Journal : Alternative libre à MATLAB

7 oct. 2026
27

Posté par **uneTanche** (**site web personnel**, **Mastodon**) le 07 octobre 2026 à 12:08. **Licence CC By-SA.**
Étiquettes : **jupyter**, **scilab**, **matplotlib**, **numpy**, **python**, **matlab**

Sommaire

- Avant-propos
- Intro
- La simulation en automatique
  - Matlab/simulink
  - Scilab/xcos
  - Python/Jupyter
    - Présentation
    - Modélisation du moteur DC
    - Modélisation du régulateur
    - Simulation complète
- Bilan
- Autres pistes

Avant-propos

Je ne suis pas un spécialiste de l'automatique, au mieux un amateur éclairé. J'ai eu besoin de mettre au point différents régulateurs pour des projets de contrôle moteur et je trouvais qu'il manquait d'exemples/retour d'expérience sur les alternatives à MATLAB (en français). C'est dans ce cadre que je vous propose cette synthèse subjective des solutions que j'ai testées.

L'article original est disponible ici

Intro

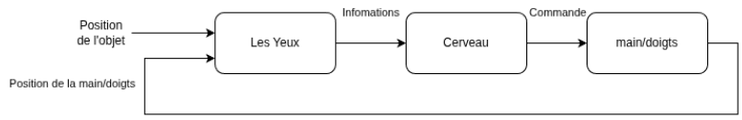
Dans cet article, je dresse un bilan de ma recherche de solution
alternative à matlab/simulink [W] pour la simulation d'algorithme de contrôle/régulation.

Se connecter
Proposer un contenu
Identifiant
Identifiant
Mot de passe
Mot de passe
Connexion automatique
Se connecter
Pas de compte ? S'inscrire...

> **Pourquoi simuler?**
> La mise au point d'asservissement ou de régulation comporte des risques. Il y a notamment un risque d'instabilité qui peut entraîner la destruction du système. Ce qui peut s'avérer coûteux et même parfois dangereux pour l'opérateur. Donc pour limiter les risques lors de la conception, on réalise sa mise au point via **une simulation numérique**.

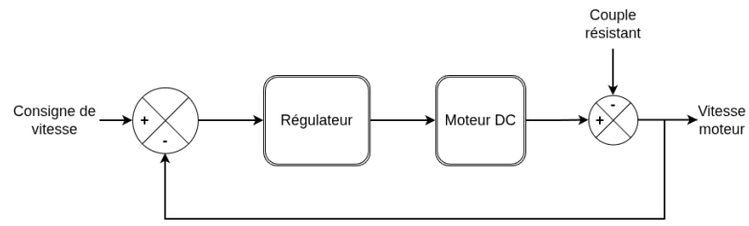
Commençons par une petite mise en contexte. Si vous n'avez aucune idée de ce qu'est l'**automatique**, je vous laisse aller lire l'article wikipédia [W]. C'est un vaste domaine qui couvre notamment, l'asservissement et la régulation de systèmes.

Un exemple: le fait d'attraper un objet avec ses doigts peut-être représenté
comme un montage en *asservissement*. Le capteur est l'oeil, l'actionneur le
bras et le cerveau le centre de calcul. La méthode classique pour documenter
ce type de fonctionnement est de le représenter sous la forme d'un **schéma fonctionnel**:



Ce type de schéma permet de mettre en évidence les boucles de rétro-action ainsi que les différents éléments composant le système et leur interface avec le monde extérieur.

Pour prendre un autre exemple, plus concret, on va s'intéresser à la régulation en vitesse d'un moteur DC. Il servira aussi de référence tout au long de cet article. Son schéma fonctionnel est le suivant:



La simulation en automatique

Commençons par définir nos attentes quant aux solutions de simulation. Pour ma part, j'ai deux critères principaux:

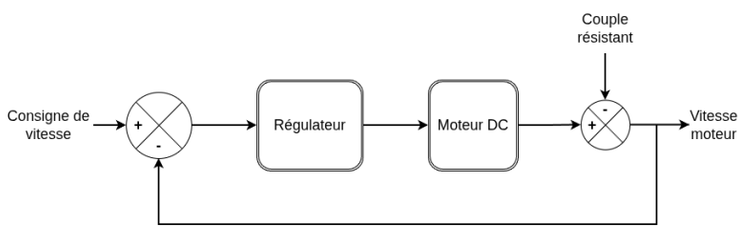
1. De bien simuler! Là c'est le rôle du solveur. Ce n'est pas vraiment mon domaine alors je ne

rentre pas dans les détails
2. D'être lisible. Comme on l'a dit en introduction, en automatique beaucoup de choses sont représentées sous la forme de schéma fonctionnel. Donc il est nécessaire de pouvoir rapidement faire la correspondance entre le code et le schéma

On pourrait aussi citer la rapidité, l'observabilité ou encore la facilité d'itération/modification, mais tous ces points sont plus ou moins liés au solveur et son interface.

Pour effectuer une comparaison des différentes solutions, on va se baser sur notre exemple d'asservissement et régulation en vitesse d'un moteur DC. Un cas d'application concret pourrait être le pilotage du moteur de brosse d'un aspirateur sans fil.

On représente ce pilotage par le schéma suivant:



On définit la fonction de transfert du moteur dans le domaine de *Laplace* tel que:

$$I(s) = \frac{U(s) - Ke\Omega_m}{Ls + R}$$

Avec:
- I: le courant traversant le moteur (A)
- U: la tension aux bornes du moteur (V)
- Ke: la force électromotrice (V/rad/s)
- Ohmega_m: la vitesse de l'arbre du moteur (rpm)
- L: l'inductance du moteur (H)
- R: la résistance du moteur (Ohm)

$$Cem = Kt \times I(s)$$

Avec:
- Kt: la constante de couple (N.m/A)

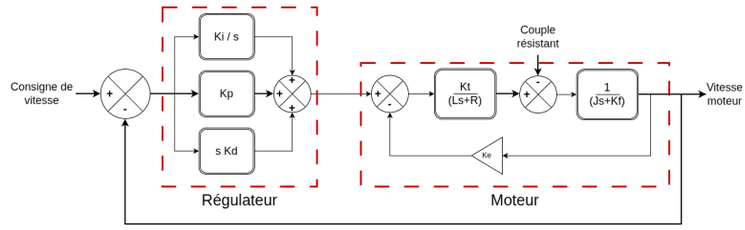
$$\frac{\Omega_m}{Cem - Cr} = \frac{1}{J} Js + Kf$$

Avec:
- Cem: le couple électromagnétique (N.m)
- J: le moment d'inertie (kg.m²)
- Kf: les frottements visqueux (N.m.s)

On en profite pour définir aussi la fonction de transfert du régulateur PID:

$$PID = Kp + \frac{Ki}{s} + sKd$$

Je ne rentre pas dans les détails de ces fonctions de transferts, ni même sur ce qu'est une fonction de transfert. Dans notre cas, on peut la considérer comme le modèle virtuel de notre moteur. En les intégrant au schéma fonctionnel précédent on obtient le schéma suivant:



Maintenant que nous avons défini notre exemple, nous allons implémenter ce système avec différents outils et en comparer l'utilisation. Vu que le logiciel MATLAB/simulink est la référence dans le domaine, on va commencer par lui.

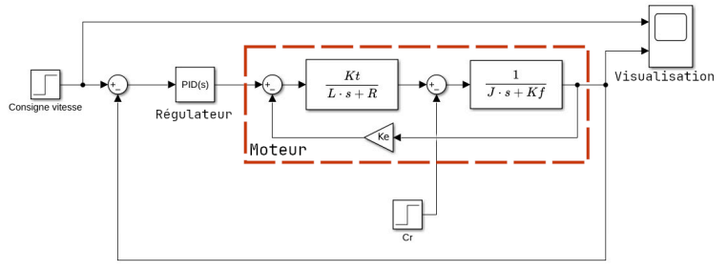
Matlab/simulink

Commençons par quelques définitions.

- *MATLAB* c'est le nom d'un langage de script, mais aussi celui de son environnement de développement. *MATLAB* a été conçu pour manipuler des matrices et afficher des courbes
- *Simulink* est un logiciel de modélisation de système dynamique reposant sur *MATLAB*. Il permet de créer des programmes sous la forme de schéma/ blocs

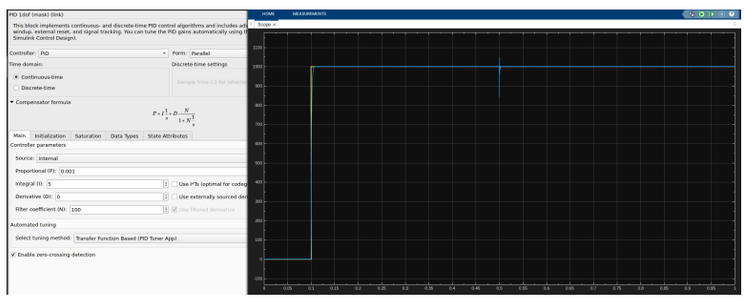
Pour schématiser grossièrement, MATLAB est le moteur et simulink l'interface
utilisateur.

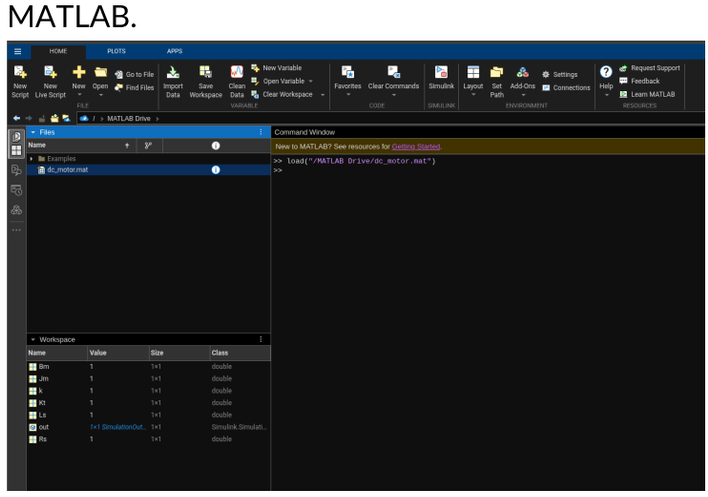
Maintenant que les présentations sont faites, passons à la mise en place de
notre exemple. *MATLAB* n'étant pas un logiciel libre, une licence est requise pour son utilisation. Heureusement pour moi, l'éditeur propose une offre découverte gratuite à utiliser via une interface web (utilisation limitée à 20h/mois). Donc on ouvre MATLAB, on lance simulink et on rentre l'équivalent schéma fonctionnel précédent, ce qui nous donne:



L'annotation en rouge est un ajout de ma part. Mais on remarque tout de suite l'un des points forts de *simulink*, son interface permet de faire un *programme* ayant une mise en forme proche du diagramme fonctionnel.

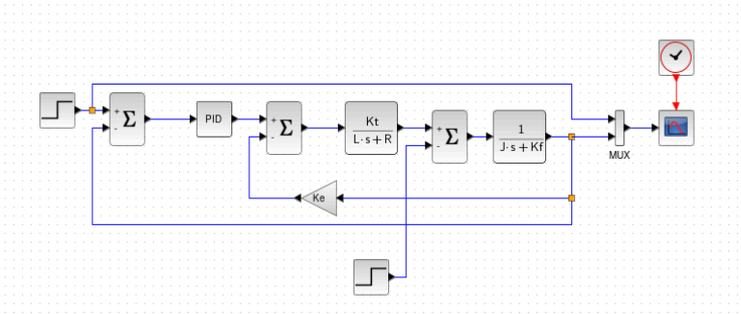
Un exemple de résultat de simulation, en jaune la consigne de vitesse et en bleu la vitesse du moteur. On peut visualiser ces valeurs grâce à la sonde nommée "*Visualisation*" sur notre schéma.



> **Remarque**
> Dans notre capture d'écran, l'on peut voir que nous avons utilisé des variables (kt, L, R...) dans les blocs que fonction de transfert. Un des inconvénients de MATLAB est que ces variables ne peuvent pas être facilement définies dans *simulink*. Le plus simple étant de créer un "*workspace*" dans MATLAB dans lequel on définit et assigne nos variables. Il faudra simplement penser à activer ce workspace avant de lancer notre simulation à la prochaine ouverture de MATLAB.
> 

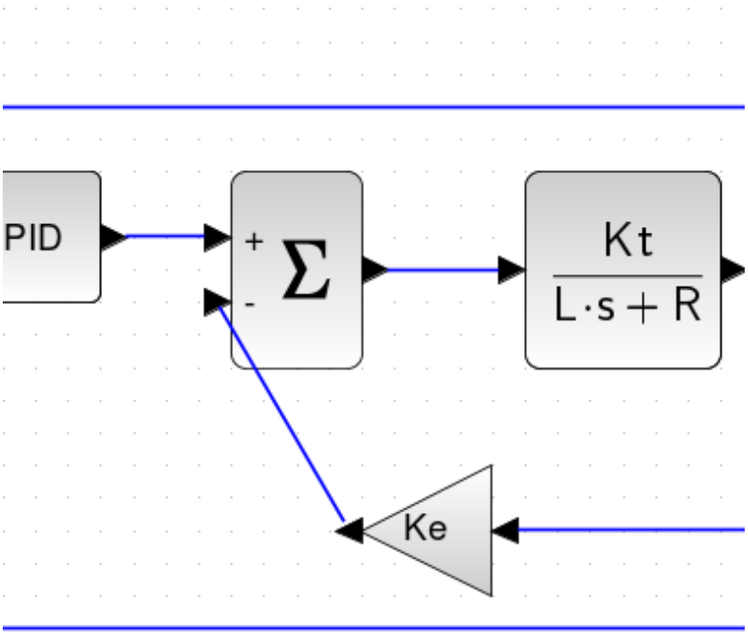
Scilab/xcos

Lorsque l'on cherche une alternative open source à *MATLAB/simulink* Scilab est LA solution évidente. En effet, ces deux solutions ont une interface utilisateur très proche ainsi qu'un langage de script assez similaire. Puisqu'une image vaut mille mots l'implémentation de notre exemple via *scilab/xcos* sera sans doute plus parlant:



La ressemblance avec *simulink* est frappante même si à l'utilisation les deux interfaces ne sont pas strictement identiques. Un avantage qu'offre *xcos*, est d'intégrer la gestion du contexte (les variables comme Ke, L ou R) directement dans le fichier de schéma-blocs. Pas besoin de créer de workspace comme sous MATLAB.

Cette solution est donc parfaitement viable. Cependant, *xcos* souffre d'un bug particulièrement pénible: l'instabilité des liaisons entre les blocs. Il n'est pas rare de rouvrir un projet de découvrir qu'une ou plusieurs liaisons ne suivent plus le routage original mais ont été remplacés par une liaison directe entre les
points d'encrage. Ceci rend le schéma nettement moins lisible, nous obligeant à refaire les liaisons lorsque ça arrive...



Une autre limite de cette solution, par rapport à *MATLAB*, est la petitesse
de sa bibliothèque de blocs comparée aux bibliothèques disponibles sur simulink. Mais tous ces défauts ne sont pas immuables et seront sans doute corrigés à l'avenir.

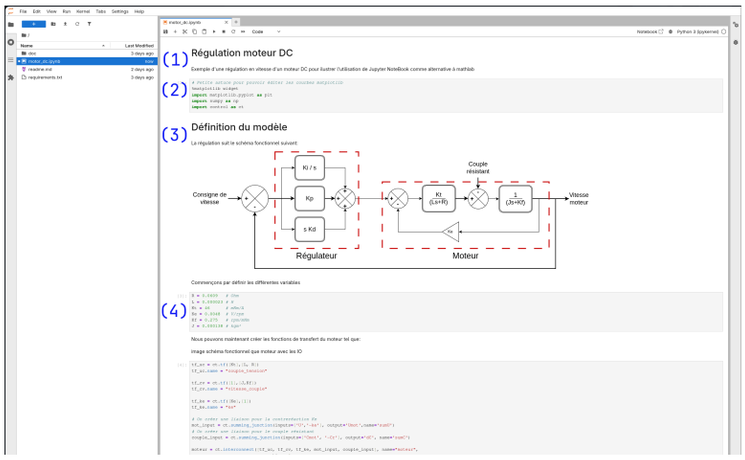
Python/Jupyter

Avec cet outil, je vous propose une autre approche, plutôt que d'écrire un
programme par schéma-bloque afin qu'il ressemble au schéma fonctionnel de la
documentation, nous allons intégrer le code dans la documentation. Ou la
documentation dans le code c'est vous qui voyez!

Cette alternative repose sur un ensemble de logiciels dont les principaux sont:
- pyton le langage de programmation
- Jupyter l'interface graphique
- control,
numpy et mathplotlib
les bibliothèques scientifiques permettant
de résoudre les équations, manipuler et afficher les données

Présentation

Comme je le disais en introduction, le principe de cette solution est de
mélanger documentation et code. Pour être honnête, c'est le principe même de
Jupyter, mais on va voir comment l'appliquer dans notre cas.
On va donc enchaîner les parties documentations au format markdown et le code en python (on peut d'ailleurs utiliser d'autres langages, mais je n'ai pas étudié cette possibilité).



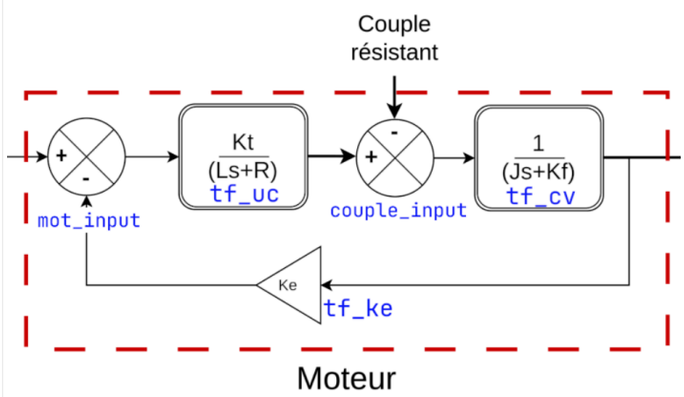
Dans cet exemple, on commence par un petit texte permettant de contextualiser le projet (1). On enchaîne ensuite avec l'importation des bibliothèques python nécessaires au projet (2). S'ensuit des explications plus précises quant au projet et le schéma fonctionnel permettant de le représenter (3). Enfin on termine la mise en contexte par la définition des différentes variables (4).

Modélisation du moteur DC

Cette approche a un gros inconvénient, elle est bien moins intuitive que
la programmation par blocs. Cependant, elle offre d'autres avantages, comme
une plus grande évolutivité et une plus grande souplesse dans la conception.
Elle permet aussi de tester/valider des sous-ensembles sans devoir refaire le programme.

Par exemple, plutôt que d'écrire le code pour l'ensemble de la simulation,
on peut commencer par faire *super-bloc* moteur qui inclut ses différents
composants:

Nous pouvons maintenant créer les fonctions de transfert du moteur Afin d'encapsuler dans un seul bloc l'ensemble du moteur:



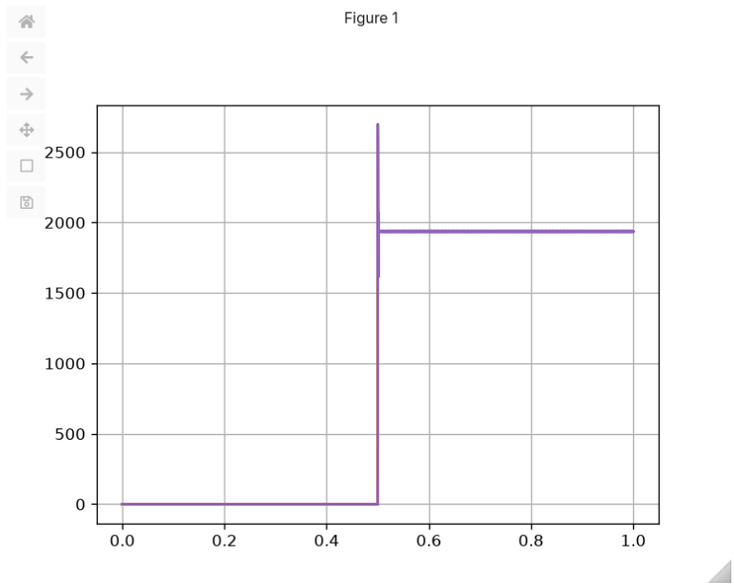

```python
# On commence par créer les 3 fonctions de transfert
tf_uc = ct.tf([Kt],[L, R])
tf_uc.name = "couple_tension"

tf_cv = ct.tf([1],[J,Kf])
tf_cv.name = "vitesse_couple"

tf_ke = ct.tf([Ke],[1])
tf_ke.name = "ke"

# Ensuite on passe aux liaison avec l'extérieur du bloc: couple résistant et tension de commandé
# On créer une liaison pour la contreréaction Ke
mot_input = ct.summing_junction(inputs=['U','-ke'], output='Umot',name='sumU')
# On créer une liaison pour le couple résistant
couple_input = ct.summing_junction(inputs=['Cmot', '-Cr'], output='dC', name='sumC')

moteur = ct.interconnect([tf_uc, tf_cv, tf_ke, mot_input, couple_input], name="moteur",
                         connections=[
                             ['couple_tension.u', 'sumU.Umot'],
                             ['sumC.Cmot', 'couple_tension.y'],
                             ['vitesse_couple.u', 'sumC.dC'],
                             ['ke.u', 'vitesse_couple.y'],
                             ['sumU.ke', 'ke.y'],
                         ],
                         inplist=['sumU.U', 'sumC.Cr'], inputs=['U', 'Cr'],
                         outlist=['vitesse_couple.y'], outputs='vitesse'
                        )
```

On remarque tout de suite que la courbe d'apprentissage de la bibliothèque
`control` est plus raide que celle de *simulink* ou *xcos*. En particulier lorsqu'il s'agit de connecter les blocs entre eux (fonction `ct.interconnect(...)`).

Mais une fois cette difficulté surmontée, nous disposons maintenant un composant `moteur` ayant pour entrées `U` et `Cr` et pour sortie `vitesse` et que nous pouvons réutiliser à notre guise.

On peut donc facilement valider le comportement du moteur en boucle ouverte
lorsqu'on le soumet à un échelon de tension de 10V:

```
t = np.linspace(0,1,10000)
U = np.concatenate((np.zeros(5000), 10* np.ones(5000)), axis=None)
Cr = np.zeros(t.shape)
moteur_step_res = ct.input_output_response(moteur, t, [U, Cr])
plt.plot(moteur_step_res.time, moteur_step_res.outputs['vitesse'])
plt.grid()
plt.show()
```



Pour faire cet essai dans *simulink* ou *xcos* nous aurions dû créer un
nouveau projet et y intégrer les blocs moteur, la commande de tension et
une sonde de mesure et ensuite alterner entre les projets en fonction des
modifications/validations à réaliser.

Cette modularité est donc très appréciable. Pour aller plus loin, on pourrait créer un fichier python séparé dans lequel on placerait notre composant moteur. Ainsi nous pourrions facilement le réutiliser dans différents projets.

Modélisation du régulateur

On peut maintenant passer au bloc du régulateur PID, enfin dans notre cas c'est seulement un PI, le terme dérivé étant nul.

Maintenant, passons à l'écriture du régulateur PID ayant les caractéristique suivante:

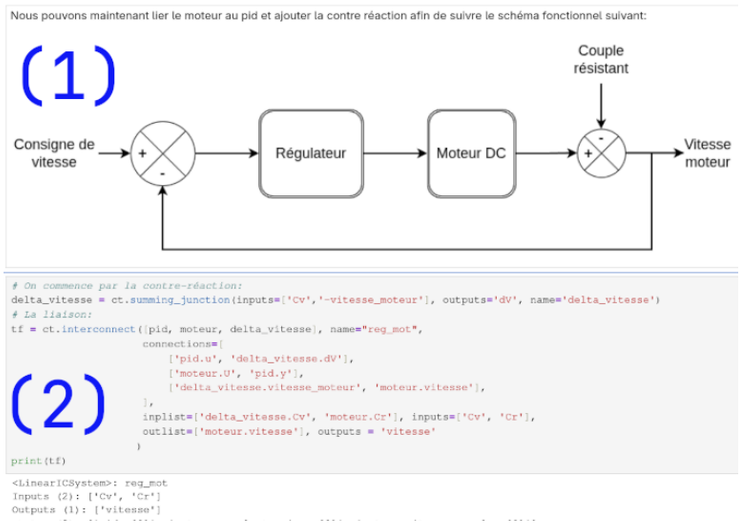
```
Kp = 0.001
Ki = ct.tf([0.05],[1, 0])
Kd = 0
pid = Kp + Ki + Kd
pid.name = "pid"
print(pid)
```

```
<TransferFunction>: pid
Inputs (1): ['u[0]']
Outputs (1): ['y[0]']

  0.001 s + 0.05
```

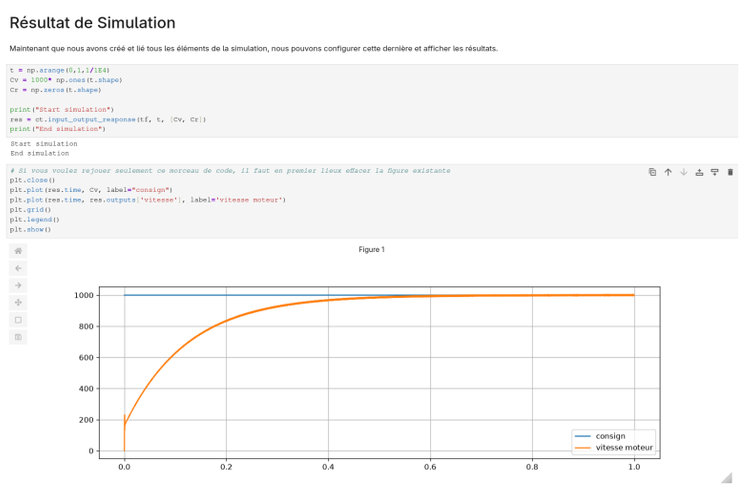
```
--------------
      s
```

Simulation complète

Nous avons maintenant tous les sous-ensembles nécessaires à la simulation de la régulation en vitesse du moteur DC. On commence donc par faire un rappel du système avec un schéma fonctionnel sans détails (1). Puis, on passe à la connexion des blocs et des entrées/sorties en python (2).



Enfin on peut afficher la réponse à un échelon de notre système. Dans notre exemple nous allons observer le comportement du moteur+asservissement pour une commande de vitesse de 1000 tr/min avec un couple résistant à zéro. Notre simulation durera une seconde avec un pas de temps 100 µs.



Ce notebook est disponible [sur codeberg](#).

Bilan

Après avoir testé trois solutions différentes pour faire de l'automatique, quel est le bilan ?

- MATLAB/Simulink
  - Avantages:
    - Standard de l'industrie

        - Programmation graphique proche du schéma fonctionnel
        - Large bibliothèque de composants
        - Nombreux exemples disponibles
    - Inconvénients:
        - Non open source
        - Nouveau langage à apprendre (MATLAB)
        - format fichier peu adapté au versionnage (simulink)
- Scilab/Xcos
    - Avantages:
        - Open Source
        - Programmation graphique proche du schéma fonctionnel
    - Inconvénients:
        - Nouveau langage à apprendre
        - versionnage
        - Bugs sur les liens (xcos) pénibles
- Jupyter/control
    - Avantages:
        - langage python
        - Modularité, on peut facilement faire ses bibliothèques de composants
        - Open source
        - simple fichier texte, facile à versionner
    - Inconvénients:
        - Courbe d'apprentissage plus raide
        - Moins visuel, relecture plus difficile
        - Communauté plus petite, donc moins de ressources

Avec cet article, je souhaitais partager mon expérience sur la recherche d'une solution open source pour l'automatique. Il y a évidemment l'incontournable Scilab/Xcos mais pas seulement. Pour les personnes qui, comme moi, sont plus à l'aise avec du texte qu'avec une interface graphique, d'autres solutions existent. Ici par exemple, on arrive en combinant l'éditeur *Jupyter* et quelques bibliothèques python, à avoir une solution offrant documentation et calculs.

Cette solution n'est pas sans inconvénients, notamment l'intégration de matplotlib dans jupyter qui n'est pas trivial… Mais elle offre aussi de nombreux avantages. Comme la facilité de créer des bibliothèques de composants, versionner efficacement le code et la documentation …

Pour ce qui est du versionnage, le code des bibliothèques est en python, la question ne se pose même pas et la partie *notebook* quant à elle est un fichier JSON, donc du texte. Cette solution vous permettra donc de gérer finement le versionnage de vos projets.

Autres pistes

Il existe encore d'autres solutions aux trois que j'ai étudiées ici, j'en ai trouvé une qui me convient alors je ne me suis pas attardé dessus mais si vous êtes intéressé voici une liste de celle que j'ai trouvées:

- OpenModelica
- octave
- Julia

Il existe certainement encore d'autres solutions que je ne connais pas...

(12 commentaires). Markdown EPUB

Non utilisateur

Posté par **Sytoka Modon** (**site web personnel**) le 07 octobre 2026 à 12:32. Évalué à 3 (+1/-0).

> Je n'utilise aucun de ces outils...
> Dans mon boulot, c'est plutôt LabView qui est utilisé pour piloter les manips (pas toujours, mais souvent). Cependant, je n'ai jamais codé et lancé LabView à mon niveau.
>
> Un critère qui me semble important est la Stabilité (Pérennité, Rétro-compatibilité...).
>
> Du peu que j'ai vu, on n'a pas envie de refaire tous les ans !
>
> Personnellement, j'ai des doutes que Scilab décolle réellement un jour et Python (et ses modules) cassent trop souvent ses API. J'en ai encore fait les frais sur de la CI avec Gitlab avant l'été.

Répondre

[^] # Re: Non utilisateur

Posté par **uneTanche** (**site web personnel**, **Mastodon**) le 07 octobre 2026 à 13:03. Évalué à 1 (+0/-0).

> En effet, c'est quelque chose de pénible.
>
>
>
> Personnellement, pour python, j'essaie de gérer ça en limitant les dépendances et en ayant une copie local du dépôt (je n'y arrive pas toujours d'ailleurs).
> Python propose aussi un système de "cache" pour gérer les dépendances mais je n'ai pas encore eu le temps de regarder de ce côté...

Répondre

[^] # Re: Non utilisateur

Posté par **fork_bomb** le 07 octobre 2026 à 18:51.
Évalué à 4 (+3/-0).

> > Python (et ses modules) cassent trop souvent ses API
>
> Je te conseille fortement d'utiliser uv : c'est un gestionnaire de dépendances qui stocke dans un fichier la version de python et des dépendances du projet, et on peut donc utiliser un projet 6 mois plus tard avec exactement les mêmes versions (y compris les dépendances transitives).

Répondre

PathSim et PathView

Posté par **bibifric05** le 07 octobre 2026 à 17:26.
Évalué à 1 (+1/-1). Dernière modification le 07 octobre 2026 à 17:26.

> Je viens de demander à ChatGPT quelles alternatives à Simulink et il m'a indiqué en plus de celles que tu as données : PathSim (framework) et PathView (visualiseur) en Python.
>
> Ça a l'air récent et je ne sais pas ce que ça vaut...
>
> Eh oui, MathWorks (l'éditeur de MATLAB / Simulink) c'est la machine à cash, ils ne vous font pas de cadeaux !

Répondre

[^] # Re: PathSim et PathView

Posté par **2PetitsVerres** (**Mastodon**) le 07 octobre 2026 à 21:58. Évalué à 7 (+5/-0).

> Eh oui, MathWorks (l'éditeur de MATLAB / Simulink) c'est la machine à cash, ils ne vous font pas de cadeaux !

Alors déjà je ne suis pas d'accord, j'ai eu des cadeaux chaque années où j'y ai travaillé.

Et plus sérieusement j'ai vu bien pire ailleurs. Alors oui, il ne font pas de cadeau dans le sens où quand un acheteur les contacte pour demander une ristourne, la réponse est non. L'avantage au moins c'est que c'est pas un prix avec une ristourne à la tête du client. Par contre j'ai vu des pratiques commerciales bien pire d'autres éditeurs logiciels.

--

Tous les nombres premiers sont impairs, sauf un. Tous les nombres premiers sont impairs, sauf deux.

Répondre

[^] # Re: PathSim et PathView

Posté par **bibifric05** le 08 octobre 2026 à 08:40.
Évalué à 4 (+3/-0).

> Je me souviens dans les années 2000, ils avaient une commerciale très sympathique, mais à part ça...
>
> Dans mon entreprise, MATLAB est partout avec tous types de licences (locations, flottantes, personnelles, avec ou sans maintenance, etc., des centaines de licences, sous Windows mais aussi sous Linux).
>
> MATLAB lui même n'est pas si cher, mais je suis choqué de voir les prix pratiqués par MathWorks pour le moindre toolbox et l'absence de tout geste commercial même vis à vis de très gros clients comme nous... Ils profitent allègrement de leur position de quasi monopole.
>
> Mais les choses sont en train de changer : de plus en plus, nous passons à Python, très progressivement car c'est difficile à cause de l'énorme existant sous MATLAB. Il n'y a pas de traducteur MATLAB => Python même si l'IA se débrouille pas mal. Ça reste un boulot énorme...
>
> Pour Simulink, le passage à Python est problématique en effet, mais Simulink n'est pas tant utilisé chez nous en raison du coût probablement... On peut faire la même chose sous MATLAB, en moins visuel bien sûr.

Sinon Octave est une très bonne alternative à MATLAB, souvent compatible sans changer une ligne de code (et on peut utiliser les MEX fonctions comme sous MATLAB en changeant juste l'interface). Mais le gros point noir c'est la rapidité : Octave ne possède pas de compilateur à la volée (le JIT de MATLAB) et dès que le code nécessite des boucles (inévitables dès qu'on fait des analyses paramétriques, type variation du rapport signal à bruit) les temps de calcul deviennent quasi prohibitifs... Python se débrouille bien mieux (notamment en utilisant Numba ou Cython). Et Octave n'a pas d'alternative à Simulink non plus.

Répondre

Ancien MathWorker...

Posté par **2PetitsVerres** (**Mastodon**) le 07 octobre 2026 à 22:21. Évalué à 10 (+8/-0). Dernière modification le 07 octobre 2026 à 22:21.

J'ai travaillé pendant des années avec Simulink (comme client de MathWorks, puis aussi chez MathWorks directement) donc je suis obligé de répondre, mais je suis surement biaisé.

> Un des inconvénients de MATLAB est que ces variables ne peuvent pas être facilement définies dans simulink

On peut assez facilement (enfin, j'imagine que c'est peut être ça qui est discutable ?) créer une variable dans simulink, et qui sera sauvegardée avec le modèle, voire dans le fichier du modèle lui-même. Depuis toujours depuis le Model Explorer (l'outil au pire raccourcis claviers du monde :-D), mais depuis quelque années normalement directement dans la fenêtre où on entre le paramètre :

Par exemple, la valeur d'un gain. Tu mets un bloc gain, puis tu vas dans ses paramètres, tu rentres "un_tres_bon_nom_de_variable" dans le champ, et si ce nom n'existe pas il te propose de le créer. Tu choisis de le créer dans le workspace du modèle ou un dictionnaire de donnée, et c'est bon.

> Simulink est un logiciel de modélisation de système dynamique reposant sur MATLAB

Alors, oui, si "reposant" veut dire que le bouton pour lancer Simulink est dans MATLAB, ou que tout ce qu'on utilise pour automatiser Simulink est dans/en MATLAB. Mais en vrai, le coeur de simulation (et à peu près tout sauf l'interface) c'est du C++ et du C. Et l'interface c'est du html/css/javascript (en tout cas maintenant ça devrait n'être plus que ça, je n'y suis plus pour vérifier qu'ils ont fini :-))

> [Simulink] permet de créer des programmes sous la forme de schéma/ blocs

Je trouve ça un peut réducteur.

Alors oui, mais surtout il permet de les simuler, de simuler en faisant varier les paramètres. Il permet de simuler des système discrets, continus ou hybrides.

Il permet aussi potentiellement de générer directement le contrôlleur qu'on a mis au point en C ou C++ et d'embarquer ce contrôlleur sur sa cible embarquée directement.

C'est parce qu'il permet de faire plein de choses que la recherche d'une alternative est compliquée. Je pense qu'il n'existe pas une alternative qui fait tout, du coup certaines personnes vont dire "X peut remplacer Simulink", mais qu'un autre utilisateur de Simulink va se dire "n'importe quoi, ça ne fait même pas 10% de ce que je fais avec".

Dans les alternatives que tu as cité en plus (et avec mes connaissances anciennes du sujet), OpenModelica est plus une alternative à SimScape (qui est un

outil à l'intérieur de Simulink), octave et julia sont plutôt des alternatives à MATLAB.

Après des gens font de la simulation robotique avec des outils issus du monde de ROS, c'est une alternative potentielle à Simulink sur ce domaine là. (et moi je fais de la robotique avec des outils maison, donc j'imagine que les outils maison, c'est une alternative aussi)

J'ai plein de noms d'alternatives sur des petits morceaux de Simulink, mais c'est surtout des logiciels non libres :-D (et souvent c'était pas aussi bien...)

--

Tous les nombres premiers sont impairs, sauf un. Tous les nombres premiers sont impairs, sauf deux.

Répondre

[^] # Re: Ancien MathWorker...

Posté par **uneTanche** (**site web personnel**, **Mastodon**) le 08 octobre 2026 à 08:08. Évalué à 4 (+3/-0).

Merci pour ce retour riche en explications :D
Je vais mettre à jour mon article avec vos informations.

En effet, ma comparaison se limite à la partie simulation, ce qui ne représente qu'une partie des possibilité de matlab/simulink.

Un autre "bon point" pour Mathworks par rapports à de nombreux autres éditeurs de logiciel métier: ils proposent une version linux.

Répondre

Octave

Posté par **Colin Pitrat** (**site web personnel**) le 08 octobre 2026 à 08:49. Évalué à 8 (+6/-0).

Alors, ça doit faire 20 ans que je n'y ai pas touché mais pendant mes études on utilisait matlab. En libriste convaincu, j'utilisais octave personnellement, et j'en étais très content.

De ce qu'on m'a dit, octave + modelica est un excellent remplaçant à matlab + simulink. Pas exactement dans le même esprit, mais d'après certains retour plus puissant et plus flexible sur le long terme, même si un peu moins "user-friendly".

> Donc à essayer plus en profondeur peut-être ?

Répondre

[^] # Re: Octave

Posté par **esdeem** (**site web personnel**) le 08 octobre 2026 à 17:20. Évalué à 2 (+0/-0).

> J'ai fais pareil, il y a une disaine d'années. Je n'ai pas eu de problèmes majeurs.
>
> --
>
> 0. Assume good faith 1. Be kind to other people 2. Express yourself 4. Apply rule 0

Répondre

utilisation de simulink

Posté par **ChocolatineFlying** le 08 octobre 2026 à 14:49. Évalué à 2 (+1/-0).

> pour ceux qui ne le savent pas encore simulink est très utilisé dans l'éducation national et fait partie des épreuves pour certain capes comme physique, électrique, mécanique.
>
> ca pique un peu lorsque vous arrivez la fleurs au fusils à l'épreuve :)
> et ce n'est pas donnée si vous comptez l'avoir chez vous pour vous entrainer.

Répondre

[^] # Re: utilisation de simulink

Posté par **BAud** (**site web personnel**) le 08 octobre 2026 à 18:26. Évalué à 2 (+0/-0).

> de l'intérêt d'avoir des projets comme ClefAgreg pour promouvoir les logiciels libres directement utilisables.
>
> Je ne sais pas où ça en est :/
>
> on a aussi eu CAELinux à une époque pour des logiciels d'ingénierie (Salome, code_aster...) mais bon ce serait pas mal que cela perdure dans le temps (même si les logiciels sont installables sur à peu près toute distribution, avec plus ou moins de facilité on va dire pour certains logiciels...).

Répondre

Envoyer un commentaire

Suivre le flux des commentaires

Note : les commentaires appartiennent à celles et ceux qui les ont postés. Nous n'en sommes pas responsables.

Revenir en haut de page

Derniers commentaires

- Re: Chambre blanche
- Re: Je suis un Amish!
- Re: Architecture de l…
- Re: Chambre blanche
- Re: Le problème est …
- 2 suggestions et qu…
- Re: C'est certaineme…
- Re: Chambre blanche
- Un fil Mastodon en …
- C'est certainement i…
- Re: Bonsoir à tous
- Re: Le problème est …

Étiquettes (tags) populaires

- intelligence_artificielle
- grands_modèles_de…
- slop
- merdification
- jihad_butlérien
- jeu_vidéo
- capitalisme
- états-unis
- chine
- meta
- administration_fran…
- politique

Sites amis

- Agenda du Libre
- April
- Éditions D-BookeR
- Éditions Diamond
- Éditions Eyrolles
- Éditions ENI
- En Vente Libre
- Framasoft
- La Quadrature du Net
- Lea-Linux
- Open Source Initiative
- Imprimerie Grafik Plus

À propos de LinuxFr.org

- Mentions légales
- Faire un don
- L'équipe de LinuxFr.…
- Informations sur le s…
- Aide / Foire aux que…
- Suivi des suggestion…
- Wiki du site
- Règles de modération
- Statistiques
- API pour le dévelop…
- Code source du site
- Plan du site