifiquement, les réseaux de neurones artificiels sont les modèles mathématiques inspirés de notre compréhension des systèmes nerveux biologiques. Ils sont attrayants comme dispositifs de calcul qui peuvent accepter un grand nombre d'entrées et d'apprendre seulement avec des échantillons d'entrainement.

Comme modèles mathématiques pour les systèmes nerveux biologiques, les réseaux de neurones artificiels sont utiles pour établir des relations entre les entrées et les sorties de n'importe quel genre de système.

En général, un réseau de neurones est une collection de neurones artificiels.

Le neurone artificiel est un modèle mathématique d'un neurone biologique sous sa forme la plus simple. Selon notre compréhension, Les neurones biologiques sont regardés en tant qu'unités élémentaires pour le traitement de l'information dans n'importe quel système nerveux.

Sans réclamer sa validité neurobiologique, le modèle mathématique d'un neurone artificiel est basé sur les thèses suivantes :

1. Les neurones sont les unités élémentaires dans un système nerveux dans lequel le traitement de l'information se produit.
2. L'information entrante est sous forme de signaux qui passent entre les neurones par des liens de raccordement.
3. Chaque lien de raccordement a un poids approprié qui multiplie le signal transmis.
4. Chaque neurone a une fonction d'activation qui dépend d'un seuil de polarisation étant appliquée à la somme pondéré des signaux d'entrée produit un signal de sortie.

Ainsi, quand des signaux d'entrée x_1, x_2,…, x_n arrivent au neurone par des liens de raccordement avec les poids associés w_1, w_2,…, w_n, respectivement, l'entrée résultante du neurone, appelée l'entrée nette, est la somme pondérée $\sum_{i=1}^{n} w_i x_i$. Si le seuil de polarisation est b et la fonction d'activation est f, alors la sortie de ce neurone est :

$$y = f(\sum_{i=1}^{n} w_i x_i - b) \qquad (2-84)$$

Dans le premier modèle de calcul pour les neurones artificiels, proposé par McCulloch et Pitts, les sorties sont binaires, et la fonction f est la fonction échelon défini par :

$$f(x) = \begin{cases} 1 \; si \; x \geq 0 \\ 0 \; si \; x < 0 \end{cases} \qquad (2-85)$$

, de sorte que l'activation de ce neurone soit :

$$f\left(\sum_{i=1}^{n} w_i x_i - b\right) = \begin{cases} 1 \; si \sum_{i=1}^{n} w_i \, x_i \geq b \\ 0 \; si \sum_{i=1}^{n} w_i x_i < b \end{cases} \qquad (2-86)$$

Comme dans la figure 2.26 :

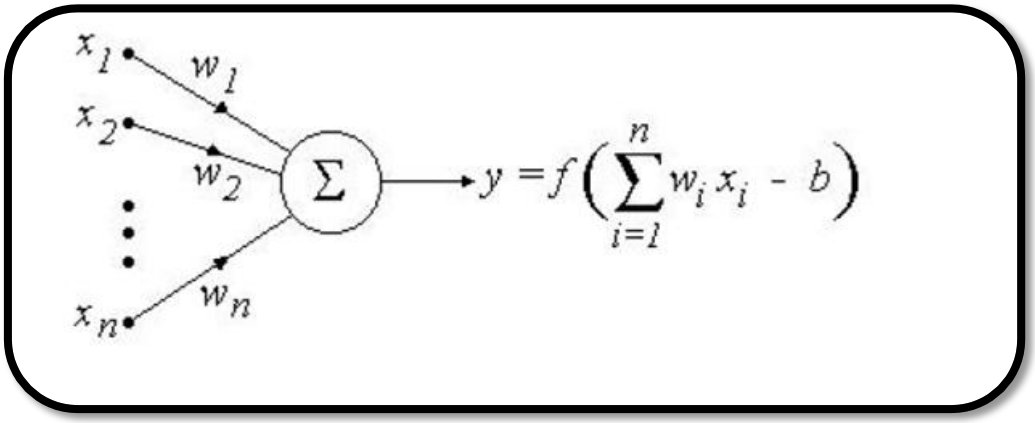


Figure2.26 premier modèle du perceptron

Un neurone artificiel est caractérisé par les paramètres $\theta = (w_1, w_2, \ldots, w_n, b, f)$

Le biais b peut être traité en tant qu'autre « poids » en ajoutant un nœud x0 d'entrée qui prend toujours la valeur d'entrée x0 = +1 et mettant w0 = −b (voir la figure 2.27).

Avec cette représentation, l'ajustement du biais et l'ajustement des poids peuvent être faits de la même manière.

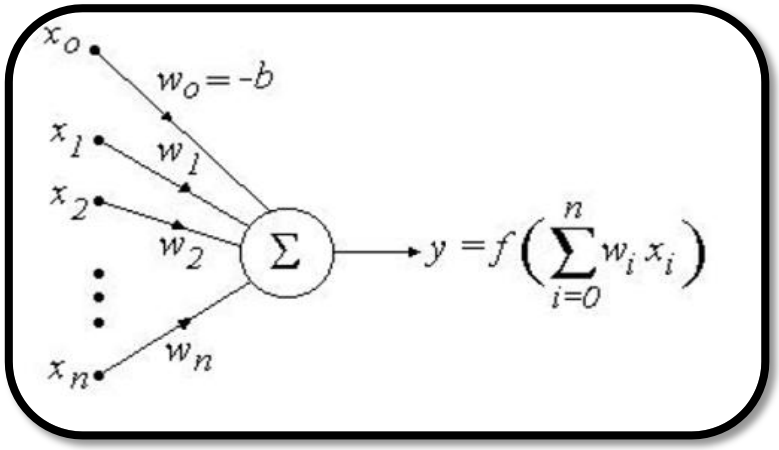


Figure2.27 neurone artificiel avec un biais comme poids

Nous considérerons ici seulement les réseaux de neurones par rétro-propagation, l'information se propage seulement vers l'avant comme indiqué par la direction des flèches.

Mathématiquement parlant, un réseau de neurones par retro propagation est un graphe orienté acyclique pondéré.

En voyant les Neurones artificiels en tant qu'unités élémentaires pour le traitement de l'information, nous arrivons aux réseaux de neurones simples en considérant plusieurs neurones à la fois.

Le réseau de neurones de la figure 2.28 se compose d'une couche d'entrée « une couche de nœuds d'entrée» et une couche de sortie. Ceci désigné sous le nom d'un réseau de neurones à une seule couche parce que la couche d'entrée n'est pas une couche de neurones, aucuns calculs ne se produit aux nœuds d'entrée. Ce réseau de neurones à une seule couche s'appelle un **perceptron**.

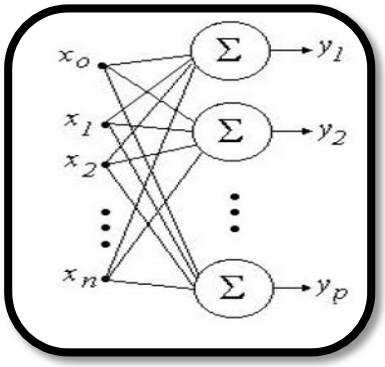


Figure2.28 : Réseau de neurones à une seule couche (Perceptron)

Un réseau neuronal multicouche est un réseau neuronal avec plus d'une couche de neurones. Noter que les fonctions d'activation des différents neurones peuvent être différentes. Les neurones d'une couche ont des connexions pondérées avec les neurones de la prochaine couche, mais aucunes connexions entre les neurones d'une même couche. La figure2.29 montre un réseau neuronal de deux-couches.

La couche d'entrée (ou couche 0) a n+1 nœuds, la couche moyenne, appelée la couche cachée, a p nœuds, et la couche de sortie a m nœuds. Ceci s'appelle un réseau neuronal « n-p-m ».

Les neurones (nœuds) dans chaque couche sont quelque peu similaires.

Des neurones dans la couche cachée, sont **« cachés »**dans le sens que nous ne pouvons pas directement observer leur sortie. Ayant des configurations d'entrée, nous pouvons seulement observer les configurations de sortie par la couche de sortie. Naturellement, un réseau neuronal multicouche peut avoir plus d'une couche cachée.

Le réseau neuronal de deux-couches représenté sur la figure 2.29 est un perceptron multicouches typique (MLP : Multi layer perceptron), un réseau neuronal multicouche dont les neurones remplissent la même fonction sur des entrées, habituellement une association d'une somme pondérée et une fonction d'activation non linéaire différentiable, ou fonction de transfert, telle qu'une fonction tangente hyperbolique. Les perceptrons multicouches sont les structures de réseau neuronal les plus utilisées généralement pour une large gamme d'applications.

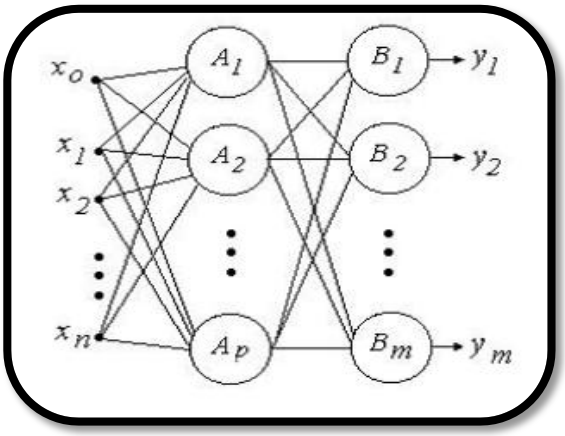


Figure2.29 : Réseau de neurone avec deux couches

2.10 Mise en œuvre des réseaux de neurones

À la différence des calculateurs qui sont programmés à l'avance pour effectuer quelques tâches particulières, les réseaux de neurones doivent être formés par des exemples (apprentissage dirigé) avant d'être employés.

Un réseau neuronal peut être conçu et formé pour effectuer une tâche particulière, par exemple, pour construire une loi de contrôle pour commander un système dynamique.

L'architecture choisie du réseau neuronal est dictée par le problème actuel. Une fois l'architecture du réseau neuronal est choisie, nous devons avoir des échantillons de formation afin d'entrainer le réseau neuronal à effectuer la tâche désirée. La formation des réseaux neuronaux constitue la phase la plus importante pour les mettre en pratiques.

Une fois ces réseaux de neurones sont formés, ils sont prêts à être employés dans les applications comme dispositif de calcul qui produit les sorties appropriées à partir des entrées.

Nous commençons par donner un exemple d'une fonction logique qui peut être mise en application par un perceptron. Noter qu'avec des paires d'entrées (x1, x2) de nombres réels, le domaine (graphe) est divisé en deux parties par la fonction d'activation

$$f(w_1x_1 + w_2x_2 - b) = \begin{cases} 1 \; si \; w_1x_1 + w_2x_2 \geq b \\ 0 \; si \; w_1x_1 + w_2x_2 < b \end{cases} \qquad (2-87)$$

À savoir, les entrées au-dessus de la ligne w1x1 +w2x2 -b = 0 produisent un résultat de 1, et les entrées en dessous de cette ligne produisent un résultat de 0.

En particulier, cette installation fournit une solution seulement si la fonction dont le réseau neuronal essaye de mettre en application est linéairement séparable.

Exemple :

Nous concevons un perceptron pour mettre en application la fonction logique OU, c.-à-d la fonction g : {0, 1} × {0, 1} → {0, 1}, défini par :

$$g(x_1, x_2) = \begin{cases} 0 \; si \;\; x_1 = x_2 = 0 \\ 1 \qquad\quad ailleurs \end{cases} \qquad (2-88)$$

En raison de la fonction g, nous considérons des réseaux neuronaux d'entrée binaire/ sortie binaire, avec deux nœuds dans la couche d'entrée, et seulement un neurone dans la couche de sortie

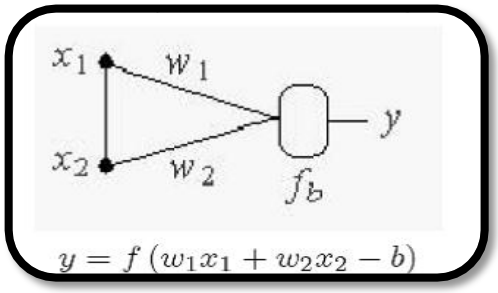


Figure2.30 : La fonction logique OU avec un perceptron

Le problème est de choisir w1, w2, et b de sorte que

$$g(x_1, x_2) = f(w_1x_1 + w_2x_2 - b) \qquad (2-89)$$

D'abord, pour voir si ce problème est résoluble, nous regardons le domaine de la fonction g, colorant en blanc le points où la valeur de la fonction est 1, et en noir le point où la valeur de la fonction est 0.

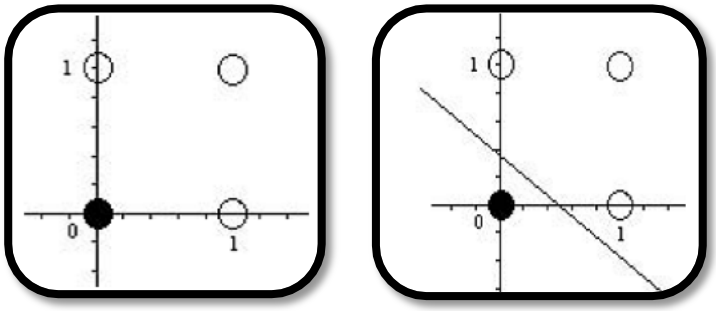


Figure2.31 : Domaine de la fonction g(OU)

Le domaine d'entrée est divisé en deux sous-ensembles B = {(0, 0)} et W = {(0, 1), (1, 1), (1, 0)} correspondant aux deux valeurs de sorties 0 et 1, respectivement.

Une solution à la conception du perceptron est une ligne droite w1x1+w2x2−b = 0, dans le plan x1-x2, et nous pouvons voir qu'il y a des lignes qui peuvent séparer ces deux sous-ensembles.

C'est-à-dire, c'est un problème **linéairement séparable**. Naturellement, il y a beaucoup de telles lignes, et n'importe laquelle donne une solution. En raison de la simplicité de cette fonction g, aucune mathématique sophistiquée n'est nécessaire. Observer juste que :

g (0, 0) = 0 implique 0 < b

g (0, 1) = 1 implique w2 ≥ b

g (1, 0) = 1 implique w1 ≥ b

g (1, 1) = 1 implique w1 + w2 ≥ b

, et choisir tous les numéros w1, w2, et b satisfaisant ces inégalités. Une solution est w1 = w2 = b = 2.

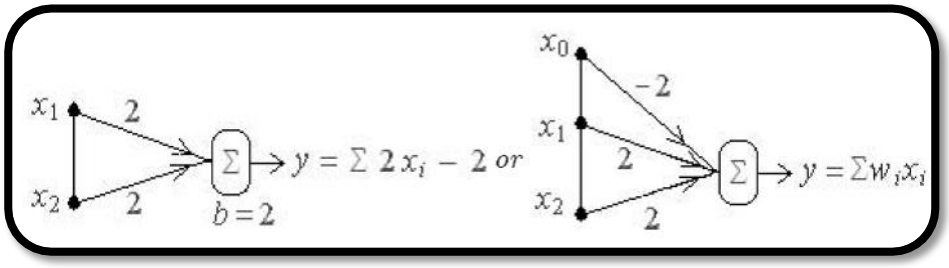


Figure2.32 : Solution trouvée pour la réalisation de la fonction g(OU)

Considérons maintenant une autre fonction logique. Dans l'exemple suivant nous considérons le OU **exclusif.**

Exemple :

Le « ou » exclusif (XOR) est la fonction logique booléenne ayant la table de vérité :

$$g(x_1, x_2) = \begin{cases} 0 \; si \;\; x_1 = x_2 = 1 \\ 0 \; si \;\; x_1 = x_2 = 0 \\ 1 \qquad\quad ailleurs \end{cases} \qquad (2-90)$$

Quand nous exposons la relation entrée-sortie exprimée par g, nous voyons que ce problème n'est pas linéairement séparable.

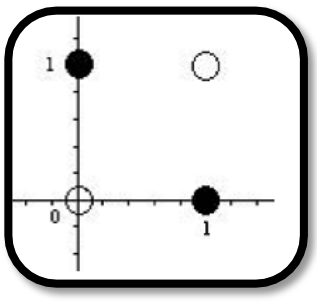


Figure2.33 : Domaine de la fonction g(XOR)

C'est-à-dire, il n'y a aucune ligne qui peut séparer les deux sous-ensembles

W = {(0, 0), (1, 1)} et B = {(0, 1), (1,0)}.

Par conséquent, la fonction g ne peut pas être implémentée en utilisant un perceptron, ainsi, nous devons considérer une couche cachée. Nous savons concevoir S∩T' avec les poids et la polarisation (w_{1A}, w_{2A}, B_A) et (w_{1B}, w_{2B}, b_B), respectivement. De l'exemple 5.1, nous savons aussi concevoir le « OU » avec des poids et la polarisation (w_{AC}, w_{BC}, b_C).

En mettant ces neurones ensemble nous obtenons une solution pour le « XOR », avec l'inconvénient d'avoir une architecture plus compliquée que celle du perceptron, à savoir, l'utilisation de deux couches :

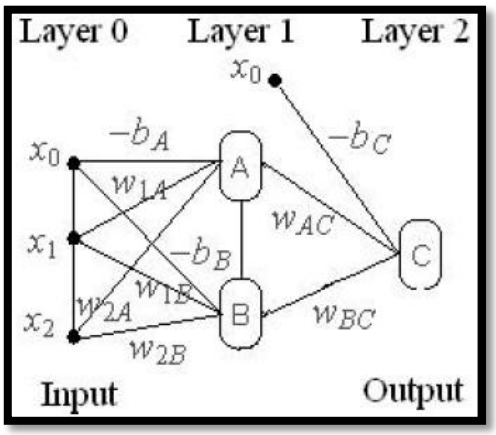


Figure2.34 : la fonction XOR avec les réseaux de neurones

C'est un réseau multicouches dans lequel la couche d'entrée (ou couche 0) a deux nœuds x1, x2 (aucun calcul), et x0 = 1 pour implémenter le biais des neurones A et B ; la couche cachée a deux neurones A et B. Avec x0 = 1 pour implémenter le biais du neurone C ; la couche de sortie se compose seulement d'un neurone C. Pour tous les neurones, la fonction d'activation f est prise comme suit :

$$g(x) = \begin{cases} 1 \; si & x_1 \geq 0 \\ 0 & ailleurs \end{cases} \qquad (2-91)$$

Considérons, par exemple, x1 = x2 = 1, nous nous attendons à ce que le réseau produit la sortie y = 0. La sortie y est calculée en plusieurs étapes. L'entrée nette au neurone A est : $w_{1A}x_1 + w_{2A}x_2 - b_A$ de sorte que la sortie du neurone A soit :

$$y1 = f : (w_{1A}x_1 + w_{2A}x_2 - b_A) \qquad (2-92)$$

De même, la sortie du neurone B est :

$$y2 = f\,(w_{1B}x_1 + w_{2B}x_2 - b_B) \qquad (2-93)$$

Par conséquent, l'entrée nette au neurone C, via les neurones A et B avec la condition qui doit être satisfaite :

$$w_{AC} f(w_{1A} x_1 + w_{2A} x_2 - b_A) + w_{BC} f(w_{1B} x_1 + w_{2B} x_2 - b_B) - b_C < 0 \qquad (2-94)$$

, a comme conséquence la sortie y = 0.

Nous concluons que, plus de couches semblent augmenter la puissance du calcul.

En d'autres termes, pour des problèmes généraux, les réseaux à une seule couche ne sont pas suffisants ; des réseaux multicouches devraient être considérés.

2.11 Capacité d'apprentissage :

Les réseaux neuronaux artificiels sont les modèles mathématiques simplifiés du cerveau, ils font du calcul parallèle. Comme indiqué plus tôt, les réseaux neuronaux doivent être entrainés ou formés par des exemples avant être mis en service. Car nous verrons que, le problème d'apprentissage n'est rien d'autre que de choisir une fonction parmi une classe donnée de fonctions selon quelques critères. D'abord, nous devons savoir quelles fonctions peuvent être mises en application ou représentées par un réseau neuronal. Seulement, après avoir répondu à cette question, nous pouvons chercher des voies pour les concevoir.

Si nous adressons des relations générales exprimées comme des fonctions de R^n vers R^n, alors des réseaux neuronaux avec des fonctions d'activations lisses (plutôt que les fonctions échelon utilisées dans le développement pour perceptrons avec des sorties binaires) peuvent approximer des fonctions continues sur un support compact. Cette propriété universelle d'approximation des réseaux neuronaux a été prouvée en utilisant le théorème Pierre-Weierstrass dans la théorie d'approximation des fonctions. Spécifiquement, toutes les fonctions continues dont le domaine est borné et fermé dans $R^{n,}$ c.-à-d., ayant un support compact, peuvent être approchées avec n'importe quel degré d'exactitude par un réseau neuronal possédant une couche cachée avec la fonction d'activation sigmoïde ou tangente hyperbolique. Ce résultat théorique signifie qu'il est possible de concevoir un réseau neuronal approprié pour représenter n'importe quelle fonction continue ayant un support compact. C'est seulement un théorème d'existence, mais pas constructif.

le théorème nous rassure, puisque la plupart des fonctions d'intérêt pratique sont continues.

En suite vient la question fondamentale. Comment trouvons-nous un réseau neuronal approprié pour représenter une relation donnée ?

Évidemment, nous avons besoin de quelques informations venant d'une certaine relation afin de répondre à la question ci-dessus. Dans le cadre de l'apprentissage, les informations sont

fournies par un ensemble sous forme de paires entrée-sortie connues. Faire l'apprentissage par des informations de ce type est désigné sous le nom d'apprentissage **supervisé**, ou apprendre avec un professeur.

Spécifiquement, le problème d'apprentissage est ceci. Supposer que nous avons un ensemble T d'exemples d'entrainement

$$T = \{(x^q, y^q), q = 1, \dots, N\}$$
$$x^q = \left(x_1^q, x_2^q, \dots, x_n^q\right) \in \mathbb{R}^n \qquad (2-95)$$
$$y^q = (y_1^q, y_2^q, \dots, y_m^q) \in \mathbb{R}^m$$

et nous souhaitons employer ces données pour ajuster les poids et les biais d'un réseau neuronal avec N nœuds à l'entrée, et m neurones en sortie et une couche cachée, par exemple.

$$o^q = \left(o_1^q, o_2^q, \dots, o_m^q\right) \in \mathbb{R}^m \qquad (2-96)$$

Alors, Nous devrions comparer la sortie de l'entrée x_q, à la sortie cible y_q correspondante.

C'est la même chose que d'approximer une fonction avec les paires d'entrée-sortie.

L'idée de correction d'erreurs est simple : Un changement des poids devrait être fait ou pas selon la comparaison de la sortie réelle avec la sortie désirée.

Cette idée est formulée en termes de mesure appropriée de performance globale.

Dans un sens général, le problème d'apprentissage peut être énoncé comme suit : ayant une classe de fonctions (ici une architecture des réseaux neuronaux) et d'un critère de performance, trouver une fonction dans cette classe qui optimise le critère de performance sur la base d'un ensemble d'exemples de formation.

Dans une certaine mesure, les réseaux neuronaux sont une classe de machines à enseigner, et l'algorithme de rétro-propagation est un cas particulier d'un principe inductif général appelé « le principe empirique de minimisation de risque ». En soi, la capacité d'apprentissage des réseaux neuronaux tombe sous la même ligne d'analyse que les machines à enseigner. Nous indiquons ici les idées utilisées dans l'analyse.

Se rappeler qu'une loi de commande est une fonction ϕ d'un espace d'entrée X vers l'espace de sortie Y qui prévoit la sortie pour une entrée donnée.

Si nous n'avons pas un modèle mathématique, ou même si nous l'avons mais nous n'avons pas une méthode analytique pour trouver ϕ, nous devons chercher d'autres solutions.

Considérer la situation où l'information qui nous aidera à trouver ϕ est un ensemble de paires (xi, yi), i= 1,…, m, où yi = ϕ (xi).

Alors le problème évident est de trouver une fonction ϕ_n qui passe par ces points de telle manière que la prévision soit bonne, pour les nouveaux x, la fonction ϕ_n produira les valeurs ϕ_n(x) assez proche de ϕ (x).

Pour effectuer le programme ci-dessus, nous avons besoin du cahier des charges des critères de performance et d'une voie de construction de ϕ_n qui répond à ces critères.

Le critère de performance pour une bonne approximation est spécifié comme étant une erreur d'approximation acceptable. Un réseau neuronal est un outil qui peut être utilisé pour réaliser ces deux conditions simultanément. Cependant, comme avec tous les outils, les réseaux neuronaux ont des limitations.

L'approche du réseau neuronal se compose de deux phases, la création d'un ensemble de modèles et fournir un mécanisme pour trouver une bonne approximation.

D'abord, l'architecture du réseau neuronal fournit un ensemble de fonctions F de X vers Y. Le fait que F est un bon ensemble de modèles pour ϕ est dû à la propriété universelle d'approximation des réseaux neuronaux, ϕ peut être approximé par un certain élément de F.

En suite, on conçoit des algorithmes de rétro-propagation pour réseaux neuronaux multicouche pour implémenter cette approximation.

Au lieu d'avoir les paires désirées, ou des échantillons, supposer que nous avons seulement les conseils des experts comme information pour trouver ϕ.

Alors nous sommes dans le domaine de l'applicabilité de la commande floue. L'ensemble des modèles est créé par les règles linguistiques des experts « si… alors ». Une propriété universelle similaire d'approximation des systèmes flous existe. Cependant, la recherche d'une bonne approximation est faite par le tuning, il n'y a aucune procédure systématique pour atteindre la fonction approximative ϕ_n.

Cet inconvénient peut être compensé en utilisant des réseaux neuronaux dans la commande floue, menant à ce qui s'appelle la commande neuro-floue, dans laquelle la formation employée pour construire ϕ est une base des règles floues représentées dans une structure de réseau neuronal de sorte que la recherche systématique par des algorithmes de rétro-propagation puisse être employée pour trouver une bonne approximation pour ϕ.

Naturellement, ceci exige des expériences sur le système commandé pour extraire les échantillons. L'avantage de cette approche est que nous avons incorporé toute l'information disponible (les conseils des experts et les données numériques) dans notre processus pour construire la loi de commande ϕ. Ce processus substituant le tuning et est désigné sous le nom de l'optimisation des règles floues. En effet, quand des étiquettes linguistiques dans des

règles floues sont modélisées par des fonctions d'appartenances paramétriques, l'algorithme d'apprentissage optimisera les paramètres, menant à l'approximation désirée de ϕ.

2.12 La règle delta :

La règle delta est un algorithme d'apprentissage pour les réseaux de neurones à une seule couche. Nous avons choisis de présenter cet algorithme de manière assez détaillée car c'est un précurseur de l'algorithme de rétro-propagation pour les réseaux de neurones multicouches. L'idée est de définir une mesure de la performance globale d'un réseau, tel que celui représenté sur le schéma 5.5, puis trouver une manière d'optimiser cette exécution. Évidemment, les algorithmes d'apprentissage devraient changer les poids de sorte que la sortie o_q devienne de plus en plus similaire à la sortie désirée y_q pour q = 1, 2,…, m, lorsque l'entrée x_q se présente au réseau. Une mesure appropriée de la performance globale est :

$$E = \sum_{q=1}^{N} E^q \qquad (2-97)$$

Où

$$E^q = \frac{1}{2}\sum_{i=1}^{m} (y_i{}^q - o_i{}^q)^2 \qquad (2-98)$$

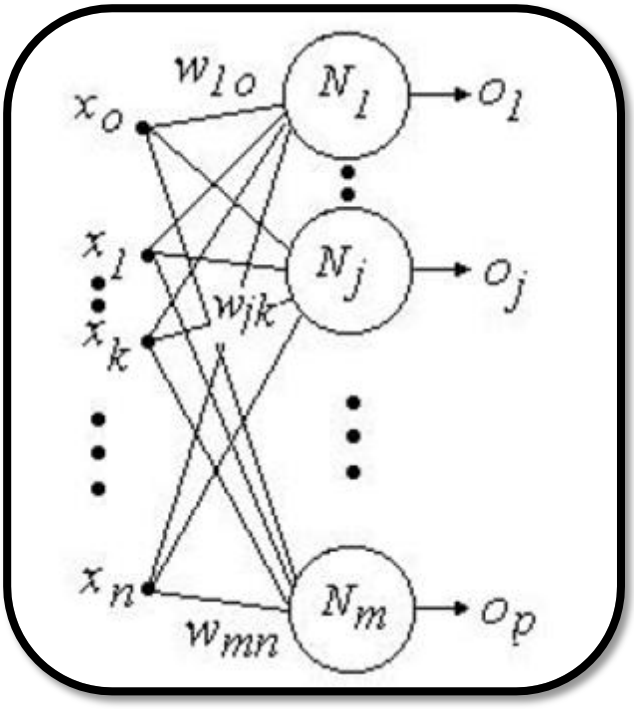


Figure2.35 réseau avec les poids w_{ij}

Le but est de minimiser E en ajustant les poids w_{ij} du réseau.

Soit $\boldsymbol{w_{ij}}$ les poids du nœud j vers le neurone i. Pour utiliser la méthode du gradient descendant pour l'optimisation, on a besoin que les $\boldsymbol{w_{ij}}$ soient différentiables.

Ceci exige que :

$$o_i^q = f_i\left(\sum_{j=0}^{n} w_{ij}\, x_j^q\right) \qquad (2-99)$$

Soit différentiable.

C'est-à-dire, la fonction f_i d'activation du ième neurone devrait être choisie pour être différentiable.

Noter que des fonctions échelons comme

$$f(x) = \begin{cases} 1 & si \quad x \geq 0 \\ 0 & si \quad x < 0 \end{cases} \qquad (2-100)$$

ne sont pas différentiables.

La fonction d'activation sigmoïde, représentée dans la (figue2.34), est différentiable.

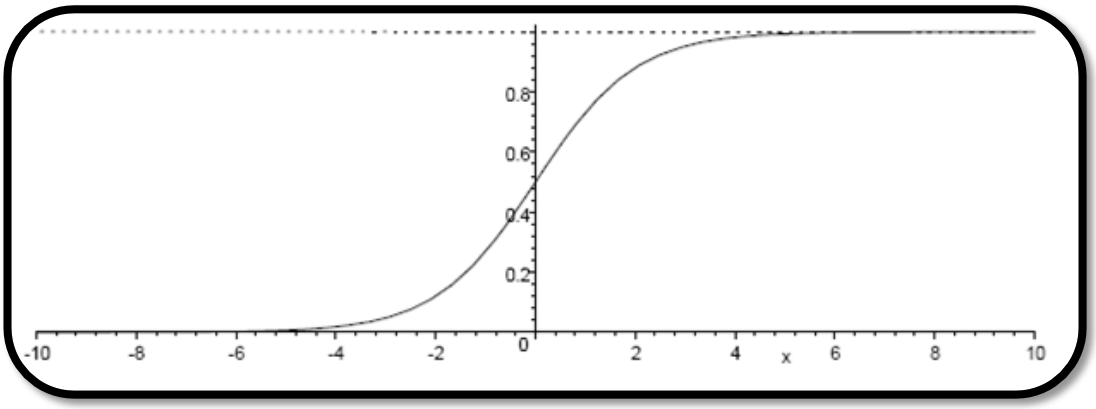


Figure2.36 : La fonction d'activation sigmoïde

Noter qu'une fonction d'activation échelon modélise les neurones dans deux états, tant dis qu'une fonction d'activation continue comme la sigmoïde offre un degré graduel de fonctionnement, une propriété floue qui est plus réaliste.

L'erreur E est une fonction des variables $\boldsymbol{w_{ij}}$, i = 1, 2,…, m, j = 1, 2,…, N.

Se rappeler que le gradient ∇E de E en un point w avec les composants $\boldsymbol{w_{ij}}$ est un vecteur aux dérivées partielles $\frac{\partial E}{\partial w_{ij}}$ .Comme la dérivée d'une fonction d'une variable, le gradient pointe toujours vers la direction ascendante de la fonction E. La direction descendante de E en

W est $-\nabla E$. Ainsi, pour réduire au minimum E, nous nous déplaçons proportionnellement vers le négatif du ∇E, menant à la mise à jour de chaque poids w_{jk} :

$$w_{jk} \longrightarrow w_{jk} + \Delta w_{jk} \qquad (2-101)$$

Où

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{ij}} \qquad (2-102)$$

et $\eta > 0$ est un nombre appelé le taux d'apprentissage.

On a :

$$\frac{\partial E}{\partial w_{ij}} = \sum_{q=1}^{N} \frac{\partial E^q}{\partial w_{ij}} \qquad (2-103)$$

Et

$$\frac{\partial E^q}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\frac{1}{2} \sum_{i=1}^{m} (y_i^q - o_i^q)^2 \right) = (o_j^q - y_j^q) \frac{\partial}{\partial w_{ij}} f_i \left(\sum_{i=0}^{n} w_{ij}\, x_i^q \right) \qquad (2-104)$$

Du moment que

$$\frac{\partial}{\partial w_{ij}} (o_j^q - y_j^q) = 0 \; pour \; i \neq j \qquad (2-105)$$

Noter que

$$o_j^q = f_i\left(\sum_{i=0}^{n} w_{ij}\, x_i^q\right) \qquad (2-106)$$

Donc

$$\frac{\partial E^q}{\partial w_{jk}} = x_k^q \left(o_j^q - y_j^q\right) f_j' \left(\sum_{i=0}^{n} w_{ji}\, x_i^q \right) = \delta_j^q x_k^q \qquad (2-107)$$

Où

$$\delta_j^q = \left(o_j^q - y_j^q\right) f_j'\left(\sum_{i=0}^{n} w_{ji}\, x_i^q\right) \qquad (2-108)$$

Pour le j ième neurone de sortie.

Par conséquent,

$$\Delta w_{jk} = \sum_{q=1}^{N} \Delta^q w_{ji} = -\eta \sum_{q=1}^{N} \frac{\partial E^q}{\partial w_{jk}} = \sum_{q=1}^{N} -\eta\, \delta_j^q x_k^q \qquad (2-109)$$

La règle delta mène à la recherche d'un d vecteur de poids w∗ tel que le gradient de E au $w*$ est égal à zéro. Pour quelques réseaux de neurones simples à une seule couche, le $w*$ est l'unique minimum absolu de $E\ (w)$.

Exemple : nous mettons en application la fonction logique ET en réduisant au minimum l'erreur E. Considérer l'ensemble d'entraînement T composé d'entrées binaires / cibles-bipolaires :

$$T = \{(x^q, y^q), q = 1,2,3,4\} \qquad (2-110)$$

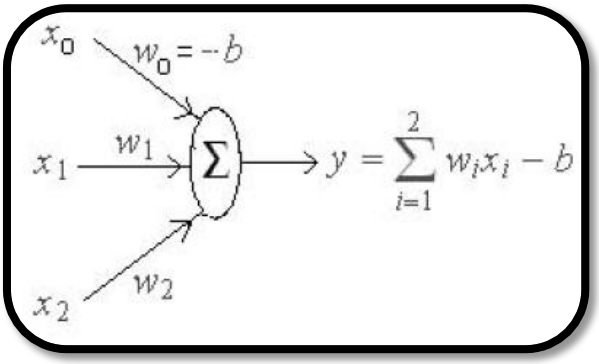
Avec

$$x^q \epsilon \{0,1\}^2 \ et \ y^q \epsilon \{-1,1\} \qquad (2-111)$$

Spécifiquement,

$$\begin{aligned} x^1 &= (x_1^1, x_2^1) = (1,1), \quad & y^1 &= 1 \\ x^2 &= (x_1^2, x_2^2) = (1,0), \quad & y^2 &= -1 \\ x^3 &= (x_1^3, x_2^3) = (0,1), \quad & y^3 &= -1 \\ x^4 &= (x_1^4, x_2^4) = (0,0), \quad & y^4 &= -1 \end{aligned} \qquad (2-112)$$

Une architecture appropriée d'un réseau de neurones pour ce problème est la suivante, avec la fonction d'activation linéaire f (x) = X.



Nous avons : ***Figure 2.37***

$$E = \sum_{q=1}^{4} \left(x_1^q w_1 + x_2^q w_2 - w_0 - y^i\right)^2 \qquad (2-113)$$

Le vecteur des poids (w_0^*, w_1^*, w_2^*) qui minimise E est la solution du système d'équations

$$\frac{\partial E}{\partial w_j} = 0, \quad j = 0,1,2 \qquad (2-114)$$

Explicitement ;

$$(1) \quad \frac{\partial E}{\partial w_0} = w_1 + w_2 + 2w_0 + 1 = 0 \qquad (2-115)$$

$$(2) \quad \frac{\partial E}{\partial w_1} = 2w_1 + w_2 + 2w_0 = 0 \qquad (2-116)$$

$$(3) \quad \frac{\partial E}{\partial w_2} = w_1 + 2w_2 - 2w_0 = 0 \qquad (2-117)$$

Ici, il est facile d'obtenir la solution de ce système d'équations linéaires. En effet, la soustraction de (3) de (1) donne $w_2 = 1$. Avec $w_2 = 1$, (2) et (3) donne

$$2w_0 = w_1 + 1 = w_1 + 2 \qquad (2-118)$$

Ce qui implique w1 = 1. Alors de (3), nous obtenons :

$$w_0 = \frac{w_1 + 2w_2}{2} = \frac{3}{2} \qquad (2-119)$$

Noter que dans la règle delta, aussi bien que dans la règle delta généralisée que nous considérerons plus tard, nous devons calculer les dérivés des fonctions d'activation impliquées.

Le choix des fonctions d'activation lisses est dicté par les gammes des variables de sortie.

Par exemple, si la gamme de sortie d'un neurone, pour une certaine application spécifique, est l'intervalle ouvert [0, 1], alors une fonction d'activation différentiable appropriée pourrait être la fonction sigmoïde représentée sur la figure 2.38. Si la variable de sortie a une gamme (−1, 1), alors une fonction d'activation appropriée devrait avoir la même gamme, par exemple

$$f(x) = \frac{2}{1 + e^{-x}} - 1 \qquad (2-120)$$

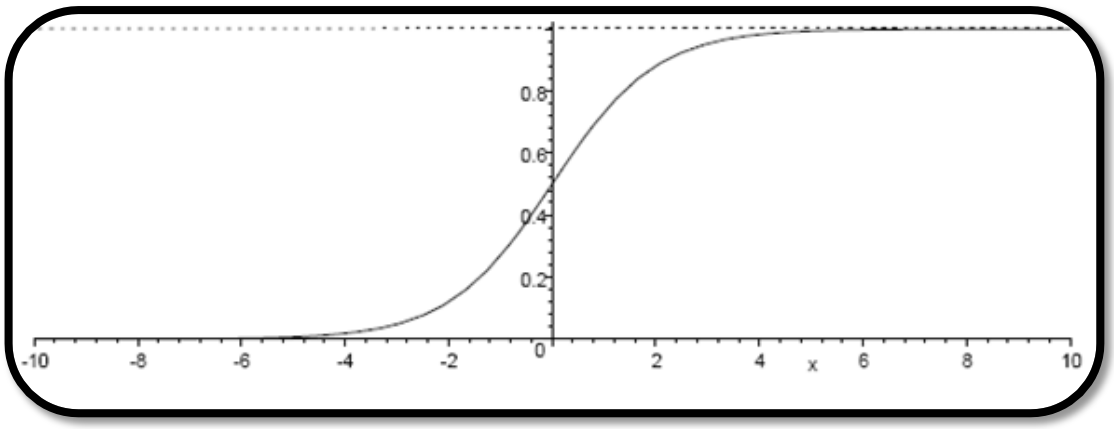


Figure 2.38 $\boldsymbol{f(x) = \frac{2}{1+e^{-x}} - 1}$

2.13 L'algorithme de rétro-propagation :

Comme nous avons vu dans l'exemple d'implémentation de la fonction XOR, nous devons considérer les réseaux de neurones multicouches pour approximer des relations plus générales. Ainsi nous avons besoin d'algorithmes d'apprentissages pour des réseaux de neurones plus compliqués.

L'algorithme d'apprentissage populaire suivant, désigné sous le nom de l'algorithme de rétro-propagation, est une généralisation de la règle delta. Si nous regardons étroitement la règle delta pour les réseaux de neurones à une seule couche, nous nous rendons compte que pour mettre à jour un poids w_{ik} quand l'entrée d'apprentissage x_q est présenté, nous avons besoin de

$$\Delta w_{ik}^{q} = -\eta \delta_i^q \mathrm{x}_{ik}^q \qquad (2-121)$$

Donc, nous devons calculer la fonction :

$$\delta_i^q = \left(o_i^q - y_i^q\right) f'\left(\sum_{j=0}^{n} w_{ij}\, x_i^q\right) \qquad (2-122)$$

, et c'est possible puisque nous avons à notre disposition la valeur $\mathrm{y}^{\mathrm{q}}{}_{\mathrm{i}}$ qui est connue pour nous comme cible de sortie pour le ième nœud de sortie.

Considérons, pour la simplicité, le réseau de neurones n-m-p de deux couches représenté sur la figure 2.39. Il semble que nous ne pouvons pas mettre à jour un poids comme w_{ik} sur le lien reliant le k-ième nœud d'entrée dans la couche d'entrée au ième neurone caché, puisque nous n'avons pas les caractéristiques de la cible du ième neurone depuis l'entrée x^q.

Noter que les y^{q} sont les valeurs désirées des neurones dans la couche de sortie, et non pas dans les couches cachées. Ainsi, lorsqu'on regarde les erreurs $\left(o_j^q - y_j^q\right)^2$ dans la couche de sortie on ne peut pas détecter le neurone caché responsable.

Il s'avère que pour mettre à jour ces poids, il suffit d'être capable de calculer « $\partial \mathrm{E}/\partial \mathrm{O_i}$ », la dérivée partielle de l'erreur globale E avec la sortie respective o_i, $\mathrm{i} = 1, \ldots, p$. Pour des fonctions d'activation différentiables, la stratégie de descente de gradient peut encore être appliquée pour trouver la configuration du poids $\mathrm{w}*$ qui réduit au minimum l'erreur E.

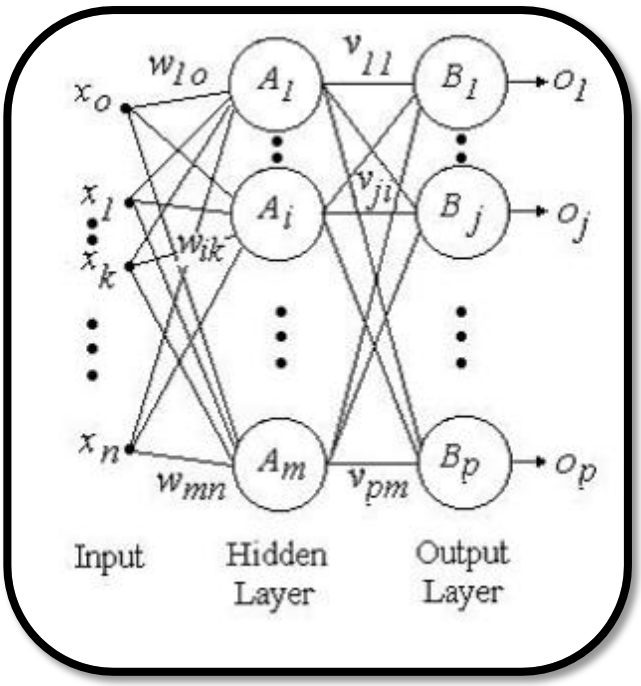


Figure 2.39 : Le réseau de neurones n-m-p de deux couches.

On écrira ci-dessous les formules pour l'actualisation des poids d'un réseau à deux couches. La généralisation pour un réseau contenant plus de deux couches sera juste une question de notation.

Soit v_{ij} le poids de la liaison qui relie le ième neurone caché avec le jème neurone de sortie, et w_{ik} le poids de la liaison reliant le kieme neurone d'entrée avec le ième neurone caché.

Premièrement, la règle d'actualisation de v_{ji} est la même que dans la règle delta, sauf en voyant la couche cachée comme une couche d'entrée. Pour le jème neurone de sortie et le ième neurone caché, le poids v_{ji} est actualisé en utilisant :

$$\Delta v_{ij} = \sum_{j=1}^{N} \Delta v_{ji}^{q} = -\eta \sum \frac{\partial E^{q}}{\partial v_{ji}} = \sum -\eta \delta_{j}^{q} \, z_{i}^{q} \qquad (2-123)$$

Où z_i^q est l'entrée du ième neurone caché,

$$z_{i}^{q} = f_{i}\left(\sum_{k=0}^{n} w_{ik} {x_k}^{q}\right) \qquad (2-124)$$

, et

$$\delta_{j}^{q} = \left(o_{j}^{q} - y_{j}^{q}\right) {f'}_{i}\left(\sum_{k=1}^{m} v_{jk} \, z_{k}^{q}\right) \qquad (2-125)$$

, pour le ième neurone caché et le kieme nœud de sortie, le pois w_{ik} est ajusté comme suit :

$$\Delta^q w_{ik} = -\eta \frac{\partial E^q}{\partial w_{ik}} \qquad (2-126)$$

$$\frac{\partial E^q}{\partial w_{ik}} = \frac{\partial E^q}{\partial o_i^q} \frac{\partial o_i^q}{\partial w_{ik}} \qquad (2-127)$$

Où o_i^q est la sortie du ième neurone :

$$o_i^q = f_i\left(\sum_{l=0}^{n} w_{il} x_l^q\right) = z_i^q \qquad (2-128)$$

On a,

$$\frac{\partial o_i^q}{\partial w_{ik}} = f'_i\left(\sum_{l=0}^{n} w_{il}\, x_l^q\right) x_k^q \qquad (2-129)$$

Soit, $\delta_i^q = \frac{\partial E^q}{\partial o_i^q}$, comment doit-on calculer ceci ? , observer que

$$\delta_i^q = \frac{\partial E^q}{\partial o_i^q} = \sum_{j=1}^{p} \frac{\partial E^q}{\partial o_j^q} \frac{\partial o_j^q}{\partial o_i^q} \qquad (2-130)$$

Où j est dans la couche de sortie. Les $\frac{\partial E^q}{\partial o_j^q}$ sont connus grâce aux calculs précédents en utilisant la règle delta. Ensuite

$$o_i^q = f_j'\left(\sum_{l=1}^{m} v_{il}\, z_l^q\right) v_{ji} \qquad (2-131)$$

Alors, le δ_i^q pour le neurone caché i, est calculé avec les valeurs déjà connues de δ_j^q pour tous les j dans la couche de sortie. Pour cette raison, la règle delta généralisée est appelée algorithme de rétro-propagation. Premièrement, on envoie les entrées x^q en avant pour atteindre la couche de sortie et ensuite on calcule les δ_j^q pour tous les neurones j. Ensuite on propage ces δ_j^q en arrière vers la couche précédente (ici, c'est la couche cachée) dans le but de calculer les δ_i^q pour tous les neurones i de cette couche.

Il existe deux stratégies pour l'entrainement :

1. l'approche batch : les poids sont changés selon

$$\Delta w_{ik} = -\eta \sum_{q=1}^{N} \frac{\partial E^q}{\partial w_{ik}} \qquad (2-132)$$

Apres avoir fait N entrainements.

2. L'approche incrémentale : on change les poids w_{ik} après avoir présenté chaque entrainement q au réseau, en utilisant : $\Delta^q w_{ik} = -\eta \frac{\partial E^q}{\partial w_{ik}}$ pour actualiser les w_{ik} .

Noter que le batch actualise Δw_{ik} est juste la somme de l'actualisation incrémentale $\Delta^q w_{ik}, q = 1,2 \dots, N$.

Dans l'algorithme de rétro-propagation, on commence par le calcul de la valeur de δ dans la couche de sortie. Ensuite, on propage le vecteur d'erreur en arrière, de la couche de sortie vers les entrées. Lorsque tous les poids sont actualisés avec $\Delta^q w_{ik}$, le prochain x^q d'entrainement est présenté, et ainsi de suite. Le critère d'arrêt sera la fonction d'erreur E.

En résumé, l'algorithme de rétro-propagation peut être décrit par les étapes suivantes :

1. **initialiser tous les poids w_{ik} avec de petites valeurs aléatoires, et sélectionner le taux d'apprentissage η.**
2. **Appliquer les x^q à la couche d'entrée.**
3. **Propager les x^q vers l'avant, de la couche d'entrée vers la couche de sortie en utilisant $o_i = f_i\left(\sum_{k=0}^{p} w_{ik} o_k\right)$**
4. **Calculer l'erreur $E^q(w)$ dans la couche de sortie par :**
   $E^q(w) = \frac{1}{2}\sum_{i=1}^{p}\left(o_i^q - y_i^q\right)^2$
5. **Calculer les valeurs de δ pour la couche de sortie en utilisant :**
   $\delta_i^q = f_i'\left(\sum_{k=1}^{n} v_{ik} z_k\right)\left(o_i^q - y_i^q\right)$
6. **Calculer les valeurs de δ pour la couche cachée en propageant les valeurs de δ en arrière,**

$$\delta_i^q = f_i'\left(\sum_{k=1}^{n} w_{ik} x_k^q\right)\sum_{j=1}^{p} v_{ij}\delta_j^q$$

7. **utiliser $\Delta^q w_{ik} = -\eta \delta_i^q o_k^q$ pour tous les w_{ik} du réseau de neurones.**
8. **$q \longrightarrow q + 1$ et aller à l'étape 2.**

Les deux approches batch et incrémentale peuvent être appliquées pour ajuster le poids w.

L'efficacité est relative au problème, mais l'approche incrémentale parait supérieure dans beaucoup de cas, surtout pour les ensembles d'entrainement redondants ou très régulier. L'algorithme mène à w*, tel que $\Delta(E)(w*) = 0$; qui est un minimum local.

2.14 Concepts flous dans les réseaux de neurones :[7]

Des concepts de logique floue peuvent être incorporés à la structure des réseaux de neurones à n'importe quel niveau. Se rappeler qu'une règle floue de Mamdani est de la forme : ***« Si x est A alors y est B »*** et une règle floue de Sugeno est de la forme : ***« Si x est A alors y est f (x) ».***Où A et B sont des ensembles flous ou les produits cartésiens d'ensembles flous, et *f* est une fonction à valeurs réelles.

Si A est le produit des sous-ensembles flous $A1, \dots, An$ de $Ui, i = 1, 2, \dots, n,$ alors :

$$A: \prod_{i=1}^{n} U_i \to [0,1]: (u_1 \dots u_n) \to min\{A_1(u_1), \dots A_n(u_n)\} \qquad (2-133)$$

Et pour $\mathbf{x} = (x1, \dots, xn)$, $\mathbf{x}$ est A veut dire ***« x1 est A1 et x2 est A2, … , et xn est An »***, et f est une fonction à valeurs réelles dans $\mathbb{R}^n$. L'inférence floue ***ALORS*** est mise en application en général par le **« minimum »**, c'est l'implication de Mamdani, ou bien par le produit. Les règles sont combinées par le OU-flou, à savoir par une certaine t-conorm, habituellement le **« maximum».**

Une modification de la structure du réseau de neurones est de remplacer une partie ou tous les composants d'un neurone par des opérations de logique floue. Par exemple, si l'addition (comme opération d'assemblage) dans un neurone est remplacée par le minimum, nous avons un neurone **flou-minimum**. Un neurone qui emploie le maximum comme une opération d'assemblage est un neurone **flou-maximum**. Un réseau de neurones avec des neurones flous devient un réseau flou- neural multicouche.

Noter que des réseaux de neurones conventionnels sont employés pour approximer des fonctions grâce aux couples d'entrée-sortie numériques $(xi, yi), i = 1, 2, \dots, n.$

Les réseaux flou-neuro sont des structures de calcul plus générales avec lesquelles l'approximation des fonctions peut être étendue aux données linguistiques.

Considérer un ensemble de règles floues de la forme :

$$R_i: si\ \mathbf{x}\ est\ A^i\ alors\ \mathbf{y}\ est\ B^i \qquad (2-134)$$

$i = 1, \ldots, n,$ Où Ai et Bi sont des produits cartésiens d'ensembles flous :

$$A^i(\mathbf{x}) = \left(\prod_{j=1}^{N} A^i_j\right)(\mathbf{x}) = \bigwedge_{j=1}^{N} A^i_j(x_j) \qquad (2-135)$$

$$B^i(\mathbf{y}) = \left(\prod_{k=1}^{M} B^i{}_k\right)(\mathbf{y}) = \bigwedge_{k=1}^{M} B^i{}_k(y_k) \qquad (2-136)$$

Chaque règle dans (2-134) peut être interprétée comme un modèle d'entrainement pour un réseau de neurones multicouche, où la partie antécédente de la règle est l'entrée du réseau de neurones, et la partie conséquente de la règle est la sortie désirée du réseau neuronal. L'équation (2-134) donc constituerait :

Un système à une entrée et une sortie (SISO), si $N = M = 1$.

Un système à plusieurs entrées et une sortie (MISO), si $N > M = 1$.

Un système à plusieurs entrées et plusieurs sorties (MIMO), si $N > M > 1$.

L'ensemble d'entrainement se compose des paires d'entrée-sortie (A_i, B_i). Pour un système flou MIMO, l'ensemble d'entrainement peut être écrit sous la forme :

$$\left(\{A^i_j\}_{j=1}^{N}, \{B^i_k\}_{k=1}^{M}\right), i = 1, \ldots, n \qquad (2-137)$$

Pour un système MISO, ceci se simplifie à :

$$(\{A^i_j\}_{j=1}^{N}, B_i), i = 1, \ldots, n \qquad (2-138)$$

Et pour un système de SISO, l'ensemble d'entrainement peut être écrit sous la forme :

$$(A_1, B_1), \ldots, (A_n, B_n) \qquad (2-139)$$

Nous devons également incorporer des opérations de défuzzification aux réseaux de neurones. Ces opérations, lorsque nécessaire, ont lieu au niveau final du réseau de neurones et sont les mêmes que ceux discuté dans la partie de la logique floue.

2.15 Principes de base des systèmes flous-neuro :

Les règles exprimant le comportement d'un système nous permettent de spécifier la sortie, étant donné une entrée. Cette propriété est particulièrement utile du fait que nous pouvons définir la sortie désirée pour obtenir la meilleure performance du système commandé. Pour un contrôleur PID, ceci donne la flexibilité de spécifier les divers gains, à savoir, le gain proportionnels, le gain dérivé et le gain d'intégrale pour réaliser la performance désirée.

Dans cette section nous verrons comment les règles floues ***« si… alors… »*** expriment le comportement d'entrée-sortie des systèmes flous-neuraux. La connaissance des règles nous

permet de concevoir des architectures entrainables appropriées des réseaux de neurones. Nous montrons comment la structure des règles floues peut être transformée en réseaux de neurones, donnant naissance aux systèmes flous-neuraux.

Il y a deux approches principales pour implémenter les règles floues par un réseau de rétro-propagation standard, la première méthode proposée d'abord par Umano et Ezawa en 1991, et une modification par Uehara et Fujise en 1992 qui emploie un nombre fini d'ensembles d'α-niveau pour représenter des nombres flous.

En 1993, Jang a étendu ces méthodes qui ont mené au développement d'un système d'inférence flou adaptatif aux réseaux (ANFIS), qui a engendré de nombreuses applications.

Nous discuterons ANFIS plus tard dans ce chapitre.

Dans la méthode proposée par Umano et Ezawa, un ensemble flou est représenté par un nombre fini de ses valeurs d'appartenance. Supposer que nous avons un ensemble de règles floues de la forme :

R_i: $si\ x\ est\ A_i\ alors\ y\ est\ B_i,\ \ i = 1, \dots, n,$ où A_i et B_i sont des ensembles flous. Soit l'intervalle $[a1, a2]$ le support de tout les A_i, et aussi de toute autre fonction que nous pourrons avoir comme entrée au système.

Et aussi, soit $[b1, b2]$ est le support de tous les B_i, et aussi de toute autre fonction que nous pourrons obtenir comme sorties du système.

Nous pouvons travailler avec un nombre fini de valeurs d'appartenance en prenant des entiers positifs $M \geq 2, N \geq 2$ et mettons:

$$x_j = a_1 + \frac{j-1}{N-1}(a_2 - a_1) \qquad (2-140)$$

$$y_k = b_1 + \frac{k-1}{M-1}(b_2 - b_1) \qquad (2-141)$$

$Pour\ 1 \leq j \leq N, 1 \leq k \leq M.$ Ceci nous donne une version discrète de l'ensemble d'entrainement continue comprenant les paires d'entrée-sortie :

$$\big(A_i(\mathbf{x}), B_i(\mathbf{y})\big) = \Big(\big(A_i(x_1), \dots, A_i(x_N)\big), \big(B_i(y_1), \dots, B_i(y_M)\big)\Big) \qquad (2-142)$$

, pour $i = 1, \dots, n,$ et le réseau flou-neuro devient un réseau standard à N entrées et M sorties qui peut être entrainé par la règle delta généralisée.

Exemple :

Supposons que notre base de règles floues se compose de trois règles :

R1 : Si x est petit alors y est négatif.

R2 : Si x est moyen alors y est environ zéro.

R3 : Si x est grand alors y est positif.

Où les fonctions d'appartenance des termes flous sont définies par :

$$\mu_{petit}(u) = A_1(u) = \begin{cases} 1 - 2u \; si \; 0 \le u \le \frac{1}{2}. \\ 0 \qquad\qquad ailleur. \end{cases} \qquad (2-143)$$

$$\mu_{moyen}(u) = A_2(u) = \begin{cases} 1 - 2\left|u - \frac{1}{2}\right| si \; 0 \le u \le \frac{1}{2}. \\ 0 \qquad\qquad\qquad ailleur. \end{cases} \qquad (2-144)$$

$$\mu_{grand} = A_3(u) = \begin{cases} 2u - 1 \; si \quad \frac{1}{2} \le u \le 1. \\ 0 \qquad\qquad ailleur \end{cases} \qquad (2-145)$$

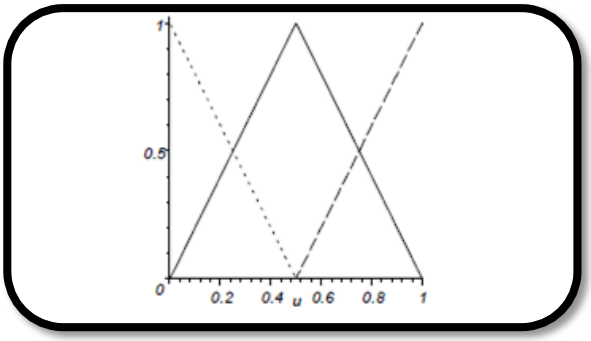


Figure 2.40: Fonctions d'appartenances : …petit - - - grand —— moyen

$$\mu_{négatif}(u) = B_1(u) = \begin{cases} -u \; si \; -1 \le u \le 0. \\ 0 \qquad\qquad ailleur. \end{cases} \qquad (2-146)$$

$$\mu_{environ\ zero}(u) = B_2(u) = \begin{cases} 1 - 2|u| \; si \; -\frac{1}{2} \le u \le \frac{1}{2}. \\ 0 \qquad\qquad\qquad ailleur. \end{cases} \qquad (2-147)$$

$$\mu_{positif} = B_3(u) = \begin{cases} u \quad si \quad 0 \le u \le 1. \\ 0 \qquad\qquad ailleur \end{cases} \qquad (2-148)$$

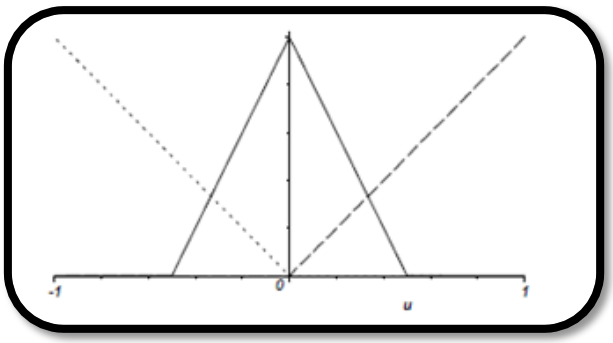


Figure2.41 : Fonctions d'appartenances : … négatif - - - positif —— environ zéro

L'ensemble flou d'entrainement dérivé de cette base de règle peut être écrit sous la forme {(Petit, négatif), (moyen, environ zéro), (grand, positif)}.

Soit $[0, 1]$ le support de tous les ensembles flous que nous pourrions avoir comme entrée au système. Et aussi, soit $[-1, 1]$ le support de tous les ensembles flous que nous pouvons obtenir comme sorties du système. Soit $M = N = 5$ et :

$$x_j = \frac{j-1}{4} \qquad (2-149)$$

$$y_k = -1 + \frac{(k-1)2}{4} = -1 + \frac{(k-1)}{2} = -\frac{3}{2} + \frac{k}{2} \qquad (2-150)$$

Pour $1 \leq j \leq 5\ et\ 1 \leq k \leq 5$, substituant pour i and j, on aura

$$\mathbf{x} = (x_1, x_2, x_3, x_4, x_5) = (0,0.25,0.5,0.75,1) \qquad (2-151)$$

$$\mathbf{y} = (y_1, y_2, y_3, y_4, y_5) = (-1,-0.5,0,0.5,1) \qquad (2-152)$$

Une version discrète de l'ensemble de formation continue se compose de trois paires d'entrée-sortie pour l'usage dans un réseau standard de rétro-propagation :

$$(A_1(\mathbf{x}), B_1(\mathbf{y})) = ((1\ \ 0.5\ \ 0\ \ 0\ \ 0\ ), (1\ \ 0.5\ \ 0\ \ 0\ \ 0))$$

$$(A_2(\mathbf{x}), B_2(\mathbf{y})) = ((0\ \ 0.5\ \ 1\ \ 0.5\ \ 1\ ), (0\ \ 0\ \ 1\ \ 0\ \ 0)) \qquad (2-153)$$

$$(A_3(\mathbf{x}), B_3(\mathbf{y})) = ((0\ \ 0\ \ 0\ \ 0.5\ \ 1), (0\ \ 0\ \ 0\ \ 0.5\ \ 1))$$

Pour faire la distinction entre un réseau de neurones régulier et un réseau flou-neural, nous considérons brièvement le cas d'un réseau de neurones qui a été discuté en détail précédemment. Considérer un réseau neuronal simple comme celui de la figure 2.42. Tous les signaux et poids sont des nombres réels. Les deux neurones d'entrées ne changent pas le signal d'entrée, ainsi, leur sortie est identique que leur entrée.

Le signal x_i avec le poids w_i donne le produit :

$$p_i = w_i x_i\ , i = 1,2. \qquad (2-154)$$

L'information « p_i » d'entrée est agrégée par l'addition ; produit l'entrée :

$$net = p_1 + p_2 = w_1 x_1 + w_2 x_2. \qquad (2-155)$$

Le neurone emploie sa fonction de transfert f, souvent une fonction sigmoïdale, pour calculer la sortie :

$$y = f(net) = f(w_1 x_1 + w_2 x_2) \qquad (2-156)$$

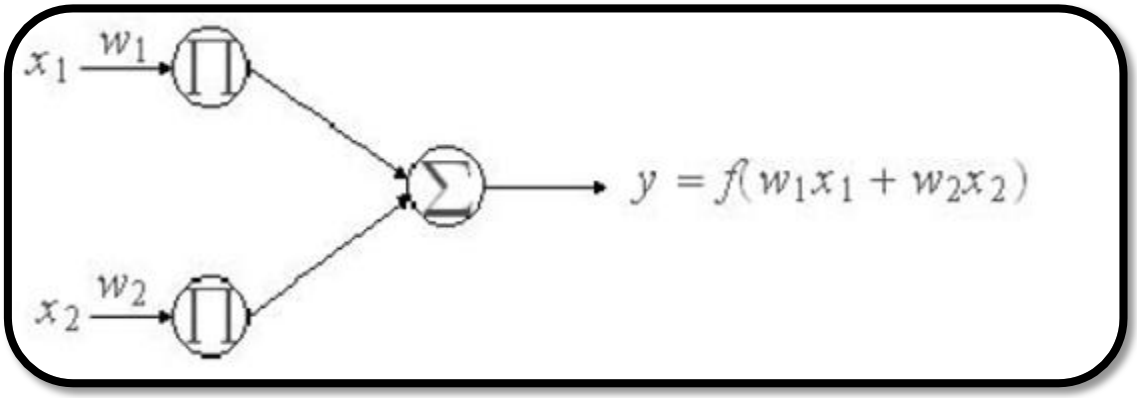


Figure2.42 : neurone standard

Dans ce cas, nous employons des opérateurs d'addition et de multiplication pour exécuter l'agrégation des signaux d'entrée ; figure 2.42. Cependant, ce n'est pas possible dans le cas des nombres flous. Si nous employons des opérateurs comme une t-norme ou une t-conorm pour combiner les données entrantes à un neurone, nous obtenons ce que nous appelons un réseau neuronal hybride. Ces modifications mènent à une architecture floue-neurale basée sur des opérations arithmétiques floues. Un réseau neuronal hybride peut ne pas employer la multiplication, l'addition, ou une fonction sigmoïdale parce que les résultats de ces opérations ne sont pas nécessairement dans l'intervalle unitaire.

Définition :

Un réseau neuronal hybride est un réseau neuronal avec des signaux, des poids et une fonction de transfert pour lesquels :

1- Les signaux et les poids x_i et w_i qui se situent dans l'intervalle [0, 1], peuvent être combinés utilisant une t-norme, une t-conorm, ou une autre opération continue.
2- Les résultats p_1, p_2 peuvent être agrégés avec une t-norme, t-conorm, ou autre fonction continue de $[0, 1]$ à $[0, 1]$.
3- La fonction de transfert f peut être n'importe quelle fonction continue de [0, 1] à [0, 1].

Un élément de traitement d'un réseau neuronal hybride s'appelle un neurone flou.

Nous soulignons que toutes les entrées, sorties, et poids d'un réseau neuronal hybride sont des nombres réels pris de l'intervalle unitaire [0, 1].

Exemple :

La figure2.43 montre un ET flou-neurone. Les signaux x_i et w_i de poids sont combinés par un conorm triangulaire S pour produire la sortie :

$$p_i = S(w_i, x_i), i = 1,2. \qquad (2-157)$$

L'information p_i d'entrée est agrégée par une norme triangulaire T pour produire la sortie du neurone.

$$y = ET(p_1, p_2) = T(p_1, p_2) = T\big(S(w_1, x_1), S(w_2, x_2)\big) \qquad (2-158)$$

Si $T = min$ et le $S = maximum$, alors le Et-neurone réalise la composition en max-min

$$y = min\{w_1 \vee x_1, w_2 \vee x_2\} \qquad (2-159)$$

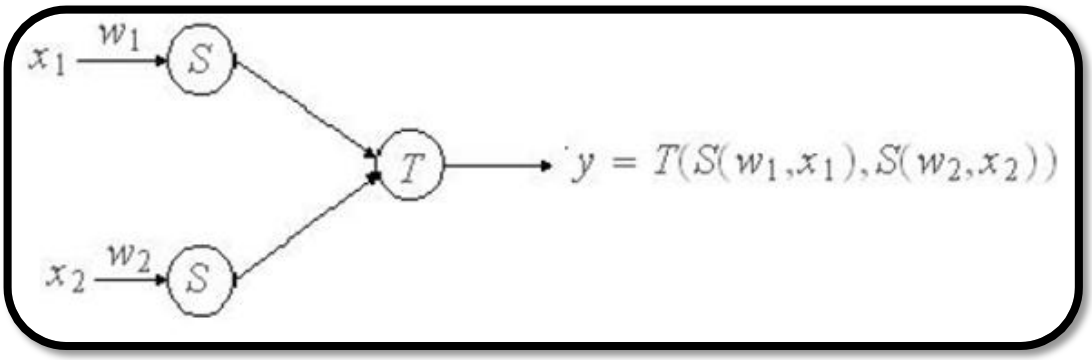


Figure2.43 : ET neuro_ flou

Exemple :

La figure 2.44, montre le ou-neurone flou. Le signal x_i et les poids « w_i » sont combinés par une norme triangulaire T pour produire :

$$p_i = T(w_i, x_i), i = 1,2 \qquad (2-160)$$

L'information d'entrée pi est agrégée par un conorm triangulaire S pour produire la sortie du neurone :

$$y = OU(p_1, p_2) = S(p_1, p_2) = S\big(T(w_1, x_1), T(w_2, x_2)\big) \qquad (2-161)$$

Si $T = min$ et le $S = maximum$, alors le OU-neurone réalise la composition en max-min

$$y = max\{w_1 \wedge x_1, w_2 \wedge x_2\} \qquad (2-162)$$

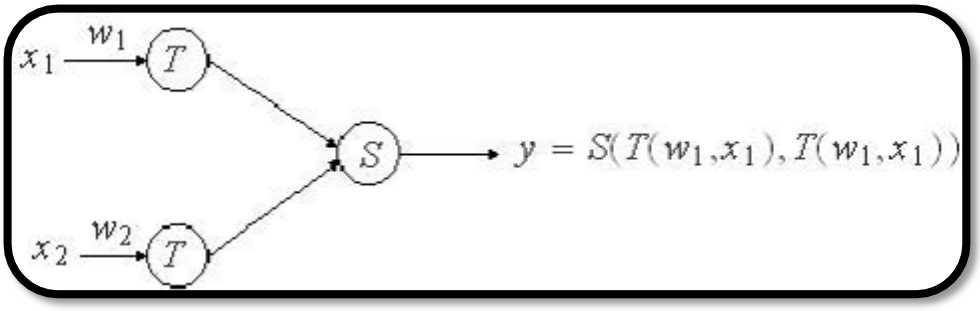


Figure2.44 : OU neuro-flou

2.16 Principes de base des systèmes neuro-flous :

Afin de traiter des règles floues par les réseaux de neurones, il est nécessaire de modifier la structure standard du réseau de neurones convenablement.

Puisque les systèmes flous sont des approximations universelles, on s'attend à ce que leurs représentations équivalentes par réseau de neurones possèdent la même propriété.

Comme indiqué plus tôt, la raison de représenter un système flou en termes de réseau de neurones est d'utiliser les possibilités d'apprentissage des réseaux de neurones pour améliorer la performance, comme l'adaptation des systèmes flous. Ainsi, l'algorithme d'apprentissage dans les réseaux de neurones modifiés devrait être examiné.

2.16.1 Réseau adaptatif muni d'un système inférence flou :

Pour illustrer l'utilisation des réseaux de neurones pour l'inférence floue, nous présentons certains réseaux de neurones adaptatifs réussis des systèmes d'inférence floue, avec des algorithmes d'apprentissage connus sous le nom d'ANFIS.

Ces structures, également connues sous le nom de systèmes d'inférence neuro-flous adaptatifs ou de réseaux adaptatifs aux systèmes d'inférence flous, ont été proposé par Jang.

Il convient de noter que des structures similaires ont été également proposées indépendamment par Lin et Lee et Wang et Mendel. Ces structures sont utiles pour la commande et pour beaucoup d'autres applications.

Pour fixer les idées, considérons le problème de représenter la commande réalisée dans le modèle de Sugeno-Takagi. Pour un simple exemple, considérons une base de règles floues se composant seulement de deux règles :

R1 : Si x_1 est A_1 et x_2 est B_1 alors $y = f_1(x)$

R2 : Si x_1 est A_2 et x_2 est B_2 alors y = $f_2(x)$

Ou A_i et B_i sont des ensembles flous et :

$$f_1(\mathbf{x}) = z_{11}x_1 + z_{12}x_2 + z_{13} \qquad (2-163)$$

$$f_2(\mathbf{x}) = z_{21}x_1 + z_{22}x_2 + z_{23} \qquad (2-164)$$

Se rappeler que quand l'entrée numérique $X = (x1, x2)$ est présentée, le mécanisme d'inférence produira la sortie numérique :

$$y^* = \frac{A_1(x_1)B_1(x_2)f_1(\mathbf{x}) + A_2(x_1)B_2(x_2)f_2(\mathbf{x})}{A_1(x_1)B_1(x_2) + A_2(x_1)B_2(x_2)} \qquad (2-165)$$

Pour implémenter ce qui précède, un réseau neuro- flou est montré sur la figure 2.45.

L'entrée observée $X = (x1, x2)$ est présentée à la couche 1via la couche d'entrée 0. La sortie de la couche 1 est :

$$(O_{11}, O_{12}, O_{13}, O_{14}) = (A_1(x_1), A_2(x_1), B_1(x_2), B_2(x_2)) \qquad (2-166)$$

Où la fonction d'appartenance $Ai, Bi, i = 1, 2$, sont spécifiés d'une manière paramétrique d'une famille de fonctions d'appartenance, telles que triangulaire ou gaussien. La couche 2 se compose de neurones flous avec un opérateur d'agrégation étant une certaine t-norme. Nous employons la t-norme « produit » dans cet exemple, en raison que le produit est employé dans le procédé de l'inférence de Sugeno-Takagi.

La sortie de la couche 2 est :

$$(O_{21}, O_{22}) = (A_1(x_1)B_1(x_2), A_2(x_1)B_2(x_2) \qquad (2-167)$$

La couche 3 est un normalisateur. La sortie de la couche 3 est :

$$(O_{31}, O_{32}) = \left(\frac{O_{21}}{O_{21} + O_{22}}, \frac{O_{22}}{O_{21} + O_{22}}\right) \qquad (2-168)$$

$$= \left(\frac{A_1(x_1)B_1(x_2)}{A_1(x_1)B_1(x_2) + A_2(x_1)B_2(x_2)}, \frac{A_2(x_1)B_2(x_2)}{A_1(x_1)B_1(x_2) + A_2(x_1)B_2(x_2)}\right) \qquad (2-169)$$

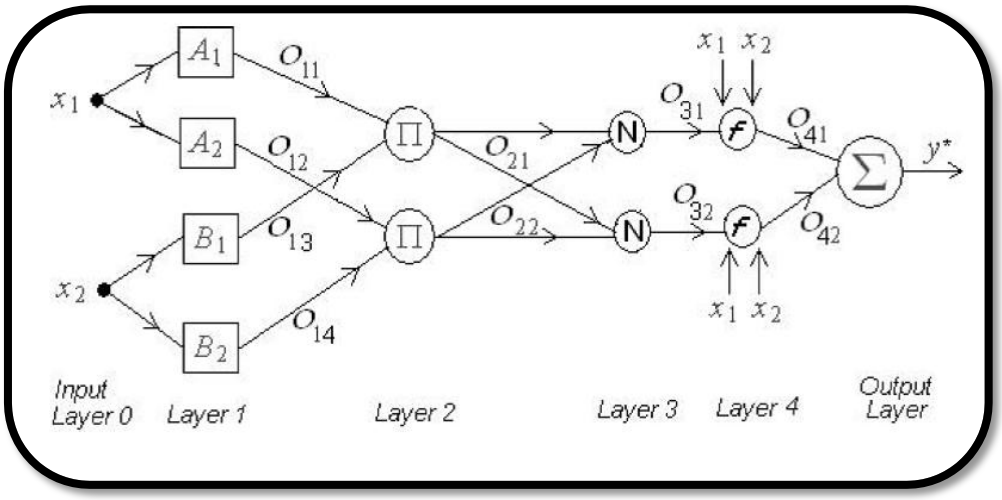


Figure2.45 : Modèle de Sugeno de premier ordre avec deux règles

Les neurones flous de la couche 4 donnent les valeurs de sortie :

$$(O_{41}, O_{42}) = (O_{31} f_1, O_{32} f_2) \quad (2-170)$$

$$= \left(\frac{A_1(x_1)B_1(x_2)(z_{11}x_1 + z_{12}x_2 + z_{13})}{A_1(x_1)B_1(x_2) + A_2(x_1)B_2(x_2)}, \frac{A_2(x_1)B_2(x_2)(z_{21}x_1 + z_{22}x_2 + z_{23})}{A_1(x_1)B_1(x_2) + A_2(x_1)B_2(x_2)} \right) \quad (2-171)$$

Finalement, la couche de sortie calcule l'action de commande en additionnant :

$$y^* = O_{41} + O_{42} \quad (2-172)$$

$$= \frac{A_1(x_1)B_1(x_2)(z_{11}x_1 + z_{12}x_2 + z_{13})}{A_1(x_1)B_1(x_2) + A_2(x_1)B_2(x_2)} + \frac{A_2(x_1)B_2(x_2)(z_{21}x_1 + z_{22}x_2 + z_{23})}{A_1(x_1)B_1(x_2) + A_2(x_1)B_2(x_2)} \quad (2-173)$$

Naturellement, le type de réseau ci-dessus pour une base deux règles peut être étendu d'une manière évidente à un nombre arbitraire de règles.

2.16.2 Algorithme d'apprentissage d'ANFIS :

Dans la section précédente, La représentation du réseau de neurones est simplement un affichage graphique des étapes de calcul du procédé de Sugeno-Takagi.

Pour que cette représentation soit plus utile lors de l'implémentation de la loi de commande, on doit l'équiper d'un algorithme d'apprentissage efficace.

Dans les réseaux de neurones conventionnels, l'algorithme de rétro-propagation est employé pour l'apprentissage, ou pour ajuster des poids sur les flèches reliant entre les neurones grâce aux échantillons d'entrainement « entrée-sortie ».

Dans la structure d'ANFIS, les paramètres des prémisses et des conséquences jouent le rôle des poids. Spécifiquement, quand les fonctions d'appartenance A^{j}_{i} utilisées dans une partie « si » des règles sont spécifiées paramétriquement, la forme est spécifiée et la fonction est déterminée par un nombre fini de paramètres, ces paramètres s'appellent les paramètres des prémisses, tandis que les paramètres a_i, le b_i, c_i, I = 1, 2 dans la partie « alors » des règles sont désignés sous le nom des paramètres de conséquence. Dans la structure d'ANFIS, les paramètres des prémisses et des conséquences jouent le rôle des poids.

L'algorithme d'apprentissage d'ANFIS consiste à ajuster l'ensemble de paramètres ci-dessus grâce aux échantillons

$$((x_1^k, x_2^k), y^k), k = 1, \dots, N \quad (2-174)$$

Il est important de mettre à l'esprit que lorsque nous développons un prototype d'ensemble de règles floues, nous somme en fait probablement entrain de représenter des relations non linéaires d'entrée-sortie.

L'efficacité des modèles flous pour représenter des relations non linéaires d'entrée-sortie dépend des fonctions d'appartenances impliquées. Ainsi, le tuning des fonctions d'appartenance est une question importante dans la modélisation floue. Cette tâche de tuning peut être vue comme un problème d'optimisation ; les réseaux de neurones offrent une possibilité pour résoudre ce problème.

Afin de former un réseau neuro-flou, nous avons besoin d'un ensemble de données de formation sous forme d'entrée-sortie, et des spécifications des règles, y compris une définition préliminaire des fonctions d'appartenance correspondantes.

Une approche standard est d'assumer une certaine forme pour les fonctions d'appartenance de sorte qu'elles dépendent des paramètres qui peuvent être ajustés par un réseau de neurones.

Nous décrivons une méthode pour l'apprentissage des fonctions d'appartenance pour les parties antécédentes et conséquentes des règles floues « si…alors ». Supposer une fonction inconnue, ou une loi de commande, qui doit être réalisé par le système d'inférence flou est connu seulement par l'ensemble d'entrainement :

$$\{(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^K, y^K)\} \; ou \; \mathbf{x}^K = (x_1^k, \dots, x_n^k) \in \mathbb{R}^n et \; y^K \in \mathbb{R} \qquad (2-175)$$

Pour modéliser la fonction inconnue, nous employer les règles floues « si… alors. » Ri, i= 1,…, m, du type suivant :

$$R_i: si \; x_1^k est \; A_i^1 \; et \dots et \; x_n^k est \; A_i^n \; alors \; y = \sum_{j=1}^{n} z_i^j x_j^k + z_i \qquad (2-176)$$

Où les A^j_i sont les fonctions d'appartenance floues et les z^j_i sont des nombres réels.

En admettant que les O_k soient les sorties du système flou correspondant à l'entrée x^k.

Supposons que le et-flou de chaque règle est implémenté par un produit, de sorte que l'antécédent de la ième règle soit donné par :

$$\alpha_i^k = \prod\nolimits_{j=1}^{n} A_i^j\left(x_j^k\right) \qquad (2-177)$$

Nous pourrions également employer d'autres t-normes pour modéliser le ET logique. Nous calculons la sortie du système comme :

$$O^k = \frac{\sum_{i=1}^{m} \alpha_i^k (\sum_{j=1}^{n} z_i^j x_j^k + z_i^0)}{\sum_{i=1}^{m} \alpha_i^k} = \frac{\sum_{i=1}^{m} (\prod_{j=1}^{n} A_i^j(x_j^k))(\sum_{j=1}^{n} z_i^j x_j^k + z_i^0)}{\sum_{i=1}^{m} \prod_{j=1}^{n} A_i^j(x_j^k)} \qquad (2-178)$$

et on défini la mesure d'erreur pour le kieme entrainement comme :

$$E^k = \frac{1}{2}(O^k - y^k)^2 \qquad (2-179)$$

, où O^k est la sortie du système flou correspondante à l'entrée x^k , et y^k est la sortie désirée, k = 1,…,K. Des méthodes standards d'apprentissage sont employés pour faire un entrainement aux z^j_i , j = 0, 1,…, n dans la partie conséquente de la règle floue R_i .

Exemple :

Nous illustrons le processus de tuning ci-dessus par un exemple simplifié.

Considérer deux règles floues, avec la variable d'entrée x et une variable de sortie y, de la forme :

R1 : Si x est A_1alors y = z_1

R2 : Si x est A_2 alors y = z_2

, où les ensembles flous A_1 et A_2 ont des fonctions d'appartenance sigmoïdes défini par :

$$A_1(x) = \frac{1}{1 + e^{b_1(x-a_1)}} \qquad (2-180)$$

$$A_2(x) = \frac{1}{1 + e^{b_2(x-a_2)}} \qquad (2-181)$$

, et a_1, a_2, b_1, et b_2 sont les paramètres à régler pour les prémisses, et l'antécédent de la règle Ri est simplement la valeur de la fonction d'appartenance A_i (x).

La sortie O (x) du système, calculé par la méthode de défuzzification du centre de gravité discrète, est :

$$O(x) = \frac{A_1(x)z_1 + A_2(x)z_2}{A_1(x) + A_2(x)} \qquad (2-182)$$

Etant Donné un ensemble d'entrainement $\{(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^K, y^K)\}$, nous voulons fournir aux deux règles floues des fonctions d'appartenance appropriées et des parties conséquentes pour produire des paires d'entrée-sortie.

C'est-à-dire, nous voulons faire l'apprentissage les paramètres a_1, a_2, b_1, et b_2 des fonctions d'appartenance sigmoïdes, et des valeurs z1 et z2 des parties conséquentes.

La mesure de l'erreur pour le kieme pair d'entrainement est définie par :

$$E^k = E^k(a_1, b_1, a_2, b_2, z_1, z_2) = \frac{1}{2}\left(O^k(a_1, b_1, a_2, b_2, z_1, z_2) - y^k\right)^2 \qquad (2-183)$$

k = 1,…, K, où O_k est la sortie calculée du système d'inférence floue correspondant à l'entrée x_k , et y_k est la sortie désirée. La méthode de la descente la plus raide est employée pour l'apprentissage des z_i dans la partie conséquente de la ième règle floue, et les paramètres des formes des fonctions d'appartenance A_1 et A_2. C'est-à-dire :

$$z_i(t+1) = z_i(t) - \eta\frac{\partial E^k}{\partial z_i} = z_i(t) - \eta\left(O^k - y^k\right)\frac{A_i(x^k)}{A_1(x^k) + A_2(x^k)} \qquad (2-184)$$

$$a_i(t+1) = a_i(t) - \eta\frac{\partial E^k}{\partial a_i} \qquad (2-185)$$

$$b_i(t+1) = b_i(t) - \eta\frac{\partial E^k}{\partial b_i} \qquad (2-186)$$

Où le $\eta > 0$ est la constante d'apprentissage et « t » indexe le nombre d'ajustements de z_i, a_i, et b_i.

Supposant plus loin que $a_1 = a_2 = a$ et $b_1 = b_2 = b$ simplifie les règles d'apprentissage parce que, dans ce cas-ci, l'équation $A_1(x) + A_2(x) = 1$ se tient pour tout les x du domaine de A_1 et d e A_2. Le réglage des poids deviennent dans ce cas :

$$z_i(t+1) = z_i(t) - \eta(O^k - y^k)A_i(x^k) \qquad (2-187)$$

$$a(t+1) = a(t) - \eta(O^k - y^k)(z_1 - z_2)bA_1(x^k)A_2(x^k) \qquad (2-188)$$

$$b(t+1) = b(t) + \eta(O^k - y^k)(z_1 - z_2)(x^k - a)A_1(x^k)A_2(x^k) \qquad (2-189)$$

Dans la pratique, la situation est typiquement comme suit. Dans l'identification système pour la commande neurale indirecte, ou dans la commande neurale directe nous recherchons à modéliser la loi de commande comme un réseau neuronal, nous modélisons une relation d'entrée-sortie, approximant une certaine fonction y = f (x) depuis les données exprimées comme un ensemble de règles floues.

Une structure appropriée d'ANFIS est choisie pour le problème d'approximation, guidé par un théorème sur l'approximation universelle. Dans les simulations, divers choix de fonctions d'appartenance des règles floues, aussi bien que le choix du nombre des règles, illustrent la capacité d'approximation d'ANFIS. Cependant, dans des applications réelles où la loi de commande est inconnue, ces choix appartiennent à la « l' art de l'ingénieur. » Puisque le

système commandé prévu peut être examiné, une bonne approximation peut être obtenue avec du temps et la patience. Le point est ceci :

La propriété universelle d'approximation d'ANFIS, comme un théorème mathématique, est la directive théorique pour l'usage d'ANFIS.

Chapitre : 3

Conception d'un contrôleur MPPT flou « CF »

3.1 Introduction :

Notre PFE consiste à concevoir un contrôleur MPPT qui repose sur l'approche neuro-floue. Puisque on ne possède pas une base expérimentale prête, on était obligé de passer par la conception d'un contrôleur flou sous Matlab et d'utiliser les résultats de la simulation comme base de données pour entrainer le réseau ANFIS.

Ce contrôleur utilise la logique floue dans le but de rendre la conception plus simple, La poursuite du point MPP est divisée en deux phases : la première sera la recherche rapide, avec un pas important, la seconde sera une phase fine avec un pas très petit, ce qui assure une stabilité du système en diminuant les oscillations autours du point MPP.

3.2 Conception du contrôleur MPPT flou :

Le contrôleur MPPT flou a été conçu et simulé en utilisant la boite à outil ***Fuzzy*** sous ***Simulink*** représentée dans la (figure 3.1) :

Un CLF (contrôleur par la logique flou) se compose de trois blocs (figure 3.2) qui sont :

- La Fuzzification.
- l'Inférence.
- la Défuzzification.

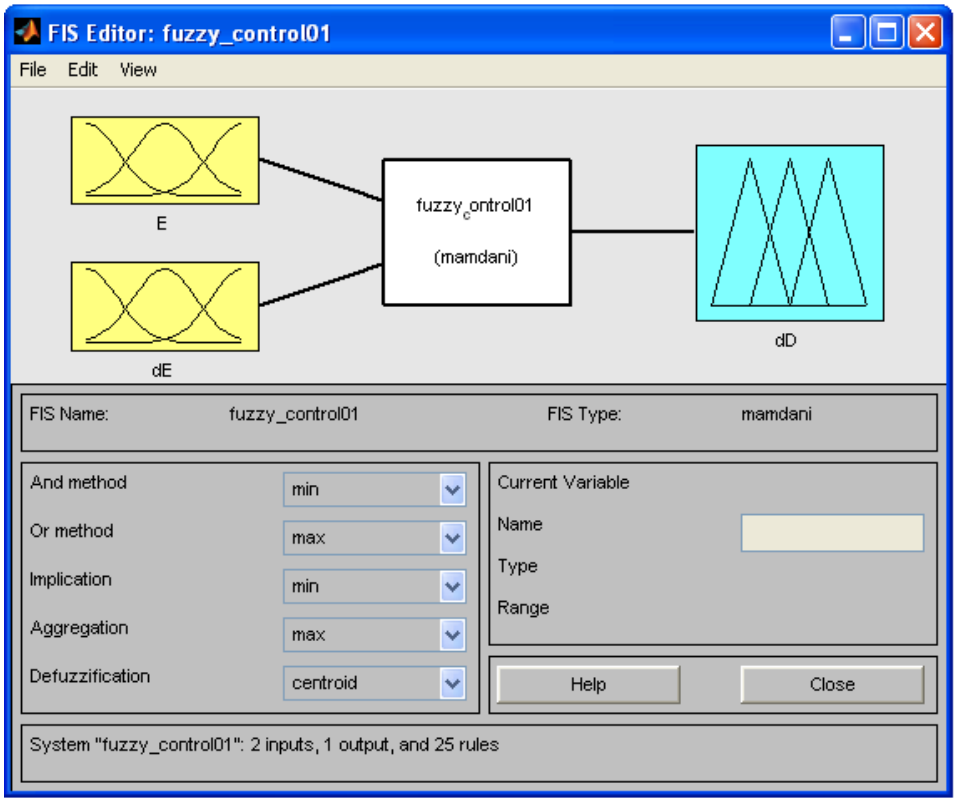


Figure 3.1 Schéma général du contrôleur MPPT flou sous MATLAB

3.2.1 La Fuzzification :

On suppose que le contrôle se fait par la satisfaction de deux critères relatifs à deux variables d'entrées du contrôleur flou proposé, qui sont:

- L'erreur *E* et le changement de l'erreur *dE* à des instants échantillonnés *k*.

Les variables *E* et *CE* sont exprimées comme suit:

$$E(k) = \frac{P(k) - P(k-1)}{V(k) - V(k-1)} \qquad (3-1)$$

$$dE = E(k) - E(k-1) \qquad (3-2)$$

Ou $P(k)$ et $V(k)$ sont respectivement: la puissance et la tension du générateur photovoltaïque.

L'entrée *E(k)* montre si le point de fonctionnement de la charge est situé a gauche ou a droite du point de puissance maximale de la courbe P-V .Si cette valeur est positive, alors le point de fonctionnement se trouve à gauche du point MPP, sinon, le point de fonctionnement

est à droite du point MPP. , L'entrée *CE(k)* par contre montre la direction et nous permet d'estimer la vitesse de convergence vers le point MPP du point de fonctionnement.

Connaissant ces deux entrées on peut décider quel sera la variation du rapport cyclique qu'on doit imposer en agissant sur un hacheur dévolteur : pour augmenter la tension de point de fonctionnement D doit être diminué et vice versa.

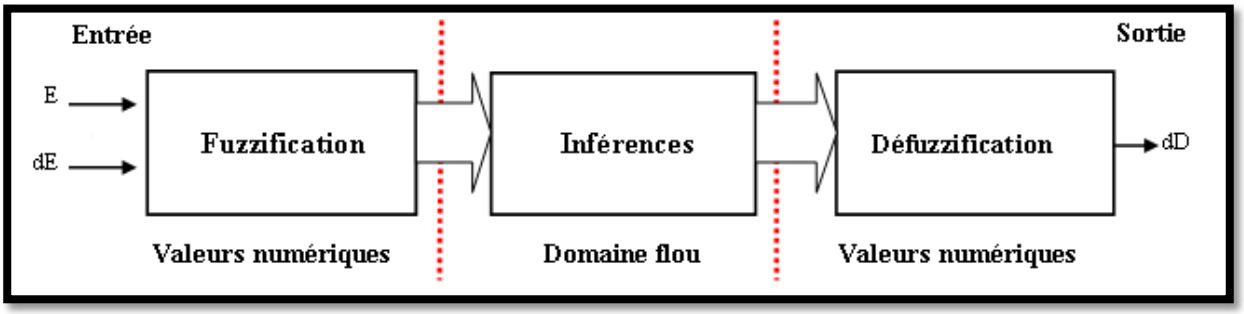


Figure.3.2. Structure de base du contrôleur MPPT flou

3.2.2 Les variables linguistiques :

On peut exprimer les variables d'entrées et de sortie en termes de variables linguistiques suivantes :

- PG: (positif grand).
- PP: (positif petit).
- ZE: (Zéro).
- NP: (négatif petit).
- NG: (négatif grand).

3.2.3 Fonctions d'appartenances :

La génération de fonctions d'appartenance s'effectue soit d'une manière itérative, soit avec le tâtonnement ou encore en se basant sur l'expérience humaine. Il n'existe jusqu'à présent aucune méthode généralisée pour la formulation des stratégies de la logique floue.

[***Le système photovoltaïque utilisé pour la simulation est présenté dans la figure 3.3***]

Après plusieurs essais sous différentes conditions de température et d'éclairement on a défini pour la résolution des problèmes de poursuite du point de puissance maximale les fonctions d'appartenances suivantes : figures (3.4, 3.5, 3.6).

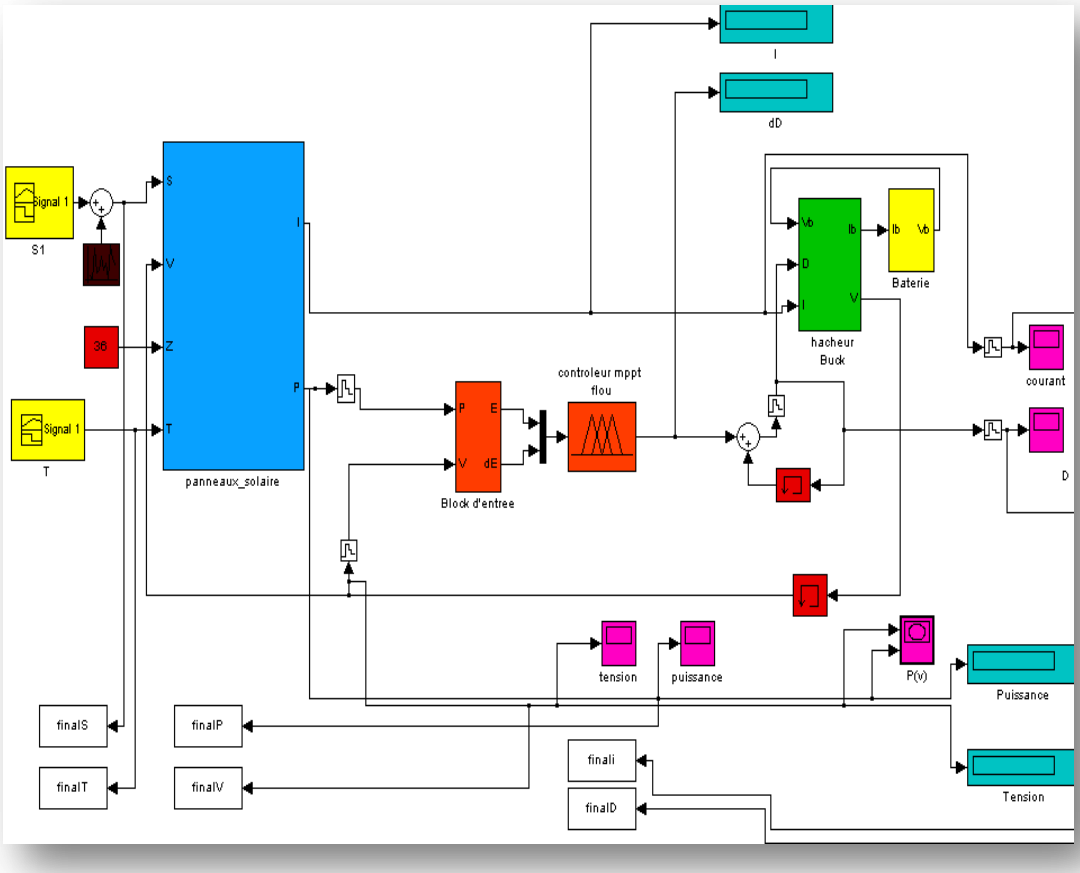


Figure 3.3 : Schéma synoptique d'un système photovoltaïque avec le contrôleur flou sous Simulink.

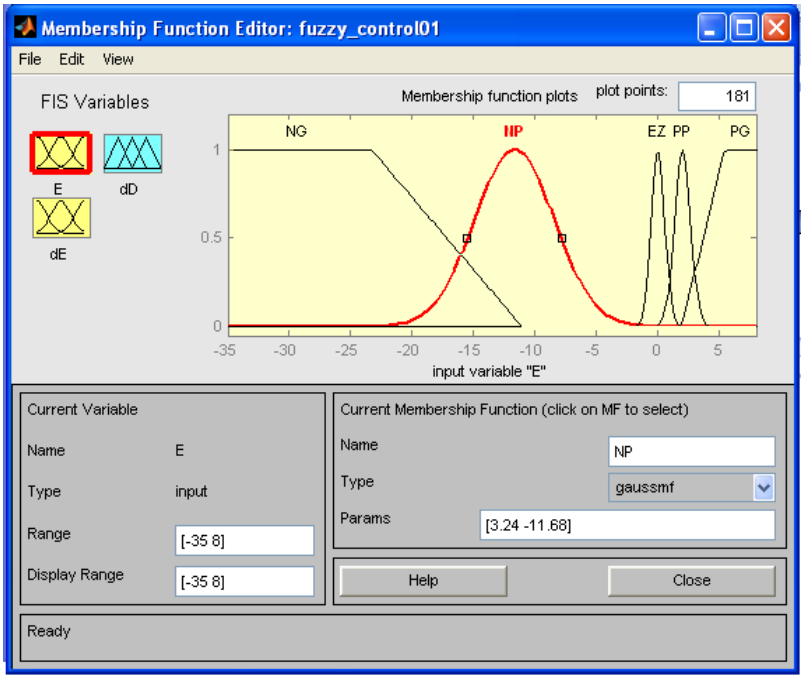


Figure 3.4 : fonctions d'appartenance pour la variable d'entrée E

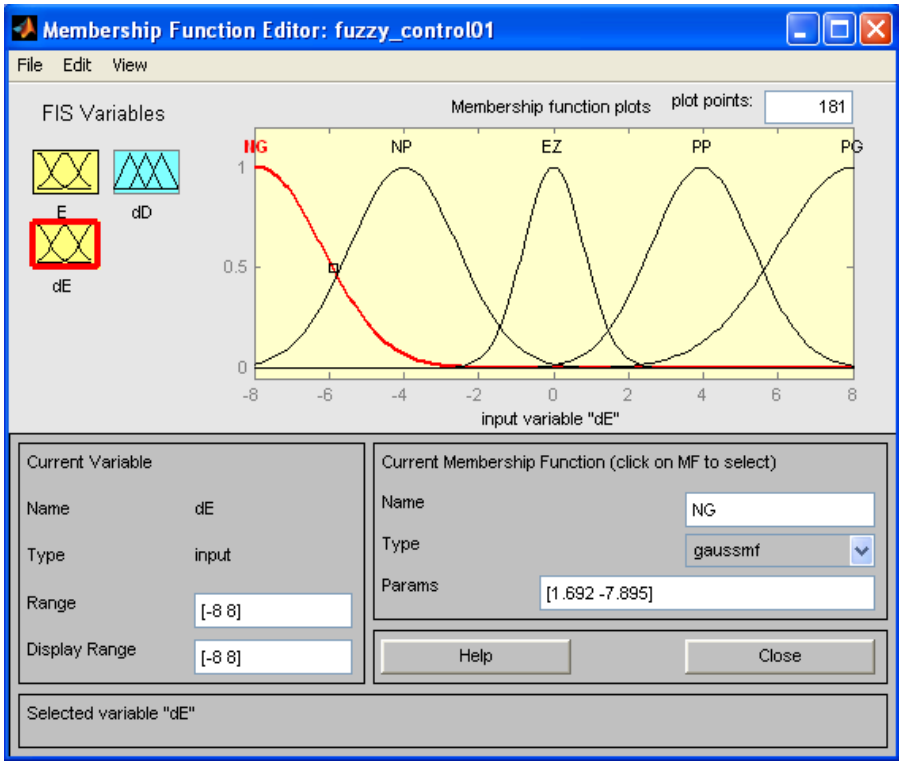


Figure 3.5 : fonctions d'appartenance la variable d'entrée dE

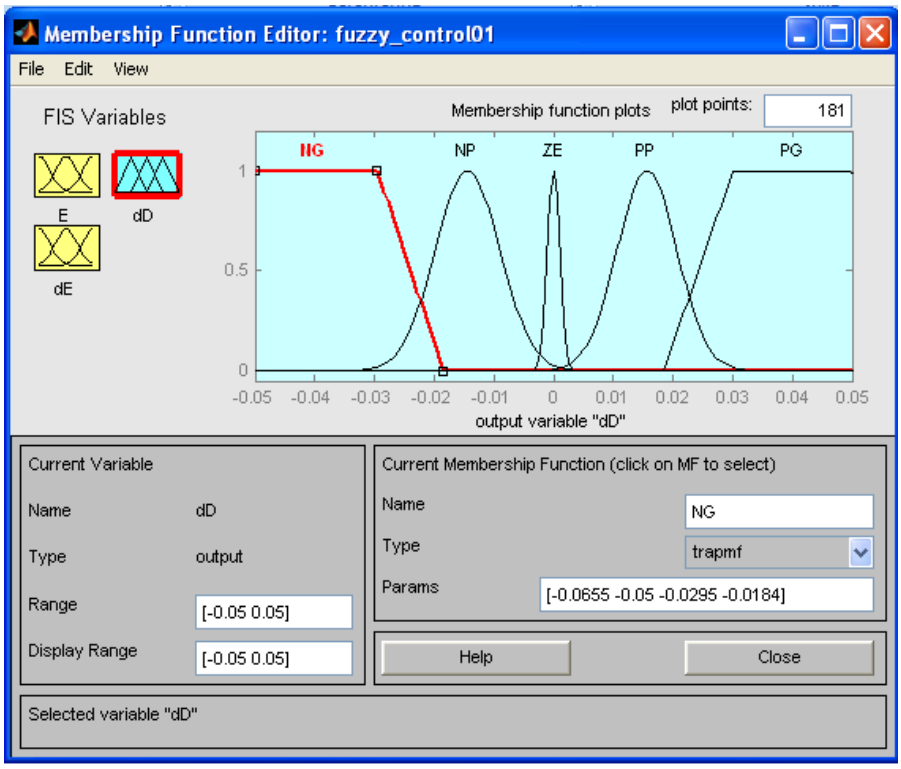


Figure3.6 : fonctions d'appartenance pour la variable de sortie dD

3.2.4 Inférence :

La méthode d'inférence utilisée est celle de « Mamdani », c'est généralement la méthode la plus utilisée. Elle utilise l'opérateur **MIN** pour le **« ET »** et l'opérateur **MAX** pour le « OU ». Les règles d'inférence permettent de prendre la bonne décision pour la sortie dD à partir des valeurs des entrées E et dE. Dans notre travail on a choisi les règles présentées dans le (tableau 5.1) :

E / ***dE***	**NG**	**NP**	**ZE**	**PG**	**PP**
NG	PG	PG	PG	PG	PG
NP	PG	PP	PP	PP	ZE
ZE	PP	PP	ZE	NP	NP
PG	NP	NP	NP	NP	NP
PP	NG	NP	NP	NP	ZE

Tableau 3.1 : Table d'inférence du régulateur flou

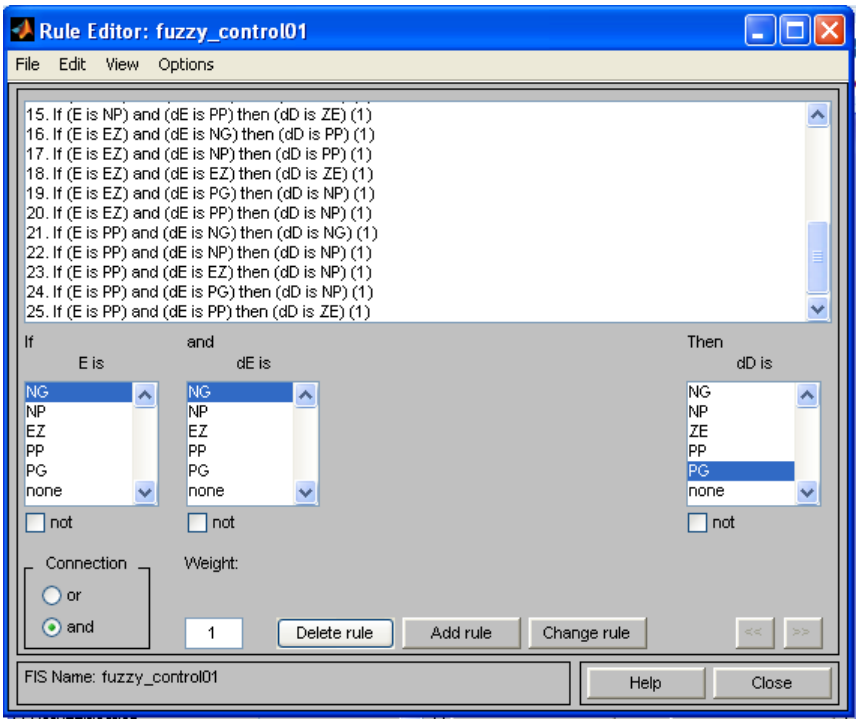


Figure 3.7 : Saisie des règles floues pour le CLF

3.2.5 La défuzzification :

La méthode de défuzzification, utilisée dans le CLF est la méthode du centre de gravité. La figure 3.8 donne le graphe en 3 dimensions de la sortie dD en fonction des deux entrées E et dE.

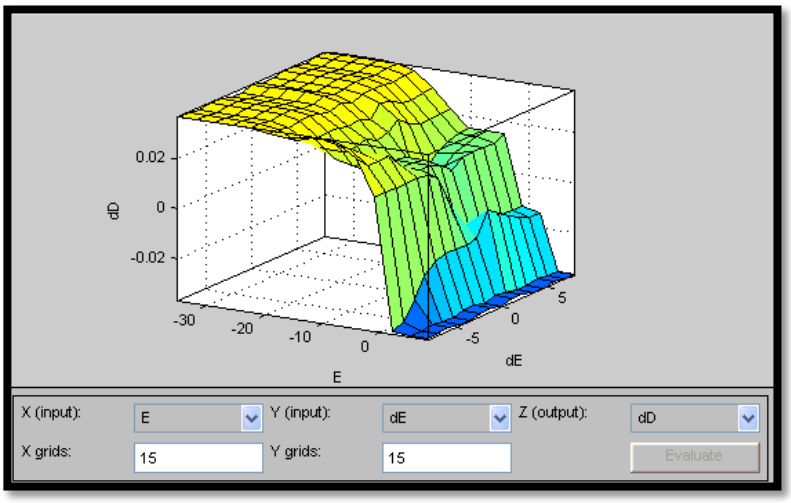


Figure 3.8 Résultat de défuzzification dD = f(E, dE)

3.3 Simulation de l'application de CLF à la poursuite du point de puissance maximale :

Dans le but de confirmer l'efficacité de notre CLF on a pris un signal d'ensoleillement et de température variables avec un bruit à l'entrée de telle manière qu'on aura 5 points de puissance maximale différents que le contrôleur MPPT doit suivre. Les résultats de la simulation du SIMULINK sont représentés dans les figures (3-9, 3-10, 3-11, 3-12, 3-13 et 3-14) :

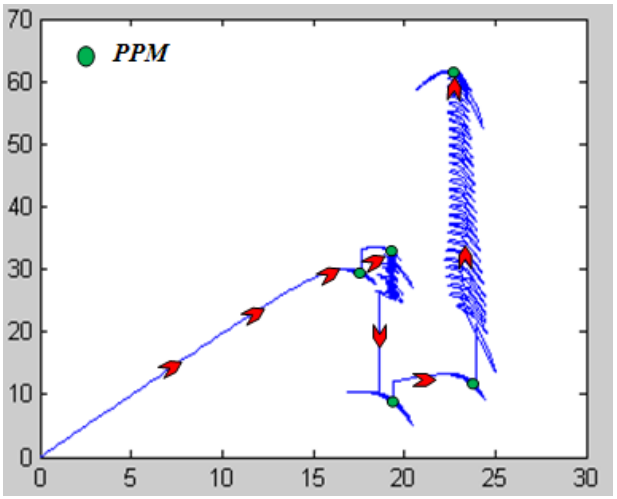


Figure 3.9 : suivi de point de puissance maximale dans des conditions variables de température et d'ensoleillement ; la courbe représente la puissance à la sortie du GPV en fonction de la tension de celle-ci.

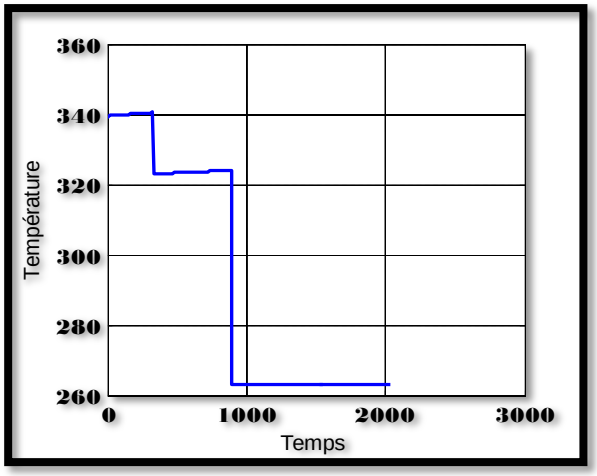


Figure3.10 le signal de Température(K) appliqué à l'entrée du GPV

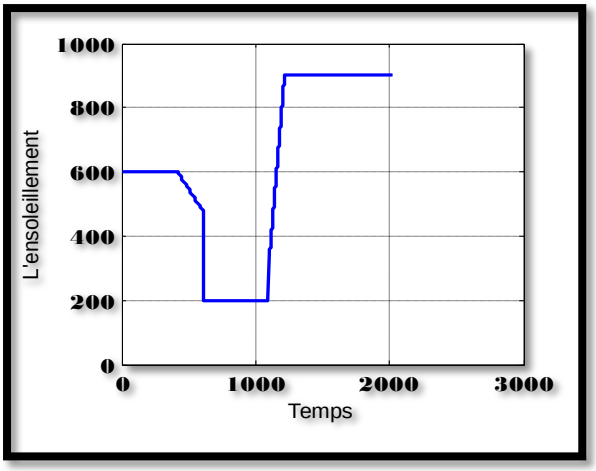


Figure3.11 le signal d'ensoleillement (w/m^2) appliqué à l'entrée du GPV

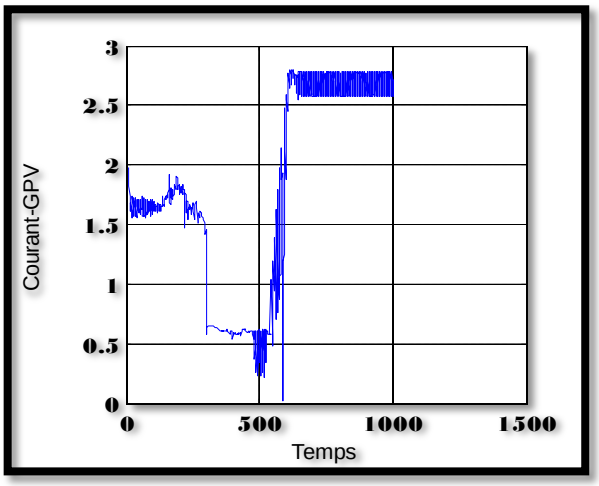


Figure 3.12 Variation du courant (A) à la sortie du GPV

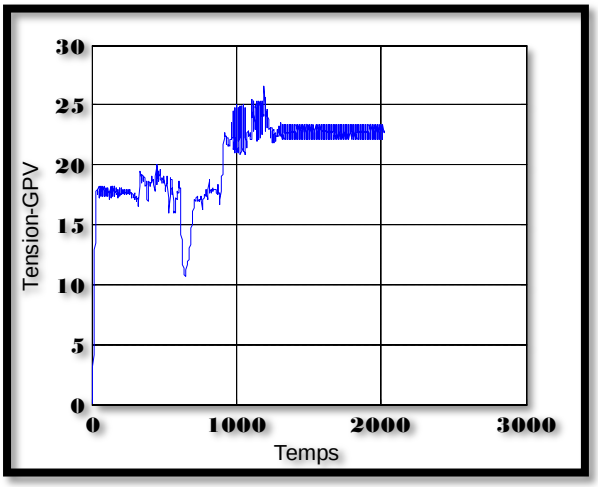


Figure 3.13 Variation du Tension (V) à la sortie du GPV

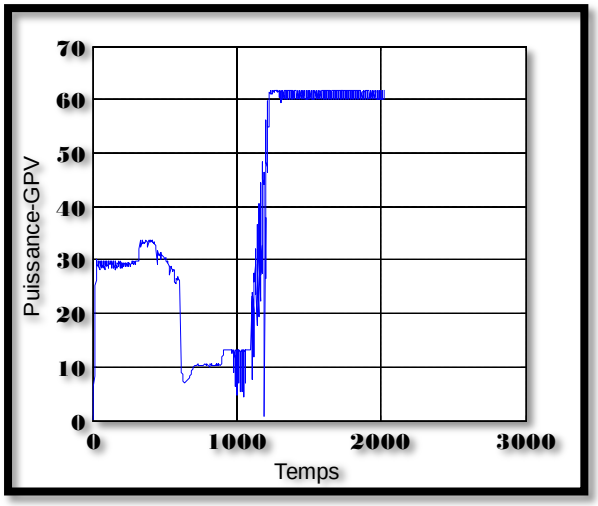


Figure 3.14 La puissance (w) fournie par le GPV

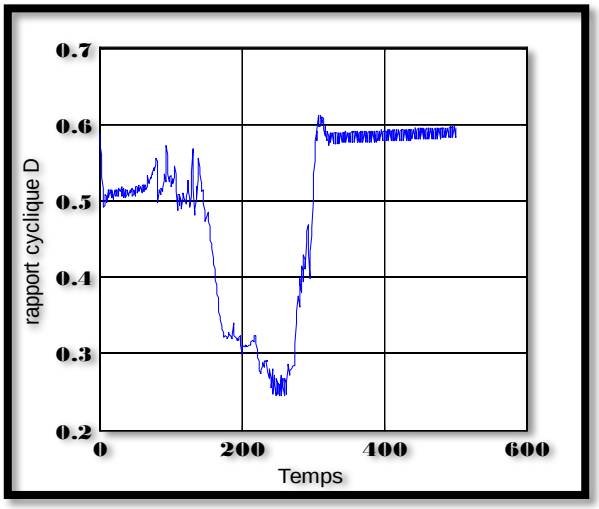


Figure 3.15 Variation du rapport cyclique à la sortie du CLF

3.4 Avantages et inconvénients du réglage par logique floue :

3.4.1. Avantages :

Le réglage par logique floue présente les avantages suivants :

- Le non nécessité du modèle du système (souplesse de la méthode)
- Possibilité d'entreprendre des systèmes complexes (systèmes non linéaire et difficile à modéliser).
- La disponibilité de système de développement efficace, soit par microprocesseur ou PC, soit par circuits intégrés. Dans notre projet par l'implémentation est faite sur le circuit FPGA.

3.4.2. Inconvénients :

- Le manque de directives précises pour la conception d'un réglage (détermination de la fuzzification, des inférences et de la défuzzification).
- L'approche artisanale et asymptotique (implantation des connaissances de l'opérateur souvent difficile).
- L'impossibilité de démontrer de la stabilité du contrôleur.

3.5 Conclusion :

Dans ce chapitre, nous avons présenté la démarche détaillée pour la conception d'un contrôleur MPPT flou en utilisant la boite à outil Fuzzy de Matlab. La simulation nous a permis de voir le bon fonctionnement de CLF pour la poursuite du PPM. Par la suite, comme on a signalé dans l'introduction, le but est d'extraire une base de donnée : $dD = f(E, dE)$ pour l'utilisée plus tard dans le contrôleur neuro-flou, pour cela on a essayé lors de choix des triplets (E, dE, dD) (inférences) de balayer tout l'intervalle de variation de E et dE afin d'avoir un bon CLF. La conception de contrôleur MPPT neuro-flou sera faite dans le chapitre suivant.

Chapitre : 4

Conception d'un contrôleur MPPT par l'approche neuro-floue

4-1 Introduction :

L'architecture des réseaux neuro-flous combine les caractéristiques des réseaux de neurones en profitant de leurs capacités d'apprentissage et le caractère incertain de la logique floue qui aide beaucoup à la résolution des problèmes non linéaires. Dans ce chapitre on va présenter les étapes de la conception d'un contrôleur neuro-flou pour la commande d'un hacheur série dans le but d'atteindre le point de puissance maximale du système photovoltaïque. Afin d'accomplir cette tâche on a utilisé la boite à outil ***ANFIS editor*** proposé par MATLAB, (figure 4.1).

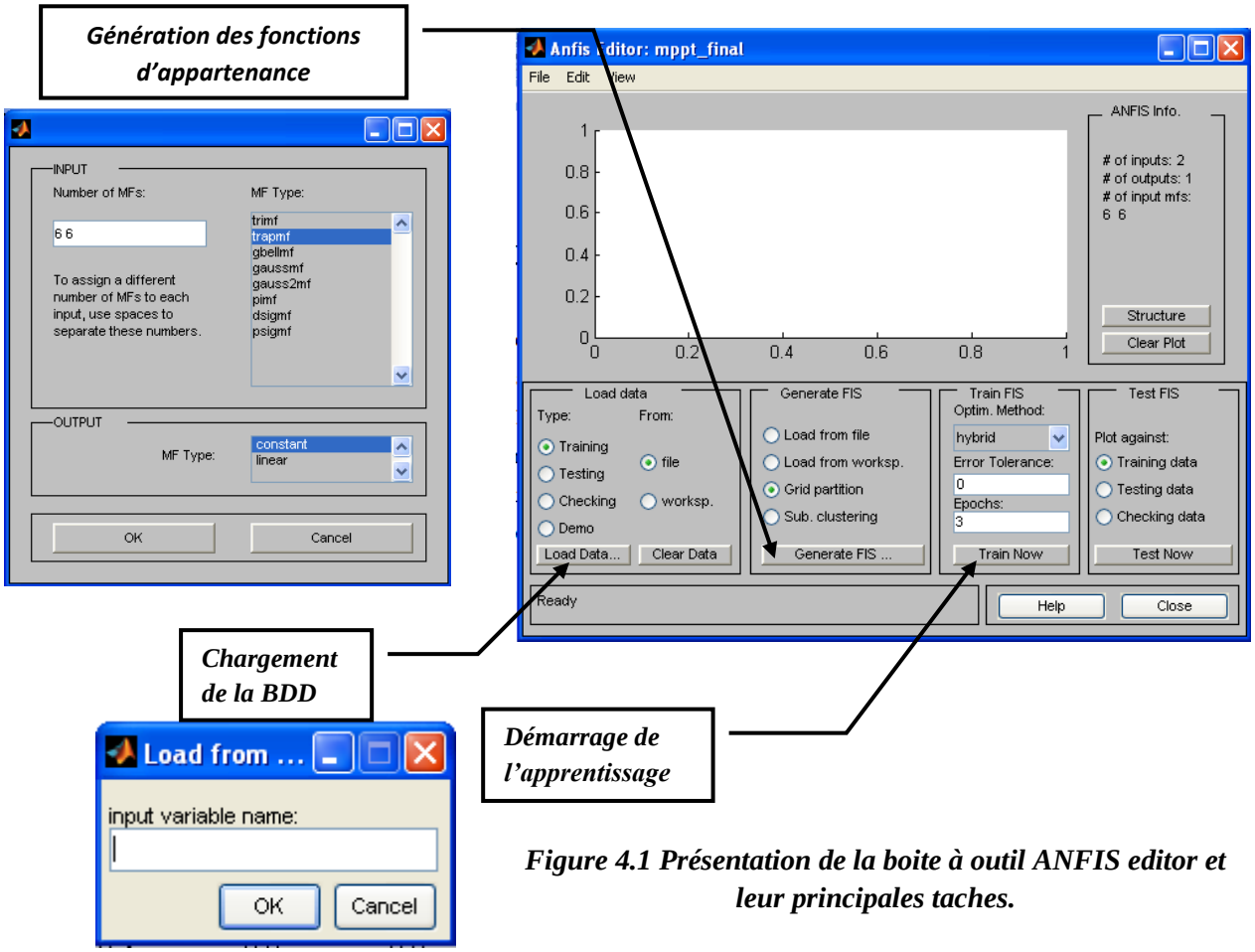


Figure 4.1 Présentation de la boite à outil ANFIS editor et leur principales taches.

4.2 Structure du contrôleur MPPT neuro-flou proposé :

Le contrôleur neuro-flou a deux entrées E et dE, et une seule sortie dD qui représentent respectivement l'erreur, la variation de l'erreur, et la variation du rapport cyclique pour la commande du convertisseur DC/DC pour assurer l'adaptation de la puissance fournie par le GPV.

Notre contrôleur neuro-flou est un contrôleur **ANFIS** qui utilise un système d'inférence flou (SIF) du type Takagi Sugeno. Il est composé de cinq couches comme représentées sur les figures 4.2, 4.3. Ce contrôleur permet la génération des règles floues :

Si e est A_1 et de est B_1 . Alors $d_1=f(e, de)$

Si e est A_1 et de est B_2 . Alors $d_2= f\ e, de)$

.

.

Si e est A_6 et de est B_6 . Alors $d_{36}=f(e, de)$

Où ***E, dE***, sont des variables d'entrée et ***A_1, A_2, . . ., A_6 et B_1, B_2,.....B6*** sont des ensembles flous. Lorsque ***d= constante***, on obtient un modèle de Sugeno d'ordre zéro. Lorsque d est une combinaison linéaire des entrées ***: d= a E + b dE + c***, on obtient un modèle de Sugeno de premier ordre.

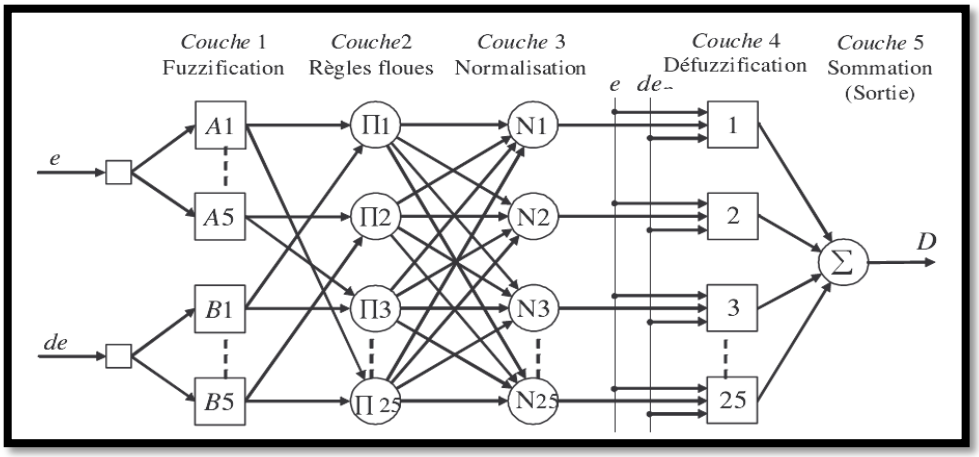


Figure 4.2 : Architecture du modèle ANFIS.

Les cinq couches du réseau neuro-flou ont pour rôle :

Couche 1 :

Les neurones de cette couche réalisent les ensembles flous qui serviront dans les antécédents des règles.

Couche 2 :

Chaque neurone dE cette couche correspond à une règle floue Sugeno. Il reçoit les sorties des neurones de fuzzification et calcule son activation. La conjonction des antécédents est réalisée avec l'opérateur produit.

Couche 3 :

Chaque neurone calcule le degré de vérité normalisé d'une règle floue donnée. La valeur obtenue représente la contribution de la règle floue au résultat final.

Couche 4 :

Chaque neurone i de cette couche est relié à un neurone de normalisation correspondant et aux entrées initiales du réseau. Il calcule le conséquent.

Couche 5 :

Comprend un seul neurone qui fournit la sortie de ANFIS en calculant la somme des sorties de tous les neurones de défuzzification

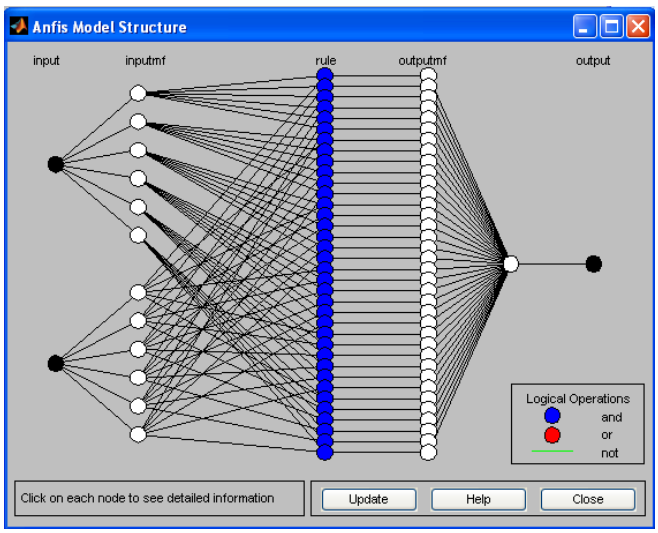


Figure 4.3 La structure neuronale du modèle proposé par MATLAB

4.3 Conception du contrôleur MPPT neuro-flou :

Une fois que le vecteur d'entrée-sortie est obtenu, et comme ANFIS fait partie de la boite à outil "logique floue" de Matlab, il a été utilisé pour construire l' ensemble des règles floues « ***Si Alors*** » avec leurs fonctions d'appartenance appropriées. La manière dont ANFIS fonctionne est décrite dans l'organigramme de la figure 4.4.

L'apprentissage du contrôleur est effectué par l'algorithme de rétro propagation, il détermine les paramètres liés aux fonctions d'appartenance, et les paramètres conséquents. Ce qui a pour appellation "apprentissage hybride".

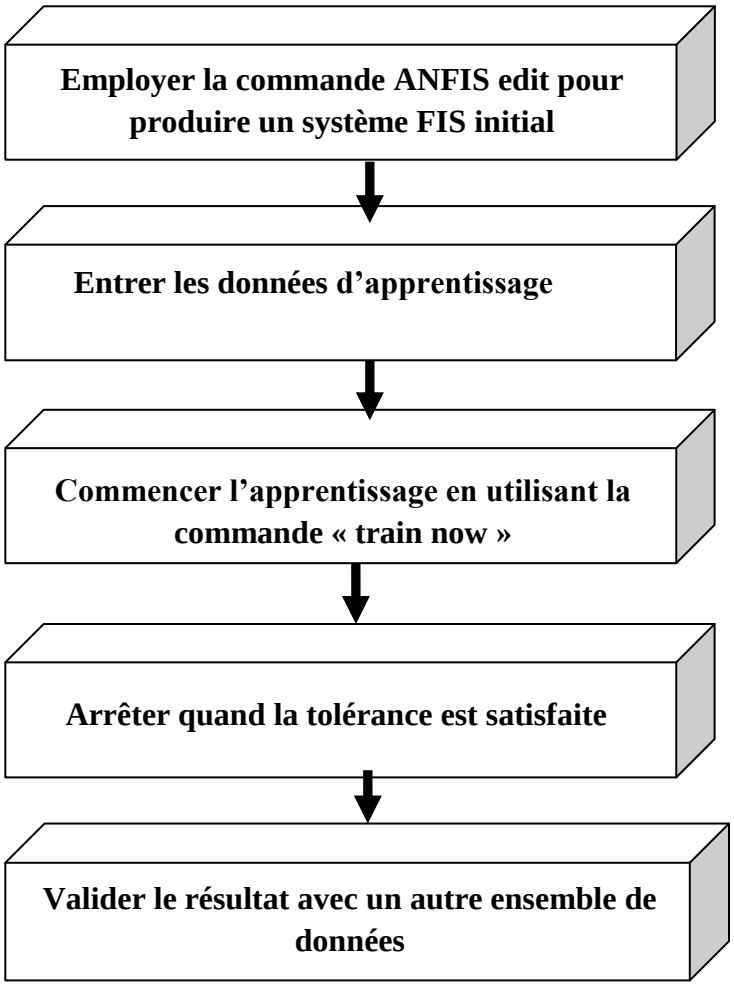


Figure 4.4 Les étapes de la conception d'système neuro-flou sous MATLAB.

Le contrôleur MPPT neuro-flou est représenté globalement par la (figure 4.5) :

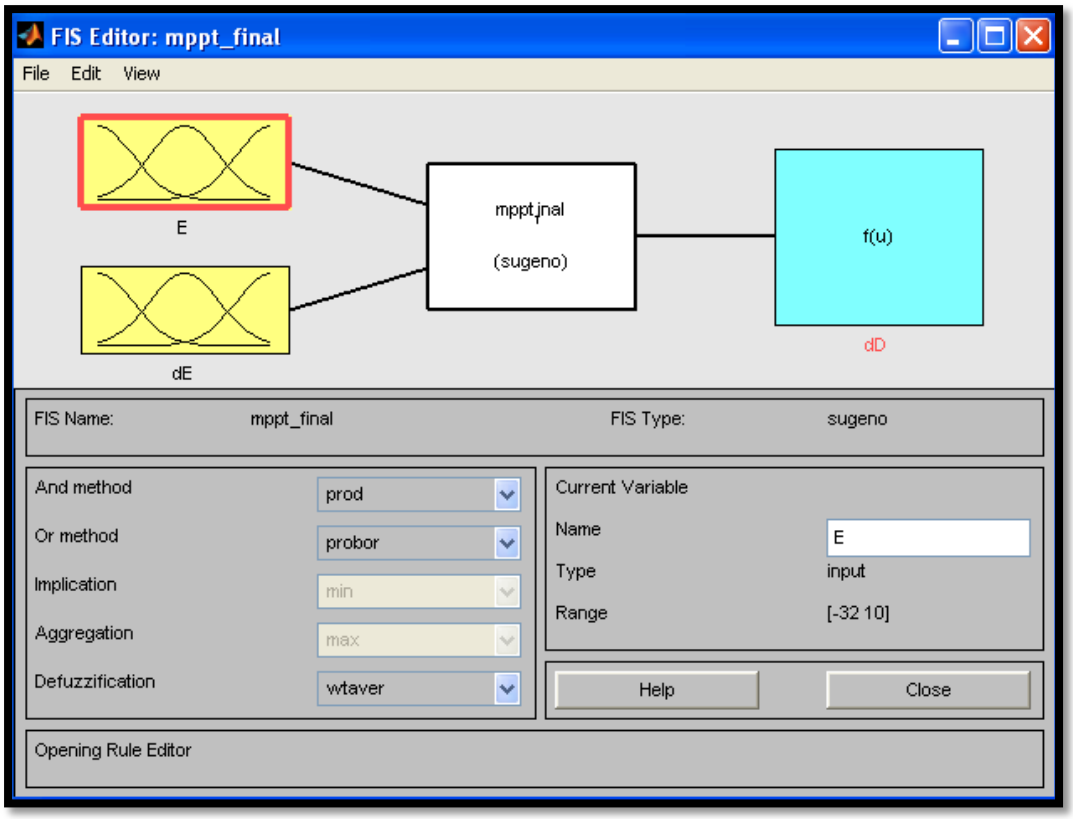


Figure 4.5 : Structure neuronale du modèle proposé sous Matlab.

Il est important de noter qu'il n'existe pas une méthode exacte pour déterminer la table des règles et le nombre de fonctions d'appartenance assigné à chaque variable d'entrée ainsi que leurs formes. Généralement, ce nombre est choisi empiriquement en examinant son influence sur la réaction du système. Dans notre cas et après plusieurs changements on a choisi 6 fonctions d'appartenance pour chaque entrée (E, dE) avec la forme d'un trapèze, et un modèle de Sugeno d'ordre zéro (figures 4.6 et 4.7) et (tableau 4.1) :

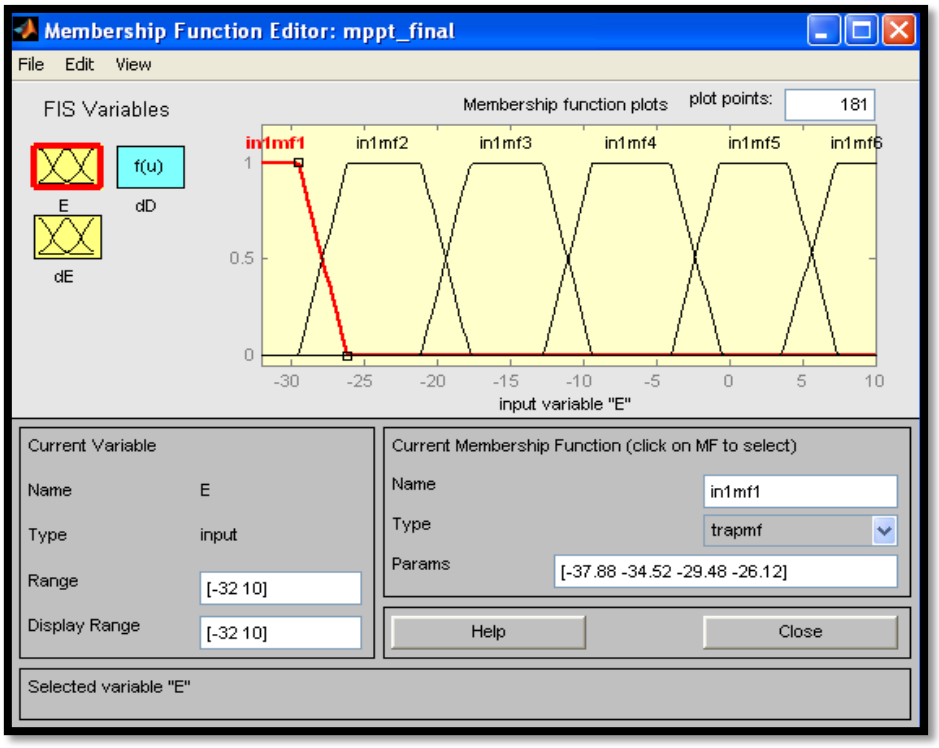


Figure 4.6 Les fonctions d'appartenance de l'erreur "e" générées par ANFIS après apprentissage.

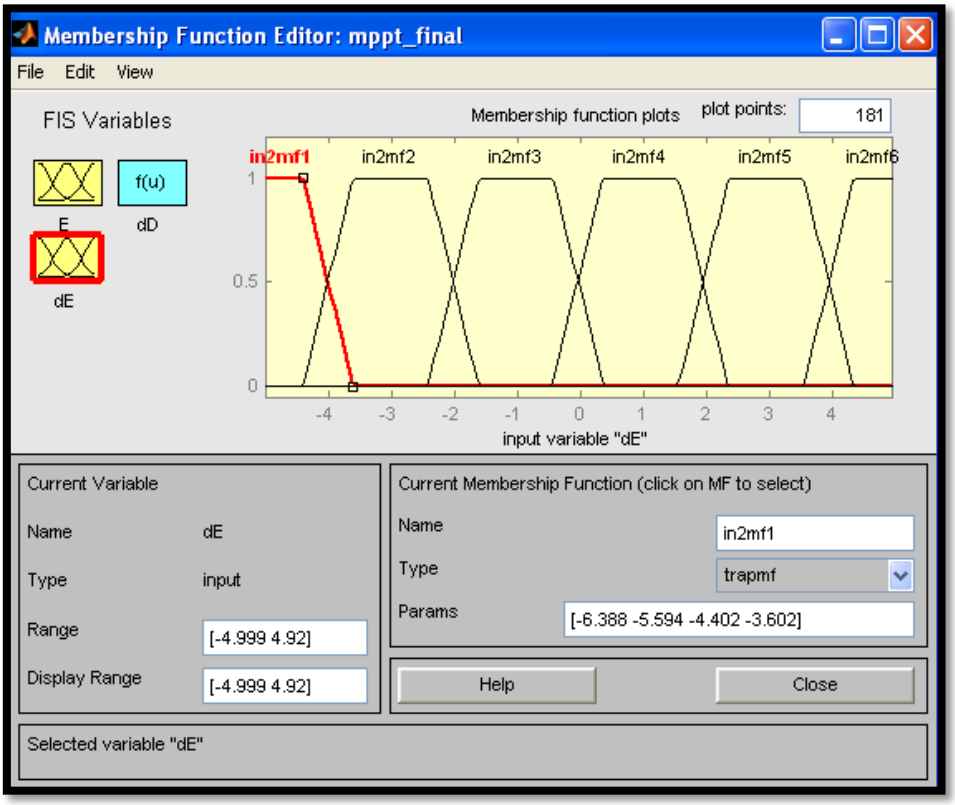


Figure 4.7 Les fonctions d'appartenance du changement de l'erreur "de" générées par ANFIS après apprentissage.

Le tableau 4.1 suivant montre les règles générées par l'ANFIS :

E↓ dE→	B1	B2	B3	B4	B5	B6
A1	D1= 0.03686	D2= 0.03606	D3= 0.03582	D4= 0.03599	D5= 0.03611	D6= 0.03704
A2	D7= 0.03761	D8= 0.03687	D9= 0.03624	D10= 0.03641	D11= 0.03649	D12= 0.03714
A3	D13= 0.02323	D14= 0.02369	D15= 0.0247	D16= 0.02425	D17= 0.02594	D18= 0.02194
A4	D19= 0.02765	D20= 0.01921	D21= 0.01729	D22= 0.01544	D23= 0.009089	D24= 0.01423
A5	D25= -0.004325	D26= -0.00140	D27= -0.00523	D28= -0.0137	D29= -0.01415	D30= -0.01554
A6	D31= -0.04664	D32= -0.04796	D33= -0.04616	D34= -0.04528	D35= -0.04383	D36= -0.04283

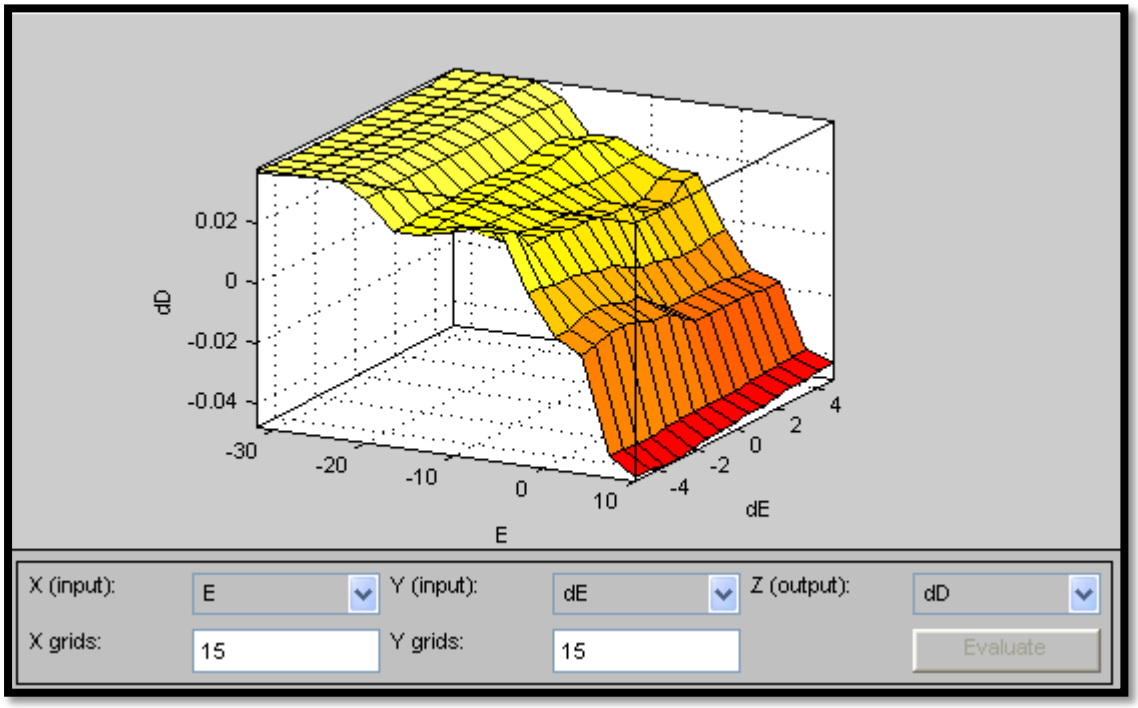


Figure 4.8 la courbe en 3D de la sortie dD en fonction des deux entrées E ,dE du contrôleur neuro-flou.

4.4 Simulation de l'application de l'approche neuro-floue à la poursuite du point de puissance maximale :

Une fois le contrôleur neuro-flou est testé, il est inséré dans le système photovoltaïque. Voici les résultats de la simulation.

1/ On prend la température variable et descendante avec ensoleillement constant=1000w/m^2.

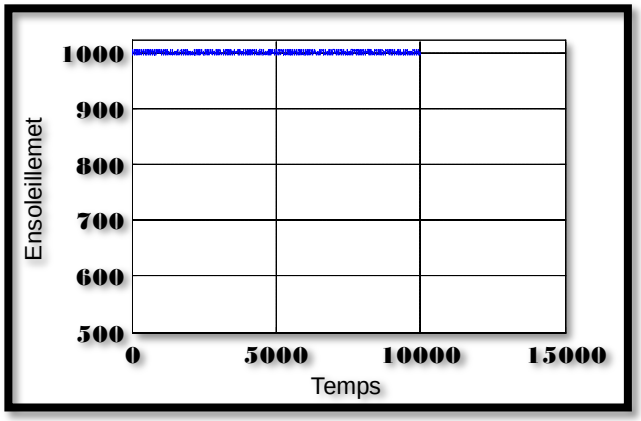


Figure4.10 le signal d'ensoleillement (w/m^2) appliqué à l'entrée du GPV

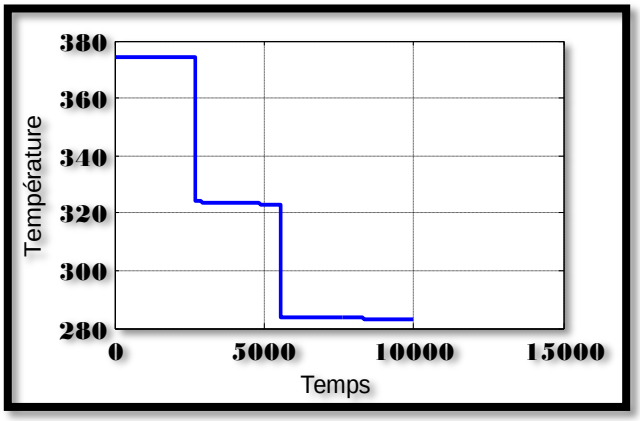


Figure4.9 le signal de Température (K) appliqué à l'entrée du GPV

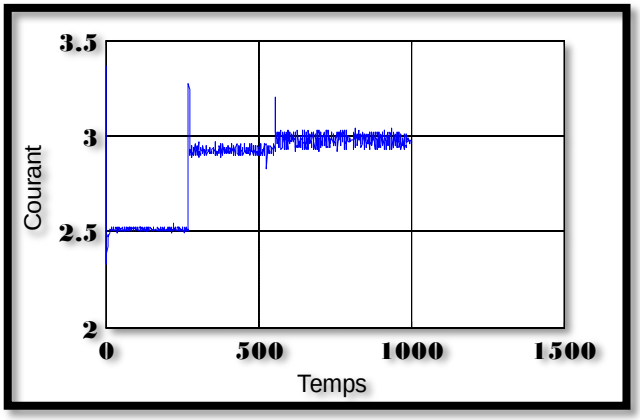


Figure 4.11 Variation du courant(A) à la sortie du GPV

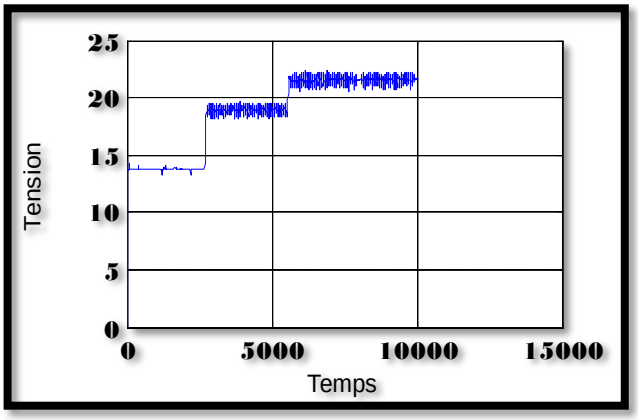


Figure 4.12 Variation du Tension(V) à la sortie du GPV

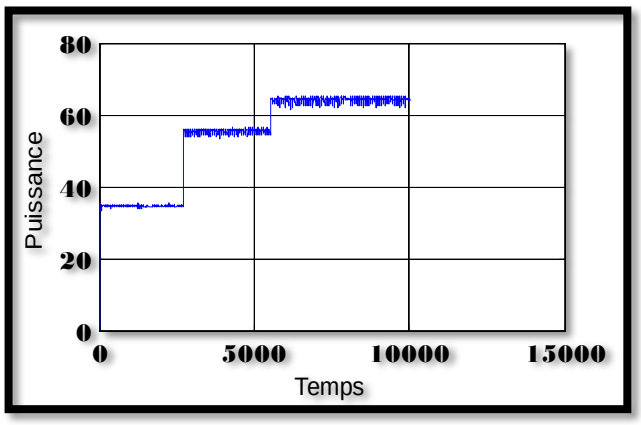


Figure 4.13 La puissance (w) fournie par le GPV

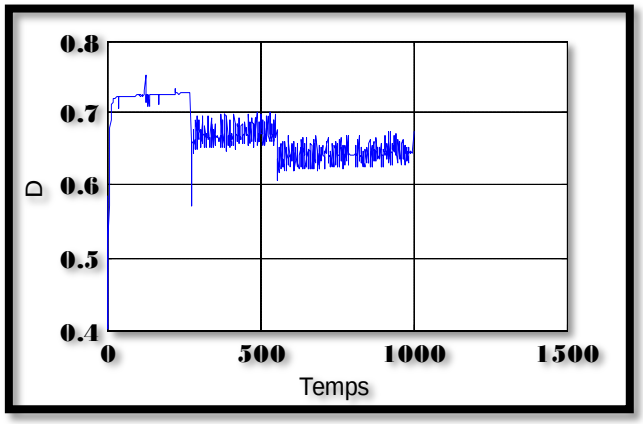


Figure 4.14 Variation du rapport cyclique à la sortie du contrôleur neuro-flou

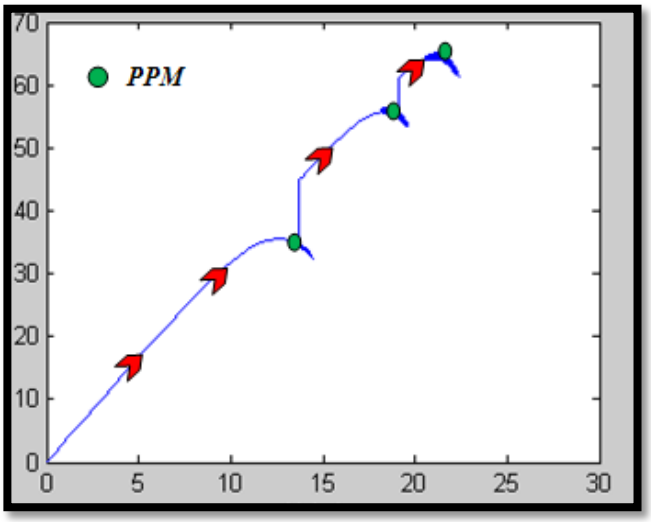


Figure 4.15 : suivi de point de puissance maximale dans des conditions variables de température ; la courbe représente la puissance fournie par le du GPV en fonction de la tension de celui-ci.

2/ On prend la température variable et ascendante avec ensoleillement constant=1000w/m^2.

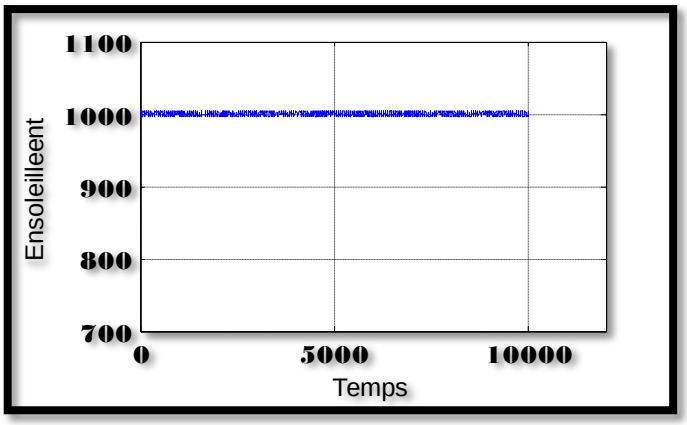


Figure4.16 le signal d'ensoleillement (w/m^2) appliqué au GPV

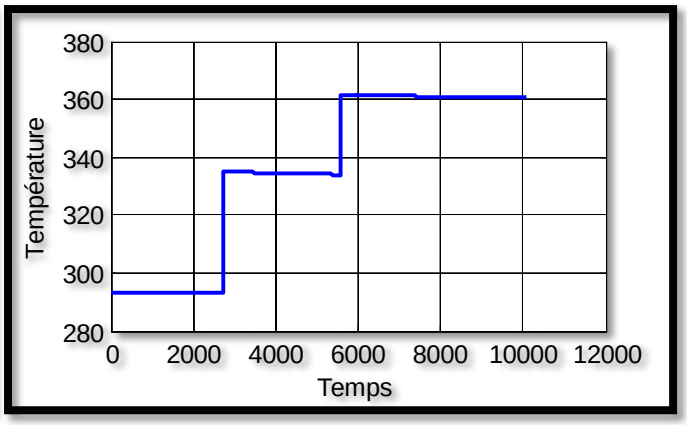


Figure4.17 le signal de Température (K) appliqué au GPV

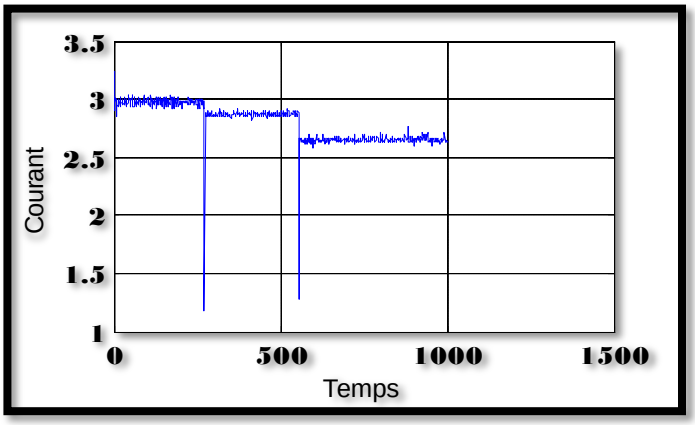


Figure 4.18 Variation du courant (A) à la sortie du GPV

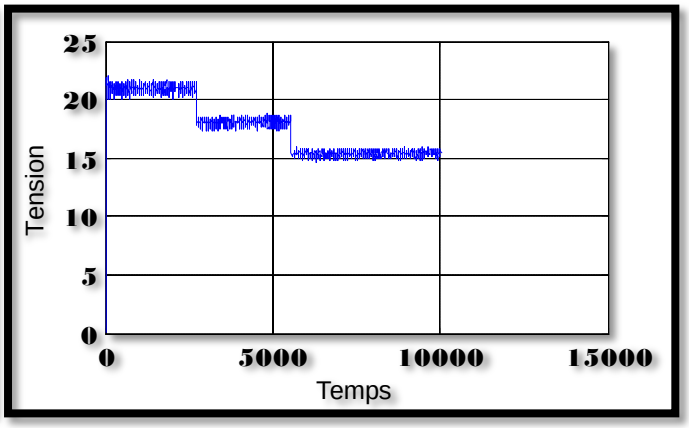


Figure 4.19 Variation du Tension (V) à la sortie du GPV

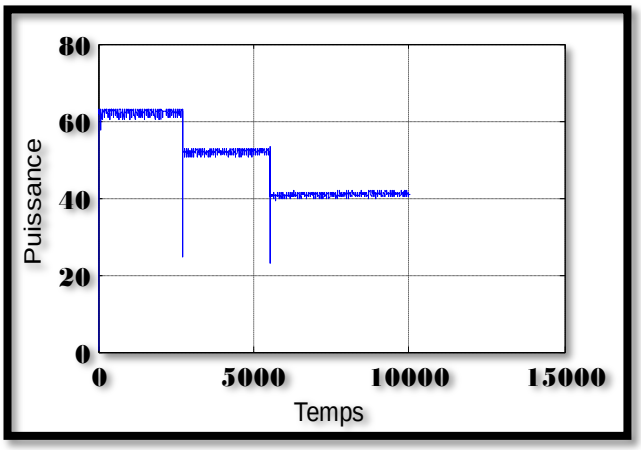


Figure 4.20 La puissance (w) fournie par le GPV

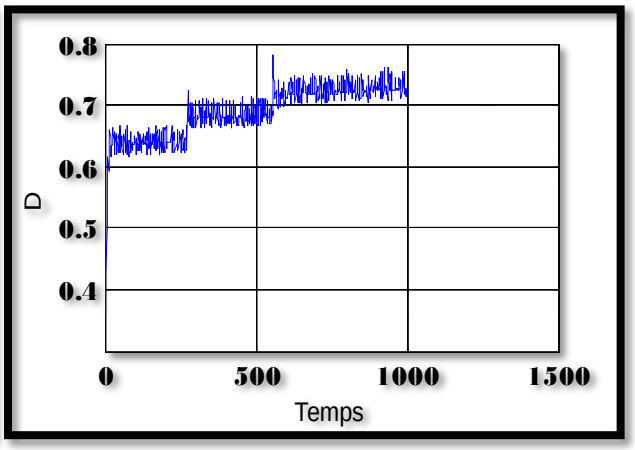


Figure 4.21 Variation du rapport cyclique à la sortie du contrôleur neuro-flou

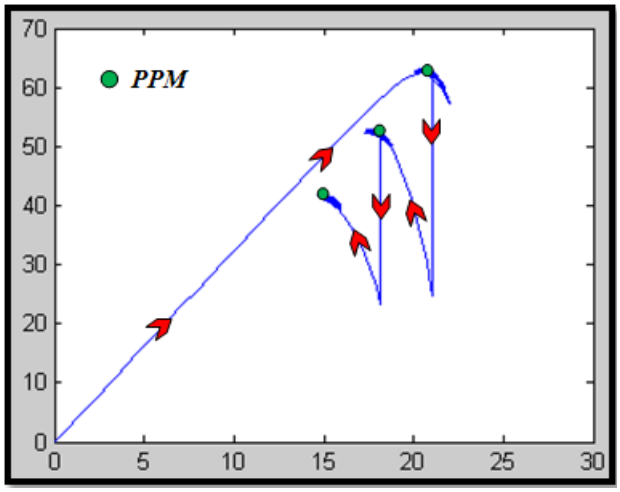


Figure 4.22 : suivi de point de puissance maximale dans des conditions variables de température ; la courbe représente la puissance fournie par le du GPV en fonction de la tension de celui-ci.

3/ On prend la température constante=273K avec un signal d'ensoleillement variable :

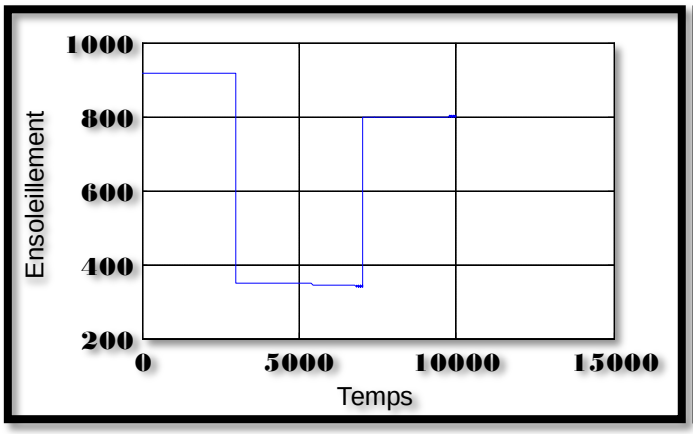


Figure4.23 le signal d'ensoleillement (w/m^2) appliqué au GPV

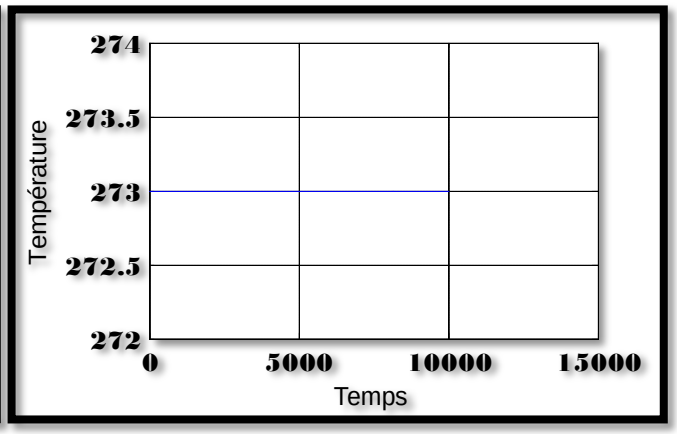


Figure4.24 le signal de Température (K) appliqué au GPV

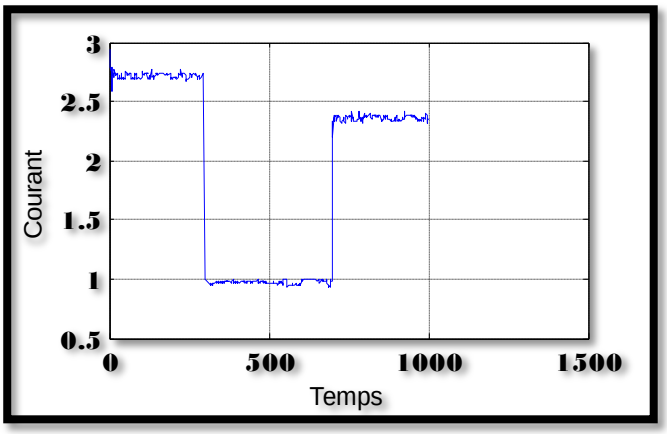


Figure 4.25 Variation du courant (A) à la sortie du GPV

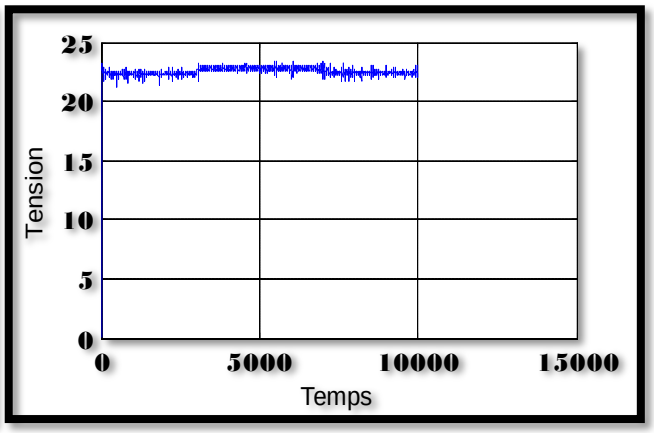


Figure 4.26 Variation du Tension (V) à la sortie du GPV

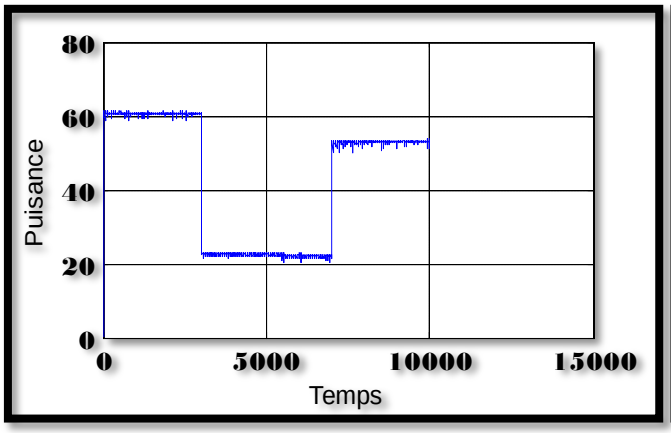


Figure 4.27 La puissance (w) fournie par le GPV

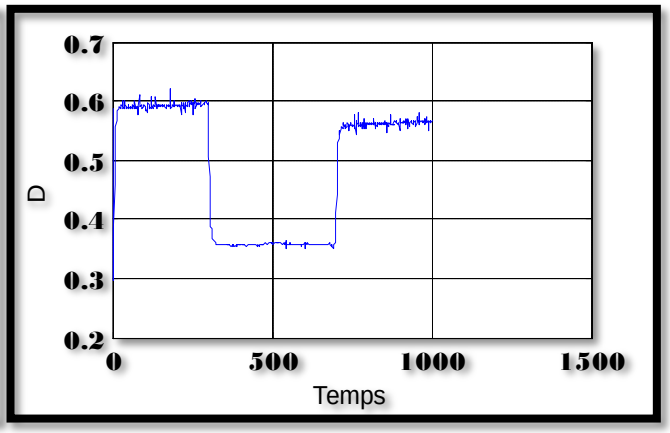


Figure 4.28 Variation du rapport cyclique à la sortie du contrôleur neuro-flou

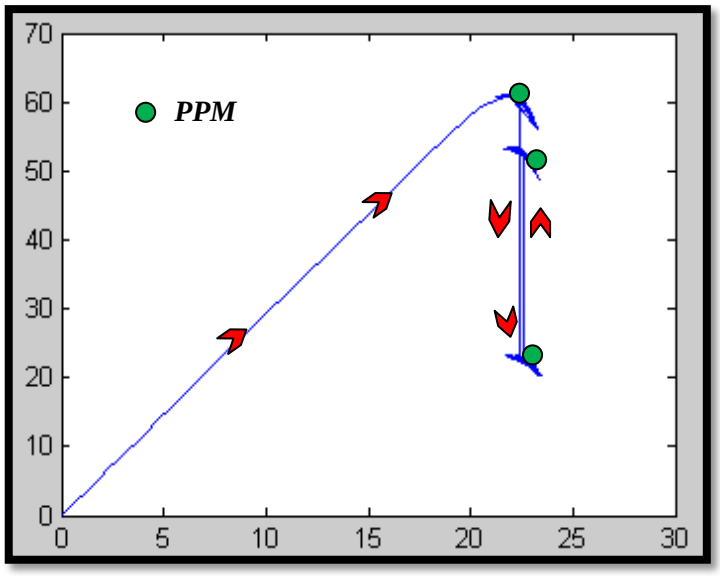


Figure 4.29 : suivi de point de puissance maximale dans des conditions variables d'ensoleillement ; la courbe à coté représente la puissance fournie par le du GPV en fonction de la tension de celui-ci.

4/ On prend la température variable l'ensoleillement variable.

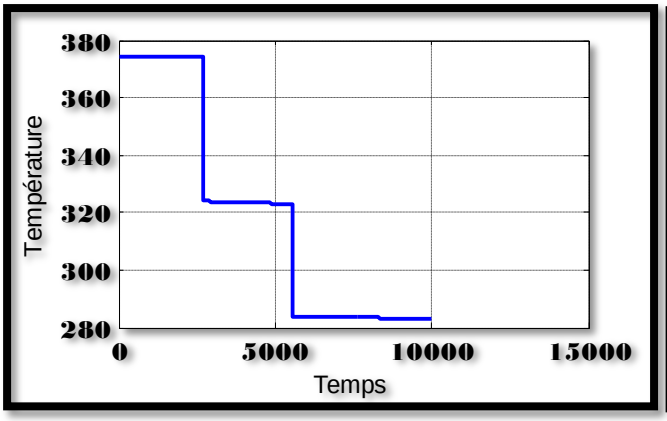


Figure4.30 le signal de Température (K) appliqué au GPV

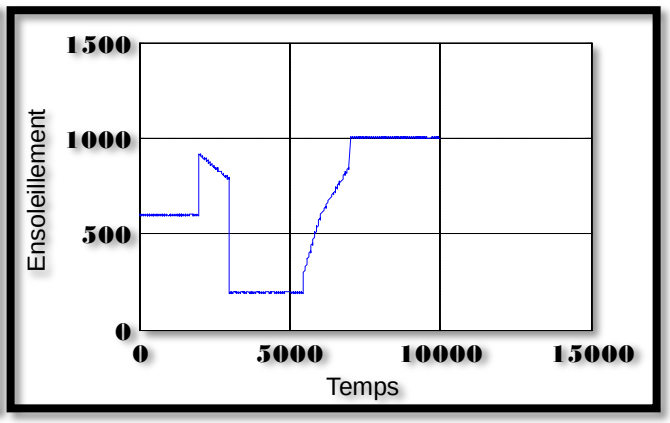


Figure4.31 le signal d'ensoleillement (w/m^2) appliqué au GPV

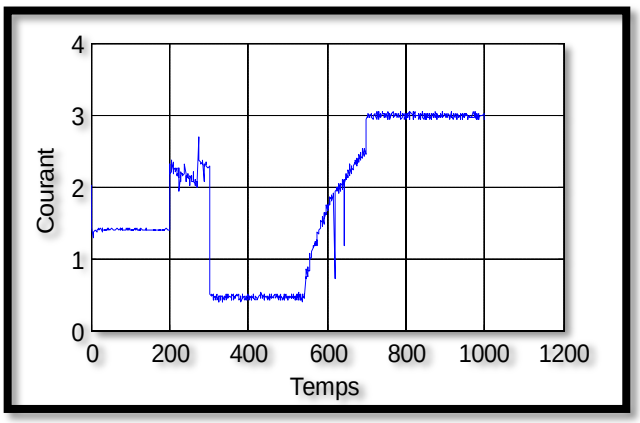


Figure 4.32 Variation du courant (A) à la sortie du GPV

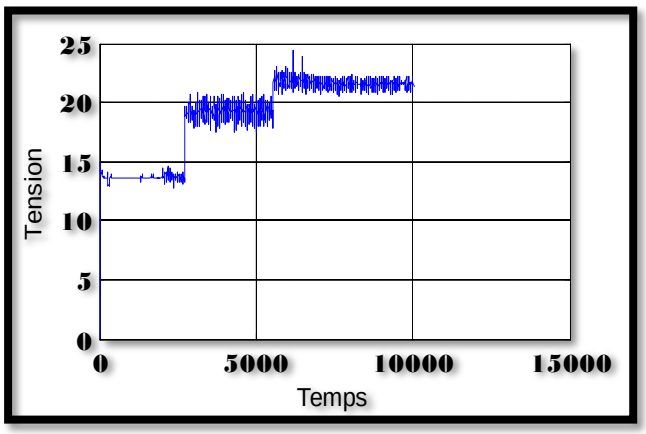


Figure 4.33 Variation du Tension (V) à la sortie du GPV

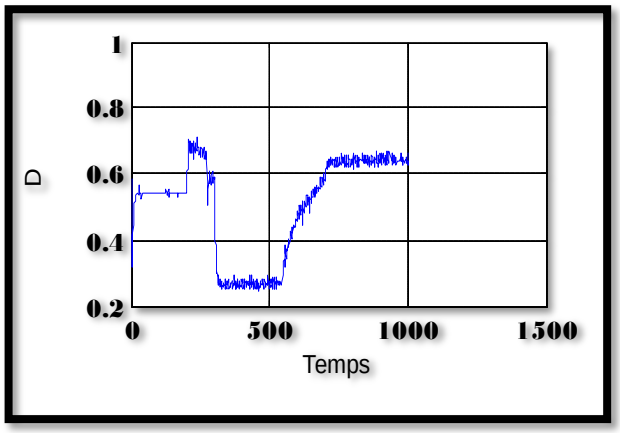


Figure 4.34 Variation du rapport cyclique à la sortie du contrôleur neuro-flou

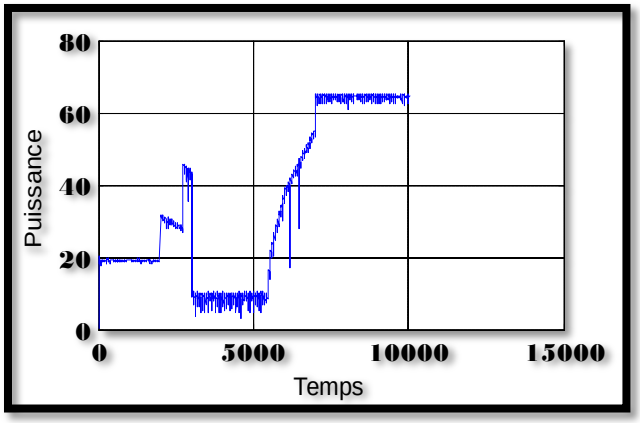


Figure 4.35 La puissance (w) fournie par le GPV

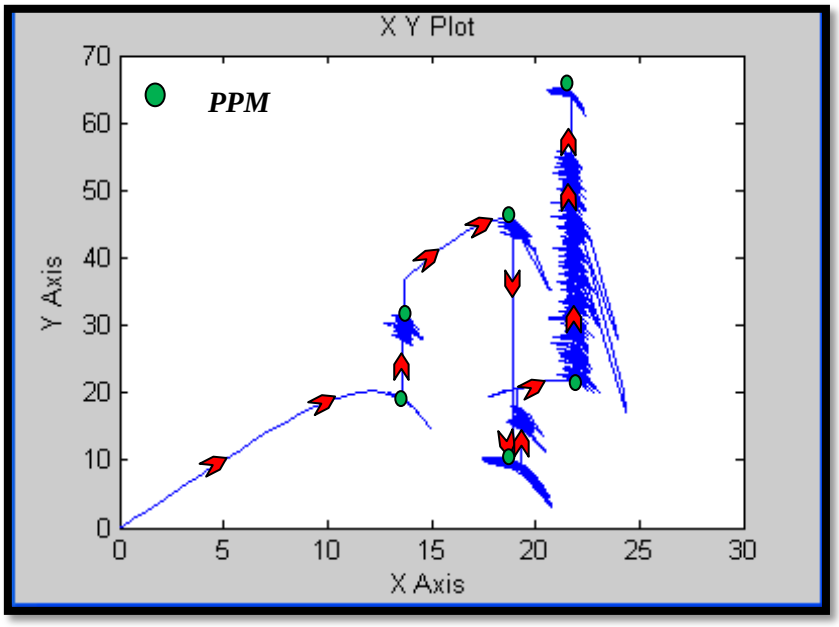


Figure 4.36 : suivi de point de puissance maximale dans des conditions variables de température et d'ensoleillement ; la courbe représente la puissance à la sortie du GPV en fonction de la tension de celui-ci.

4.5 Conclusion :

Dans ce chapitre on a présenté une autre commande intelligente «l'approche neuro-floue» pour la poursuite de point de puissance maximale d'un générateur photovoltaïque qui est une hybridation entre les deux approches neuronale et logique floue.

La simulation faite sous MATLAB nous a montré l'efficacité de notre contrôleur neuro-flou sous différentes conditions de température et d'ensoleillement ; le prochain chapitre sera consacré à l'implémentation de cette architecture neuro-floue sur un circuit FPGA en utilisant le langage VHDL.

Chapitre : 5

Implémentation de la commande neuro-floue sur FPGA

5.1 Introduction :

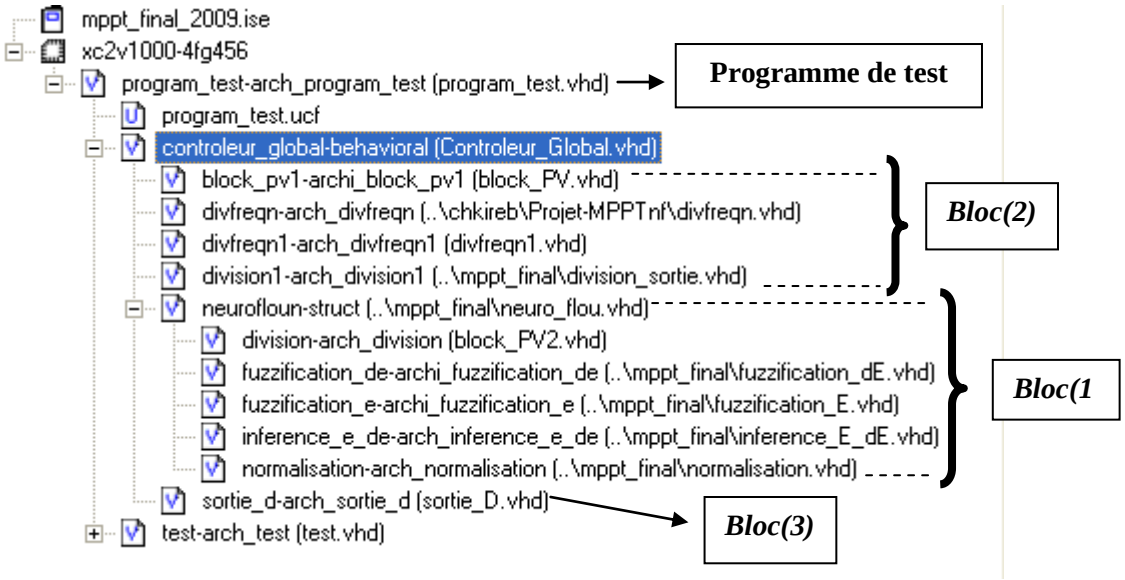
Dans ce chapitre on va présenter et expliquer l'implémentation sur FPGA de la commande neuro-floue en utilisant le langage VHDL .Notre travail est constitué en plusieurs parties (Blocs) ensuite le tout est assemblé pour construire le circuit global.

5-2 Explications et Simulations des constituants du contrôleur neuro-flou :

Le problème est divisé en quatre parties :

1. ***Un bloc(1) qui calcul la variation du rapport cyclique dD en fonction des deux entrées E, dE.***
2. ***Un bloc(2) qui sert à générer le signal de sortie en ayant la valeur du rapport cyclique à son entrée.***
3. ***Un bloc(3) qui calcul les deux entrées E et dE du contrôleur neuro-flou en fonction des deux entrées I (courant à la sortie de panneau) et V (tension au borne du panneau)***
4. ***On a fait un petit programme(4) de test qui contient un vecteur des valeurs du courant et de la tension qu'on va injecter à l'entrée du contrôleur successivement pour s'assurer d'avoir un signal modulé à la sortie.***
5. ***Un autre test qui montre la poursuite de MPP est fait***

Les premières quatre parties sont présentées dans la liste suivante :



5.2.1 Le bloc(1) :

Ce bloc (figure5.1) constitue le cœur de contrôleur ; il est constitué de 05 sous-blocs comme représenté dans la (figure 5.2).

- Bloc pour la fuzzification de l'entrée dE.
- Un autre pour la fuzzification de l'entrée E.
- Un troisième pour l'application des règles d'inférences (ici dans le modèle ANFIS la T-norme est le produit naturel)
- Les deux autres blocs ont pour rôle, de la normalisation et la défuzzification.

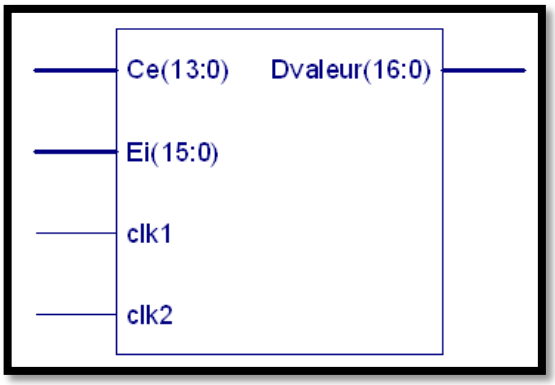


Figure5.1 : Bloc(1)

Ce bloc a 04 entrées et 02 sorties:

- ***Ce*** : a 14 bits (mot signé) reçoit la valeur de dE. Puisque les nombres réels ne sont pas synthétisables, donc on peut travailler qu'avec des nombres entiers, alors on les a multipliées par 1000 afin d'augmenter la précision cette entrée varie dans l'intervalle [-5000,5000].

- ***Ei*** : avec 16 bits (mot signé) reçoit la valeur de E multipliée par 1000 cette entrée varie dans l'intervalle [-32000,10000].
- Une entrée d'horloge (clk1) pour prendre des nouvelles valeurs de E et dE à chaque front montant.
- Une deuxième entrée d'horloge (clk2) très rapide par rapport à la première utilisée pour faire des calculs de division dans le cinquième sous-bloc.

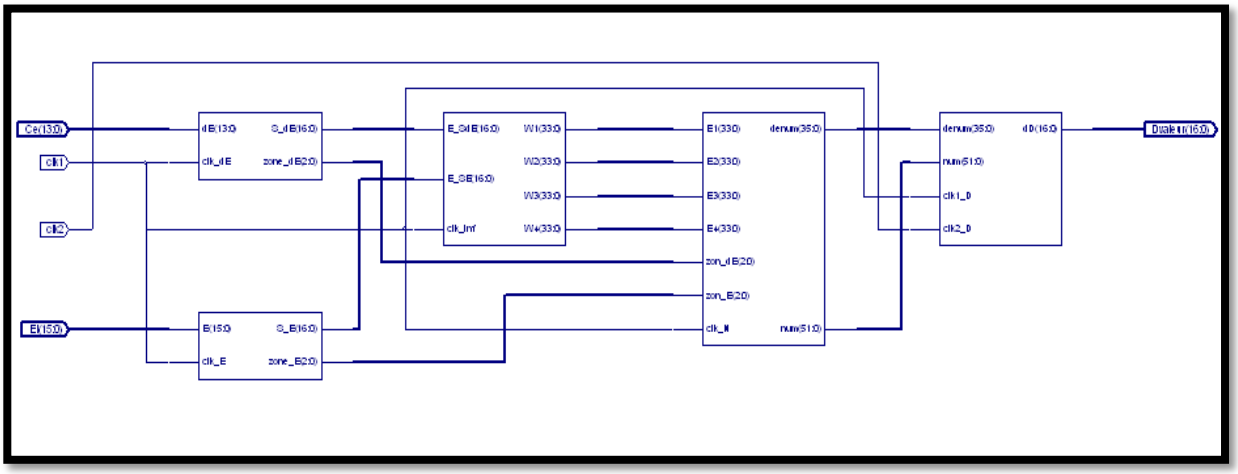


Figure 5.2 : Schéma global des constitutions du bloc(1)

5.2.1.1 Le bloc de fuzzification de l'entrée dE (figure5.3):

La sortie de ce bloc (**S_d**E) est multipliée par 10^5, elle est représentée avec un mot non signé de 17 bits ; son intervalle de variation est [0, 100000].La sortie **Zone_dE** indique la position de la valeur dE dans son univers de discours dans le but de simplifier les calculs ; puisque pour chaque entrée dE, (et comme on a 6 fonctions d'appartenances ,voir le chapitre précédent) on aura seulement deux qui donnent une valeur non nulle pour cette entrée ; et l'une des deux peut être calculée en fonction de l'autre, on aura besoin que d'une seule sortie au lieu de 6.

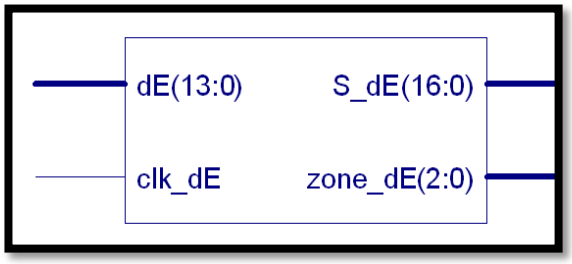


Figure5.3 : bloc de fuzzification de l'entrée dE

Maintenant on fait une petite simulation (figure5.4) en comparant les résultats donnés par ce bloc avec celles de calcul exacte.

Tableau5.1 :

La valeur de dE	La valeur de fuzzification utilisant le programme VHDL.	La valeur de fuzzification exacte.
0.958	1	1
3.970	0.44625	0.44420
-1.710	0.11712	0.11485

On remarque bien que les résultats donnés par la simulation sont très proches des valeurs exactes.

Figure5.4 : Simulation du bloc de fuzzification de l'entrée dE en prenant la valeur 0.958

5.2.1.2 Le bloc de fuzzification de E (figure5.5):

L'explication de ce bloc est comme le précédent.

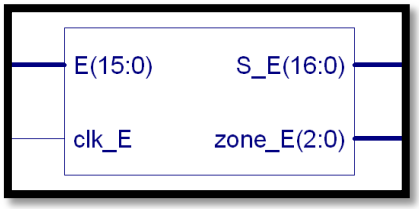


Figure5.5 : bloc de fuzzification de l'entrée E

Tableau5.2 :

La valeur de E	La valeur de défuzzification utilisant le programme VHDL.	La valeur de défuzzification exacte.
-2.673	0.62536	0.59777
-5.220	1.00000	1.00000

Les valeurs données par la simulation sont très proches des valeurs réelles.

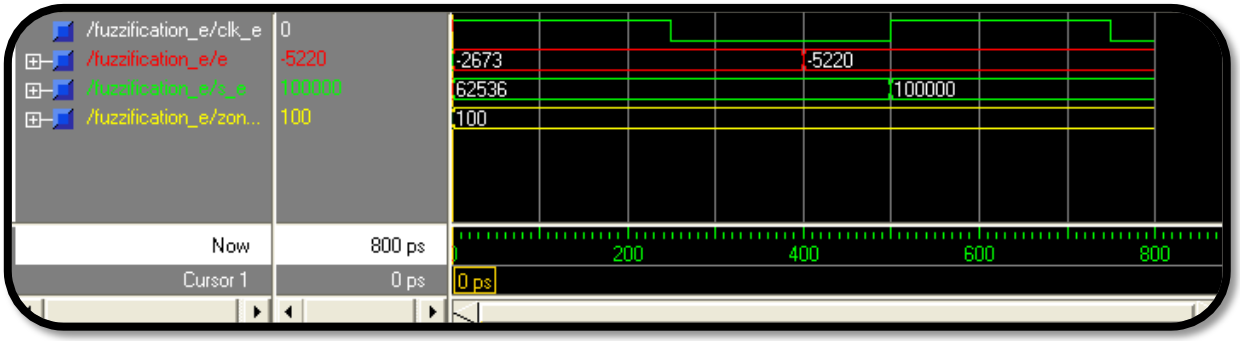


Figure5.6 : Simulation du bloc de fuzzification de l'entrée E en prenant les deux valeurs -2.673 et -5.220.

5.2.1.3 Le bloc des inférences (figure5.7)

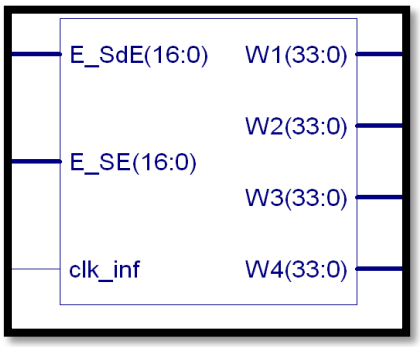


Figure5.7 : Bloc d'inférences

Ce bloc sert à calculer les quatre produits :

- W1= E_SdE * E_SE | W2= E_SE * (100000 - E_SdE)
- W3 = (100000 – E_SE) * E_SdE | W4 = (100000 – E_SdE) * (100000 - E_SE)

  A partir des deux entrées E_SdE et E_SE qui sont au préalable multipliées par $\mathbf{10^5}$; les quatre valeurs Wi sont donc multipliées par $\mathbf{10^{10}}$.

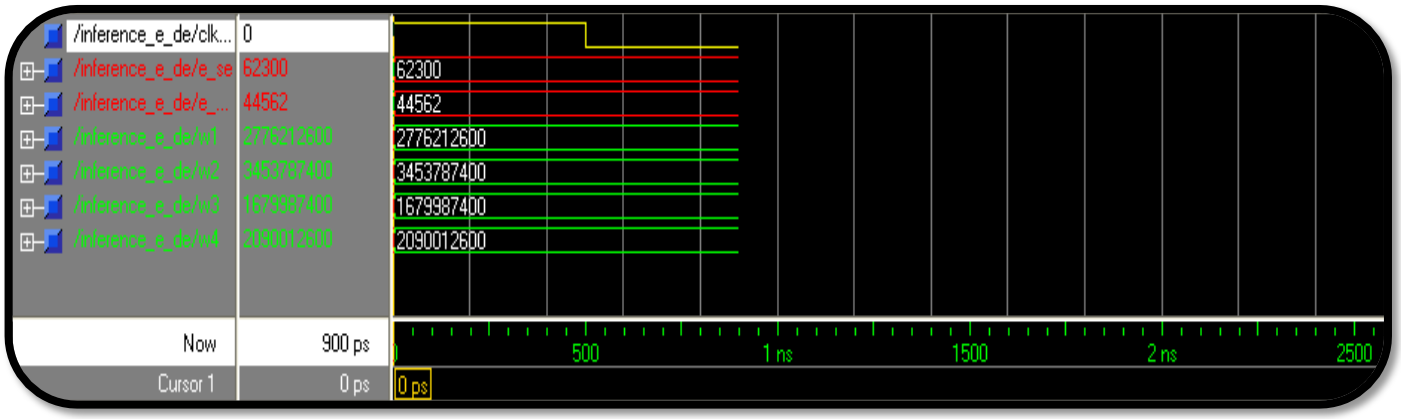


Figure5.8 : Simulation du bloc d'inférences en prenant :

E_SE=62300 (0.623 *10^5) et E_SdE = 44562(0.44562*10^5)

5.2.1.4 la normalisation et la défuzzification :

Après le calcul des quatre Wi (au lieu de 36 puisque les autre sont nulles), on applique la relation suivante pour le calcul de la sortie finale:

$$dD = \frac{W_1 d_1}{\sum_{i=1}^{36} W_i} + \frac{W_2 d_2}{\sum_{i=1}^{36} W_i} + \cdots + \frac{W_{36} d_{36}}{\sum_{i=1}^{36} W_i} \qquad (5-1)$$

Le calcul des termes $\frac{W_j}{\sum_{i=1}^{36} W_i}$ est l'étape de normalisation, et la sommation de tous ces termes multipliés par les d_j est la défuzzification ; pour ne pas faire beaucoup de divisions, on calcul d'abord les deux expressions (figure5.9) :

Num $=W_1 d_1 + W_2 d_2 + \cdots + W_{36} d_{36}$ $(5-2)$

Denum $=\sum_{i=1}^{36} W_i$ $(5-3)$.

Dans notre cas : Num $=W_1 d_1 + W_2 d_2 + W_3 d_3 + W_4 d_4$. Tel que d1, d2, d3, d4 sont choisis parmi les 36 valeurs données dans le tableau 6.1 en sachant les deux zones des deux entrées E et dE. Et Denum=$W_1 + W_2 + W_3 + W_4$.

Ensuite on fait la division Num/Denum.

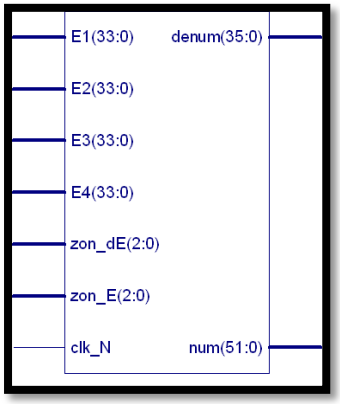


Figure 5.9 Le bloc qui calcule le Num et le Denum à partir de Wi ;i=1…4

voici une petite simulation du bloc de la (figure 5.10) :

Prenant :

E= -2.673 (la valeur de E*1000 en binaire est : 1111010110001111)

dE= -0.154 (la valeur de dE*1000 en binaire est : 11111101100110)

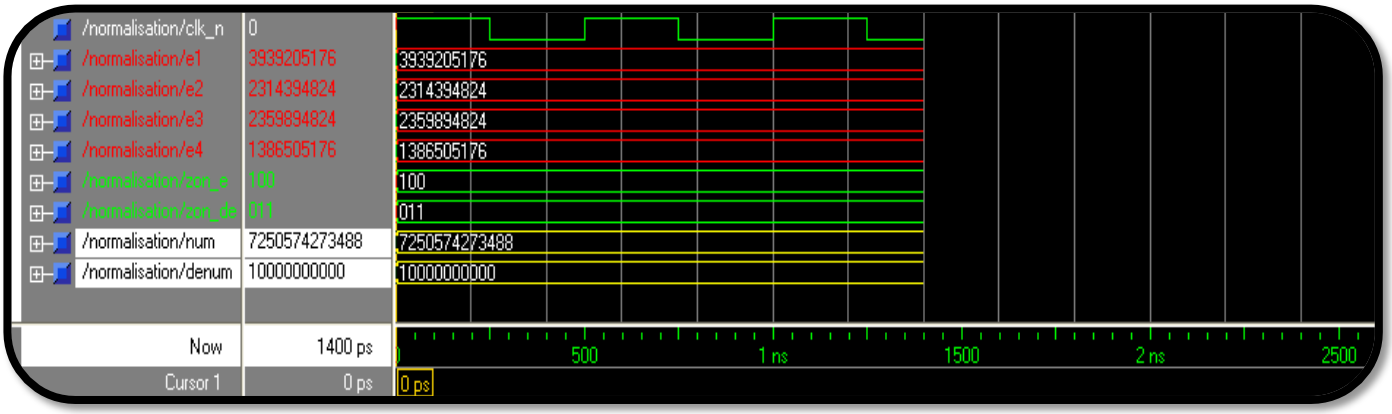
On aura :

W1=E1=3939205176. W2=E2=2314394824. W3=E3=2359894824. W4=E4= 1386505176 et Zone_dE=100, Zone_E=011.

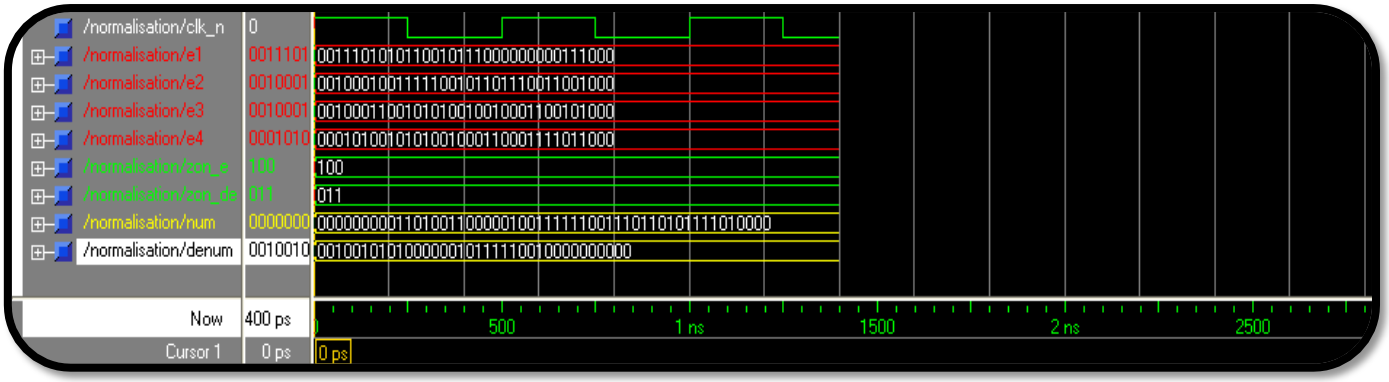
Avec ces deux zones : $d1*10^5$= 1729, $d2*10^5$=1544, $d3*10^5$=-523, $d4*10^5$=-1370.

Alors : Num = 7250574273488/ Denum=10000000000.

Le calcul exacte est bien confirmé par les deux figures suivantes :



Format décimale



Format binaire

Figure5.10 : Simulation du bloc qui calcule le Num et le Denum à partir de Wi ; i=1…4

En fin ; la division Num/Denum sera faite par le bloc de la (figure5.11). à la sortie du bloc précédent on aura la valeur de Num multipliée par 10^{10} par rapport à sa valeur réelle, aussi, la valeur de Denum sera multipliée par 10^5 par rapport à sa valeur réelle, (la multiplication de Num par 10^{10} devant tandis que Denum est multipliée juste par 10^5 nous aidera à faire la division avec la précision voulue) . cette divisionest faite par soustractions successives en utilisant utilisant une horloge très rapide (clk2_D) par rapport à celle qui est utilisée pour la synchronisation (clk1_D) ; le résultat final sera bien sûr multiplié par 10^5.

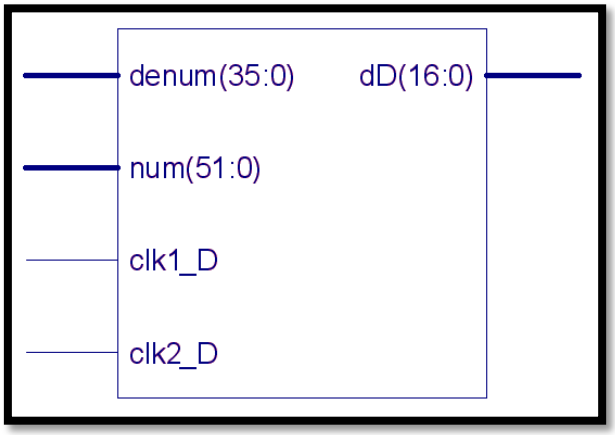


Figure 5.11 Le bloc qui calcule la valeur de dD multipliée par 10^5

Prenant : Num = 7250574273488/ Denum=10000000000.

On aura : dD=0.00725 * 10^5. Ce résultat est confirmé par la simulation suivante :

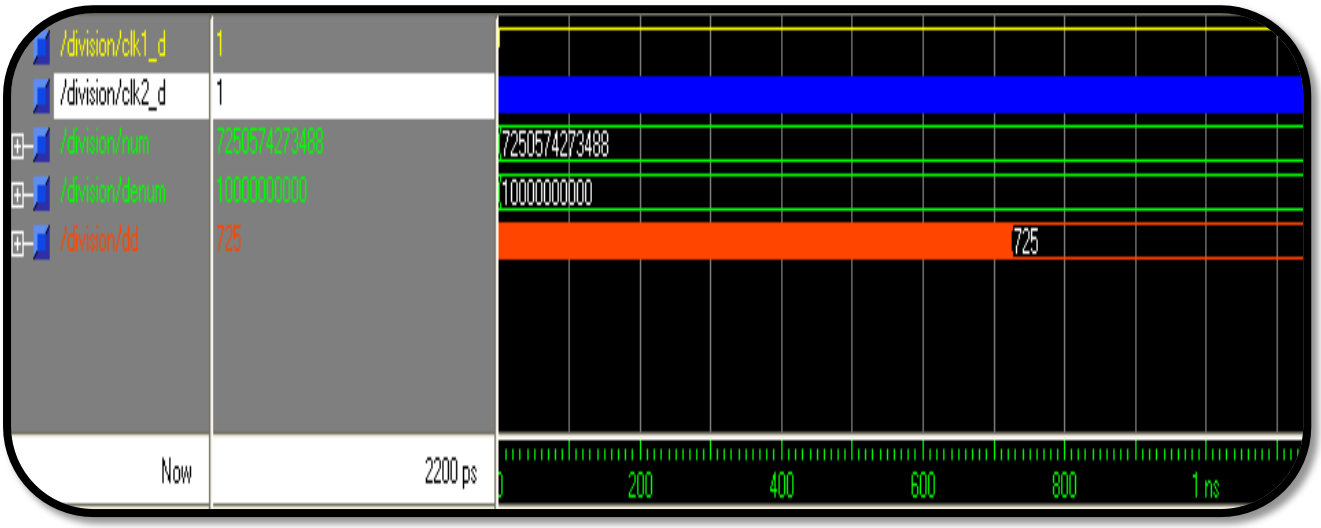


Figure5.12 : Simulation du bloc de division

5.2.1.5 Simulation globale du Bloc(1) :

Dans cette section on va comparer les résultats donnés par le programme VHDL avec ceux de MATLAB (voir figure5.13).

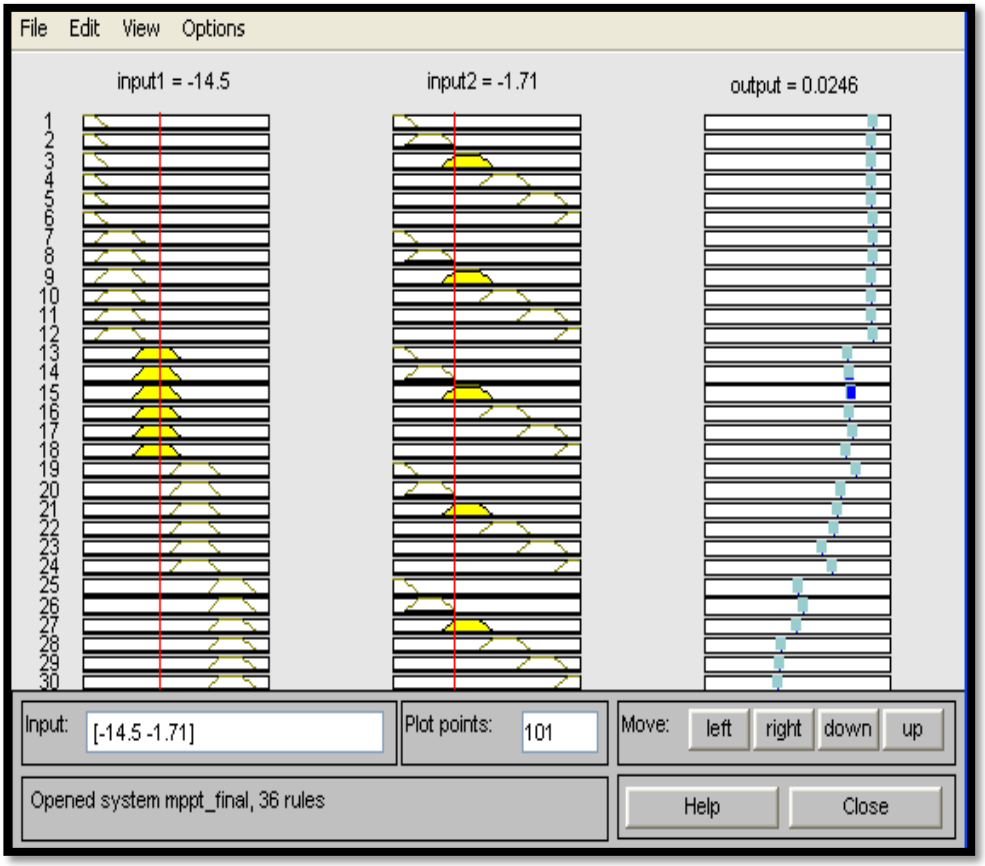


Figure5.13 : La boite de visualisation des règles sous MATLB

Le tableau suivant (5.1), présente quelques comparaisons:

	dD avec VHDL	**dD avec MATLAB**
E=-2.673 dE=-0.1549	0.00725	0.00674
E=-5.960 dE=0.958	0.01729	0.01540
E=-8.300 dE=3.97	0.01193	0.01200
E=-14.5 dE=-1.71	0.02458	0.02460

La dernière essaie (E=-14.5 et dE=-1.71) est présentée avec les deux approches (voir les deux figures5.13 /5.14)

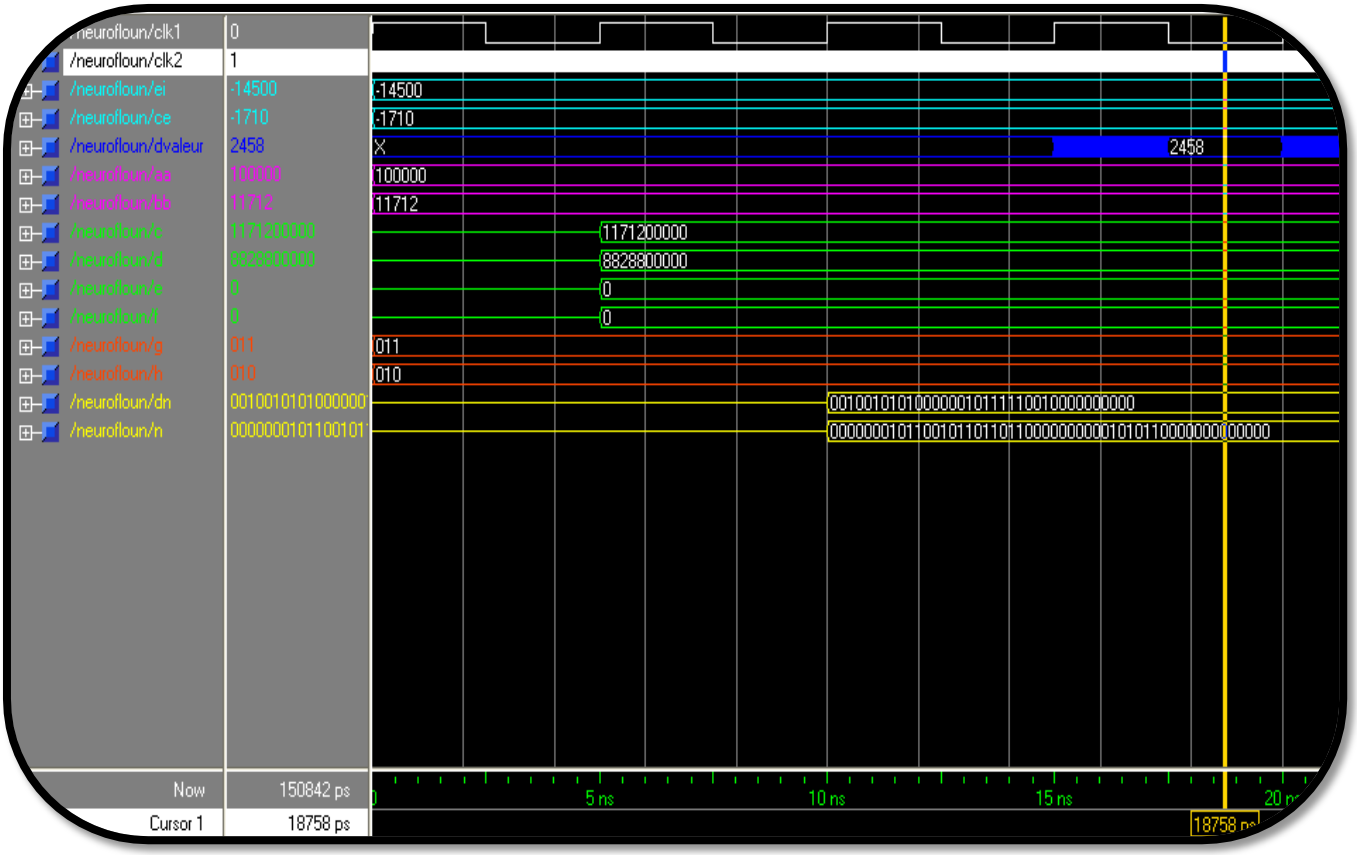


Figure5.14 Simulation de fonctionnement global du bloc(1)

Remarque5.1 :

On voit bien d'après la simulation ci- dessus que le résultat pour de nouvelles entrées ne sera visualisé qu'après quatre périodes. Donc on doit prendre cette remarque en considération lorsqu'on voudra faire l'acquisition de nouvelles valeurs E et dE.

5.2.2 Le bloc (2) :

Ce bloc (figure5.15) ajoute la valeur dD produite par le bloc(1) à la valeur de D actuelle et génère un signal carré modulé en largeur, (voir figure5.16) ; si l'entrée reset est au niveau 1, la valeur de D est toujours égale à D_entre.

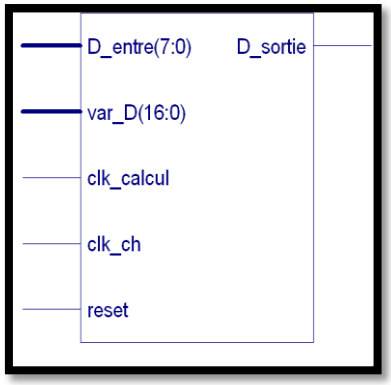


Figure5.15 Le bloc(2)

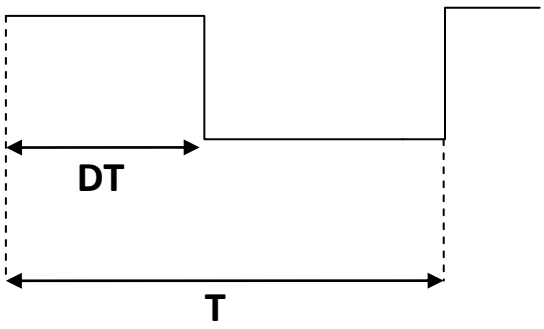


Figure5.16 le format de signal généré par le bloc(2)

Une simulation est faite pour ce bloc (voir figure 5.17) : initialement on prend D = 0.5 avec un changement dD = 0.08191 et on met reset à '1' on remarque que la valeur de D ne change pas tant que reset = '1' ; lorsque reset devient '0' on voit bien qu'après chaque front montant de clk_ch la valeur de D augmente de 0.08191.

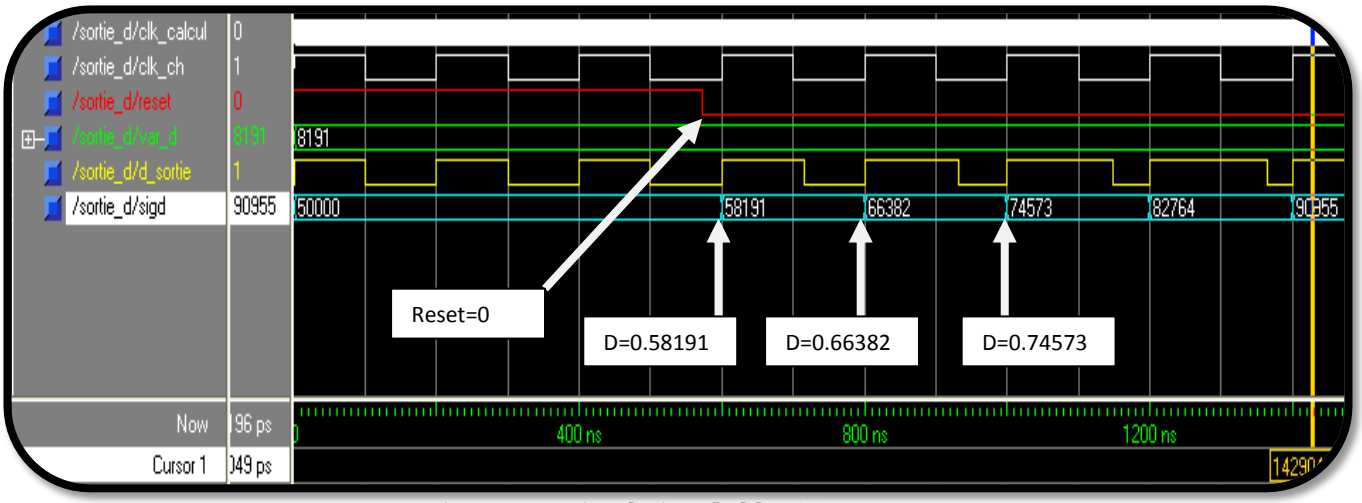


Figure5.17 Simulation de bloc(2)

5.2.3 Le bloc(3) :

Ce bloc est divisé en deux parties :

- La première (figure5.18) sert à calculer dP=P(k)-P (k-1) et dV=V(k)-V (k-1)
- La deuxième (figure5.19) sert à calculer E et dE.

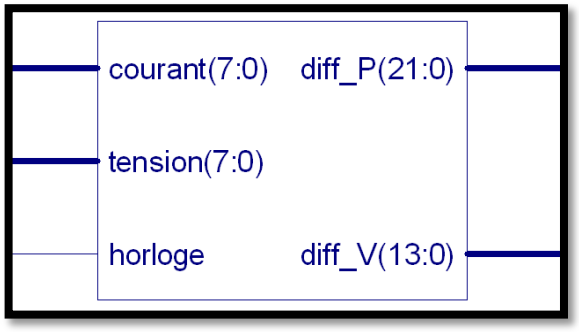


Figure5.18 la première partie du bloc(3)

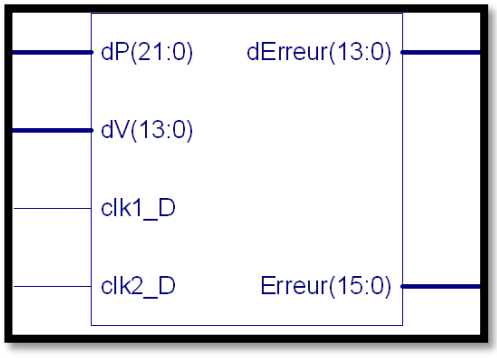


Figure5.19 la deuxième partie du bloc(3)

Les deux entrées « courant » et « tension » sont deux mots de 8bits, la valeur maximale correspond à la valeur maximale du courant ou de la tension mesurée.

voici une simulation pour tester le bon fonctionnement de ces deux blocs :

On prend quatre couples (courant(A), tension(V)) :

(3.1, 40) = (10011011, 11001000).

(3, 44) = (10010110, 11011100).

(2.9, 46) = (10010001, 11100110).

(2.8, 48) = (10001100, 11110000).

P1 =124w, P2=132w, P3=133.4, P4=134.4.

(dP1=8w, dV1=4v), (dP2=1.4w, dV2=2v), (dP3=1w, dV3=2v)

La simulation présentée à la (figure5.20) montre que notre programme donne les mêmes résultats trouvés par le calcul exacte. La même chose pour la (figure5.21) avec les valeurs de Ei et dEi.

E1=dP1/dV1=2 ; E2=dP2/dV2=0.7 ; E3=dP3/dV3=0.5.

dE1=E1-0=E1=2 ; dE2=E2-E1=-1.3 ; dE3=E3-E2=-0.2.

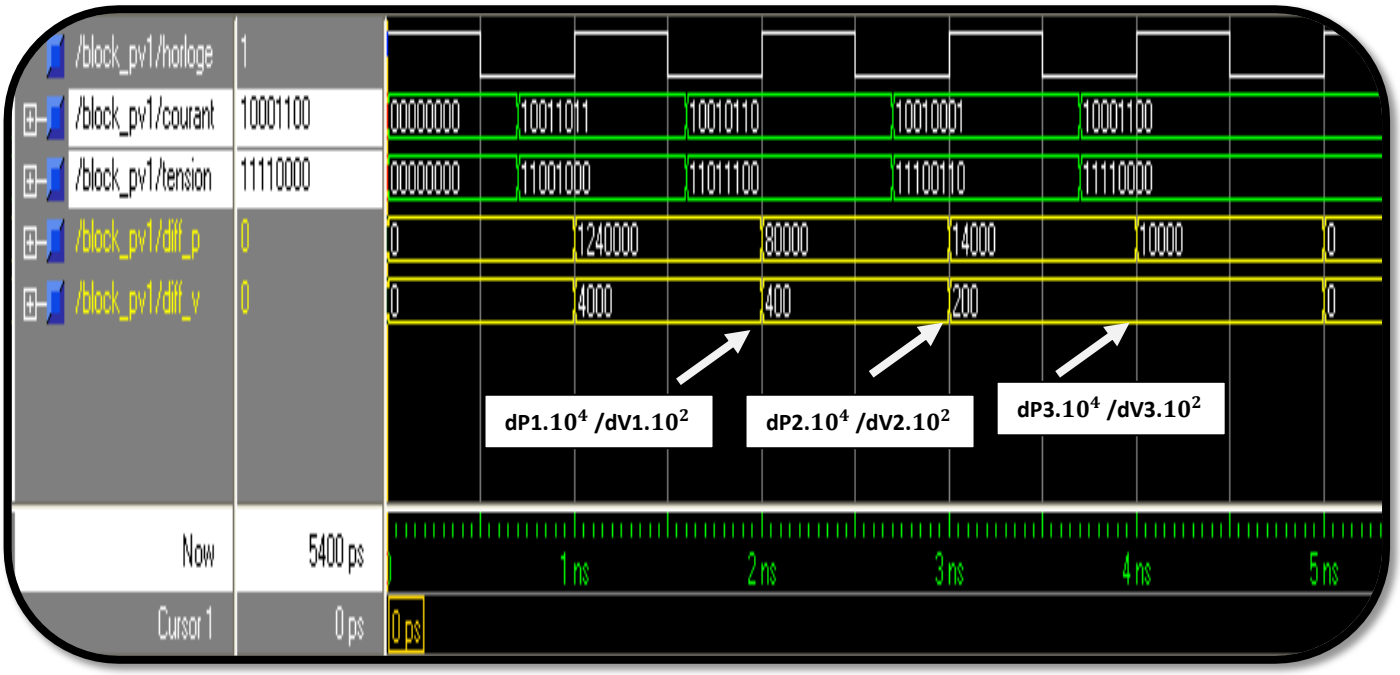


Figure5.20 Simulation de la première partie du bloc (3)

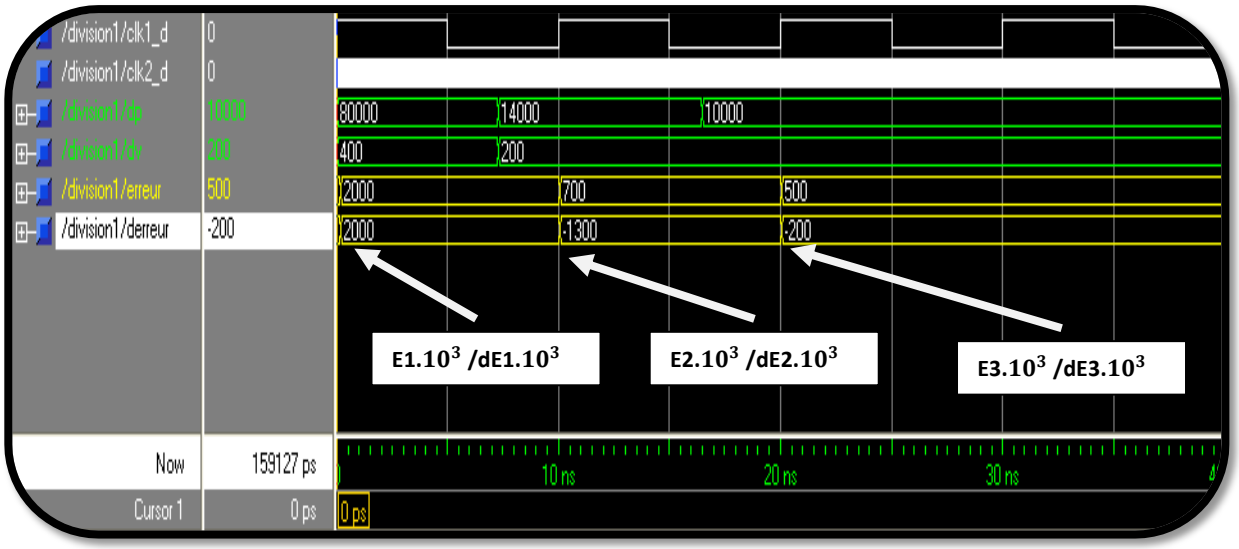


Figure5.21 Simulation de la deuxième partie du bloc (3)

5.2.4 l'assemblage des trois blocs(1),(2),(3) :

Ayant les trois blocs présentés précédemment, un calcul E et dE à partir de I et V ; le second est une l'architecture neuro-floue qui sert à calculer dD en ayant E et dE à son entrée, et le troisième génère le signal modulé pour la commande de l'hacheur ; le tout sera notre contrôleur complet (voir figure5.22/5.23).

Avec :

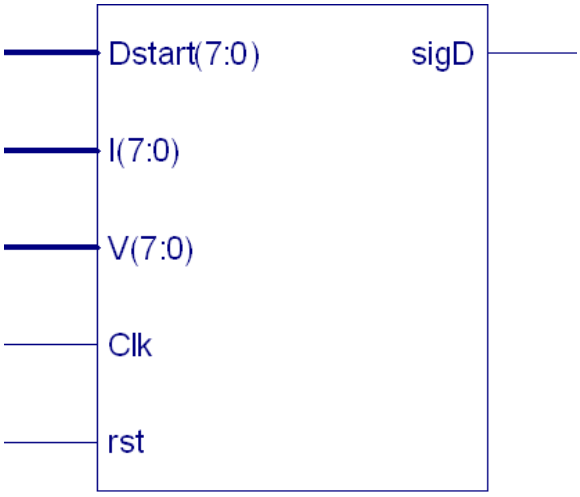


Figure5.22 Le contrôleur global

Dstart : représente la valeur initiale du rapport cyclique.

I : représente l'entrée de la valeur de courant mesurée.

V : représente l'entrée de la valeur de tension mesurée.

Clk : représente l'horloge du système :on a utilisé une horloge avec une fréquence de 24MHz.

Rst : initialisation du système en travaillant avec la valeur Dstart.

SigD : c'est le signal de commande généré par le contrôleur.

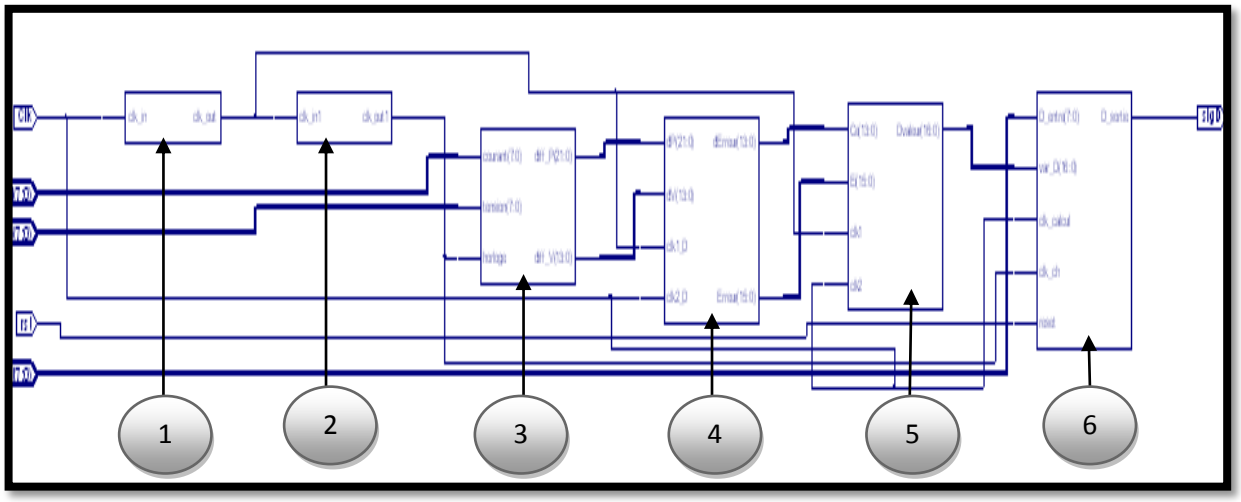


Figure 5.23 : Schéma des constitutions du contrôleur global

Explication des six parties de la (figure5.23) :

1. Représente un premier diviseur de fréquence qui divise la fréquence du système par 10^4.

2. Représente un deuxième diviseur de fréquence qui divise la fréquence du signal sortant de la partie 1 par 10.

3. Il sert à calculer dP et dV à partir de I, V ; il est synchronisé par un signal de 1000Hz.

4. son rôle est le calcul de E et dE ; il est synchronisé par un signal de fréquence 10kHz, cette fréquence est suffisamment grande (10×1kHz) pour pouvoir faire tous les calculs pour avoir une nouvelle valeur de D (sortie de la partie 5) correspondant aux valeurs actuelles de I et V.

5. Cette partie calcule la valeur du rapport cyclique nécessaire en fonction de E et dE, il est synchronisé par un signal de 10 kHz, comme pour le bloc précédent.

6 Ce dernier bloc génère le signal de commande de l'hacheur avec une fréquence de 1000Hz

5.2.5 Simulation de la poursuite du MPP par le contrôleur neuro-flou :

Afin de tester notre programme, nous avons écrit un programme de simulation en réalisant une boucle fermé entre le couple (tension, courant) et le rapport cyclique D, explicitement, chaque couple (i, v) nous donne un D, et le D nous donnera de nouvelles valeurs de (i, v) et ainsi de suite ; les résultats de cette simulation sont sur la figure suivante :

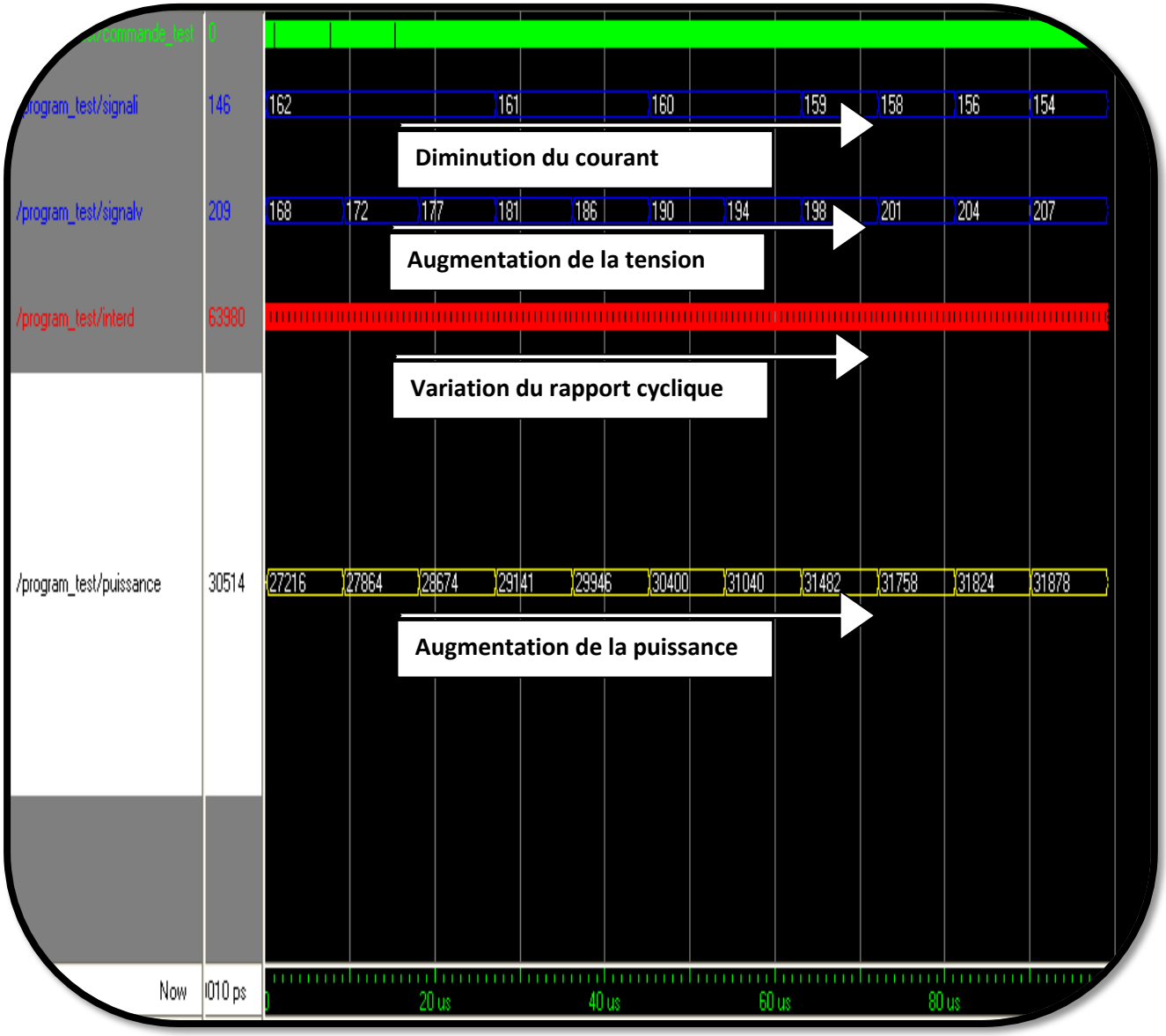


Figure 5.24 : Simulation de la poursuite de la MPP par le réseau neuro_flou

Nous avons tracé l'allure de l'évolution de la puissance au cours du temps pour avoir une vision plus claire.

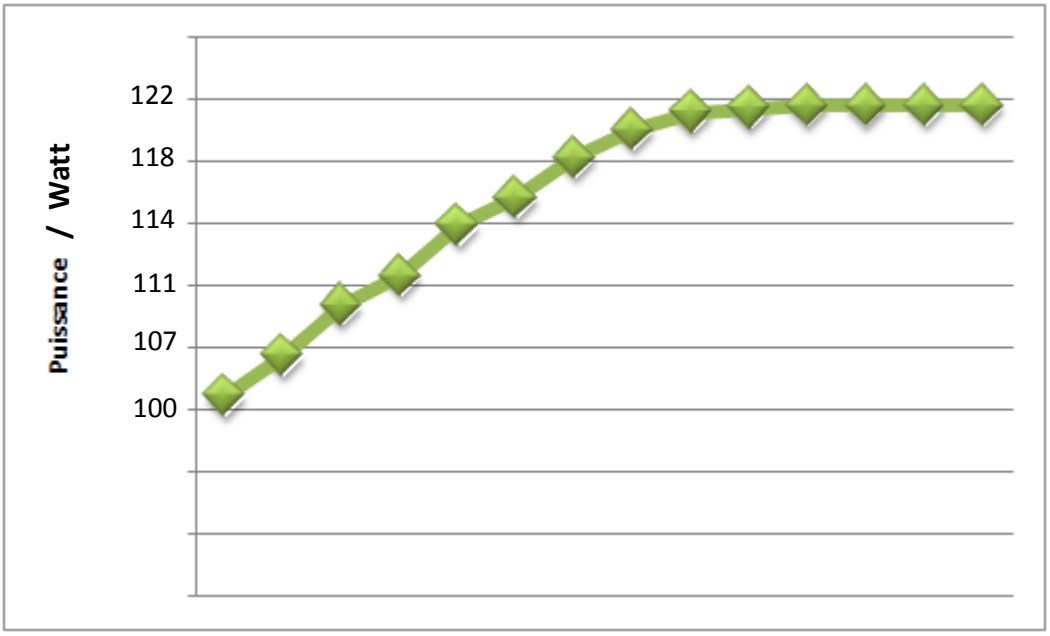


Figure 5.25 Simulation de la poursuite de la MPP par le réseau neuro_flou

5.2.6 Commande d'un hacheur série par le contrôleur neuro-flou pour alimenter une charge résistive :

Dans la dernière partie de notre travail, on a essayé de commander un convertisseur DC-DC pour alimenter une charge résistive, ce qui connu par l'étage de puissance, pour voir la réalité des choses et sortir du cadre de la simulation.

O n a utilisé pour cela un hacheur Buck, avec la topologie suivante :

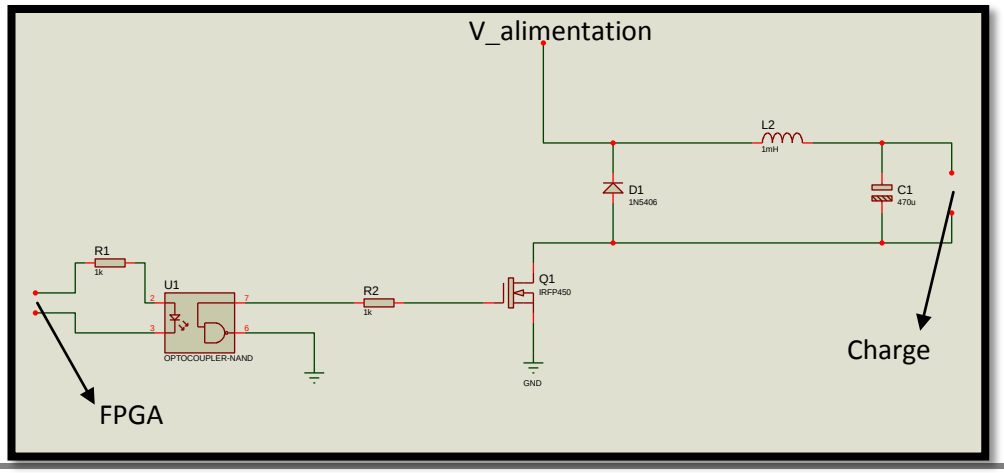


Figure 5.26 Convertisseur DC-DC

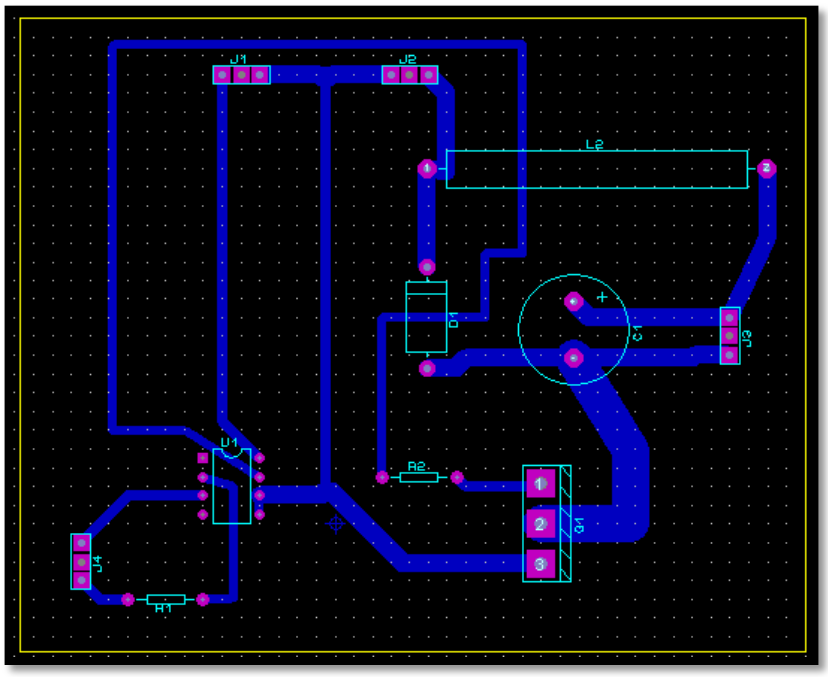


Figure 5.27Ciricuit imprimé du Convertisseur DC-DC

La sortie de la carte FPGA est de 3.3 volts, il faut la ramener à un niveau acceptable pour pouvoir attaquer la grille du **MOSFET** avec une tension entre 12 et 20 volts et aussi isoler l'hacheur de la carte pour éviter tout retour de courant qui pourra endommager l'endommager, l'optocoupleur peut réaliser ces deux fonctions, il donnera une tension de sortie avec la même valeur que sa tension d'alimentation.

En fin, un filtre RC pour diminuer l'ondulation du courant en éliminant les harmoniques, car l'application la plus commune consiste à faire la charge des batteries, bien entendu avec un courant continu, la fréquence du filtre RC se calcule comme suit :

$$F = \frac{1}{2\pi\sqrt{LC}} \qquad (5-5)$$

Pour notre application, on a essayé avec une simple résistance pour commander l'ouverture et la fermeture du MOSFET, le signal de sortie sera évidement de la même forme que le signal d'entrée mais d'amplitude est différente, les deux signaux prit en photos sont dans la (figure 5.30) :

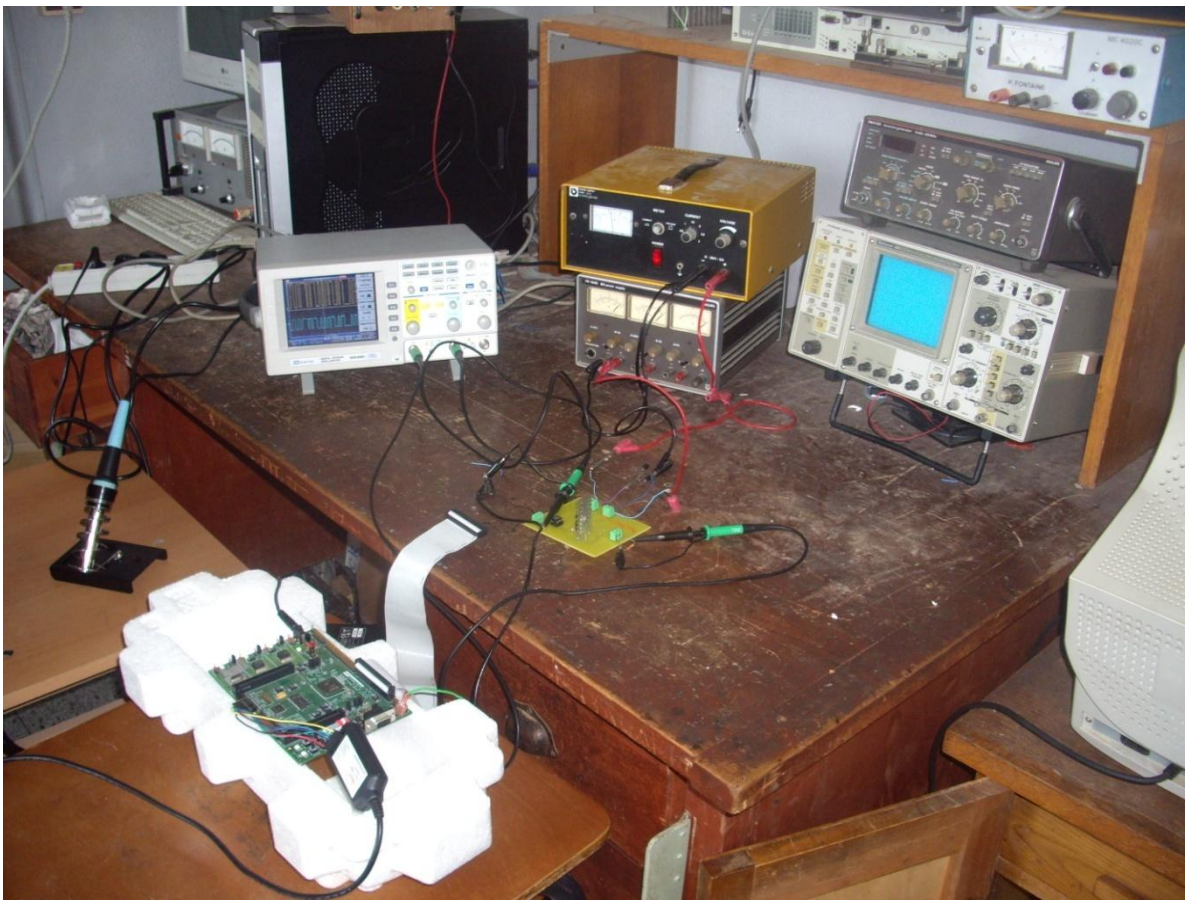


Figure 5.28 Vue générale d'environnement de test

Un programme injecté dans la carte contenant un vecteur courant et un vecteur tension permet La génération du signal de sortie; le rapport cyclique (figure 5.29).

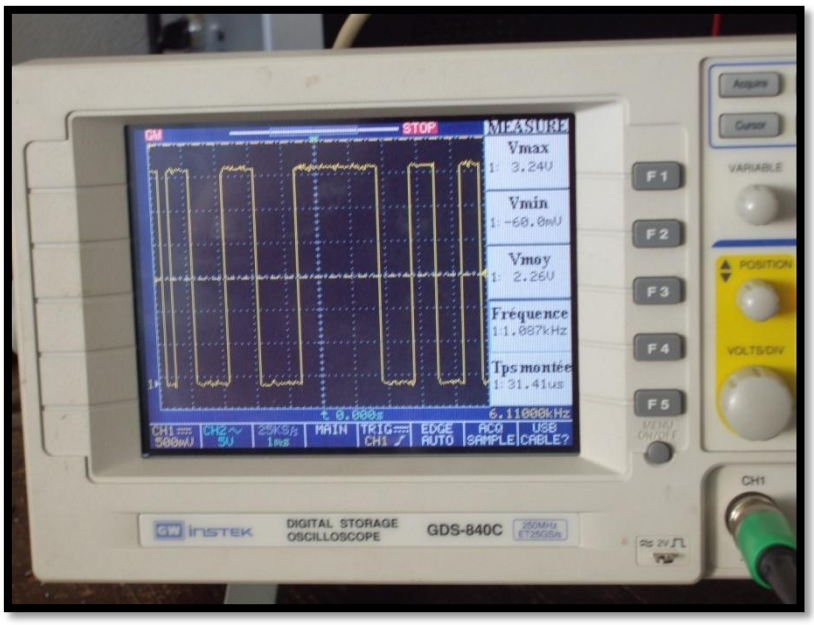


Figure5.29 Signal de sortie de la carte FPGA (rapport cyclique variable)

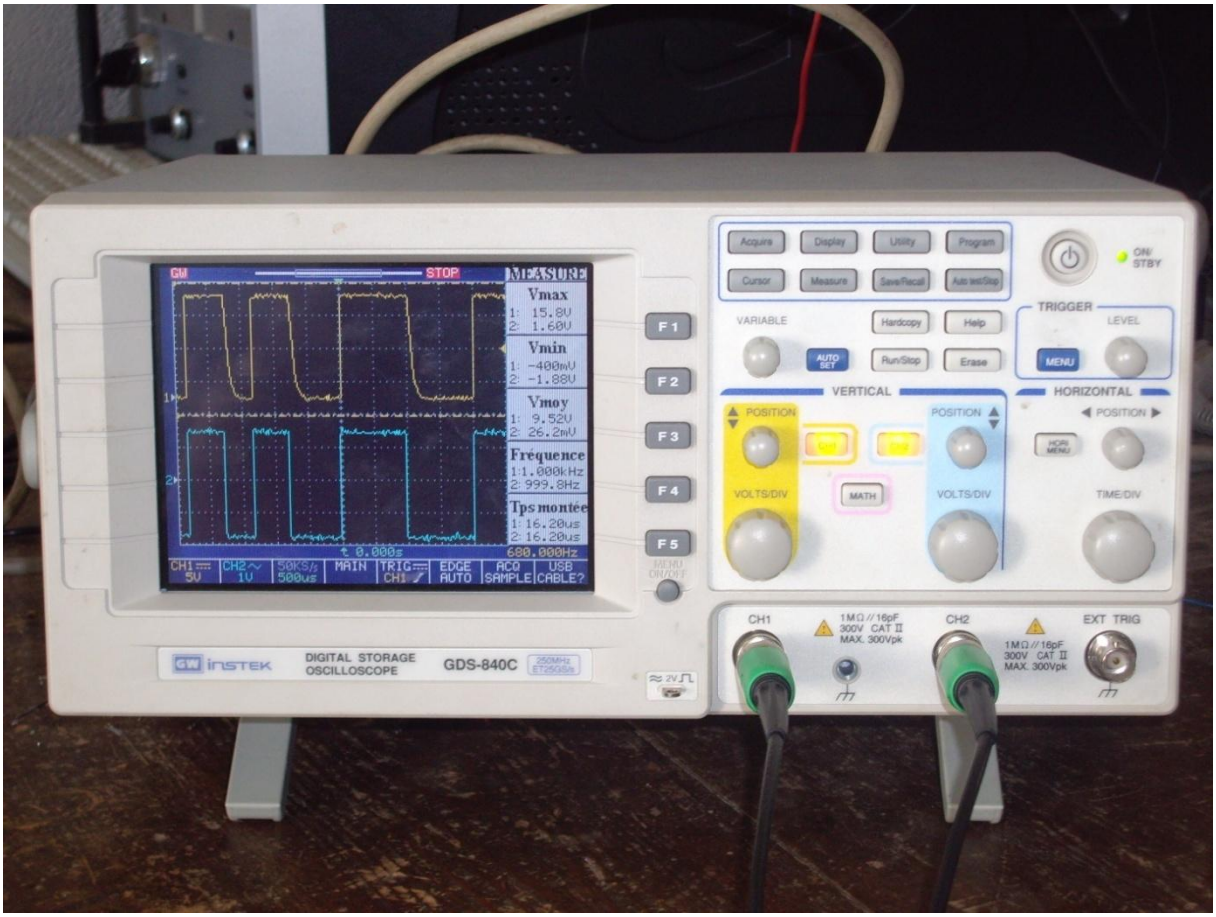


Figure 5.30 Signal de commande (en Bas) et signal au borne de la charge (en haut)

5.3 Conclusion:

Tous les exemples qui ont été fait dans cette partie en tant que simulation sont juste pour tester le bon fonctionnement des programme, sinon, une validation plus poussée doit se faire avec des modules photovoltaïques et pour cela il reste un peut de travail pour faire l'étage de conversion analogique-numérique des tensions et courants prélevés du générateur photovoltaïque et à ce moment là le test sera beaucoup plus efficace.

Notre PFE constituera peut être une aide pour les étudiants qui s'intéressent à l'énergie solaire en utilisant la commande neuro-floue et veulent aller jusqu'à l'aboutissement du projet et avoir des résultats concrets.

Conclusion générale

Les énergies renouvelables constituent une excellente alternative aux énergies fossiles et nucléaires vue leurs différents avantages : inépuisables, non polluantes, et décentralisées.

Parmi ces énergies renouvelables, l'énergie solaire photovoltaïque connait un intérêt grandissant ces dernières années.

Un générateur photovoltaïque présente une caractéristique tension-puissance en cloche qui dépend de plusieurs paramètres extérieurs comme l'ensoleillement, la température, la position du soleil. En d'autres termes, il existe un seule point de fonctionnement pour extraire la puissance maximale du générateur et ce point n'est pas fixe. Il change avec les conditions extérieures, donc, plus on est loin de lui, plus la puissance délivrée par le module photovoltaïque est faible et plus le rendement de notre installation est mauvais.

Pour faire face à ce problème les ingénieurs on mit au point des procédés pour garder le point de fonctionnement l'installation sur le point de puissance maximale quelque soient les conditions météorologiques. Ces méthodes de poursuite s'appellent « maximum power point tracking » ou MPPT.

Le choix optimal d'une méthode de poursuite passe par le cahier des charges, car chaque méthode présente des avantages et des inconvénients.

Dans ce mémoire nous nous sommes intéressés à une commande MPPT neuro-flou. Elle combine logique floue et réseaux de neurones pour faire la poursuite de point de puissance maximale, en profitant de la souplesse de la logique floue et de la capacité d'apprentissage de réseaux de neurones.

Cette commande MPPT neuro-floue a été d'abord validée par simulations sous Matlab où de bons résultats ont été obtenues.

Nous avons ensuite implémenté cette commande sur circuit FPGA en utilisant la carte de développement ***Virtex-II V2MB1000 de Memec.***

La simulation avec Modelsim a montré le bon fonctionnement de cette commande.

Enfin, nous avons testé cette commande avec un hacheur dévolteur. De bons résultats ont été obtenus.

Bibliographie :

[1] F.Chekired , étude et implémentation d'une commande MPPT neuro-flou sur un circuit FPGA, mémoire de magister ENP, 2008.

[2] T.Obeidi , Application des algorithmes génétiques dans la commande des hacheurs MPPT, mémoire de magister ENP, 2006.

[3] H.J. Möller, *Semiconductors for Solar Cells*, Artech House, Inc, Norwood, MA, 1993.

[4] A.F. Boehinger. *Self Adaptive DC Converter for Solar Spacecraft Power Supply*. IEEE Transaction on Aerospace and Electronic Systems, AES-4, n°1, pp 102-111, 1968.

[5] H. Knopf, *Analysis, Simulation, and Evaluation of Maximum Power Point Tracking (MPPT) Methods for a Solar Powered Vehicle*, Master of Science in Electrical and Computer Engineering, Portland State University, 1999.

[6] S.Saadi Commande d'une poursuite du point de puissance maximum (MPPT) par les Réseaux de Neurones, mémoire de magister ENP, 2006.

[7] Hung T. Nguyen • Nadipuram R. Prasad Carol L. Walker • Elbert A. Walker. A first course in fuzzy and neural control , CHAPMAN & HALL/CRC, 2003.

[8] *Volnei A. Pedroni* , *Circuit Design with VHDL* , MIT Press, Cambridge, Massachusetts, London, England,2004.

ANNEXE A

SIMULINK MATLAB

A .1 Introduction :

MATLAB est une abréviation de MATrix LABoratory. Ecrit à l'origine, en Fortran, par Cleve Moler, MATLAB était destiné à faciliter l'accès au logiciel matriciel. La version actuelle, écrite en C par The Math Works Inc., existe en version "professionnelle" et en version "étudiant". Sa disponibilité est assurée sur plusieurs plates-formes : Sun, Bull, HP, IBM, compatibles PC, Macintosh, et plusieurs machines parallèles.

MATLAB est conforté par une multitude de boîtes à outils (toolboxes) spécifiques dans des domaines variés. Un autre atout de MATLAB, est sa portabilité; la même portion de code peut être utilisée sur différentes plates-formes sans la moindre de modification. En complément de MATLAB, l'outil additionnel SIMULINK est proposé pour la modélisation et la simulation de systèmes dynamiques en utilisant une représentation de type schémas-blocs.
L'environnement MATLAB se présente sous la forme d'un espace de travail (Workspace)

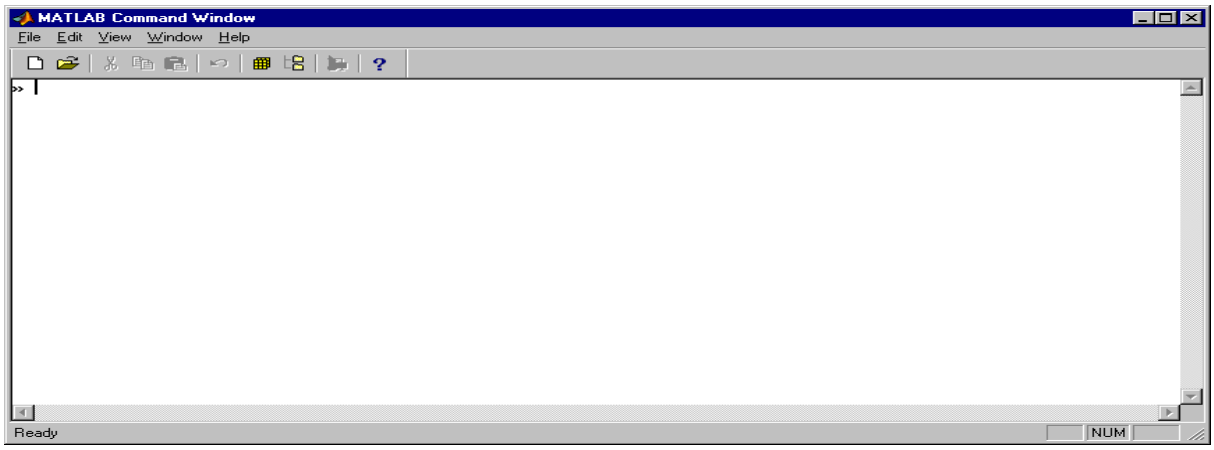


Figure A.1 espace du travail de Matlab

Où un interpréteur de commande exécute des opérations et fonctions MATLAB. Les sources de celles-ci sont disponibles, écrites en "langage" MATLAB, voir en C ou en Fortran.

L'utilisateur peut à sa guise les modifier, mais en s'en inspirant, il peut surtout créer et rajouter ses propres fonctions.

MATLAB offre également plusieurs fonctions destinées à la résolution (numérique) d'équations différentielles linéaires ou non-linéaires par la méthode de Runge-Kutta (ode23 et ode45), l'intégration numérique, la recherche des solutions d'équations algébriques ou transcendantes, la création et manipulation de polynômes (poly, polyder, polyval, conv,

deconv), la transformée de Fourier rapide (ffr, fft2, ifft). Des fonctions propres au traitement de données, comme min, max, mean, cumsum, sort, std, diff, ainsi que celles relatives à l'interpolation (polyfit, interp1) sont autant d'outils très pratiques pour l'ingénieur analysant un problème. L'interface graphique de MATLAB est sans conteste l'un des points forts du logiciel et facilite le tracé de courbes et l'obtention de graphiques 2D ou 3D de grande qualité. Le "langage" MATLAB contient un minimum de structures de programmation (structure itérative, structure conditionnelle, sous-routine) mais reste très rudimentaire.

L'avantage est qu'il est très simple et très rapide à programmer, offrant une grande tolérance (syntaxe simple, pas de définition de types, etc), ce qui permet un gain appréciable en temps de mise au point. L'ingénieur peut par ce moyen être plus efficace dans l'analyse d'un problème, en concentrant ses efforts sur celui-ci et non pas sur l'outil servant à le résoudre.

Les boîtes à outils (toolbox) dédiées à des domaines techniques spécifiques, sont :

- Le traitement du signal
- La régulation automatique
- L'identification
- Les réseaux de neurones
- La logique floue
- Le calcul symbolique

Et bien d'autres encore. Ces boîtes à outils sont simplement constituées d'un ensemble de fonctions spécialisées programmées à partir des fonctions de base de MATLAB, permettant par exemple la synthèse de filtres, le calcul de FFTs, la simulation d'algorithmes flous ou encore le calcul de réponse harmoniques.

Il existe deux modes de fonctionnement:

- Mode interactif: MATLAB exécute les instructions au fur et à mesure qu'elles sont données par l'usager.
- Mode exécutif: MATLAB exécute ligne par ligne un "fichier M" (programme en langage MATLAB).

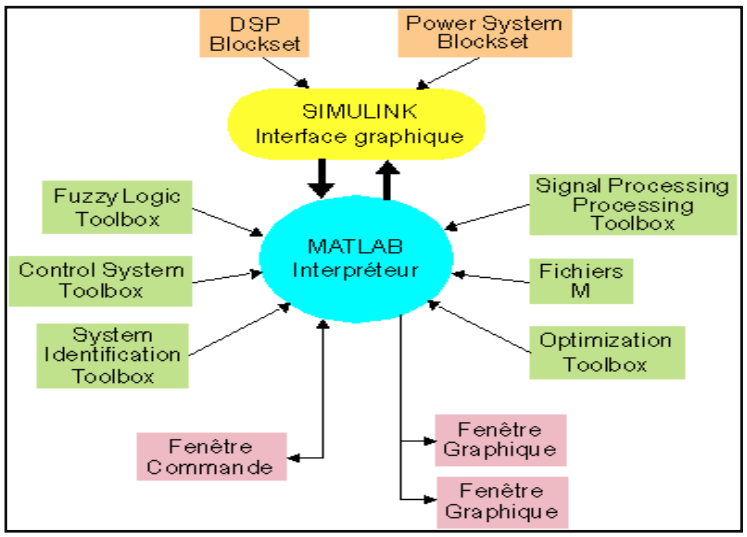


Figure A.2 Environnement du Matlab

- ***Fenêtre de Commande*:** Dans cette fenêtre, l'usager donne les instructions et MATLAB retourne les résultats.
- ***Fenêtres Graphique*:** MATLAB trace les graphiques dans ces fenêtres.
- ***Fichiers M*:** Ce sont des programmes en langage MATLAB (écrits par l'usager).
- ***Toolboxes*:** Ce sont des collections de fichiers M développés pour des domaines d'application spécifiques (Signal Processing Toolbox, System Identification Toolbox, Control System Toolbox, u-Synthesis and Analysis Toolbox, Robust Control Toolbox, Optimization Toolbox, Neural Network Toolbox, Spline Toolbox, Chemometrics Toolbox, Fuzzy Logic Toolbox, etc.)
- **Simulink :** n'est rien d'autre qu'une boîte à outils de MATLAB permettant au moyen d'une interface graphique évoluée la construction rapide et aisée ainsi que la simulation de schémas fonctionnels complexes, contenant des systèmes linéaires, non linéaires voire non-stationnaires, y compris des opérateurs logiques, des outils mathématiques d'analyse, etc.

Incontestablement, MATLAB est un formidable outil pour l'ingénieur, y compris pour celui traitant des problèmes pratiques. Avec sa boîte à outils Simulink, il est maintenant une référence au niveau mondial, non seulement dans les universités et instituts de recherche, mais aussi dans le milieu industriel.

Blocksets ce sont des collections de blocs Simulink développés pour des domaines d'application spécifiques (DSP Blockset, Power System Blockset, etc.). Ces blocksets appelé Library comme montre la (figure A.3).

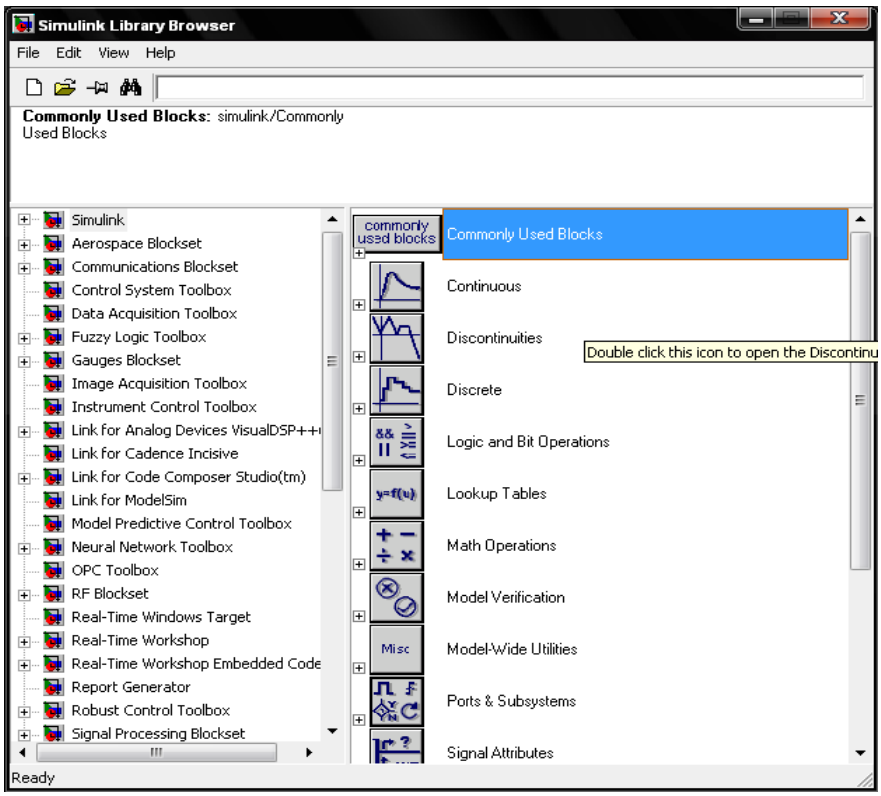


Figure A.3 Les différents Library dans Simulink

Chaque Library contient des différents blocks, si on prend la bibliothèque Simulink comme indique la (figure A.4).

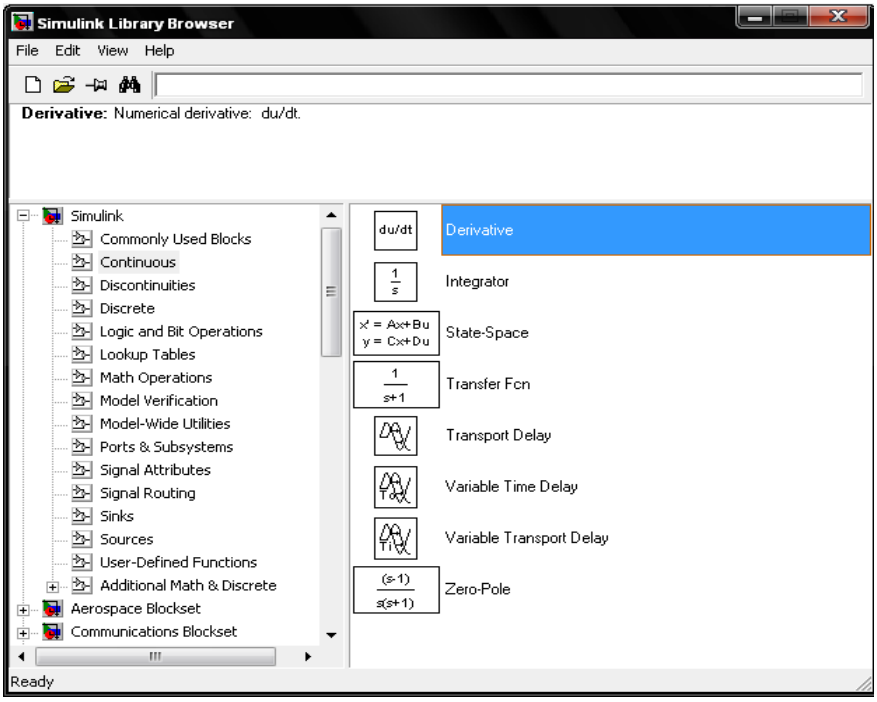
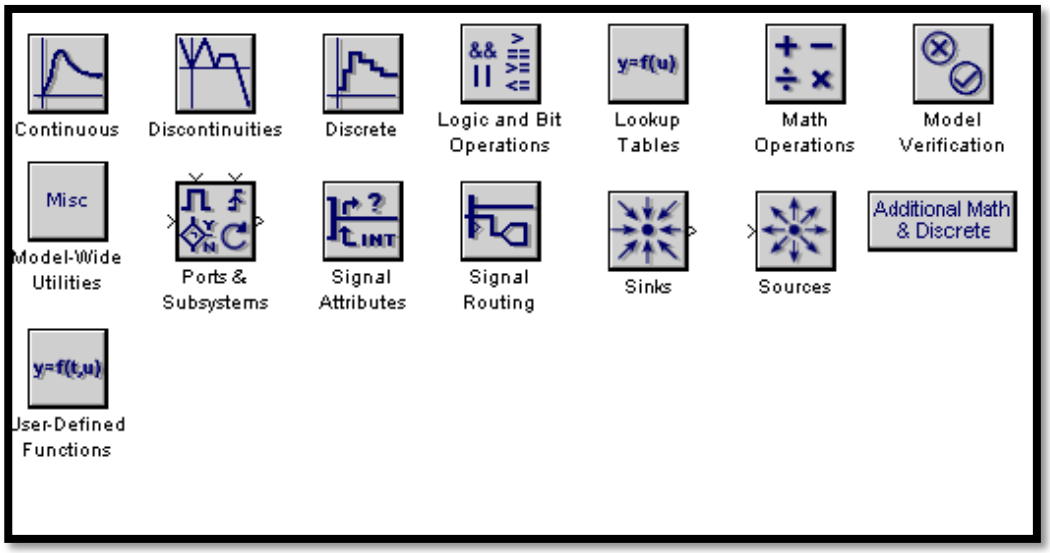


Figure A.4 la bibliothèque de Simulink

A.2 Bibliothèque Simulink :

Cette bibliothèque contient des éléments générateurs de signaux tels que saut unité, sinusoïde, fichiers de points, variable MATLAB, bruit, séquences, le temps courant de la simulation (horloge), etc.., sans oublier le générateur de signal lui-même.



FigA.5 blocs de la bibliothèque Simulink

Cette fenêtre contient des collections de blocs que l'on peut ouvrir par double clique :

- Sources : Sources de signaux.
- Discrete : Blocs discrets.
- Sinks : Les blocs d'affichages.
- Math operations : Les opérations mathématiques.
- Signal routing : les blocs de branchement.
- Logic and bit opérations : Les opérations logiques

- Model verification : Vérification des modèles
- Continuous : Les systèmes linéaires à temps continus.
- Discontinuities : Les systèmes non-linéaires à temps discret.

A.3 description du système solaire global :

La figure A.6 montre le schéma fonctionnel sous Simulink du système solaire générale, ce dernier se compose des éléments suivants :

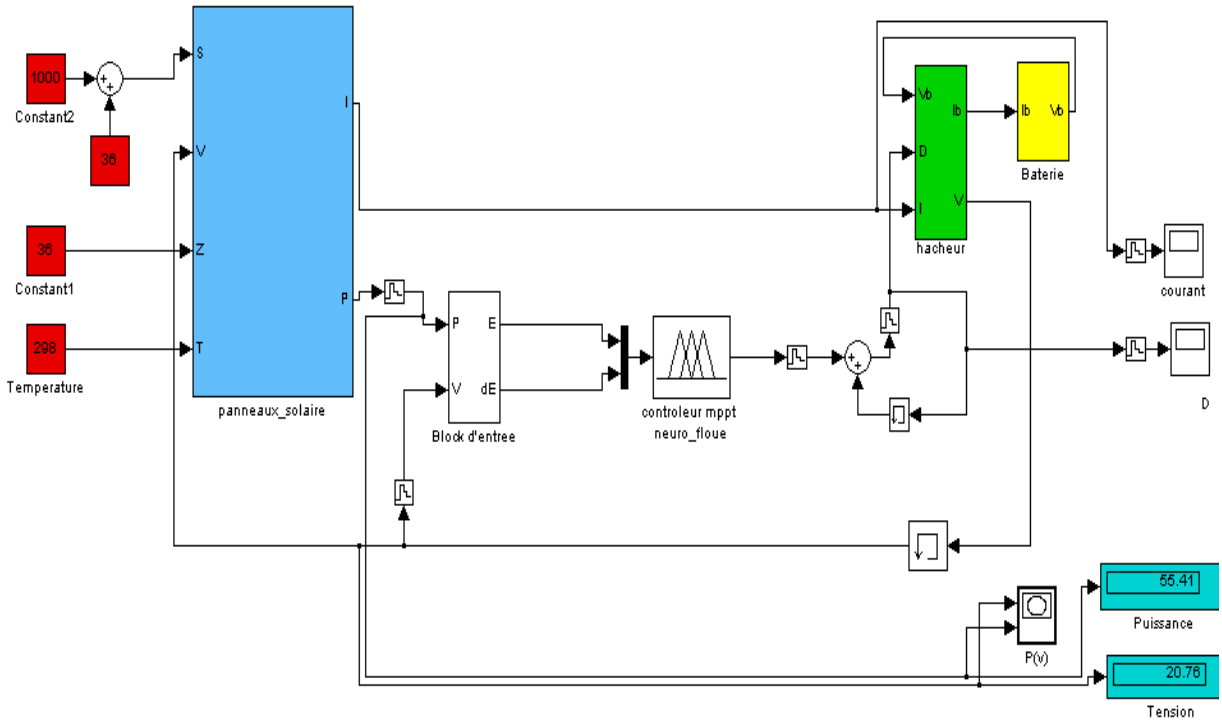


Figure A.6 Schéma synoptique pour la simulation du système photovoltaïque

A.3.1 le panneau solaire :

La (figure A.7) montre le schéma fonctionnel du panneau solaire sous Simulink, ce schéma modélise les équations mathématique du panneau solaire vu dans le chapitre 1 équation (1-1/2/3/4/5/6). On donne:

$I_{ph} = 3.25\ A$ (à $T = 298K$).

$R_p = 30\Omega$.

$R_s = 30\ .10^{-3}\Omega$

$E_g = 1.1\ eV$.

$n_1 = 1$.

$n_2 = 2$.

$K = 1.38\ 10^{-23} J/K$.

$q = 1.16\ 10^{-19} C$.

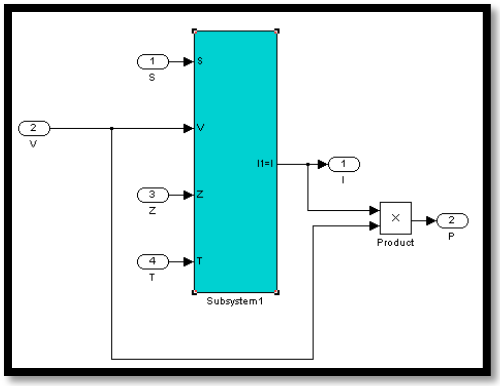


Figure A.7 Modèle SIMULINK du panneau solaire

A.3.2 le modèle de batterie :

Dans le modèle de Batterie on a pris les valeurs de R_{bs}, R_{b1}, R_{bp}, C_{b1}, C_{pb} comme suit:

$R_{bs} = 0.0013\ \Omega$

$R_{b1} = 2.84\ \ \Omega$

$R_{bp} = 1010^3\ \Omega$

$C_{b1} = 2.5\ \ \ KF$

$C_{bp} = 4.6501\ KF$

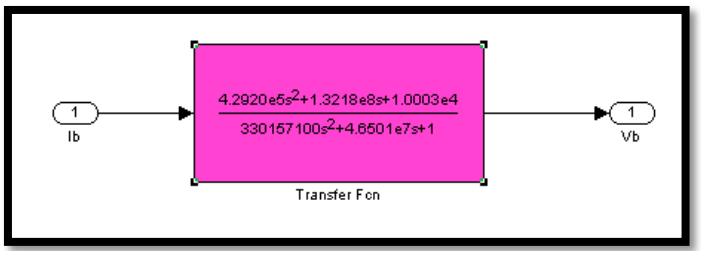


Figure A.8 Modèle SIMULINK de la batterie

Modélisation de l'hacheur :

Le modèle mathématique de convertisseur dévolteur a été donné dans le chapitre1. Composants utilisés:

$C1 = 5,6mF$

$C1 = C2$

$L = 3.5mH$

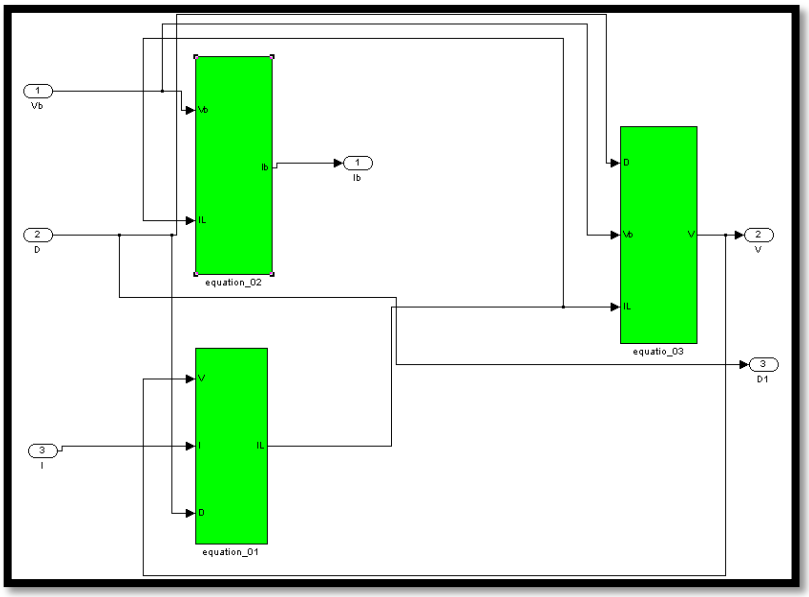


Figure A.9 Modèle SIMULINK du convertisseur Buck

ANNEXE B

Implémentation des projets sur la carte FPGA

B.1 Introduction :

Les progrès technologiques continus dans le domaine des circuits intégrés ont permis la réduction des coûts et de la consommation. Les circuits intégrés spécifiques ont permis une réduction de la taille des systèmes numériques ainsi que la réalisation de circuits de plus en plus complexes, tout en améliorant leurs performances et leur fiabilité. Aujourd'hui les techniques de traitement numérique occupent une place majeure dans tous les systèmes électroniques modernes grand public, professionnels ou militaires. De plus, les techniques de réalisation de circuits spécifiques, tant dans les aspects matériels (composants programmables, circuits pré caractérisés et bibliothèques de macro fonctions) que dans les aspects logiciels (placement-routage, synthèse logique) font désormais de la microélectronique une des bases indispensables pour la réalisation de systèmes numériques performants. Elle impose néanmoins une méthodologie de développement très structurée.
Les circuits FPGA (Field Programmable Gate Array) sont certainement les circuits reprogrammables ayant le plus de succès. Ce sont des circuits entièrement configurables par programmation qui permettent d'implanter physiquement, par simple programmation, n'importe quelle fonction logique. De plus, ils ne sont pas limités à un mode de traitement séquentiel de l'information comme avec les microprocesseurs ; et en cas d'erreur, ils sont reprogrammables électriquement sans avoir à extraire le composant de son environnement. Nous allons d'abord faire une description des circuits FPGA, ensuite nous introduire le langage VHDL, et nous terminerons par donner les étapes nécessaires au développement d'un projet sur un circuit FPGA, de la programmation jusqu'au chargement sur la carte.

B.2 Circuits FPGA :

Les FPGA (Field Programmable Gate Array) sont des circuits à architecture programmable qui ont été inventés par la société XILINX en 1985. Ils sont entièrement reconfigurables et ne demandent donc pas de fabrication spéciale en usine, ni de systèmes de développement coûteux ; ceci permet de les reprogrammer à volonté afin d'accélérer notablement certaines phases de calculs. Un autre avantage de ces circuits est leur grande souplesse qui permet de les réutiliser à volonté dans des algorithmes différents en un temps très court (quelques millisecondes).

De nombreuses familles de circuits programmables et reprogrammables sont apparues Depuis les années 70 avec des noms très divers suivant les constructeurs. La (figure B.1) donne une classification possible des circuits numériques en précisant où se situent les circuits FPGA dans cette classification. Les FPGA sont utilisés dans de nombreuses applications, on en cite dans ce qui suit quelques unes :

- Prototypage de nouveaux circuits.
- Fabrication de composants spéciaux en petite série.
- Adaptation aux besoins rencontrés lors de l'utilisation.

- Systèmes de commande à temps réel.
- DSP (Digital Signal Processor).
- Imagerie médicale.

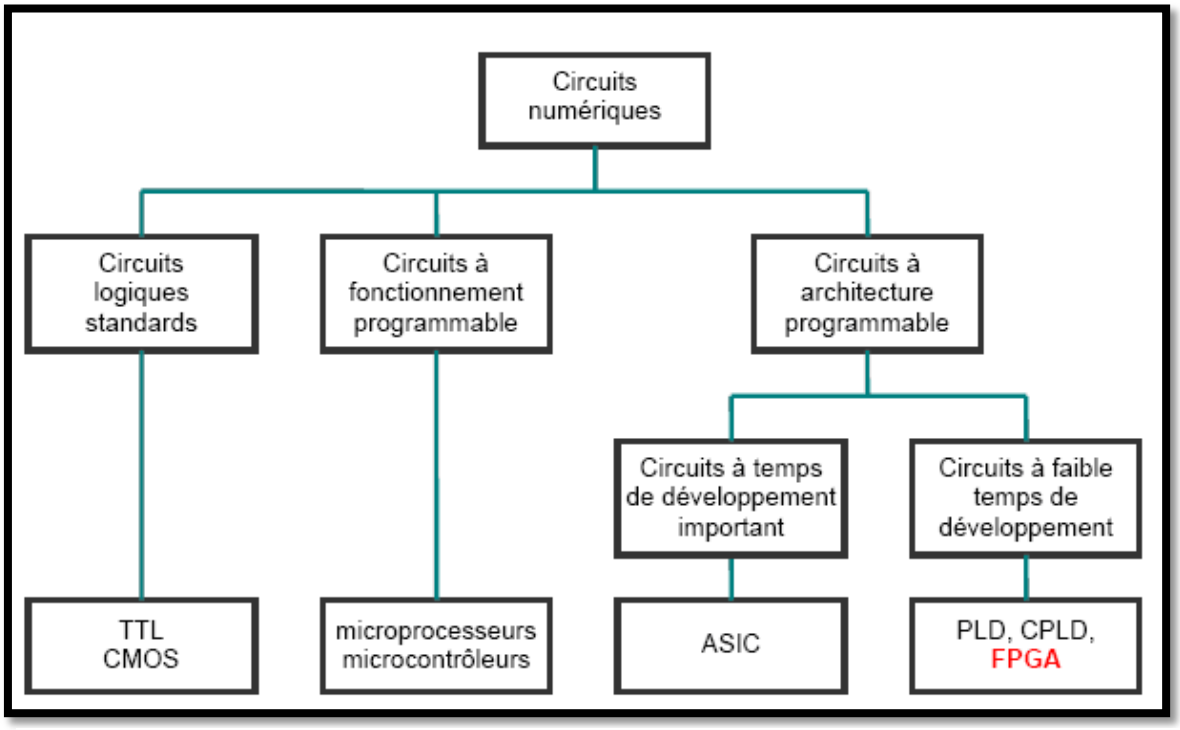


Figure. B.1 Classification des circuits numériques

B.2.1 Architecture des circuits FPGA :

Les circuits FPGA possèdent une structure matricielle de deux types de blocs (ou cellules). Des blocs d'entrées/sorties et des blocs logiques programmables. Le passage d'un bloc logique à un autre se fait par un routage programmable. Certains circuits FPGA intègrent également des mémoires RAM, des multiplieurs et même des noyaux de processeurs. Actuellement deux fabricants mondiaux se disputent le marché mondial des FPGA : Xilinx et Altera. De nombreux autres fabricants, de moindre envergure, proposent également leurs propres produits avec des technologies et des principes organisationnels différents. Dans ce qui suit, on va faire une description de l'architecture utilisée par Xilinx, car c'est sur des circuits Xilinx que nous allons va implémenter notre programme. L'architecture retenue par Xilinx se présente sous forme de deux couches :

- Une couche appelée circuit configurable.
- Une couche réseau mémoire SRAM (Static Read Only Memory).

Circuit configurable

La couche dite (circuit configurable) est constituée d'une matrice de blocs logiques configurables (CLB) permettant de réaliser des fonctions combinatoires et des fonctions

séquentielles. Tout autour de ces blocs logiques configurables, nous trouvons des blocs d'entrées/sorties (IOB) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs (figure B.2). La programmation du circuit FPGA, appelé aussi LCA (Logic Cells Arrays), consistera en l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ servant à interconnecter les éléments des CLB et des IOB, afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM.

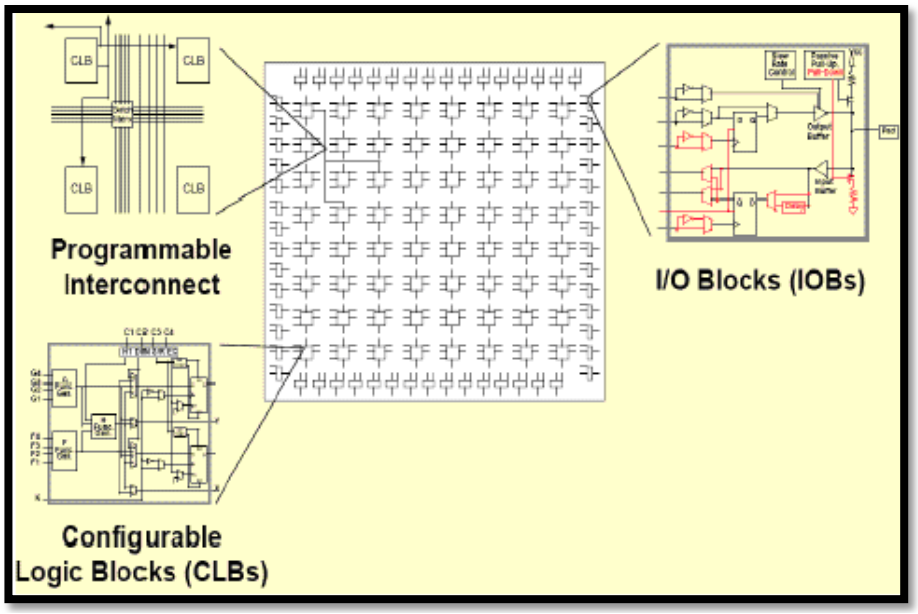


Figure. B.2 Architecture interne du FPGA

<u>Réseau mémoire SRAM :</u>

La programmation d'un circuit FPGA est volatile, la configuration du circuit est donc mémorisée sur la couche réseau SRAM et stockée dans une ROM externe.
Un dispositif interne permet à chaque mise sous tension de charger la SRAM interne (figure B.3) à partir de la ROM. Ainsi on conçoit aisément qu'un même circuit puisse être exploité successivement avec des ROM différentes puisque sa programmation interne n'est jamais définitive. On voit tout le parti que l'on peut tirer de cette souplesse en particulier lors d'une phase de mise au point. Une erreur n'est pas rédhibitoire, mais peut aisément être réparée. Les blocs logiques configurables sont les éléments déterminants des performances du circuit FPGA. Chaque CLB est un bloc de logique combinatoire composé de générateurs de fonctions à quatre entrées (LUT) et d'un bloc de mémorisation/synchronisation composé de bascules D. Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du CLB. La LUT (Look Up Table) est un élément qui dispose de quatre entrées, il existe donc $2^4 = 16$ combinaisons différentes de ces entrées. L'idée consiste à mémoriser la sortie correspondant à chaque combinaison d'entrée dans une petite table de 16 bits, la LUT devient ainsi un petit bloc générateur de fonctions. La (figure B.4) ci-dessous montre le schéma simplifié d'un CLB de la famille XC4000 de Xilinx.

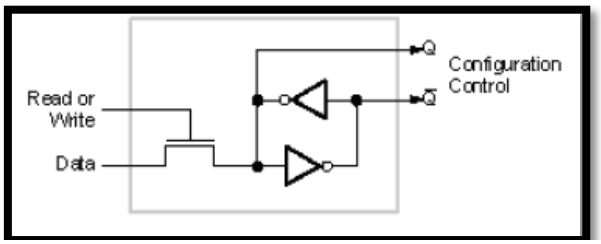


Figure B.3. Structure d'une cellule SRAM

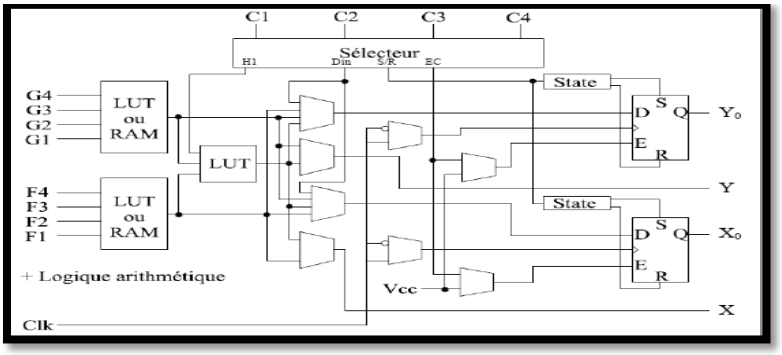


Figure B.4. Schéma d'une cellule logique (XC4000 de Xilinx)

B.2.2 Les IOB (Input Output Bloc :

Ces blocs d'entrée/sortie permettent l'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signal bidirectionnel ou être inutilisé (état haute impédance). La (figure B.5) présente la structure de ces blocs.

Configuration en entrée

Premièrement, le signal d'entrée traverse un buffer qui, selon sa programmation, peut détecter soit des seuils TTL soit des seuils CMOS. Il peut être routé directement sur une entrée directe de la logique du circuit FPGA ou sur une entrée synchronisée.
Cette synchronisation est réalisée à l'aide d'une bascule de type D, le changement d'état peut se faire sur un front montant ou descendant. De plus, cette entrée peut être retardée de quelques nanosecondes pour compenser le retard pris par le signal d'horloge lors de son passage par l'amplificateur. Le choix de la configuration de l'entrée s'effectue grâce à un multiplexeur (program controlled multiplexer). Un bit positionné dans une case mémoire commande ce dernier.

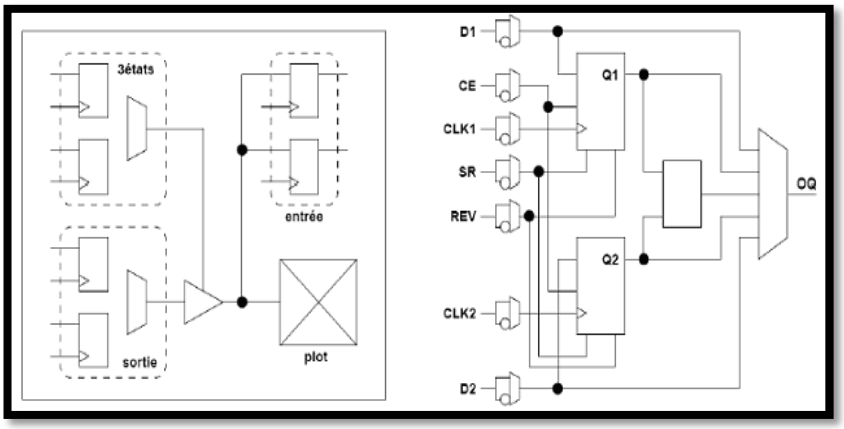


Fig. B.5 Schéma d'un bloc d'entrée/sortie (IOB)

Configuration en sortie :

Nous distinguons les possibilités suivantes :

- inversion ou non du signal avant son application à l'IOB.
- synchronisation du signal sur des fronts montants ou descendants d'horloge.
- signaux en logique trois états ou deux états, le contrôle de mise en haute impédance et la réalisation des lignes bidirectionnelles sont commandés par le signal de commande (Out Enable), lequel peut être inversé ou non.

Chaque sortie peut délivrer un courant de 12mA. Ainsi toutes ces possibilités permettent au concepteur de connecter au mieux une architecture avec les périphériques extérieurs.

B.2.3 Les différents types d'interconnexion :

Les connexions internes dans les circuits FPGA sont composées de segments métallisés. Parallèlement à ces lignes, nous trouvons des matrices programmables réparties sur la totalité du circuit, horizontalement et verticalement entre les divers CLB. Elles permettent les connexions entre les diverses lignes, celles-ci sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM (Random Access Memory). Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les blocs d'entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé possible. Pour parvenir à cet objectif, Xilinx propose trois sortes d'interconnexion selon la longueur et la destination des liaisons. Nous disposons :

- d'interconnexions à usage général.
- d'interconnexions directes.
- de longues lignes.

Les interconnexions à usage général

Des aiguilleurs appelés aussi (switch matrix) sont situés à chaque intersection.
Leur rôle est de raccorder les segments entre eux selon diverses configurations, ils assurent ainsi la communication des signaux d'une voie vers l'autre. Ces interconnexions sont utilisées pour relier un CLB à n'importe quel autre CLB. Pour éviter que les signaux traversant les grandes lignes ne soient affaiblis, nous trouvons généralement des buffers implantés en haut et à droite de chaque matrice de commutation (figure B.6).

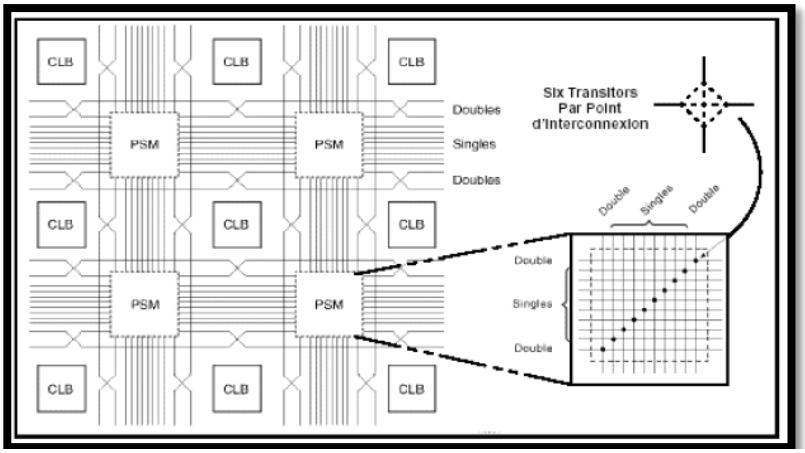


Figure. B.6 Connexions à usage général et détail d'une matrice de commutation

Les interconnexions directes

Ces interconnexions permettent l'établissement de liaisons entre les CLB et les IOB avec un maximum d'efficacité en termes de vitesse et d'occupation du circuit. De plus, il est possible de connecter directement certaines entrées d'un CLB aux sorties d'un autre (figure B.7).

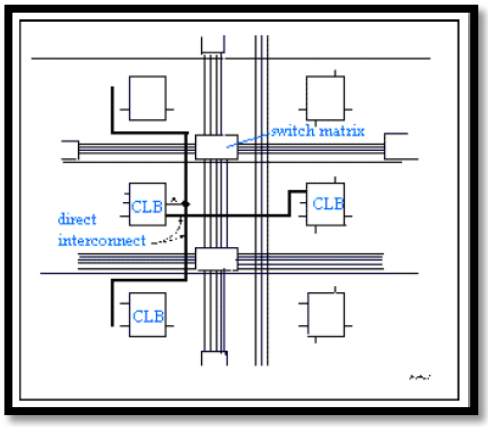


Figure. B.7 Les interconnexions directes

Les longues lignes

Les longues lignes sont de longs segments métallisés parcourant toute la longueur et la largeur du composant, elles permettent éventuellement de transmettre avec un minimum de retard les signaux entre les différents éléments dans le but d'assurer un synchronisme aussi parfait que possible. De plus, ces longues lignes permettent d'éviter la multiplicité des points interconnexion (figure B.8).

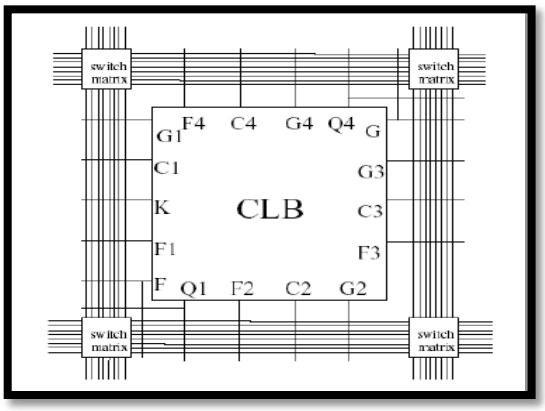


Figure B.8 Les longues lignes

B.3 Kit de développement Virtex-II V2MB1000 de Memec :

Le kit de développement Virtex-II V2MB1000 de Memec Design, qu'on a utilisé pour développer notre application, fournit une solution complète de développement d'applications sur la famille Virtex-II de Xilinx. Il utilise le circuit (FPGA XC2V1000-4FG456C) qui appartient à la famille Virtex-II de Xilinx et qui est équivalent à 1 million de portes logiques. La haute densité d'intégration des portes ainsi que le nombre important d'entrées/sorties

disponibles à l'utilisateur permettent d'implémenter des systèmes complets de solutions sur la plate forme FPGA. La carte de développement inclut aussi une mémoire 16M x 16 DDR, deux horloges, un port série RS-232 et des circuits de support additionnels. Une interface LVDS est disponible avec un port de transmission 16-bit et un port de réception 16-bit, en plus de signaux d'horloge, d'état et de contrôle pour chacun de ces ports. La carte supporte également le module d'expansion Memec Design P160, qui permet d'ajouter facilement des modules pour des applications spécifiques. La famille FPGA Virtex-II possède les outils avancés pour répondre à la demande applications de haute performance.
Le kit de développement Virtex-II fournit une excellente plateforme pour explorer ces outils. L'utilisateur peut alors utiliser toutes les ressources disponibles avec rapidité et efficacité. La figure B.9 présente une photo de la carte de développement et de ses outils.

B.3.1 Description de la carte de développement :

Un diagramme simplifié de la carte de développement Virtex-II est illustré à la (figure B.10) La carte de développement Virtex-II contient le circuit FPGA XC2V1000-4FG456C. Ce circuit fait partie de la famille Virtex-II, qui est une famille de circuits développés pour des applications haute performance telles que celles les télécommunications, l'imagerie et les applications DSP. Il possède 456 broches dont 324 peuvent être utilisées en entrées/sorties. Il se compose d'une matrice de 40x32 CLB et il contient un total de 10.240 LUT et 10.240 bascules (flip-flop). Sa capacité maximale en Select RAM est de 163.840 bits. La carte contient également une mémoire DDR de 32MB. Elle présente 2 générateurs d'horloges internes, générant des signaux d'horloge à 100MHz (CLK.CAN2) et 24MHz (CLK.CAN1). Un troisième signal d'horloge externe est disponible et peut être utilisé si besoin est. Elle contient aussi circuit de remise à zéro (Reset) activé par un bouton poussoir (SW3), un bouton poussoir (PROGn) pour initialiser la configuration et charger le contenu de la PROM dans le circuit FPGA (SW2) ainsi que deux boutons poussoirs (SW5 et SW6) qui peuvent être utilisés pour générer des signaux actifs. Deux afficheurs 7 segments à cathode commune sont présents sur la carte, et peuvent être utilisés durant la phase de test et de debugging. Il existe aussi 8 entrées exploitables par l'utilisateur (DIP switch) qui peuvent être mis statiquement à un état haut ou bas.

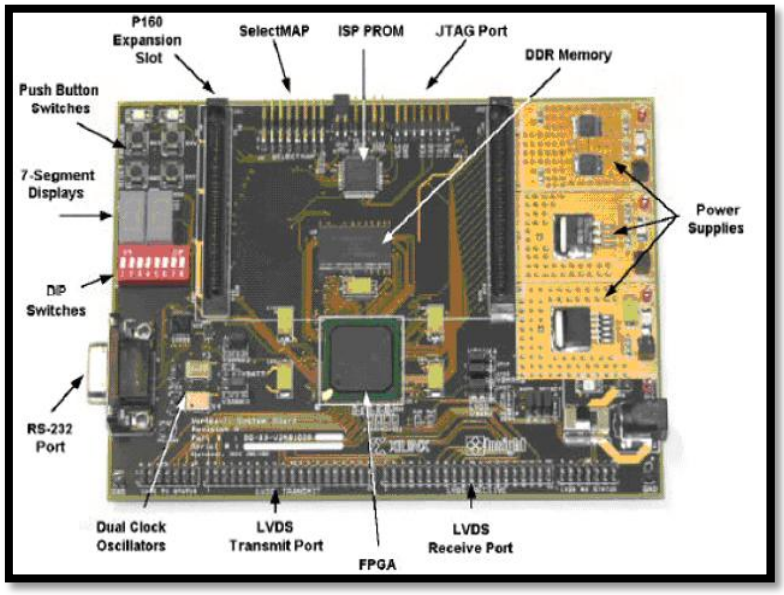


Figure. B.9 Carte de développement (Memec Design Virtex-II)

La carte possède une interface RS232, une interface JTAG pour programmer l'ISP PROM et configurer le circuit FPGA ainsi qu'un connecteur de câble parallèle IV qui peut aussi être utilisé pour configurer le FPGA via la configuration Maître/Esclave.
Il existe également des régulateurs internes de tension qui génèrent, à partir de l'alimentation principale de 5,0V, des tensions internes d'alimentation à 1,5V, 2,5V et 3,3V. Les entrées/sorties sont regroupées dans 8 différents groupes, et chaque groupe peut être configuré pour opérer dans le mode 2,5V ou 3,3V. La carte de développement peut être configurée pour travailler en mode Master Serial, Slave Serial, Master SelectMap, Slave SelectMap ou JTAG selon la position des jumpers M0, M1, M2 et M3.

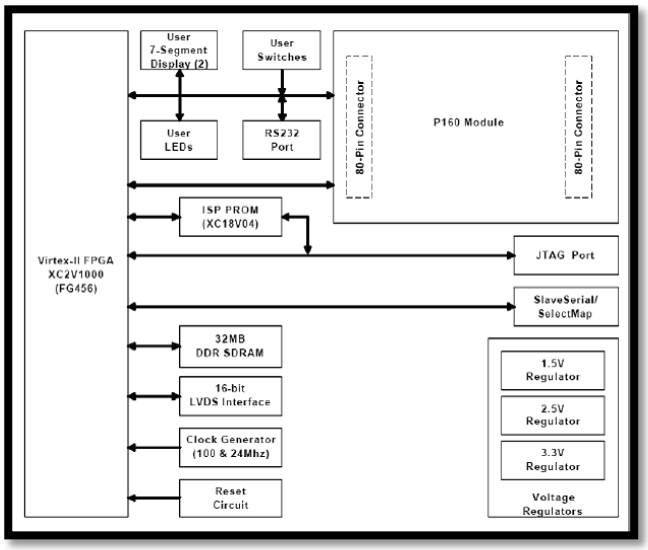


Figure. B.10 Diagramme de la carte de développement

B.3.2 Chargement du programme sur la carte de développement :

La carte de développement Virtex-II supporte plusieurs méthodes de configuration de son circuit FPGA. Le port JTAG peut être utilisé directement pour configurer le FPGA, ou pour programmer l'ISP PROM. Une fois l'ISP PROM programmée, elle peut être utilisée pour configurer le FPGA. Le port SelectMap/Slave Serial sur cette carte peut aussi être utilisé pour configurer le FPGA. La (figure B.11) montre l'installation pour toutes les configurations de modes supportés par la carte de développement Virtex-II.

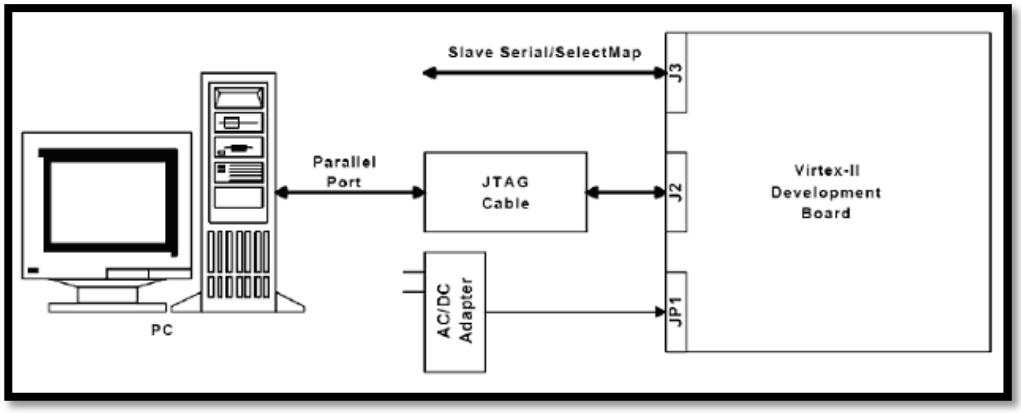


Figure B.11 Chargement du programme sur la carte

Utilisation de l'interface JTAG :

Le câble Memec Design JTAG est connecté d'un côté à la carte de développement, et de l'autre au port série du PC. On utilise alors l'outil de programmation du JTAG de Xilinx (iMPACT) pour charger le programme binaire soit directement sur le circuit FPGA en mode JTAG, soit sur l'ISP PROM en mode Master Serial ou Master SelectMap. Dans ce dernier, cas il faut appuyer sur le bouton poussoir PROGn (SW2) pour initialiser la configuration dans le circuit FPGA.

B.4 Langage de description VHDL :

Auparavant pour décrire le fonctionnement d'un circuit électronique programmable, les techniciens et les ingénieurs utilisaient des langages de bas niveau (ABEL, PALASM, ORCAD/PLD,..) ou plus simplement un outil de saisie de schémas. Actuellement la densité de fonctions logiques (portes et bascules) intégrées dans les PLD (Programmable Logic Device) est telle (plusieurs milliers voire millions de portes) qu'il n'est plus possible d'utiliser les outils d'hier pour développer les circuits d'aujourd'hui.

Les sociétés de développement et les ingénieurs ont voulu s'affranchir des contraintes technologiques des circuits. Ils ont donc créé des langages dits de haut niveau à savoir VHDL et VERILOG. Ces deux langages font abstraction des contraintes technologies des circuits PLD. Ils permettent au code écrit d'être portable, c'est-à-dire qu'une description écrite pour un circuit peut être facilement utilisée pour un autre circuit.

Il faut avoir à l'esprit que ces langages dits de haut niveau permettent de matérialiser les structures électroniques d'un circuit. En effet les instructions écrites dans ces langages se traduisent par une configuration logique de portes et de bascules qui est intégrée à l'intérieur des circuits PLD. C'est pour cela qu'on préfère parler de description VHDL ou VERILOG que de langage. Dans ce qui suit, on s'intéressera uniquement au VHDL et aux fonctionnalités de base de celui-ci lors des phases de conception ou synthèse.

L'abréviation VHDL signifie VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. Ce langage a été développé dans les années 80 par le DoD (Departement of Defense) des Etats-Unis ; l'objectif était de disposer d'un langage commun avec les fournisseurs pour décrire les circuits complexes. En 1987, une première version du langage est standardisée par l'IEEE (Institute of Electrical and Electronics Engineers) sous la dénomination IEEE Std. 1076-1987 (VHDL 87). Une évolution du VHDL est normalisée en 1993 (IEEE Std. 1076-1993 ou VHDL-93) ; cette nouvelle version supprime quelques ambiguïtés de la version 87, et surtout met à disposition de nouvelles commandes. Actuellement, tous les outils dignes de ce nom supportent le VHDL-93. En 1997, de nouvelles libraires ont été normalisées de manière à ajouter des fonctions pour la synthèse (conception) de circuits logiques programmables PLD, et plus particulièrement les FPGA. La dernière révision est celle de 2002 (IEEE Std. 1076-2002 ou VHDL-2002).

B.5 Relation entre une description VHDL et un circuit FPGA :

L'implantation d'une ou de plusieurs descriptions VHDL dans un circuit FPGA va dépendre de l'affectation que l'on fera des broches d'entrées/sorties et des structures de base du circuit logique programmable.

La (figure B.12) représente un exemple de descriptions VHDL ou de blocs fonctionnels implantés dans un FPGA. Lors de la phase de synthèse chaque bloc sera matérialisé par des portes et/ou des bascules. La phase suivante sera d'implanter les portes et les bascules à l'intérieur du circuit logique. Cette tâche est réalisée par le logiciel placement/routage (Fitter), au cours de laquelle les entrées et sorties seront affectées à des numéros de broches. On peut remarquer sur le schéma la fonction particulière du bloc VHDL N °5. En effet dans la description fonctionnelle d'un FPGA on a souvent besoin d'une fonction qui sert à cadencer le fonctionnement de l'ensemble, celle-ci est très souvent réalisée par une machine d'états synchronisée par une horloge, voir (figure B.12).

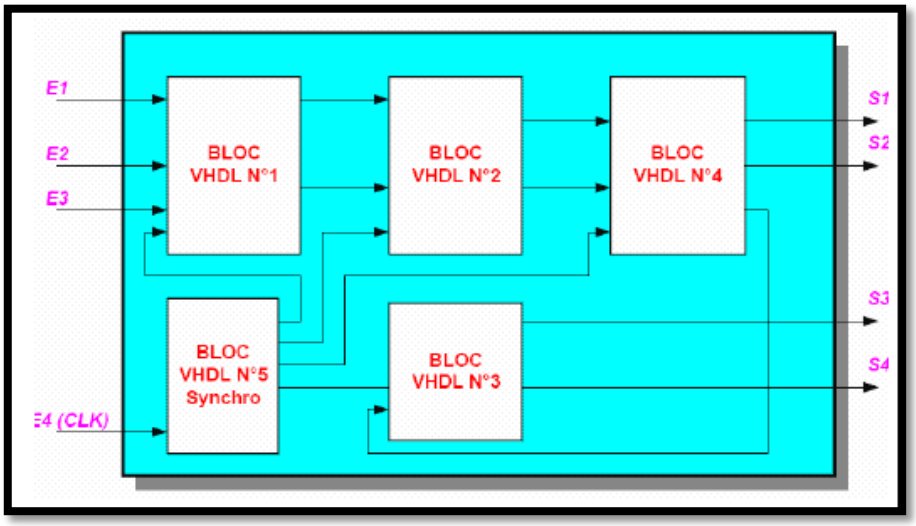


Figure B.12 Schéma fonctionnel d'implantation d'une description VHDL dans un circuit FPGA

B.5.1 Saisie du texte VHDL :

La saisie du texte VHDL se fait sur le logiciel « ISE Xilinx Project Navigator », propose une palette d'outils permettant d'effectuer toutes les étapes nécessaires au développement d'un projet sur circuit FPGA. Il possède également des outils permettant de mettre au point une entrée schématique ou de créer des diagrammes d'état, qui peuvent être utilisés comme entrée au lieu du texte VHDL. Figure B.13 montre comment se présente le logiciel (ISE Xilinx Project Navigator).

B.5.2 Synthèse :

La synthèse permet de réaliser l'implémentation physique d'un projet. Le synthétiseur a pour rôle de convertir le projet, en fonction du type du circuit FPGA cible utilisé, en portes logiques et bascules de base. L'outil « View RTL Schematic » permet de visualiser les schémas électroniques équivalents générés par le synthétiseur.
De plus, le synthétiseur permet à l'utilisateur d'imposer de technologie (User contraints) : par exemple fixer la vitesse de fonctionnement (Create Timing Constraints), délimiter la zone du circuit FPGA dans laquelle le routage doit se faire (Create Area constraints) ou affecter les broches d'entrées/sorties (Assign Package Pins).

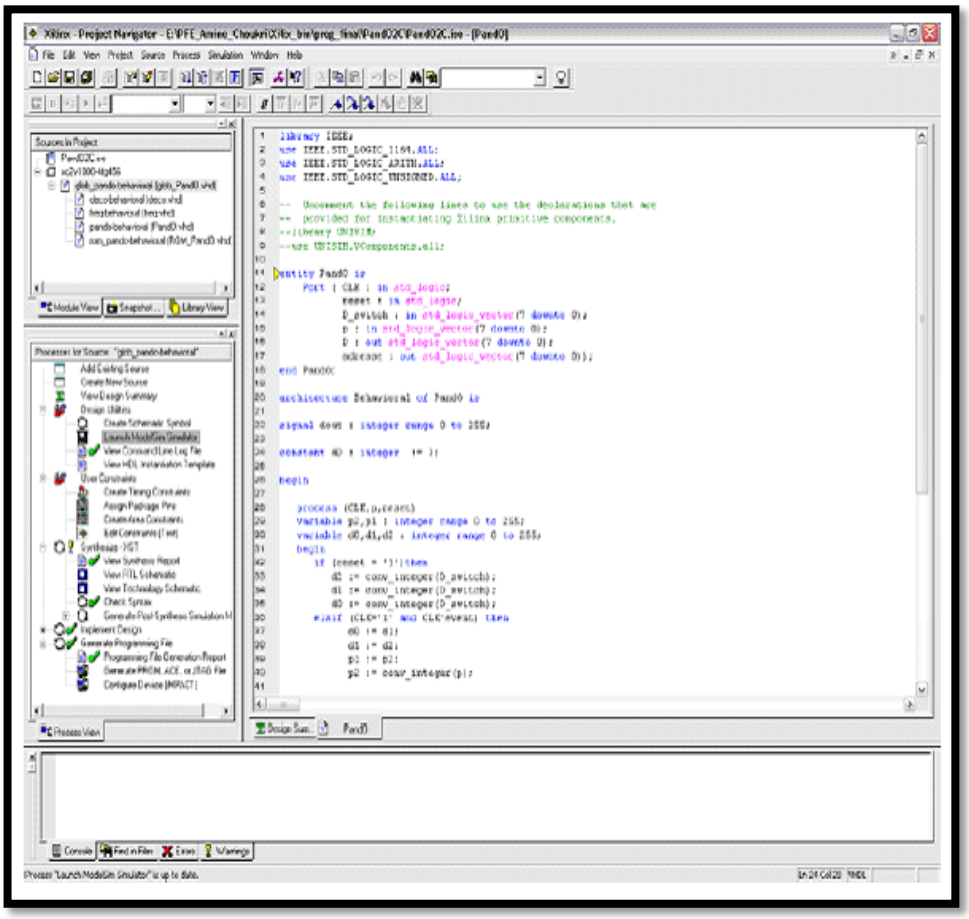


Figure. B.13 Vue d'ensemble logiciel Xilinx

B.5.3 Simulation :

La simulation permet de vérifie le comportement d'un design avant ou après implémentation dans le composant cible, le simulateur utilisé est Modelsim.

B.5.4Optimisantion, placement et routage :

Pendant l'étape d'optimisation, l'outil cherche à minimiser les temps de propagation et a occuper le moins d'espace possible sur FPGA cible.

B.5.5 programmation du composant et test :

Dans cette dernière étape (generate programming files), on génère le ficher à charger sur le circuit FPGA à travers l'interface JTAC.