Simulation DEVS d'une machine à induction triphasée

Laurent Capocchi

Simulation DEVS d'une machine à induction triphasée

Comparaison avec Simulink

Capocchi Laurent
Laboratoire UMR CNRS 6134
Université de Corse
2 Novembre 2007

Table des matières

1 Introduction

Ce rapport montre qu'il est possible de simuler le modèle mathématique d'une machine asynchrone triphasé avec le *formalisme DEVS (Discrete EVent System Specification)* et d'obtenir les mêmes résultats que ceux obtenus avec le logiciel *Matlab/Simulink*. Le modèle mathématique est obtenu par une mise en équation des courants statoriques et rotoriques du modèle orienté circuit proposé dans [7]. Il consiste en un système complet de *six équations différentielles linéaires du* 1^{er} *ordre à six inconnues*. Le modèle DEVS est basé sur la méthode de quantification QSS *(Quantized State System)* proposée par le professeur B.P Zeigler [8] et adaptée à la simulation des systèmes hybrides par E. Kofman dans [4, 6, 5, 3].

Nous utilisons une implémentation en langage Python [`http://www.python.org`] des modèles DEVS de la librairie "Continuous" proposée dans le logiciel PowerDEVS [2]. Parmis ces modèles, l'intégrateur utilise la méthode de quantification des variables de sorties QSS qui permet d'éviter la quantification temporelle à pas fixe classique utilisée par Matlab/Simulink. Cette méthode nécessite cependant de fixer le pas de quantification *qd* de chaque intégrateur qui conditionne la convergence du système modélisé.

La simulation se fait par le biais d'une implémentation de l'algorithme optimisé (arbre de simulation à plat) du simulateur PythonDEVS [1]. Les résultats de simulations seront comparés aux résultats *Matlab/Simulink* obtenus dans le précédent papier intitulé *"Simulation Maple d'une machine asynchrone triphasée"*.

Nous discuterons de notre approche et du calibrage de notre système dans une dernière section.

2 Modélisation DEVS

Cette section décrit le modèle mathématique et le modèle *DEVS* de la machine à induction triphasée que nous allons simuler par la suite. Nous présentons le modèle mathématique puis la modélisation de celui-ci dans le formalisme DEVS.

2.1 Modèle mathématique

Le modèle orienté circuit simplifié d'une machine à induction triphasée est donné sur la figure 1. Ce modèle peut être séparé en trois parties distinctes : Le stator, le rotor et le couplage magnétique entre ces deux entités.

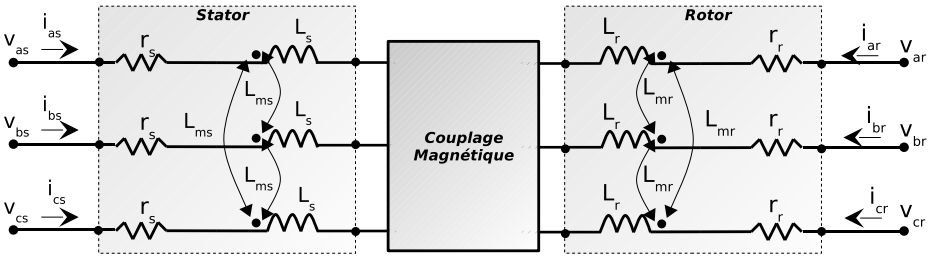


FIG. 1 – Modèle orienté circuit

Le stator est alimenté par un système triphasé équilibré composé des tensions sinusoïdales $v_{as}(t)$, $v_{bs}(t)$ et $v_{cs}(t)$:

$$v_i(t) = V_m \sin(2\pi ft - (i-1)\frac{2\pi}{3}) \;\; i = 1,2 ou 3$$

Les tensions composées constituent un système triphasé équilibré en avance de $\frac{\pi}{6}$ sur le système de tensions simples est d'amplitude $\sqrt{3}$ fois plus grande.

Chaque phase est caractérisée par une résistance r_s et une inductance L_s. Les interactions magnétiques entre chaque phase du stator sont fonction d'une inductance mutuelle L_{ms} et des courants statoriques voisins.

De même, le rotor est alimenté par un système triphasé équilibré composé des tensions sinusoïdales $v_{ar}(t)$, $v_{br}(t)$ et $v_{cr}(t)$. Chaque phase est caractérisée par une résistance r_r et une inductance L_r. Les interactions magnétiques entre chaque phase du rotor sont fonction d'une inductance mutuelle L_{mr} et des courants rotoriques voisins.

Les effets du rotor sur le stator *(resp. du stator sur le rotor)* sont fonction d'une inductance mutuelle L_{sr} *(resp.* L_{rs}*)*, des courants rotoriques $i_{ar}(t)$, $i_{br}(t)$ et $i_{cr}(t)$ *(resp.* $i_{as}(t)$, $i_{bs}(t)$ *et* $i_{cs}(t)$*)* et de la position électrique du rotor $\theta_r(t)$.

La vitesse mécanique $\Omega(t)$ est la solution d'un système de deux équations différentielles du 1er ordre avec pour coefficient, la force de frottement f et l'inertie J. Cette équation n'est pas homogène car elle est égale à la différence entre le couple électromagnétique T_e et le couple mécanique T_l.

Les entités observées en sortie du système sont les courants *(statoriques et rotoriques)* ainsi que le couple électromagnétique de la machine $T_e(t)$. Pour plus de détails sur la modélisation mathématique du système, le lecteur peut se rapporter au rapport précédent intitulé *"Simulation Maple d'une machine asynchrone triphasé"*. Le modèle complet est donné par le système d'équations 1 :

$$\left\{\begin{array}{rl}
v_{as}(t) & = r_s.i_{as} + L_s\frac{d}{dt}i_{as} - \frac{L_{ms}}{2}\left[\frac{d}{dt}i_{bs} + \frac{d}{dt}i_{cs}\right] + L_{sr}\frac{d}{dt}\left[i_{ar}.\cos(\theta_r(t)) + i_{br}.\cos(\theta_r(t)+\frac{2\pi}{3}) + i_{cr}.\cos(\theta_r(t)-\frac{2\pi}{3})\right] \\
v_{bs}(t) & = r_s.i_{bs} + L_s\frac{d}{dt}i_{bs} - \frac{L_{ms}}{2}\left[\frac{d}{dt}i_{as} + \frac{d}{dt}i_{cs}\right] + L_{sr}\frac{d}{dt}\left[i_{ar}.\cos(\theta_r(t)-\frac{2\pi}{3}) + i_{br}.\cos(\theta_r(t)) + i_{cr}.\cos(\theta_r(t)+\frac{2\pi}{3})\right] \\
v_{cs}(t) & = r_s.i_{cs} + L_s\frac{d}{dt}i_{cs} - \frac{L_{ms}}{2}\left[\frac{d}{dt}i_{as} + \frac{d}{dt}i_{bs}\right] + L_{sr}\frac{d}{dt}\left[i_{ar}.\cos(\theta_r(t)+\frac{2\pi}{3}) + i_{br}.\cos(\theta_r(t)-\frac{2\pi}{3}) + i_{cr}.\cos(\theta_r(t))\right] \\
v_{ar}(t) & = r_r.i_{ar} + L_r\frac{d}{dt}i_{ar} - \frac{L_{mr}}{2}\left[\frac{d}{dt}i_{br} + \frac{d}{dt}i_{cr}\right] + L_{rs}\frac{d}{dt}\left[i_{as}.\cos(\theta_r(t)) + i_{bs}.\cos(\theta_r(t)-\frac{2\pi}{3}) + i_{cs}.\cos(\theta_r(t)+\frac{2\pi}{3})\right] \\
v_{br}(t) & = r_r.i_{br} + L_r\frac{d}{dt}i_{br} - \frac{L_{mr}}{2}\left[\frac{d}{dt}i_{ar} + \frac{d}{dt}i_{cr}\right] + L_{rs}\frac{d}{dt}\left[i_{as}.\cos(\theta_r(t)+\frac{2\pi}{3}) + i_{bs}.\cos(\theta_r(t)) + i_{cs}.\cos(\theta_r(t)-\frac{2\pi}{3})\right] \\
v_{cr}(t) & = r_r.i_{cr} + L_r\frac{d}{dt}i_{cr} - \frac{L_{mr}}{2}\left[\frac{d}{dt}i_{ar} + \frac{d}{dt}i_{br}\right] + L_{rs}\frac{d}{dt}\left[i_{as}.\cos(\theta_r(t)-\frac{2\pi}{3}) + i_{bs}.\cos(\theta_r(t)+\frac{2\pi}{3}) + i_{cs}.\cos(\theta_r(t))\right] \\
T_e(t) - T_l & = J.\frac{d}{dt}\Omega(t) + f.\Omega(t) \\
\Omega(t) & = \frac{d}{dt}\theta_r(t)
\end{array}\right. \quad (1)$$

Le couple T_e peut se calculer grâce à la formule donnée ci-dessous :

$$\begin{aligned}
T_e(t) \quad = \quad & -i_{as}(t).p.L_{sr}[i_{ar}(t).\sin(p.\theta_r) + i_{br}(t).\sin(p.\theta_r + \frac{2.\pi}{3}) + i_{cr}(t).\sin(p.\theta_r - \frac{2.\pi}{3})] \\
& -i_{bs}(t).p.L_{sr}[i_{ar}(t).\sin(p.\theta_r - \frac{2.\pi}{3}) + i_{br}(t).\sin(p.\theta_r) + i_{cr}(t).\sin(p.\theta_r + \frac{2.\pi}{3})] \\
& -i_{cs}(t).p.L_{sr}[i_{ar}(t).\sin(p.\theta_r + \frac{2.\pi}{3}) + i_{br}(t).\sin(p.\theta_r - \frac{2.\pi}{3}) + i_{cr}(t).\sin(p.\theta_r)]
\end{aligned} \quad (2)$$

Nous allons à présent donner le schéma bloc DEVS permettant la modélisation et la simulation du système d'équations 1.

2.2 Modèle DEVS

Le modèle DEVS du système d'équations 1 est donné sur la figure 2. On peut distinguer 3 parties principales qui correspondent à la partie stator (modèle couplé à gauche de la figure), la partie rotor (modèle couplé à droite de la figure) et la partie force électromotrice constituée des deux modèles couplés (bloc rectangulaire en bas de la figure). La mécanique est représentée par un modèle couplé nommé 'mécanique' (en bas à gauche) et la partie permettant le calcul du couple électromoteur T_e par un modèle couplé nommé 'couple_electro3' (en bas à droite).

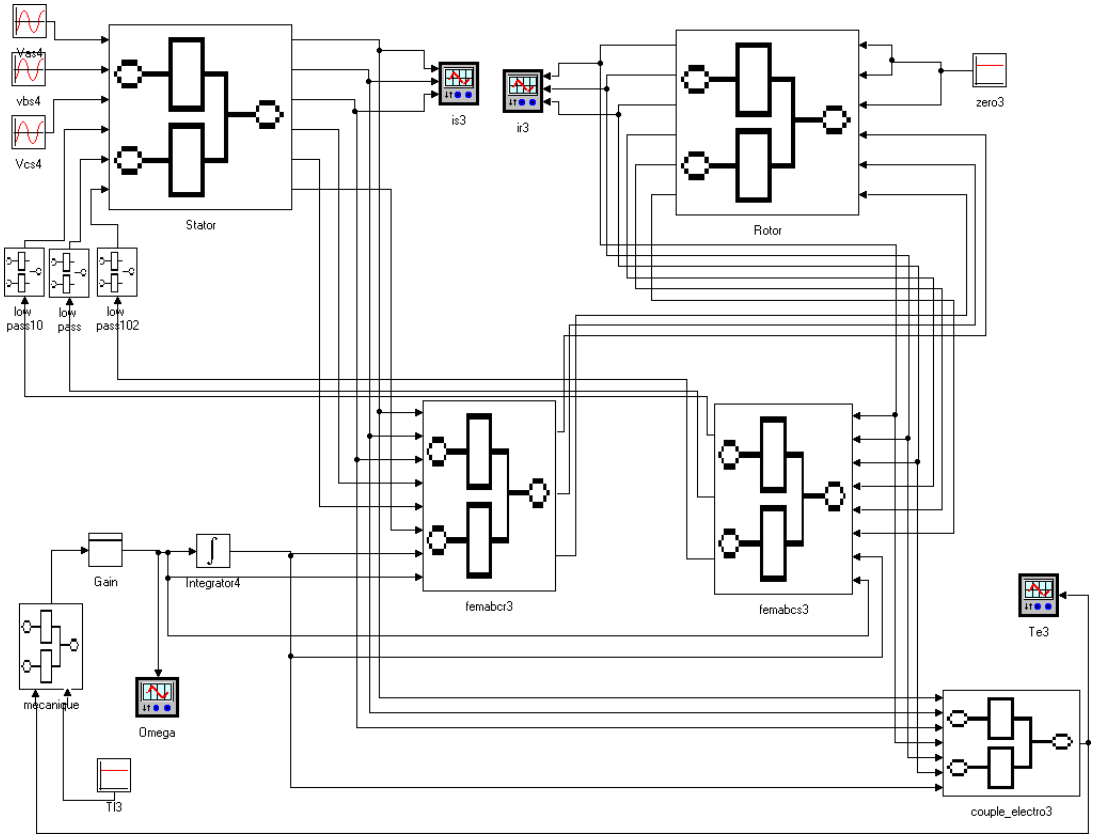
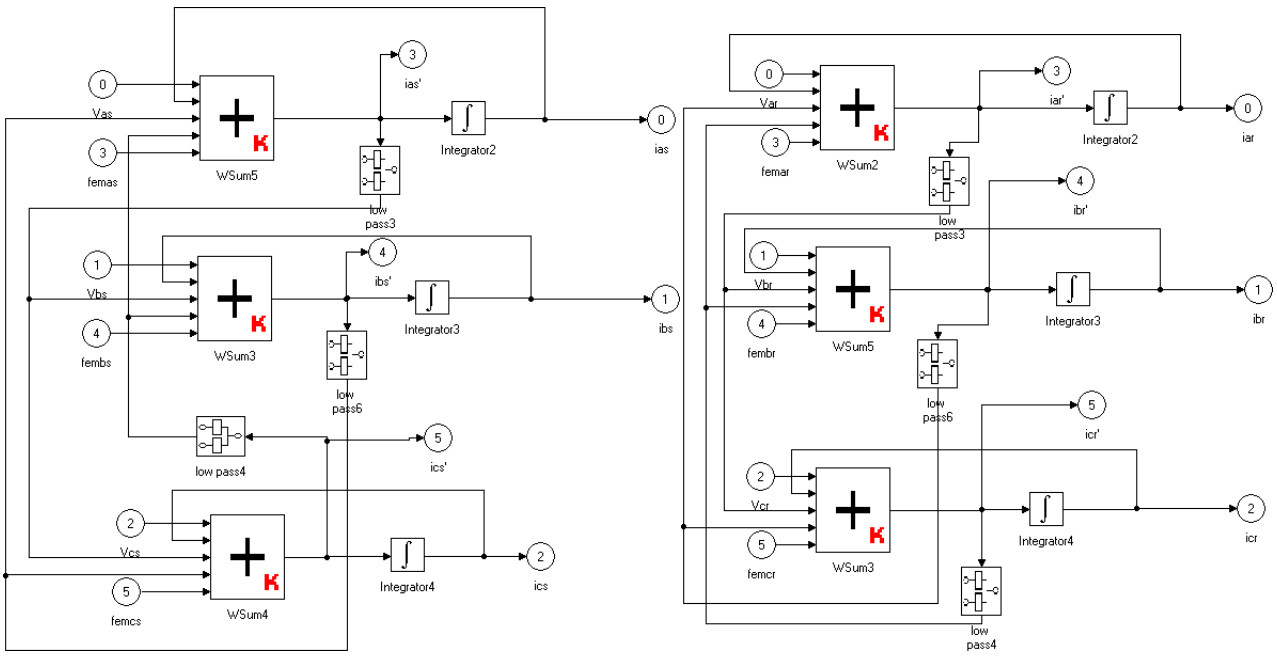


FIG. 2 – Schéma bloc Simulink de la MADA

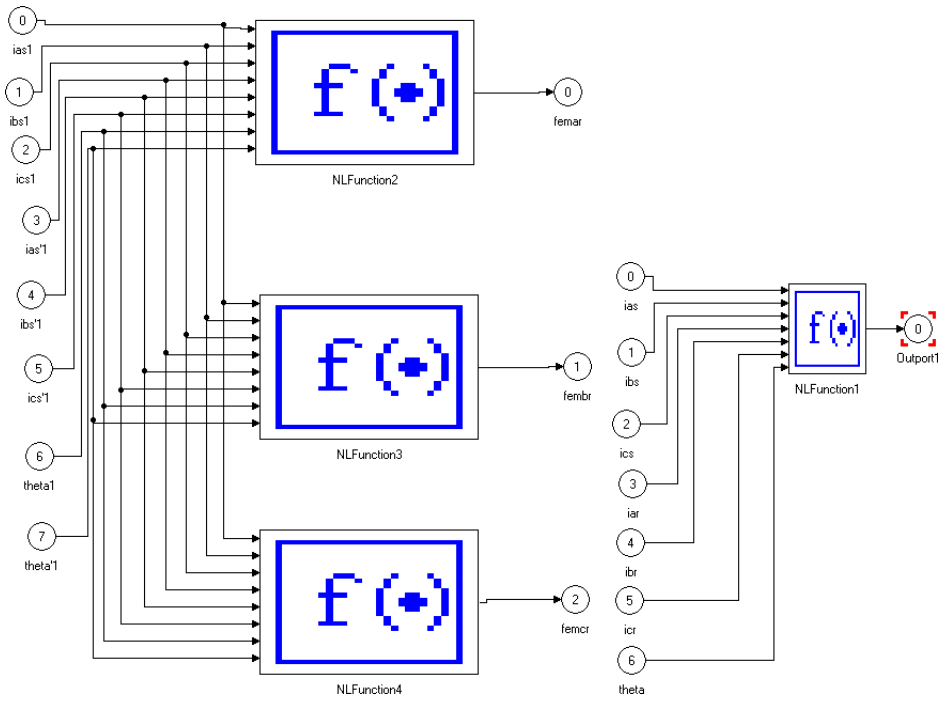
Le schéma de la figure 3 montre le contenu des deux modèles couplés *Stator* et *Rotor*. Ils sont composés des trois phases qui présentent un sommateur, un intégrateur et surtout un filtre passe bas indispensable à la correction du bruit numérique introduit en début de simulation par l'intégrateur. Le calibrage de ces filtres sera discutée plus tard.



(a) Stator (b) Rotor

FIG. 3 – Modèles couplés du stator et du rotor

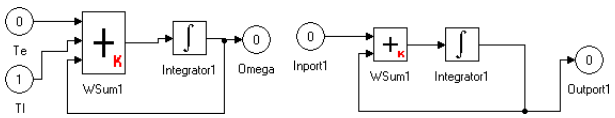
La figure 4 montre le contenu des modèles couplés FEM et Couple. Ils présentent les modèles atomiques NLFunction permettant de modéliser les fonctions non linaires pour le calcul des inductances mutuelles et pour le calcul du couple T_e donnée par la formule 2.



(a) FEM (b) Couple

FIG. 4 – Modèles couplés de la force électromotrice et du couple

La figure 5 montre le contenu du modèle couplé *Mécanique* et du modèle couplé *LowPass*.



(a) Mecanique (b) Low Pass

FIG. 5 – Modèles couplés de la partie mécanique et du filtre passe bas

Les modèles atomiques V_{as}, V_{bs} et V_{cs} permettent la génération des tensions sinusoïdales $v_{as}(t)$, $v_{bs}(t)$ et $v_{cs}(t)$. Les modèles atomiques I_{as}, I_{bs}, T_e et *Omega* permettent d'observer les courants statoriques et rotoriques (resp. $I_{s\{a,b,c\}}$ et $I_{r\{a,b,c\}}$) ainsi que le couple électromoteur T_e et la vitesse mécanique $\Omega(t)$. Nous observerons éventuellement les forces électromotrices coté stator et rotor (resp. fem_s et fem_r). Nous allons à présent simuler le système dans différentes configurations.

Remarque : Les figures sont des captures d'écran du logiciel PowerDEVS que nous avons utilisé dans un premier temps afin de valider nos modèles. Par la suite nous avons développé un environnement en langage python permettant de reproduire le comportement du logiciel PowerDEVS. Cette implémentation nous permettra par la suite de rendre plus simple le développement des algorithmes de simulations concurrentes de fautes dont on ne parle pas ici mais qui reste notre objectif final.

3 Simulation DEVS

Cette section présente quelques modes de fonctionnement simple de la machine MADA 5.5 KW (Machine Asynchrone à Double Alimentation). Pour plus de détails sur l'explication physiques des phénomènes, le lecteur peut se rapporter aux papiers précédants. Enfin, nous présentons une discussion sur le calibrage du modèle *(coefficients des filtres et des pas de quantification des intégrateurs)* ainsi que sur la comparaison des résultats obtenus avec Matlab/Simulink. Les valeurs des paramètres du système 1 choisi pour les besoins de la simulation sont résumées dans le tableau 1 :

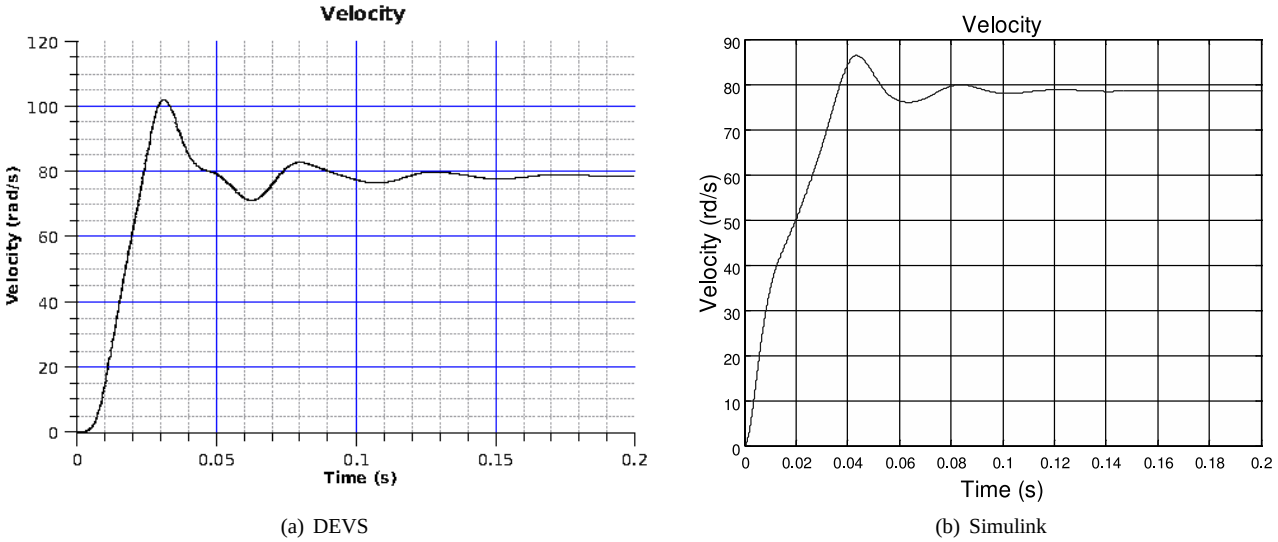
Tension composée efficace (U_m)	380V
Fréquence (f)	50Hz
Pôles (p)	4
Coefficient d'inertie (J)	0.1kg.m²
Coefficient d'atténuation (f)	0.001Nm.s/rad
Couple de charge nominale (T_{ln})	73Nm
Résistance au stator (r_s)	0.528Ω
Résistance au rotor (r_r)	0.282Ω
Inductance au stator (L_s)	0.04732H
Inductance au rotor (L_r)	0.01452H
Inductance magnétique au stator (L_{ms})	0.01732H
Inductance magnétique au stator (L_{mr})	0.005852H
Inductance mutuelle ($L_{sr} = L_{rs}$)	0.02259H

TAB. 1 – Valeurs des paramètres pour la machine 5.5 KW

3.1 Résultats de simulations

Nous considérons le système complet avec l'intégration des équations mécaniques. Nous allons simuler ce système pour un couple de charge $T_l = 0\,Nm$ et $T_l = 75\,Nm$ et une vitesse initiale nulle. **Les résultats graphiques sont obtenus avec dq=0.001 (pour tous les intégrateurs du système) pour DEVS et une méthode ode5 (pas fixe de 10^{-5} seconde) pour Simulink.**

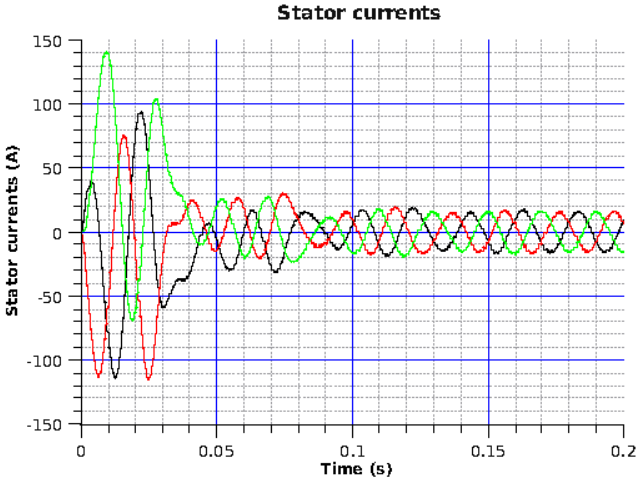
3.1.1 Machine à vide avec vitesse initiale nulle



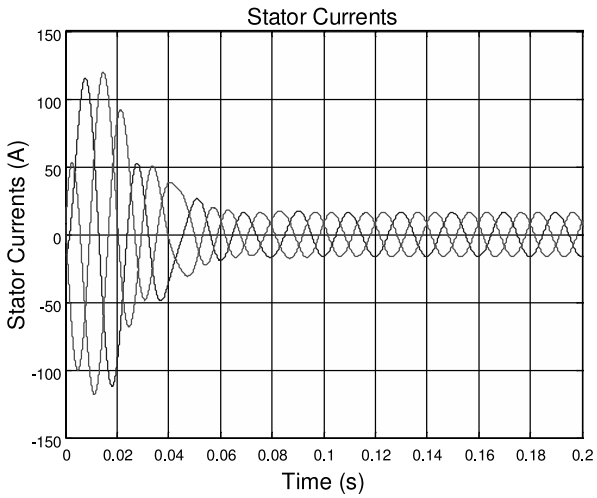
(a) DEVS (b) Simulink

FIG. 6 – Vitesse mécanique $\Omega(t)$ pour $T_l = 0$

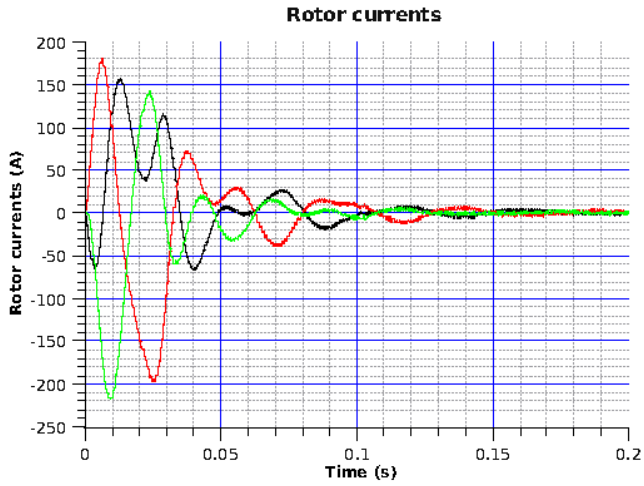
Lorsque $T_l = 0$, cela correspond à l'absence de charge sur la machine. Lorsque la machine est alimentée, elle passe par un régime transitoire durant lequel la vitesse rotorique augmente pour se stabiliser comme on peux le voir sur la figure 6(a).



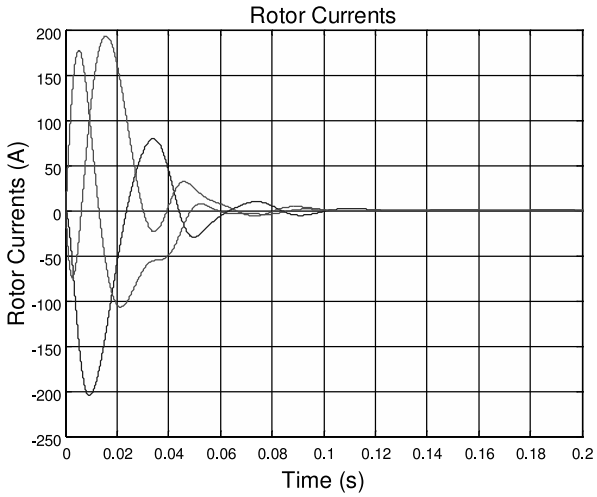
(a) DEVS



(b) Simulink



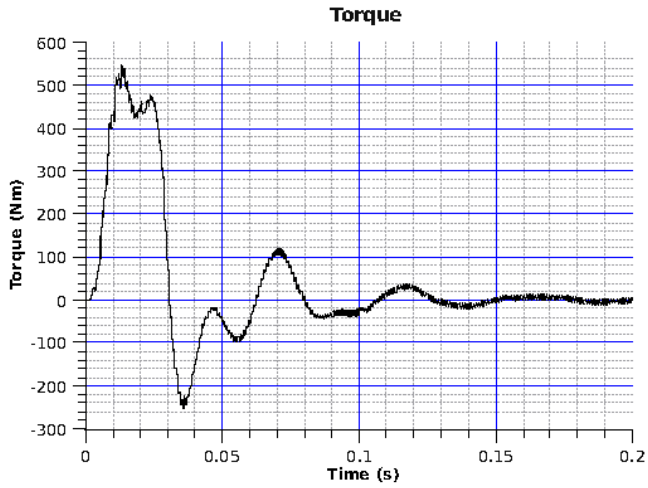
(c) DEVS



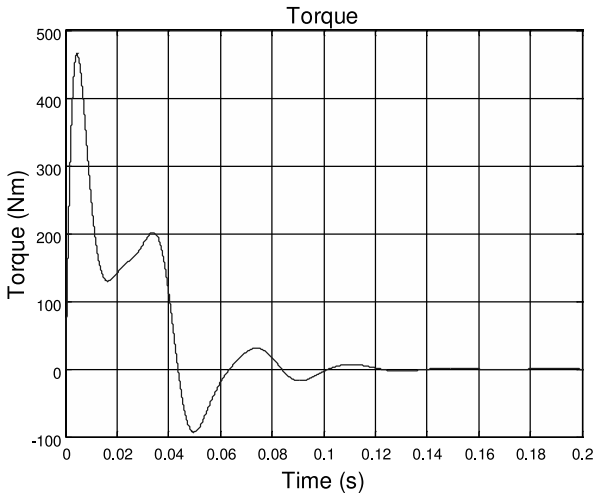
(d) Simulink

FIG. 7 – Courants statoriques et rotoriques à vide, $T_l = 0$

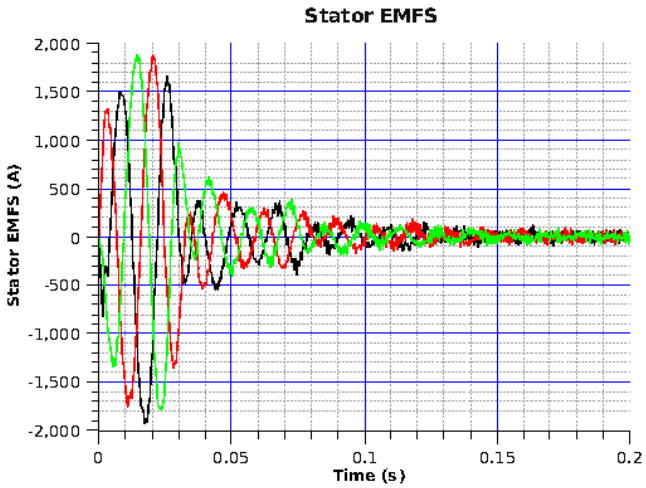
Les courants statoriques et rotoriques présentent le même régime permanent. Les courants statoriques oscillent entre -24 et 24 ampère, alors que les courants rotoriques tendent vers une valeur nulle.



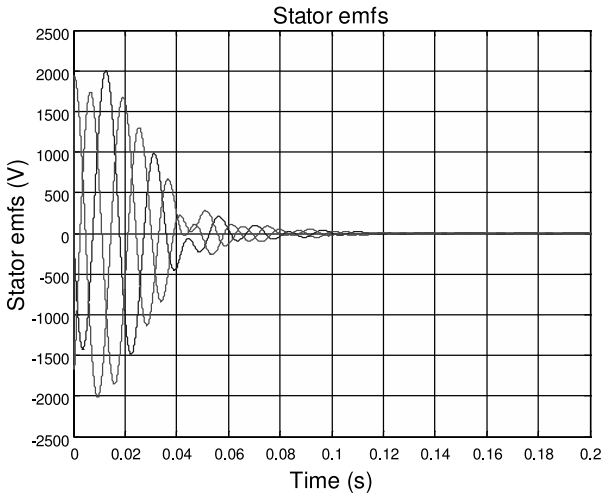
(a) DEVS



(b) Simulink



(c) DEVS



(d) Simulink

FIG. 8 – Couple électromoteur et force électromotrice coté stator avec $T_l = 0\,Nm$

On remarque que les courbes du couple éléctromoteur ne sont pas identiques. Les forces éléctromotrices du coté du stator évoluent de la même manière.

Les temps de simulation sont résumés dans le tableau 2 ci dessous :

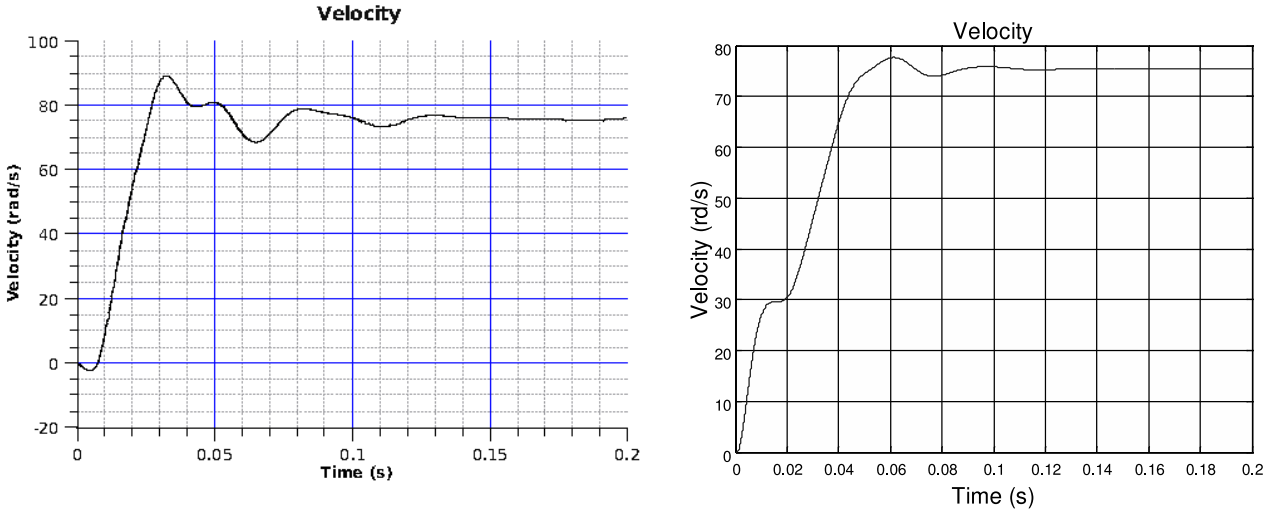
dq	**DEVS**		
	real	*user*	*sys*
0.001	**87m13.446s**	**77m49.724s**	**1m10.660s**
0.01	32m47.700s	29m54.212s	0m35.150s
0.1	14m13.573s	11m43.832s	0m14.105s
1	div	div	div

TAB. 2 – Temps de simulation DEVS en fonction des pas de quantification

Les simulations sont effectuées avec ordinateur portable DELL Latitude D800 possédant un processeur pentium M cadencé à 1.4 GHz. Les temps de simulation pour dq=1 ne sont pas notés car ils sont trop importants du fait d'une divergence des résultats de la simulation. L'erreur numérique en fonction de la quantification sera discutée plus bas.

3.1.2 Machine en pleine charge avec vitesse initiale nulle

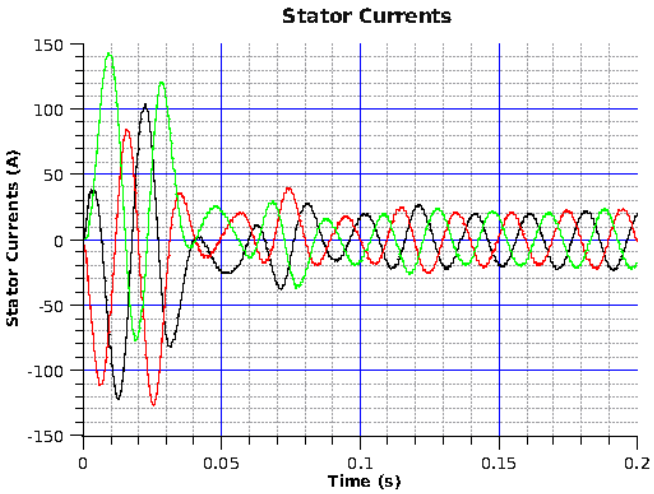
Nous imposons à présent un couple de charge $T_l = 75\,Nm$ égale au couple de charge nominal.



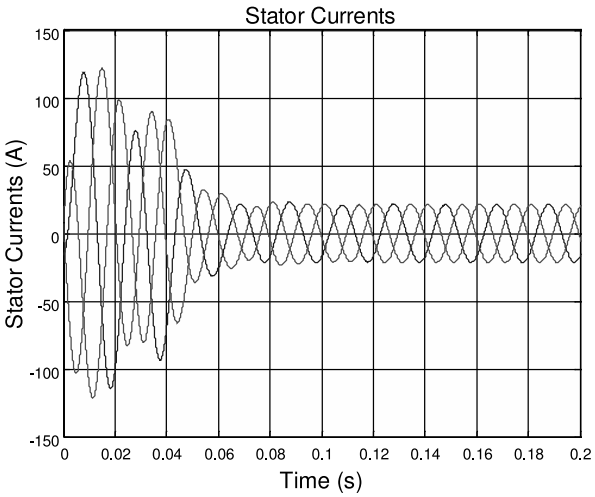
(a) DEVS (b) Simulink

FIG. 9 – Vitesse mécanique Ω(t) pour $T_l = 75$

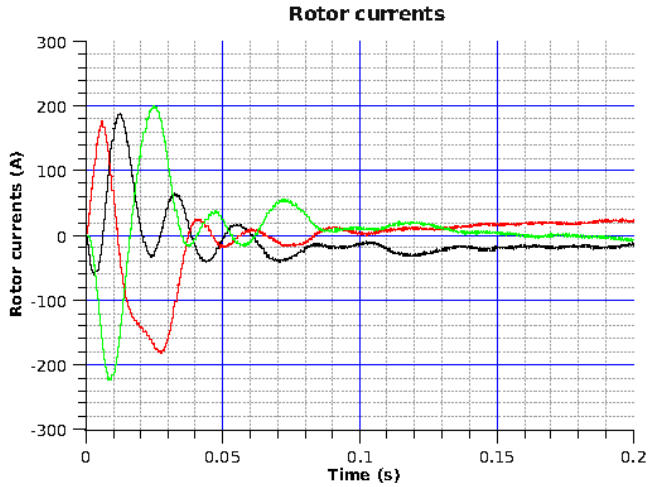
La vitesse mécanique n'a pas le même régime transitoire mais converge vers la même valeur : 75.5 rad/s.



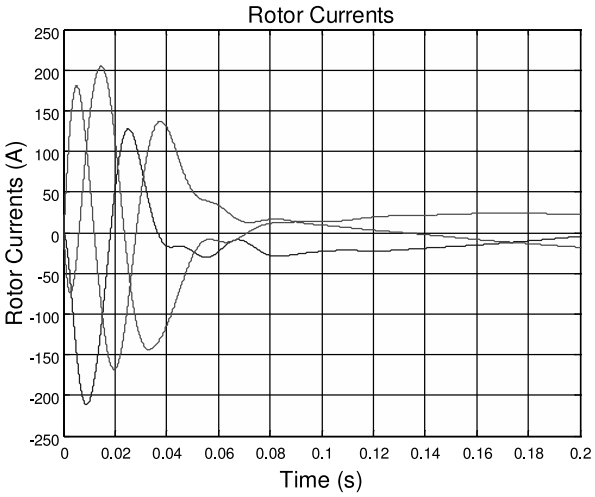
(a) DEVS



(b) Simulink



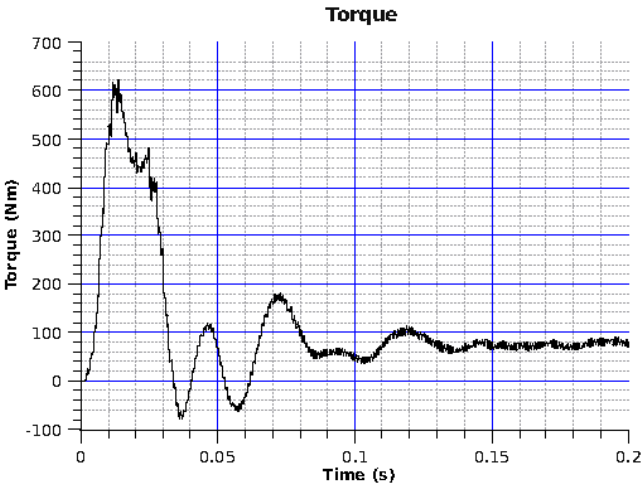
(c) DEVS



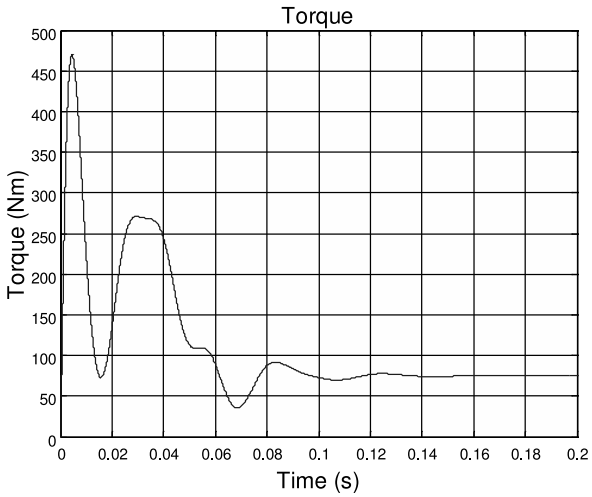
(d) Simulink

FIG. 10 – Courants statoriques et rotoriques avec $T_l = 75\,Nm$

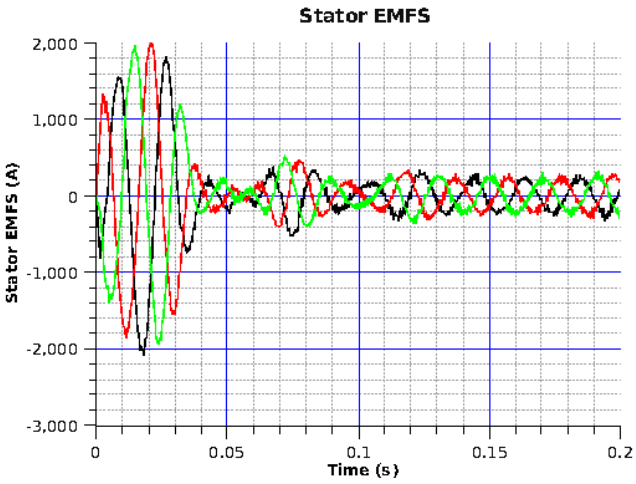
Les courants présentés sur la figure 10 montrent que l'imposition d'un couple de charge $T_l = 75\,Nm$ implique un appel de courant au niveau du stator et de ce fait une augmentation de l'amplitude des courants rotoriques. Une fois de plus les courants ne présentent pas les mêmes régimes transitoires mais ils convergent vers les mêmes valeurs.



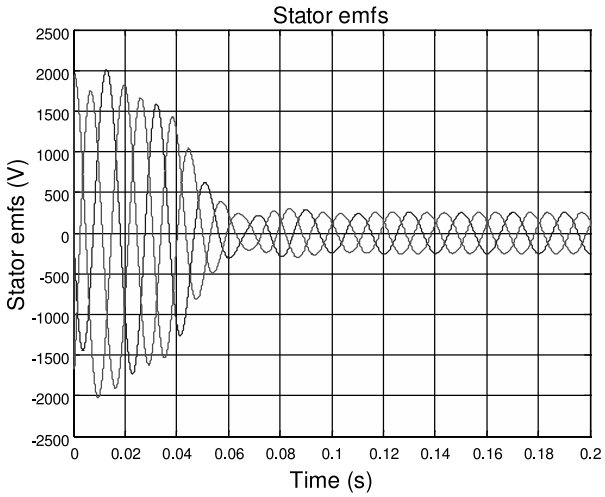
(a) DEVS



(b) Simulink



(c) DEVS



(d) Simulink

FIG. 11 – Couple électromoteur et force électromotrice coté stator avec $T_l = 75\,Nm$

Nous remarquons que le couple électromoteur ne converge pas vers la même valeur et nous discuterons de ce résultat par la suite.

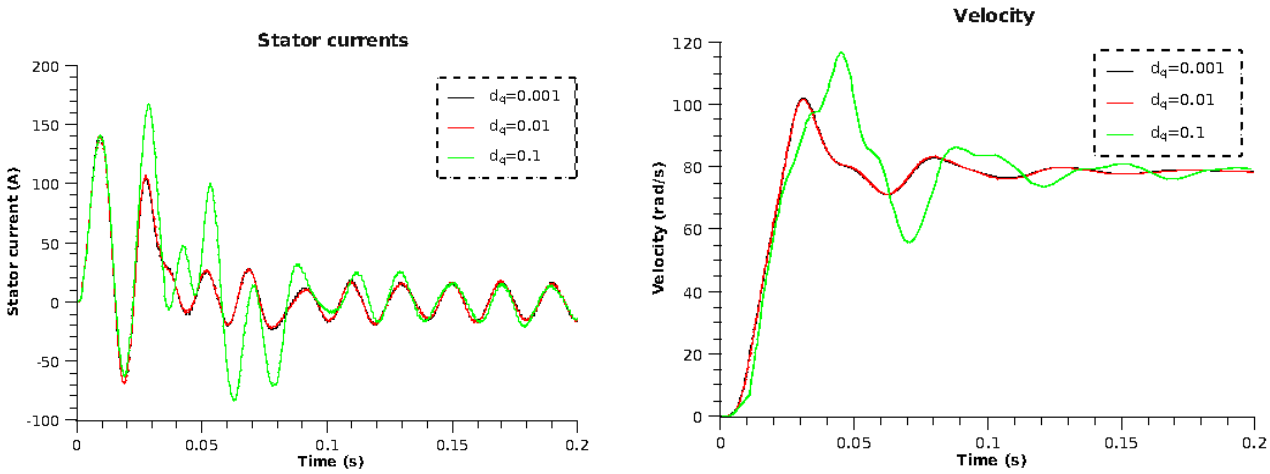
dq	**DEVS**		
	real	*user*	*sys*
0.001	**113m25.916s**	**93m38.399s**	**2m3.936s**
0.01	32m17.818s	29m49.524s	0m35.290s
0.1	13m5.053s	11m40.780s	0m14.313s
1	div	div	div

TAB. 3 – Temps de simulation DEVS en fonction des pas de quantification

Les ordres de grandeurs des temps de simulation sont identiques que ceux observé dans le tableau 2. Le temps de simulation est multiplié par 8 lorsqu'on passe de dq=0.1 à dq=0.001.

3.2 Discussion

Les tableaux 2 et 3 montrent les temps de simulations DEVS en fonction des pas de quantifications *qd* utilisés dans tous les intégrateurs du modèle complet (dans les filtres, au stator, au rotor...). Ces résultats sont en accords avec le fait que plus dq et petit plus les temps de simulations sont importants.



(a) courant statorique ias

(b) vitesse mécanique

FIG. 12 – Erreur en fonction de dq pour $T_l = 0$

La figure 12 montre l'erreur commise sur les signaux lorsque le pas de quantification augment. On peut noter que seul le régime transitoire est affecté par la valeur de dq. **On peut donc, à priori, choisir un pas de quantification de l'ordre de 0.1 puisque seul le régime permanent est étudié par la suite.** Le choix du pas de quantification est basé sur l'expérience et un travail devra être fait de manière à automatiser le choix de dq afin d'optimiser les temps de simulations.

Comme le montre le schéma 2, nous avons introduit des filtres pas bas au sein du système pour répondre à un problème de blocage de la simulation. En effet, au début de la simulation, le résultat de l'intégration (au sein du stator par exemple) donne des valeurs importantes qui impliquent une activation des modèles à des temps qui dépasse le temps final de simulation. Par conséquent, et en accord avec les algorithmes DEVS, aucun modèle est activé et la simulaiton ne démarre pas. L'introduction de filtre permet de lisser les valeurs brutales et de donner des temps d'activation des modèles raisonnable (inférieur au temps finale de simulation). La nouvelle version du logiciel Simulink effectue automatiquement ces rectifications mais les anciennes versions nécessitaient la même procédure.

L'introduction de filtre passe bas permet donc de débloquer la simulation mais peu introduire une erreur numérique suivant la valeur des coefficients des filtres. Dans ce papier les coefficients des filtres sont tous égaux à 2000 de manière à pouvoir laisser passer le maximum d'informations. Les valeurs de ces coefficients sont également suffisantes pour débloquer la simulation car les temps de réponse des filtres sont largement inférieurs aux constantes de temps des sous-systèmes filtrés. A l'heure actuelle, nous pouvons dire que le choix des valeurs des coefficients influe sur la réponse mais aussi sur le temps de simulation. Plus les coefficients des filtres sont bas, plus les réponses filtrés sont tronquées et plus les temps de simulations sont importants du fait peut être de la forme exagéré des signaux traités. Ces affirmations reposent sur l'expérience car il n'existe aucune méthode déterministe pour fixer ces coefficients et un travail devra être fait dans ce sens.

Nous sommes également conscient qu'une étude des performances de simulation aurait été plus juste avec un choix personnalisé des pas de quantification. Nous avons choisi de fixer des valeurs des pas identiques pour tous les intégrateurs alors que les dynamiques des signaux d'entrées sont différentes.

En ce qui concerne la comparaison des résultats avec Simulink, nous pouvons dire que les régimes permanents sont similaires à l'exception du couple électromoteur T_e. Les régimes transitoires évoluent différament mais présente le mme enveloppe. **La simulation DEVS permet donc d'aboutir aux mêmes résultats que Simulink.**

4 Conclusion

Nous pouvons conclure sur le fait que la simulation DEVS conduit aux mêmes résultats que ceux obtenus avec Matlab/Simulink *(sous certaines conditions liées au choix de dq)*. A l'heure actuelle, bien que le simulateur DEVS que nous avons implémenté soit basé sur un algorithme optimisé, les temps de simulation sont trop importants. Cela est dû à la méthode de quantification QSS qui calcule le temps d'activation des modèles en fonction de la valeur du pas de quantification. Le meilleur moyen d'accélérer les simulations est d'augmenter ces pas de quantification au risque d'augmenter l'erreur numérique conduisant à une divergence des résultats de simulation. De plus, la présence des filtres passe bas, qui ont été introduit afin d'éviter le blocage des simulations, ont une influence sur les temps de simulation : plus les pas de quantification des filtres sont bas plus les temps de simulations sont importants. Les coefficients des filtres ont également une influence sur la stabilité des résultats. Lorsque les coefficients sont élevés (fréquence de coupure haute), les solutions sont plus réaliste (plus proche de celle obtenu avec Simulink) et moins bruités numériquement.

Simulink est basé sur des algorithmes utilisant des matrices et procède par inversion de ces matrices pour obtenir, par une discrétisassions en temps à pas fixe, des résultats avec des temps beaucoup plus raisonnables. DEVS utilise la simulation à événements discrets qui devrait accélérer la simulation en terme de pas de calcul. Cependant, face à des systèmes bouclés complexes comme celui de la machine asynchrone, la méthode QSS demande des pas de quantification petits et donc des temps de simulation importants.

L'avantage d'utiliser DEVS n'apparaît pas clairement face aux résultats obtenus avec Matlab/Simulink. Cela dit, DEVS offre la possibilité d'implémenter des algorithmes comparatifs et concurrents permettant de faire du diagnostic de pannes. C'est la raison pour laquelle nous avons choisis cette approche.

En perspective, pour diminuer les temps de simulation important avec DEVS, il faut donc se pencher sur une méthode de détermination ou d'adaptation des pas de quantification au cours de la simulation. De plus, nous sommes en train de développer une méthode de simulation concurrente des défauts les plus courants qui peuvent apparaître au sein de ces systèmes asynchrones.

Références

[1] Jean-Sébastien Bolduc and Hans Vangheluwe. pythonDEVS : A modeling and simulation package for classical hierarchal DEVS. In *Rapport technique, MSDL, Université de McGill*, juin 2001.

[2] E.Kofman, M.Lapadula, and E.Pagliero. PowerDEVS : A DEVS-based environment for hybrid system modeling and simulation. Technical report, Rosario National University, 2003. http ://fceia.unr.edu.ar/~kofman/pubs.html.

[3] E. Kofman. Quantization-based simulation of differential algebraic equation systems. *Trans. Soc. Comput. Simul. Int.*, 79(7) :363–376, 2003.

[4] E. Kofman. A third order discrete event simulation method for continuous system simulation. part i : Theory. Technical report, 2005. http ://fceia.unr.edu.ar/~kofman/pubs.html.

[5] Ernesto Kofman. A second order approximation for DEVS simulation of continuous systems. *Trans. Soc. Comput. Simul. Int.*, 78(2) :76–89, 2002.

[6] Ernesto Kofman. Discrete event simulation of hybrid systems. *SIAM Journal on Scientific Computing*, 25(5) :1771–1797, 2004.

[7] A. Yazidi, H. Henao, G.A. Capolino, D. Casadei, and F. Filippetti. Double-fed three-phase induction machine abc model for simulation and control purposes. In *Proceedings of IEEE Industrial Electronics Conference (IECON'05)*, volume 4, pages 2560–2565, November 2005.

[8] Bernard P. Zeigler. DEVS theory of quantized systems. Technical report, ACIMS Laboratory, University of Arizona, 2004. www.acims.arizona.edu/PUBLICATIONS/ CDRLs/UnivArizonaCDRL1.doc.

Annexes

2 novembre 2007

Fichier param.py

```
# -*- coding : iso-8859-1 -*-
# definition des macros
VERBOSE = False # si 1 print à l'ecran
FINAL_TIME = 0.2 # temps final de simulation
##5.5
Um =380
freq=50
J =0.1
F =0.0001
rsa =0.528
rsb =0.528
rsc =0.528
Ls =0.04732
Lms =2*0.01736
rra =0.282
rrb =0.282
rrc =0.282
Lr =0.01452
Lmr =2*0.005852
Lsr =Lrs=0.02259
Tl =0 # 75 en pleine charge
p =4
Is_n=25 Ir_n=Is_n
```

Fichier main.py

```
Machine = Master()
# All models
Sa = Machine.addSubModel( SinGen(Um*sqrt(2.0)/sqrt(3.0),freq,0.0,20,"QSS2") )
Sb = Machine.addSubModel( SinGen(Um*sqrt(2.0)/sqrt(3.0),freq,-(2*pi/3),20,"QSS2") )
Sc = Machine.addSubModel( SinGen(Um*sqrt(2.0)/sqrt(3.0),freq,-(4*pi/3),20,"QSS2") )
Zeroa = Machine.addSubModel( ConstGen(0) )
Zerob = Machine.addSubModel( deepcopy(Zeroa) )
Zeroc = Machine.addSubModel( deepcopy(Zeroa) )
Zerofema = Machine.addSubModel( ConstGen(0) )
Zerofemb = Machine.addSubModel( ConstGen(0) )
Zerofemc = Machine.addSubModel( ConstGen(0) )
Zero = Machine.addSubModel( ConstGen(0) )
S = Machine.addSubModel( Stator() )
R = Machine.addSubModel( Rotor() )
FEM_S = Machine.addSubModel( FEM_S(Lsr) )
FEM_R = Machine.addSubModel( FEM_R(Lrs) )
m=[2000,-2000]
```

```
LPF1 = Machine.addSubModel( LPF(m, 500, "QSS3") )
LPF2 = Machine.addSubModel( LPF(m, 500, "QSS3" ) )
LPF3 = Machine.addSubModel( LPF(m, 500, "QSS3" ) )
Meca = Machine.addSubModel( Meca([1/J, -1/J, -F/J], 0.01, "QSS3") )
Integr = Machine.addSubModel( Integrator("QSS3", 0.01, 0) )
Gain = Machine.addSubModel( Gain(p) )
T = Machine.addSubModel( ConstGen(Tl) )
Couple = Machine.addSubModel( Couple() )
V_To_Disk = Machine.addSubModel( To_Disk(3,"vas") )
IS_To_Disk = Machine.addSubModel( To_Disk(3,"is") )
IR_To_Disk = Machine.addSubModel( To_Disk(3,"ir") )
Omega_To_Disk = Machine.addSubModel( To_Disk(1,"omega") )
Te_To_Disk = Machine.addSubModel( To_Disk(1,"Te") )
Theta_To_Disk = Machine.addSubModel( To_Disk(1,"theta") )
FEM_To_Disk = Machine.addSubModel( To_Disk(3,"fem") )
FFEM_To_Disk = Machine.addSubModel( To_Disk(3,"filter_fem") )
# Connecting
Machine.connectPorts(Sa.OUT, S.IN1)
Machine.connectPorts(Sb.OUT, S.IN2)
Machine.connectPorts(Sc.OUT, S.IN3)
Machine.connectPorts(FEM_S.OUT1, LPF1.IN)
Machine.connectPorts(FEM_S.OUT2, LPF2.IN)
Machine.connectPorts(FEM_S.OUT3, LPF3.IN)
Machine.connectPorts(LPF1.OUT, S.IN4)
Machine.connectPorts(LPF2.OUT, S.IN5)
Machine.connectPorts(LPF3.OUT, S.IN6)
Machine.connectPorts(S.OUT1, FEM_R.IN1)
Machine.connectPorts(S.OUT2, FEM_R.IN2)
Machine.connectPorts(S.OUT3, FEM_R.IN3)
Machine.connectPorts(S.OUT4, FEM_R.IN4)
Machine.connectPorts(S.OUT5, FEM_R.IN5)
Machine.connectPorts(S.OUT6, FEM_R.IN6)
####-----------------------------------------
#FEM_R
Machine.connectPorts(Integr.OUT, FEM_R.IN7)
Machine.connectPorts(Gain.OUT, FEM_R.IN8)
#FEM_S
Machine.connectPorts(Integr.OUT, FEM_S.IN7)
Machine.connectPorts(Gain.OUT, FEM_S.IN8)
#Gain
Machine.connectPorts(Gain.OUT, Integr.IN)
#Mecanique
Machine.connectPorts(T.OUT, Meca.IN2)
Machine.connectPorts(Couple.OUT, Meca.IN1)
Machine.connectPorts(Meca.OUT, Gain.IN)
# Couple
Machine.connectPorts(S.OUT1, Couple.IN1)
Machine.connectPorts(S.OUT2, Couple.IN2)
Machine.connectPorts(S.OUT3, Couple.IN3)
Machine.connectPorts(R.OUT1, Couple.IN4)
Machine.connectPorts(R.OUT2, Couple.IN5)
Machine.connectPorts(R.OUT3, Couple.IN6)
Machine.connectPorts(Integr.OUT, Couple.IN7)
###-----------------------------------------
# Rotor
Machine.connectPorts(Zeroa.OUT, R.IN1)
Machine.connectPorts(Zerob.OUT, R.IN2)
Machine.connectPorts(Zeroc.OUT, R.IN3)
Machine.connectPorts(FEM_R.OUT1, R.IN4)
Machine.connectPorts(FEM_R.OUT2, R.IN5)
Machine.connectPorts(FEM_R.OUT3, R.IN6)
```

```
Machine.connectPorts(R.OUT1, FEM_S.IN1)
Machine.connectPorts(R.OUT2, FEM_S.IN2)
Machine.connectPorts(R.OUT3, FEM_S.IN3)
Machine.connectPorts(R.OUT4, FEM_S.IN4)
Machine.connectPorts(R.OUT5, FEM_S.IN5)
Machine.connectPorts(R.OUT6, FEM_S.IN6)
# writting
Machine.connectPorts(S.OUT1, IS_To_Disk.IPorts[0])
Machine.connectPorts(S.OUT2, IS_To_Disk.IPorts[1])
Machine.connectPorts(S.OUT3, IS_To_Disk.IPorts[2])
Machine.connectPorts(Meca.OUT, Omega_To_Disk.IPorts[0])
Machine.connectPorts(Integr.OUT, Theta_To_Disk.IPorts[0])
Machine.connectPorts(FEM_R.OPorts[0],FEM_To_Disk.IPorts[0])
Machine.connectPorts(FEM_R.OPorts[1],FEM_To_Disk.IPorts[1])
Machine.connectPorts(FEM_R.OPorts[2],FEM_To_Disk.IPorts[2])
Machine.connectPorts(LPF1.OUT,FFEM_To_Disk.IPorts[0])
Machine.connectPorts(LPF2.OUT,FFEM_To_Disk.IPorts[1])
Machine.connectPorts(LPF3.OUT,FFEM_To_Disk.IPorts[2])
Machine.connectPorts(Couple.OUT, Te_To_Disk.IPorts[0])
Machine.connectPorts(R.OUT1, IR_To_Disk.IPorts[0])
Machine.connectPorts(R.OUT2, IR_To_Disk.IPorts[1])
Machine.connectPorts(R.OUT3, IR_To_Disk.IPorts[2])
#Simulate
Root = Simulator(Machine)
Root.simulate(FINAL_TIME)
```

Fichier Stator.py

```
## All models
Ka=[1/Ls, -rsa/Ls, Lms/(2*Ls), Lms/(2*Ls), -1/Ls]
Kb=[1/Ls, -rsb/Ls, Lms/(2*Ls), Lms/(2*Ls), -1/Ls]
Kc=[1/Ls, -rsc/Ls, Lms/(2*Ls), Lms/(2*Ls), -1/Ls]
Wa = self.addSubModel( WSum(Ka,5) )
Wb = self.addSubModel( WSum(Kb,5) )
Wc = self.addSubModel( WSum(Kc,5) )
Ia = self.addSubModel( Integrator("QSS3",0.001,0) )
Ib = self.addSubModel( Integrator("QSS3",0.001,0) )
Ic = self.addSubModel( Integrator("QSS3",0.001,0) )
K=[2000,-2000]
Fa = self.addSubModel(LPF(K,500,"QSS3"))
Fb = self.addSubModel(LPF(K,500,"QSS3"))
Fc = self.addSubModel(LPF(K,500,"QSS3"))
# in/out ports
self.IN1 = self.addInPort()
self.IN2 = self.addInPort()
self.IN3 = self.addInPort()
self.IN4 = self.addInPort()
self.IN5 = self.addInPort()
self.IN6 = self.addInPort()
self.OUT1 = self.addOutPort()
self.OUT2 = self.addOutPort()
self.OUT3 = self.addOutPort()
self.OUT4 = self.addOutPort()
self.OUT5 = self.addOutPort()
self.OUT6 = self.addOutPort()
##connecting
self.connectPorts(self.IN1, Wa.IPorts[0])
self.connectPorts(self.IN2, Wb.IPorts[0])
self.connectPorts(self.IN3, Wc.IPorts[0])
self.connectPorts(Ia.OUT, Wa.IPorts[1])
```

```
self.connectPorts(Ib.OUT, Wb.IPorts[1])
self.connectPorts(Ic.OUT, Wc.IPorts[1])
self.connectPorts(Wa.OUT, Fa.IPorts[0])
self.connectPorts(Wb.OUT, Fb.IPorts[0])
self.connectPorts(Wc.OUT, Fc.IPorts[0])
self.connectPorts(Fa.OPorts[0], Wb.IPorts[2])
self.connectPorts(Fa.OPorts[0], Wc.IPorts[3])
self.connectPorts(Fb.OPorts[0], Wa.IPorts[2])
self.connectPorts(Fb.OPorts[0], Wc.IPorts[2])
self.connectPorts(Fc.OPorts[0], Wa.IPorts[3])
self.connectPorts(Fc.OPorts[0], Wb.IPorts[3])
self.connectPorts(self.IN4, Wa.IPorts[4])
self.connectPorts(self.IN5, Wb.IPorts[4])
self.connectPorts(self.IN6, Wc.IPorts[4])
self.connectPorts(Wa.OUT, Ia.IN)
self.connectPorts(Wb.OUT, Ib.IN)
self.connectPorts(Wc.OUT, Ic.IN)
self.connectPorts(Ia.OUT,self.OUT1)
self.connectPorts(Ib.OUT,self.OUT2)
self.connectPorts(Ic.OUT,self.OUT3)
self.connectPorts(Wa.OUT,self.OUT4)
self.connectPorts(Wb.OUT,self.OUT5)
self.connectPorts(Wc.OUT,self.OUT6)
```

Fichier Rotor.py

```
## All models
Ka=[1/Lr, -rra/Lr, Lmr/(2*Lr), Lmr/(2*Lr), -1/Lr]
Kb=[1/Lr, -rrb/Lr, Lmr/(2*Lr), Lmr/(2*Lr), -1/Lr]
Kc=[1/Lr, -rrc/Lr, Lmr/(2*Lr), Lmr/(2*Lr), -1/Lr]
Wa = self.addSubModel( WSum(Ka,5) )
Wb = self.addSubModel( WSum(Kb,5) )
Wc = self.addSubModel( WSum(Kc,5) )
Ia = self.addSubModel( Integrator("QSS3",0.001,0) )
Ib = self.addSubModel( Integrator("QSS3",0.001,0) )
Ic = self.addSubModel( Integrator("QSS3",0.001,0) )
K=[2000,-2000]
Fa = self.addSubModel(LPF(K,500,"QSS3"))
Fb = self.addSubModel(LPF(K,500,"QSS3"))
Fc = self.addSubModel(LPF(K,500,"QSS3"))
# in/out ports
self.IN1 = self.addInPort()
self.IN2 = self.addInPort()
self.IN3 = self.addInPort()
self.IN4 = self.addInPort()
self.IN5 = self.addInPort()
self.IN6 = self.addInPort()
self.OUT1 = self.addOutPort()
self.OUT2 = self.addOutPort()
self.OUT3 = self.addOutPort()
self.OUT4 = self.addOutPort()
self.OUT5 = self.addOutPort()
self.OUT6 = self.addOutPort()
##connecting
self.connectPorts(self.IN1, Wa.IPorts[0])
self.connectPorts(self.IN2, Wb.IPorts[0])
self.connectPorts(self.IN3, Wc.IPorts[0])
self.connectPorts(Ia.OUT, Wa.IPorts[1])
self.connectPorts(Ib.OUT, Wb.IPorts[1])
self.connectPorts(Ic.OUT, Wc.IPorts[1])
```

```
self.connectPorts(Fa.OPorts[0], Wb.IPorts[2])
self.connectPorts(Fa.OPorts[0], Wc.IPorts[3])
self.connectPorts(Fb.OPorts[0], Wa.IPorts[2])
self.connectPorts(Fb.OPorts[0], Wc.IPorts[2])
self.connectPorts(Fc.OPorts[0], Wa.IPorts[3])
self.connectPorts(Fc.OPorts[0], Wb.IPorts[3])
self.connectPorts(self.IN4, Wa.IPorts[4])
self.connectPorts(self.IN5, Wb.IPorts[4])
self.connectPorts(self.IN6, Wc.IPorts[4])
self.connectPorts(Wa.OUT, Ia.IN)
self.connectPorts(Wb.OUT, Ib.IN)
self.connectPorts(Wc.OUT, Ic.IN)
self.connectPorts(Wa.OUT, Fa.IPorts[0])
self.connectPorts(Wb.OUT, Fb.IPorts[0])
self.connectPorts(Wc.OUT, Fc.IPorts[0])
self.connectPorts(Ia.OUT,self.OUT1)
self.connectPorts(Ib.OUT,self.OUT2)
self.connectPorts(Ic.OUT,self.OUT3)
self.connectPorts(Wa.OUT,self.OUT4)
self.connectPorts(Wb.OUT,self.OUT5)
self.connectPorts(Wc.OUT,self.OUT6)
```

Fichier Couple.py

```
expr = str(Master.sdb.Lsr)+"*-"+str(Master.sdb.p)+
"*(u0*(u3*sin(u6)+u4*sin(u6+2.0944)+u5*sin(u6-2.0944))+
u1*(u3*sin(u6-2.0944)+u4*sin(u6)+u5*sin(u6+2.0944))+
u2*(u3*sin(u6+2.0944)+u4*sin(u6-2.0944)+u5*sin(u6)))"
self.IN1 = self.addInPort() # ias
self.IN2 = self.addInPort() # ibs
self.IN3 = self.addInPort() # ics
self.IN4 = self.addInPort() # iar
self.IN5 = self.addInPort() # ibr
self.IN6 = self.addInPort() # icr
self.IN7 = self.addInPort() # theta
self.OUT = self.addOutPort() # Te
self.NLF=self.addSubModel(NLFunction(expr, 7))
self.connectPorts(self.IN1,self.NLF.IPorts[0])
self.connectPorts(self.IN2,self.NLF.IPorts[1])
self.connectPorts(self.IN3,self.NLF.IPorts[2])
self.connectPorts(self.IN4,self.NLF.IPorts[3])
self.connectPorts(self.IN5,self.NLF.IPorts[4])
self.connectPorts(self.IN6,self.NLF.IPorts[5])
self.connectPorts(self.IN7,self.NLF.IPorts[6])
self.connectPorts(self.NLF.OUT, self.OUT)
```

Fichier Mecanique.py

```
self.IN1 = self.addInPort() # Te
self.IN2 = self.addInPort() # Tl
self.OUT = self.addOutPort() # Omega
self.I=self.addSubModel(Integrator(methode,dq,0))
self.W=self.addSubModel(WSum(K,3))
self.connectPorts(self.IN1,self.W.IPorts[0])
self.connectPorts(self.IN2,self.W.IPorts[1])
self.connectPorts(self.W.OUT, self.I.IN)
self.connectPorts(self.I.OUT, self.W.IPorts[2])
self.connectPorts(self.I.OUT, self.OUT)
```

Fichier FEM.py

```
self.expr1=str(Lrs)+"*(-u7*(u0*sin(u6)+u1*sin(u6-2.0944)+
u2*sin(u6+2.0944))+u3*cos(u6)+u4*cos(u6-2.0944)+u5*cos(u6+2.0944))"
self.expr2=str(Lrs)+"*(-u7*(u0*sin(u6+2.0944)+u1*sin(u6)+
u2*sin(u6-2.0944))+u3*cos(u6+2.0944)+u4*cos(u6)+u5*cos(u6-2.0944))"
self.expr3=str(Lrs)+"*(-u7*(u0*sin(u6-2.0944)+u1*sin(u6+2.0944)+
u2*sin(u6))+u3*cos(u6-2.0944)+u4*cos(u6+2.0944)+u5*cos(u6))"
## All models
self.NLF1 = self.addSubModel(NLFunction(self.expr1,8))
self.NLF2 = self.addSubModel(NLFunction(self.expr2,8))
self.NLF3 = self.addSubModel(NLFunction(self.expr3,8))
# in/out ports
self.IN1 = self.addInPort(), self.IN2 = self.addInPort()
self.IN3 = self.addInPort(), self.IN4 = self.addInPort()
self.IN5 = self.addInPort(), self.IN6 = self.addInPort()
self.IN7 = self.addInPort(), self.IN8 = self.addInPort()
self.OUT1 = self.addOutPort()
self.OUT2 = self.addOutPort()
self.OUT3 = self.addOutPort()
##connecting
self.connectPorts(self.IN1, self.NLF1.IPorts[0])
self.connectPorts(self.IN2, self.NLF1.IPorts[1])
self.connectPorts(self.IN3, self.NLF1.IPorts[2])
self.connectPorts(self.IN4, self.NLF1.IPorts[3])
self.connectPorts(self.IN5, self.NLF1.IPorts[4])
self.connectPorts(self.IN6, self.NLF1.IPorts[5])
self.connectPorts(self.IN7, self.NLF1.IPorts[6])
self.connectPorts(self.IN8, self.NLF1.IPorts[7])
self.connectPorts(self.IN1, self.NLF2.IPorts[0])
self.connectPorts(self.IN2, self.NLF2.IPorts[1])
self.connectPorts(self.IN3, self.NLF2.IPorts[2])
self.connectPorts(self.IN4, self.NLF2.IPorts[3])
self.connectPorts(self.IN5, self.NLF2.IPorts[4])
self.connectPorts(self.IN6, self.NLF2.IPorts[5])
self.connectPorts(self.IN7, self.NLF2.IPorts[6])
self.connectPorts(self.IN8, self.NLF2.IPorts[7])
self.connectPorts(self.IN1, self.NLF3.IPorts[0])
self.connectPorts(self.IN2, self.NLF3.IPorts[1])
self.connectPorts(self.IN3, self.NLF3.IPorts[2])
self.connectPorts(self.IN4, self.NLF3.IPorts[3])
self.connectPorts(self.IN5, self.NLF3.IPorts[4])
self.connectPorts(self.IN6, self.NLF3.IPorts[5])
self.connectPorts(self.IN7, self.NLF3.IPorts[6])
self.connectPorts(self.IN8, self.NLF3.IPorts[7])
self.connectPorts(self.NLF1.OUT, self.OPorts[0])
self.connectPorts(self.NLF2.OUT, self.OPorts[1])
self.connectPorts(self.NLF3.OUT, self.OPorts[2])
```