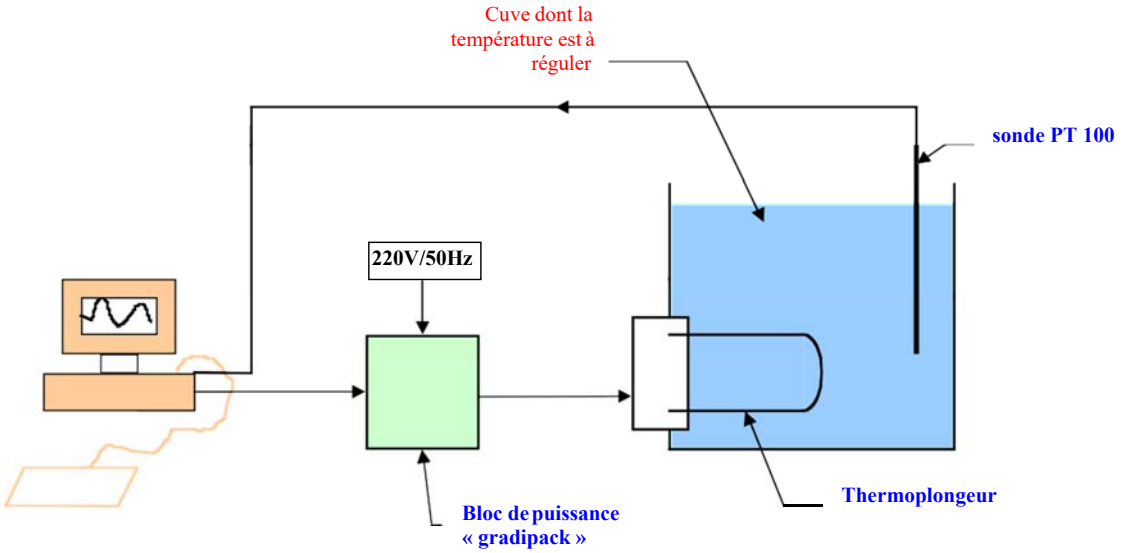
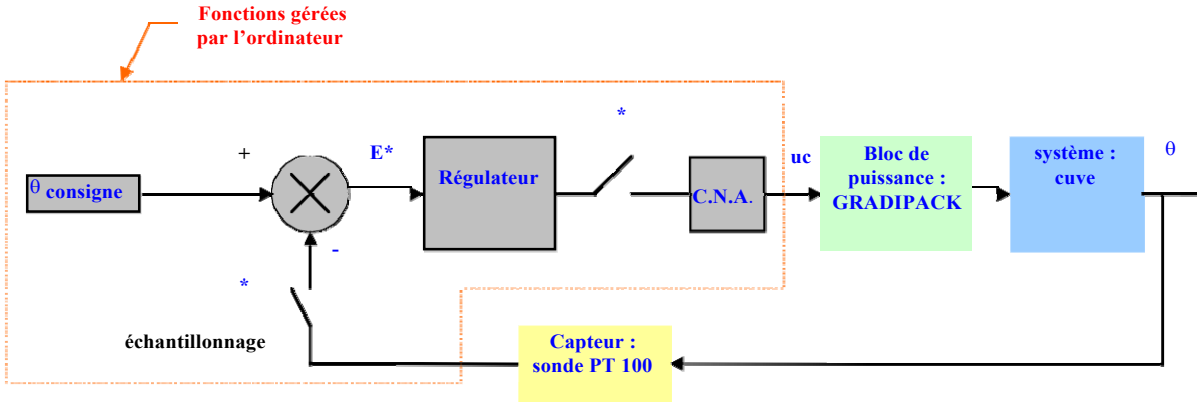
TP : Régulation thermique

A. Présentation du système à réguler

Schéma de principe



Le schéma de principe ci-dessus représente le synoptique de la régulation de température à réaliser



et dont le schéma fonctionnel est précisé ci-dessous :

Il s'agit de réguler la température de l'eau d'une **cuve** (seau d'une contenance de quelques litres) équipée d'un **thermoplongeur** apportant l'énergie calorifique. Une **sonde de température** (***PT100***) permet de mesurer la température réelle de l'eau. Il est bon de prévoir un brassage de l'eau dans la cuve afin d'homogénéiser sa température.

La puissance nominale du **thermoplongeur** (***1 kW***) est modulée grâce à un bloc de puissance **« GRADIPACK »** (dénomination commerciale). Il s'agit d'un ensemble de 2 thyristors montés tête-bêche (gradateur) permettant de faire varier la puissance électrique dissipée par variation de l'angle de retard à l'amorçage des thyristors. Celle-ci est contrôlée par une tension de commande 0 -10V continue (U_{com}) qui permet de faire varier la tension efficace (U_{eff}) appliquée au thermoplongeur.

Le rôle du micro-ordinateur sera de faire la mesure de la température de la cuve à partir du signal

délivré par la sonde PT100 et sa mise à l'échelle, de comparer cette valeur avec la température de consigne, de traiter l'écart entre ces 2 valeurs et de fournir une tension de commande utile pour le «GRADIPACK «

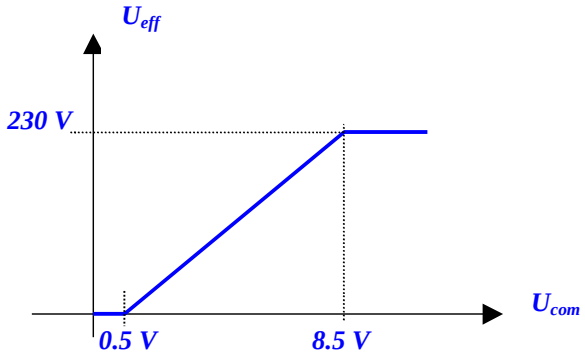
Données pour la modélisation du système

Cuve à chauffer :

On supposera que la capacité calorifique de la cuve à chauffer se réduit à l'eau qu'elle contient. En prenant un volume de 3 litres, on obtient :

$$C = m.c = 3 * 4.18E3 \approx 1.25E4JK^{-1}$$

Contrôle du thermoplongeur :



La figure ci-dessus donne la modélisation du fonctionnement du « GRADIPACK ». La tension efficace appliquée au thermoplongeur passe de 0 à 230V si la tension de commande évolue de 0.5V à 8.5V. On pourra donc écrire :

$$U_{eff} = \frac{230}{8}.(U_{com} - 0.5)$$

La puissance de chauffe sera donnée par : $P = P_N . \frac{(U_{eff})^2}{(U_N)^2}$, soit en tenant compte de l'expression précédente :

$$P = \frac{P_N}{8^2}(U_{com} - 0.5)^2 = 15.6(U_{com} - 0.5)^2 \text{ (en Watts).}$$

Evaluation des pertes thermiques :

Les pertes thermiques peuvent se mettre sous la forme $P_{th} = K.(\theta - \theta_{ext})$ (Loi de FOURIER). L'expérience montre qu'elles sont faibles. On les estimera à 50 W pour un $\Delta\theta = 25°C$. D'où

$$K = 2WK^{-1}$$

Bilan de puissance sur la cuve :

La puissance fournie par le thermoplongeur : P_T, sert à élever la température de l'eau et participe aux pertes soit :

$$P_T = C.\frac{d(\theta - \theta_{ini})}{dt} + K.(\theta - \theta_{ext})$$

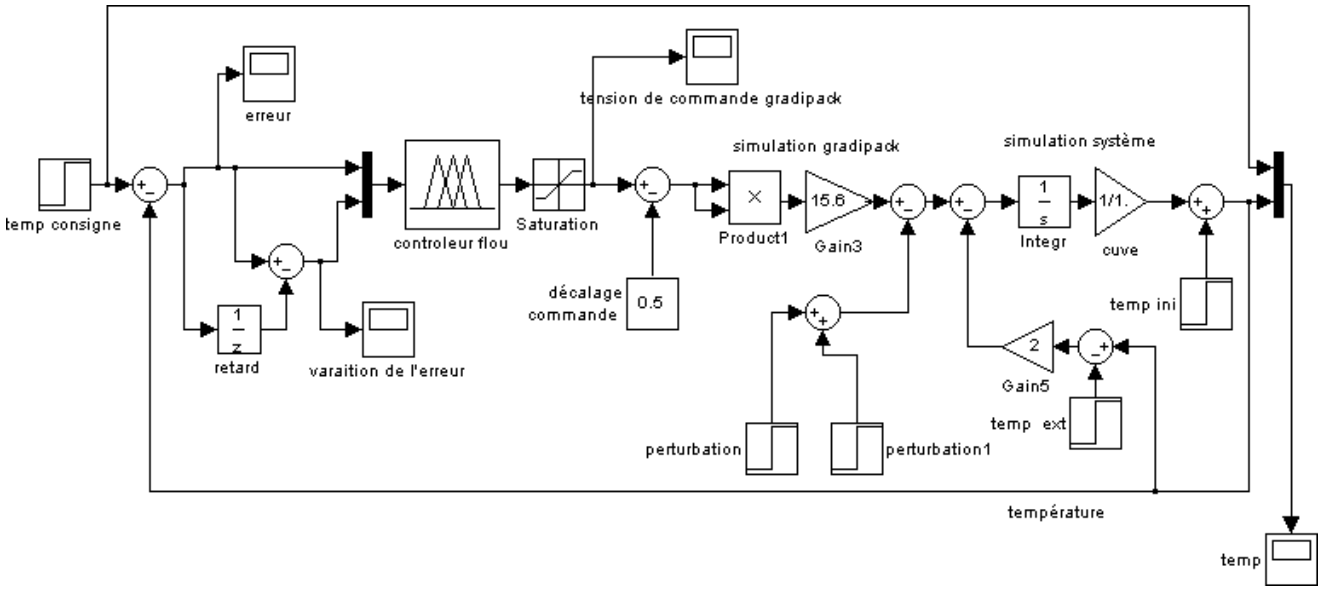
avec θ : température de l'eau
θ ini : température initiale de l'eau
θ ext : température extérieure à la cuve

B. Modélisation du problème avec MATLAB

Construction du modèle avec MATLAB

Le schéma fonctionnel ci-dessous présente la modélisation du système à étudier à partir des éléments de base décrits plus haut.

,…



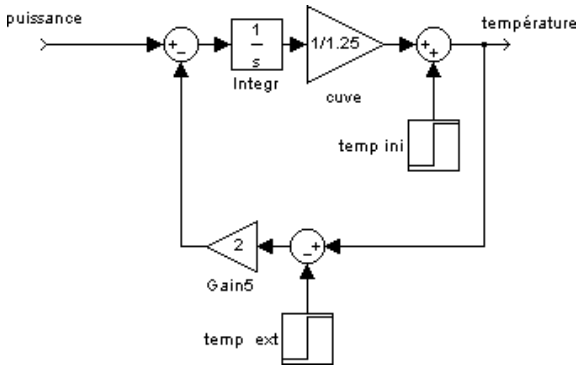
Modélisation de la cuve

En reprenant le bilan de puissance sur la cuve, il vient en utilisant la notation de Laplace :

$$P_T = C.p(\theta-\theta_{ini})+K(\theta-\theta_{ext}) \qquad P_T-K(\theta-\theta_{ext})=C.p(\theta-\theta_{ini})$$

$$(\theta-\theta_{ini})=[P_T-K(\theta-\theta_{ext})]\frac{1}{C.p} \qquad \theta=\theta_{ini}+\frac{1}{C.p}[P_T-K(\theta-\theta_{ext})]$$

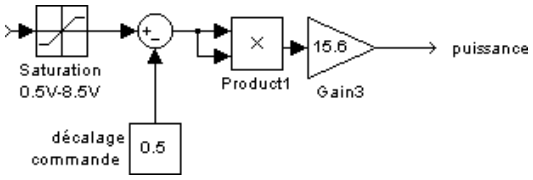
Ce qui se traduit par le schéma ***simulink*** ci dessous :



Modélisation du gradateur de puissance

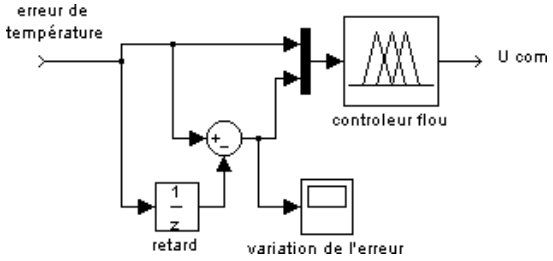
Rappelons la relation donnant la puissance de chauffe en fonction de la tension de commande

$$P = \frac{P_N}{8^2}(U_{com} - 0.5)^2 = 15.6(U_{com} - 0.5)^2$$



La saturation permet de limiter la tension de commande à la plage 0.5V-8.5V et le bloc produit permet de faire l'élévation au carré.

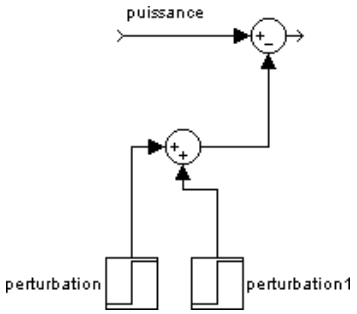
« vectorisation » du contrôleur



L'opérateur retard « $\frac{1}{z}$ » est un opérateur échantillonné, sa sortie est mise à jour toutes les 5 secondes. Il permet de mémoriser la valeur de l'erreur à l'instant « t-1 ». Ainsi, le circuit de différence permet d'obtenir la variation de l'erreur entre 2 instants d'échantillonnage. Le contrôleur sera donc sensible à l'erreur de température et à sa variation.

Prise en compte d'une perturbation

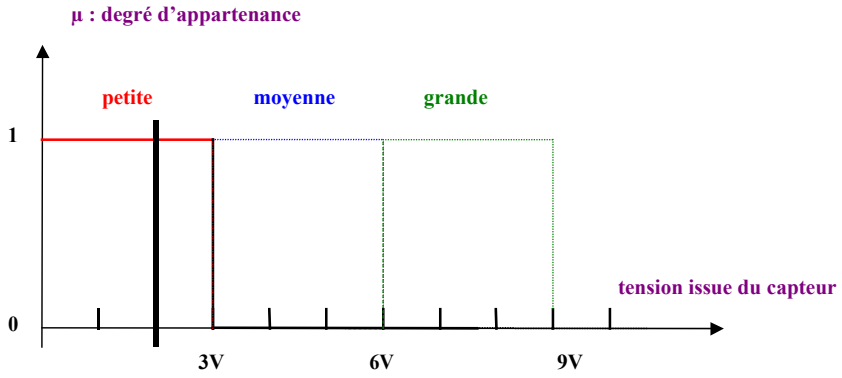
La perturbation pourra être simulée par un créneau de puissance négatif limité dans le temps (et décalé par rapport au début de la simulation) traduisant un apport d'eau froide par exemple.



Construction du correcteur flou

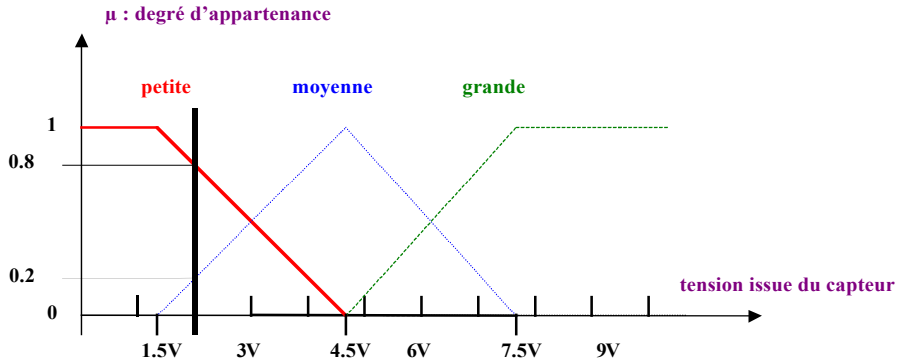
Notion d'ensemble flou

La **figure ci-dessous** représente la plage de variation d'un capteur (de température par exemple). Le signal de sortie peut varier de 0 à 10V, mais on constate qu'à partir de 9 V se produit une saturation. On va décider de ***partitionner*** la plage de variation utile (0-9V) en 3 zones qui seront dénommées : **petite** (0-3V), **moyenne** (3V à 6V), et **grande** (6V à 9V).



Si la tension est de 2V, elle est alors dans la classe **« petite »** et bien sûr ni **« moyenne »** ni **« grande »**. Le *degré d'appartenance* de la valeur 2V à la classe **« petite »** est de 1 et 0 pour les 2 autres. On retrouve là une logique binaire où l'état d'un comparateur ne peut être que « 0 » ou « 1 ». l'inconvénient de cette représentation est que la valeur 2.95V appartient à la classe **« petite »** alors que la valeur 3.05V appartient à la classe **« moyenne »** et pourtant les valeurs sont proches. Ceci est dû à la définition des 3 sous-ensembles classiques **« petite »**, **« moyenne »** et **« grande »** qui ont une « *frontière* » nette d'où cette discontinuité. Si le résultat de la mesure est légèrement bruité (cas très fréquent évidemment), on pourra basculer d'un coté à l'autre de la « frontière ».

La notion de sous-ensemble flou permet d'éviter cette ***discontinuité*** brutale. Cette opération s'appelle **« la fuzzification »**. La sortie du capteur est appelée variable **« linguistique »** puisqu'elle pourra prendre les valeurs *linguistiques* **« petite »**, **« moyenne »** ou **« grande »**, mais on va modifier comme indiqué ci-dessous les « *frontières* » entre ces valeurs *linguistiques*, c'est-à-dire les formes des fonctions d'appartenance.



Représentation des degrés d'appartenance avec 3 ensembles flous

Dans cette représentation la valeur de tension 2V sera à la fois **« petite »** et **« moyenne »**; de plus, on va préciser de combien. On définit alors un ***degré d'appartenance μ*** à chacune de ces classes :

$$\mu_{petite} = 0.8 \text{ et } \mu_{moyenne} = 0.2$$

On peut dire que l'ordre de grandeur de la tension issue du capteur est pris en compte par les valeurs linguistiques, puis la valeur précise est contenue dans les degrés d'appartenance (on parle aussi de *mesure d'appartenance* ou de *degré de certitude*).

Remarques :

Les profils des frontières peuvent être variés : triangle, trapèze, sigmoïde, courbe de Gauss, etc.

Il n'y a rien de **flou** dans cette représentation, une même mesure peut appartenir à plusieurs classes avec un degré d'appartenance **précis** dans chaque classe.

La valeur de la tension issue du capteur est ici considérée comme précise comme en automatique classique (on l'appelle singleton).

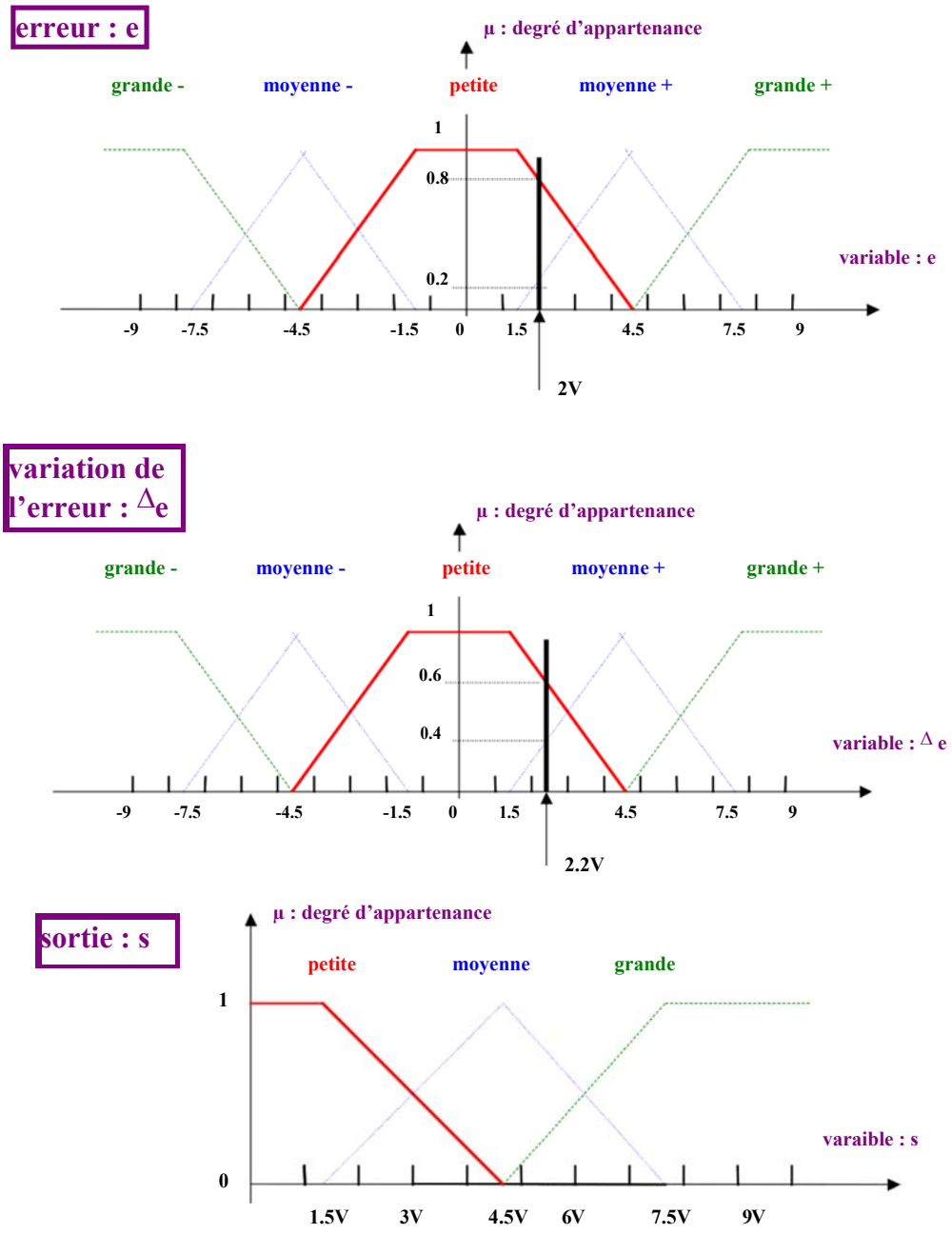
C'est le rôle de l'expert que de définir les valeurs linguistiques de la variable.

Structure d'un contrôleur flou

Opération de fuzzification

La grandeur d'entrée du contrôleur doit d'abord être ***fuzzifiée***, c'est-à-dire que l'on va fixer les valeurs linguistiques ainsi que la forme des fonctions d'appartenance. Cette opération doit être faite également sur la variable de sortie. Bien sûr cette sortie fuzzifiée n'est pas exploitable pour attaquer l'interface de puissance. Il faudra donc avoir en tête de faire une opération de ***« défuzzification »*** pour résoudre ce problème.

On va prendre un exemple où l'on va fuzzifier l'erreur (notée **e**), entrée du régulateur, mais aussi la variation de l'erreur (notée **Δe**) et la sortie (notée **s**). Les valeurs de **e** de **Δe** sont mises à jour à chaque période d'échantillonnage. On passe pour cela par un convertisseur analogique numérique.



Fuzzification des 2 variables d'entrée et de la variable de sortie

L'entrée **e** a été *partitionnée* en 5 valeurs linguistiques ainsi que **Δe**. Elles peuvent être positives ou négatives.

La sortie **s** a été partitionnée en 3 valeurs linguistiques ; elle pourra n'être que positive (cas de la figure) dans le cas où la partie puissance est commandée par une tension (0, 10V) par exemple.

On a gradué les axes donnant la variation des variables en Volt pour faciliter la compréhension, mais en pratique les tests seront faits sur les valeurs échantillonnées des entrées (on passe par un C.A.N.)

Base de règles « d'inférence »

Le rôle de l'expert est ici présent car c'est lui qui va fixer les règles de la commande qui vont porter **uniquement** sur les valeurs linguistiques des variables.

Soit par exemple la liste de règles suivantes (*qui résultent de la connaissance de l'expert*) :

-**R_1** : **SI (e)** est petite **ET (Δe)** est petite **ALORS (s)** est petite
-**R_2** : **SI (e)** est petite **ET (Δe)** est moyenne + **ALORS (s)** est petite
-**R_3** : **SI (e)** est moyenne + **ET (Δe)** est petite **ALORS (s)** est petite
-**R_4** : **SI (e)** est moyenne + **ET (Δe)** est moyenne + **ALORS (s)** est moyenne
-**R_5** : **SI (e)** est grande + **ET (Δe)** est moyenne + **ALORS (s)** est moyenne
- ....

Le tableau ci-dessous représente la base de règles en donnant les valeurs de la sortie pour les différentes valeurs linguistiques de **e** et **Δe :**

Δe \ e	grande -	moyenne -	petite	moyenne +	grande +
grande -			...	...	...
moyenne -			...	...	...
petite	...	...	(R_1) s : petite	(R_3) s : petite	...
moyenne +	...	...	(R_2) s : petite	(R_4) s:moyenne	(R_5) s:moyenne
grande +	...	...	...		

Remarques :

- « **SI (e)** est petite » : constitue une ***prémisse*** de la règle
- « **ET** »: est appelé opérateur de ***conjonction***
- « **ALORS (s)** est petite » : est appelé ***implication*** (conclusion) de la règle
- La liste des règles est appelée **base d'inférence** (**inférence** : opération logique par laquelle on admet une proposition en vertu de sa liaison avec d'autres propositions tenues pour vraies : Nouveau Petit Robert). On parle aussi de **moteur d'inférence**. Il n'est pas nécessaire que toutes les cases du tableau soient remplies.
- On peut trouver les règles énoncées critiquables, elles ne sont données qu'à titre d'exemples et doivent être adaptées en fonction de chaque processus.

Mise en œuvre de la base de règles

Supposons qu'à un instant ***t*** on ait : **e = 2 V et Δe = 2.2 V**

Ces valeurs vont constituer des ***faits*** et à ce titre activer la base de règles. Il est facile de voir à partir de la figure donnant la fuzzification de **e** et **Δe** que les règles **R_1, R_2, R_3, R_4** vont être actives puisque **e** et **Δe** sont à la fois **petite** et **moyenne +** (partie la plus grisée du tableau). Il s'agit ici des valeurs *linguistiques* des variables.

Mais l'examen des figures montre aussi que l'on peut en déduire les degrés d'appartenance à chacune de ces classes.

Pour **e** on a $\mu_{petite} = 0.8$ et $\mu_{moyenne +} = 0.2$

Pour **Δe** on a $\mu_{petite} = 0.6$ et $\mu_{moyenne +} = 0.4$

On va donc étudier maintenant le rôle de chaque règle activée et voir concrètement comment on traduit les opérations de conjonction **(ET)** et d'implication **(ALORS).**

Pour traduire le **ET** *(c'est-à-dire la conjonction des prémisses),* on peut utiliser la fonction **MIN :**

on prendra la valeur minimale des 2 degrés d'appartenance des prémisses. La signification physique de ce choix est de conserver l'information la plus sûre.

Le résultat est un nouveau degré d'appartenance au sous ensemble flou de la sortie.

Ainsi pour $\mathbf{R_1}$, on a :

le **MIN** de 0.8 et 0.6 est 0.6, donc pour **(s)** : $\mu_{petite} = 0.6$

On fait ensuite la même chose pour toutes les règles activées et on obtient les résultats suivants pour la variable linguistique de sortie **(s)** :

Pour $\mathbf{R_1}$: $\mu_{petite} = 0.6$
Pour $\mathbf{R_2}$: $\mu_{petite} = 0.4$
Pour $\mathbf{R_3}$: $\mu_{petite} = 0.2$
Pour $\mathbf{R_4}$: $\mu_{moyenne} = 0.2$

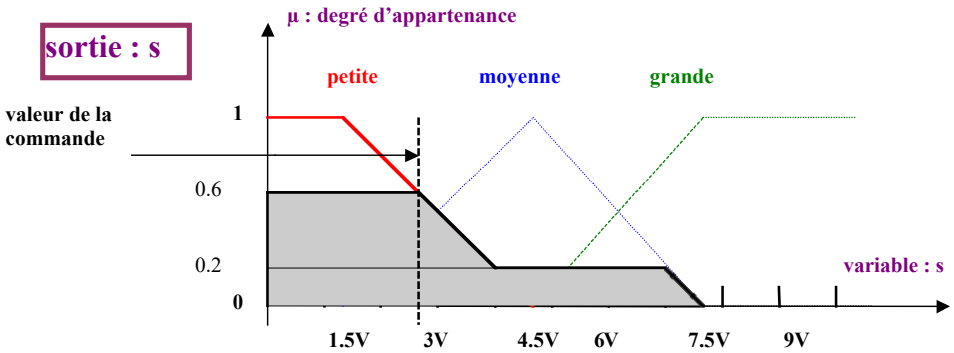
L'opération n'est pas terminée car les 4 règles étant activées en même temps, il faut faire ce que l'on appelle ***l'agrégation des règles (l'assemblage)***

Les règles $\mathbf{R_1}$, $\mathbf{R_2}$ **et** $\mathbf{R_3}$ concernent la même valeur linguistique **« petite »** de la variable de sortie alors que $\mathbf{R_4}$ porte sur la valeur **« moyenne ».** On utilise ici l'opérateur **MAX**, c'est-à-dire que pour chaque valeur linguistique de sortie concernée on va prendre la valeur **maximum** des degrés d'appartenance.

Le résultat de l'agrégation des règles donne donc la variable **s « petite »** avec un degré d'appartenance de 0.6 en même temps que **« moyenne »** avec un degré d'appartenance de 0.2.

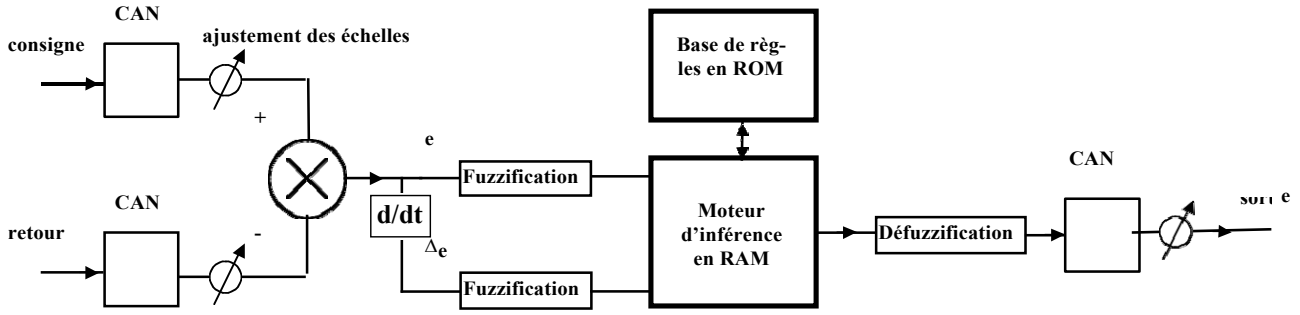
A ce stade, on a donc la sortie définie sous forme linguistique avec des degrés d'appartenance précis. Il faut maintenant passer à une grandeur qui, elle, sera interprétable par l'interface de commande.

Défuzzification



La figure ci-dessus indique comment on interprète l'agrégation des règles. Le sous ensemble **« petite »** est limité par le degré d'appartenance à 0.6 et le sous ensemble **« moyenne »** par 0.2 (on prend le **MIN** entre la fonction d'appartenance de la valeur linguistique de la sortie concernée et le μ trouvé par l'agrégation des règles). On obtient ainsi toute la surface grisée. Pour obtenir le signal de commande à envoyer à l'interface, on utilise le plus souvent la règle **« du centre de masse »** c'est-à-dire que l'on calcule le barycentre de la surface pour obtenir la valeur de la commande.

Conclusion



Les grandeurs réelles **consigne** et **retour** sont numérisées avec ces convertisseurs AN puis l'**erreur** (éventuellement la **variation de l'erreur**) est fuzzifiée. La base de règles qui sera stockée en ROM sera activée et le moteur d'inférence permettra d'élaborer une sortie qui sera ensuite défuzzifiée et convertie en signal analogique.

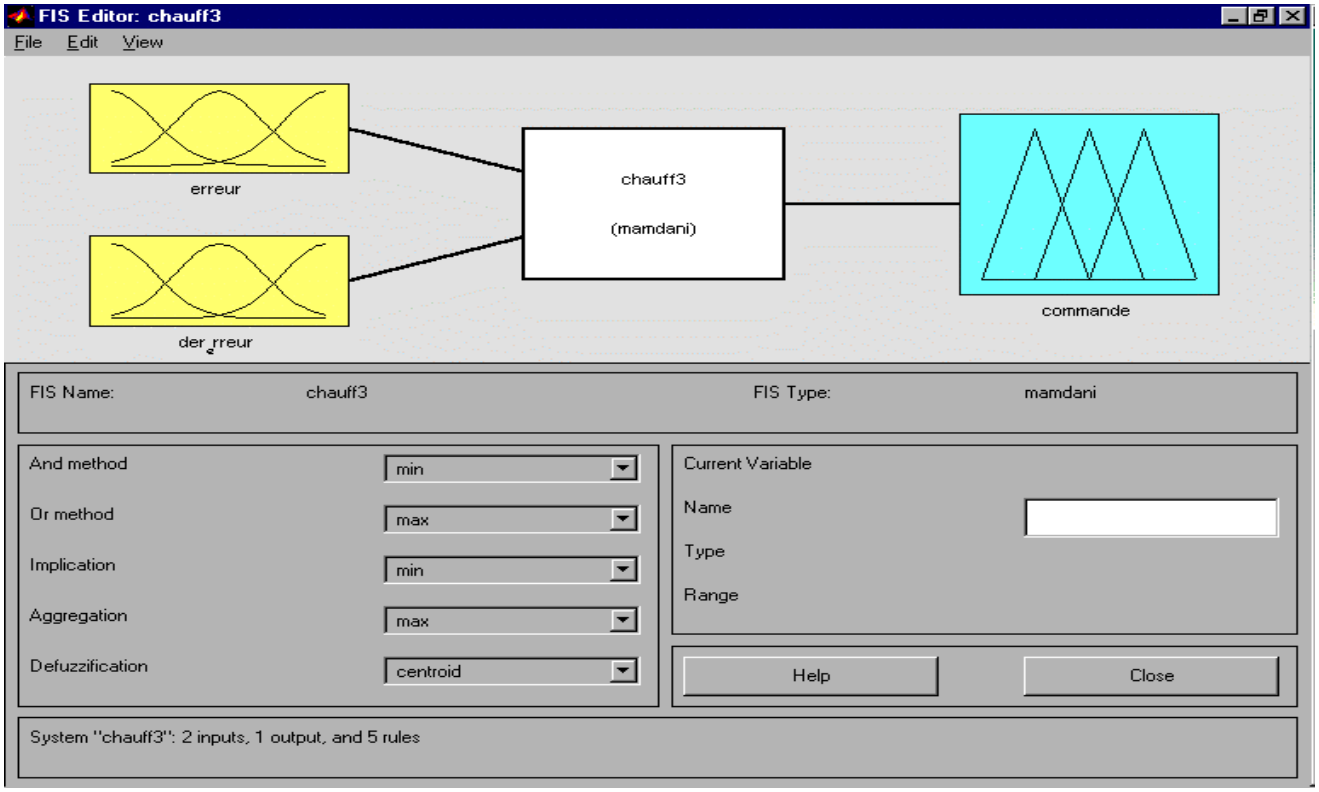
On peut constater que dans son principe même le contrôleur à logique floue n'est pas linéaire.

En effet l'activation des règles d'inférence n'est pas un processus linéaire pas plus que la traduction mathématique que l'on en donne (algorithme MAX MIN).

On a utilisé ici l'algorithme **MIN MAX** (**MIN** pour la conjonction des prémisses et l'implication des règles, **MAX** pour l'agrégation des règles). Mais, il existe d'autres possibilités : on peut par exemple utiliser l'algorithme **PROD MAX** (**PROD** pour la conjonction des prémisses et l'implication des règles**, MAX** pour l'agrégation des règles) c'est-à-dire que l'on fait le produit des degrés d'appartenance obtenu avec chaque règle pour définir le degré d'appartenance pour la sortie.

)

Utilisation de la boite à outils « Fuzzy logic »



On commence par taper **fuzzy** à l'invite de la fenêtre de commande de MATLAB : >> ce qui lance l'éditeur de contrôleur flou : Fis Editor

Opérations de fuzzification

étape 1

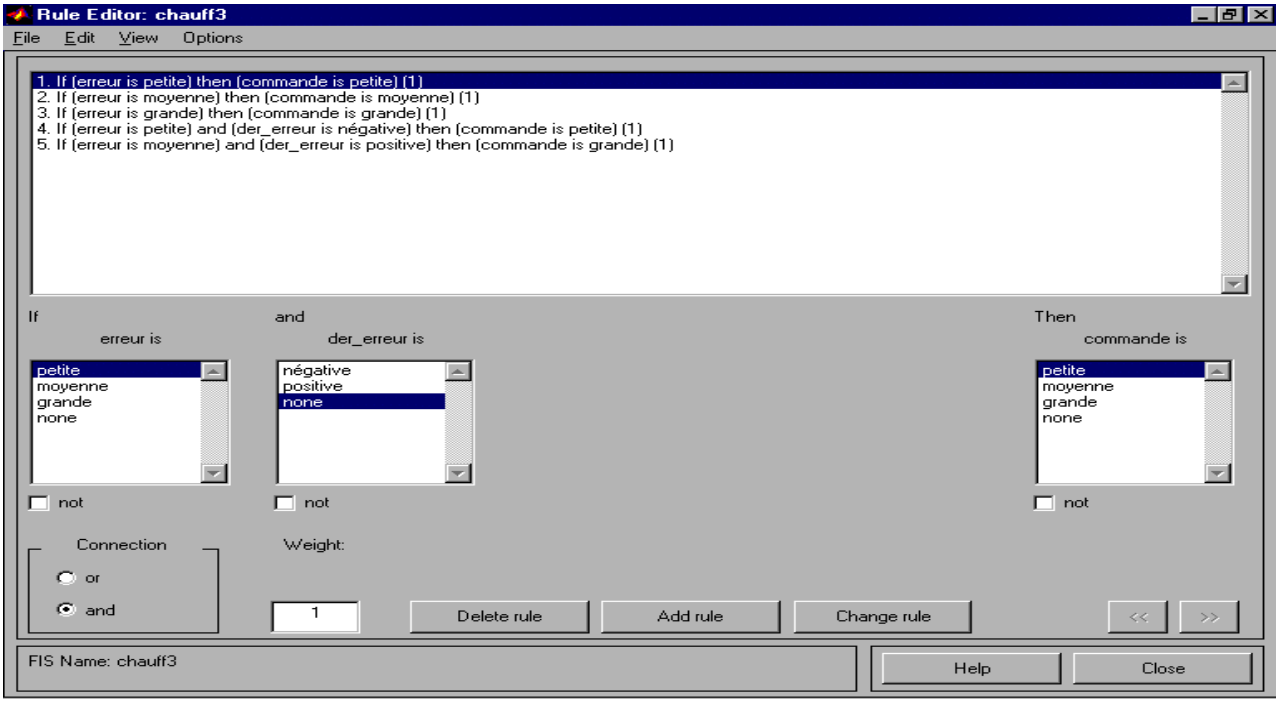
La commande **Edit** permettra de choisir le nombre d'entrées-sorties (2 entrées et 1 sortie dans notre application). On pourra renommer à loisir ces différentes variables.

étape 2

Double-cliquer sur chaque variable d'entrée ou de sortie pour définir les valeurs linguistiques et la forme des fonctions d'appartenance. Pour cela utiliser dans le nouveau menu **Edit** la commande **Add MFs**, ce qui signifie ajouter des fonctions d'appartenance. On pourra choisir des fonctions triangulaires [trimf], trapézoïdales [trapmf], etc…

étape 3

Double-cliquer sur le bloc central (mandani) pour entrer les règles d'inférences choisies.



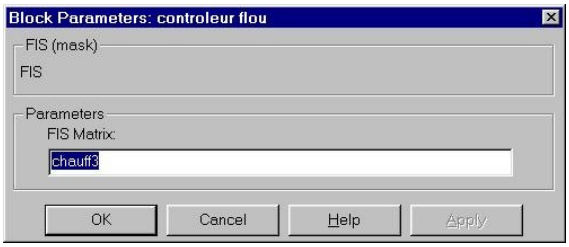
étape 4

Pour visualiser le comportement du contrôleur ainsi créé, on pourra utiliser dans le menu **View**, la commande **View rules**. Il est alors possible de fixer les valeurs des variables d'entrées, de voir quelles sont les règles d'inférences activées et de lire la valeur de la tension de commande correspondante.

Sauvegarde du contrôleur dans l'espace de travail matlab

Pour que le contrôleur ainsi créé soit disponible dans un modèle simulink, il faut qu'il soit sauvegardé dans l'espace de travail. Pour cela dans le menu **File**, utiliser l'option **Save to workspace**. Ainsi, son contenu sera mis dans un fichier de type « *.fis » *(**chauff3.fis** dans l'exemple ci-dessus)* auquel il sera fait appel dans simulink.

Pour incorporer le correcteur flou dans son propre modèle simulink, le plus simple est d'en copier un à partir d'une démo de MATLAB. Ensuite, on double-cliquera sur le modèle en entrant dans la fenêtre le nom du fichier (***chauff3*** comme indiqué précédemment)



Le modèle définitif est (enfin !) prêt à fonctionner