Programmation et simulation numérique avec Scilab

Florent Ouchet
PRAG S2I ingénierie électrique
Département mécatronique
Ecole Normale Supérieure de Rennes
florent.ouchet@ens-rennes.fr

Organisation des journées

- 9h00-9h30 : présentation des formations
- 9h30-10h30 : formation (1 & 5)
- 10h30-10h45 : pause café
- 10h45-12h15 : formation (2 & 6)
- 12h15-13h45 : repas
- 13h45-15h15 : formation (3 & 7)
- 15h15-15h30 : pause café
- 15h30-17h00 : formation (4 & 8)

Plan de la présentation

1. Présentation générale
2. Saisie graphique des systèmes

   Manipulations
3. Du modèle à l'équation différentielle

   Manipulations
4. De l'équation différentielle au résultat

   Manipulations
5. Instrumentation

   Manipulations

Présentation générale
Contexte

- Plusieurs acteurs sur le marché:
  - Scilab (gratuit, « français »)
  - Matlab (payant, anglais)
  - Octave (gratuit, anglais)
  - OpenModelica (gratuit, anglais)
  - Mapple/Mathematica/Maxima…
  - Excel…
  - GNU plot…

- Scilab mis en avant par MEN

- Matlab mis en avant dans le MES

Présentation générale Scilab

- Scilab:
  - outil de calcul numérique (non symbolique!),
    - scalaire & matriciel (attention aux dimensions),
    - nombres à virgule flottante,
    - pas d'unités
  - langage de programmation,
    - langage structuré,
      - conditions,
      - boucles,
    - sous-programmes,
    - Bibliothèques déjà présentes
  - Entrées/sorties:
    - utilisateurs
    - graphiques
    - fichiers

Présentation générale XCos

- Module de Scilab
- Blocs « fonctions » à placer sur une zone de dessin:
  - stimuli et entrées (sources)
  - fonctions mathématiques
  - affichage (sinks)
  - conditions…
  - pas d'unités!

- « fils » pour relier les entrées et les sorties:
  - Différence entre les « fils » causaux et acausaux
  - Différence entre blocs causaux et acausaux

Présentation générale XCos

- « fils » causaux:
  - Représentent un transfert orienté d'information
    - Information scalaire ou éventuellement multi-dimensionnelle
    - Information numérique sans unité
  - Flèches orientées depuis un port de sortie vers un ou plusieurs ports d'entrée

Présentation générale XCos

- Blocs causaux:
  - Nécessité pour un bloc d'avoir les valeurs à ses ports d'entrée établies pour « calculer » les valeurs de ses ports de sorties → causalité:
    - Notion de fonction de transfert
    - Résolution simple si « boucle ouverte »
    - Algorithme de résolution itératif si « boucle fermée »
  - Possibilité d'utiliser des blocs d'entrées/sorties
  - Possibilité de faire ses propres blocs (non-étudié dans cette formation)

Présentation générale XCos

- « fils » acausaux:
  - Représentent une contrainte d'équi-potentialité:
    - même potentiel électrique
    - même pression hydraulique
    - même position/vitesse en mécanique cinématique
  - Représentent une contrainte de nullité de la sommation des flux entrants:
    - « Loi des mailles » pour un « fil » électrique
    - Conservation du flux en hydraulique
    - Egalité entre force et réaction du support en mécanique cinématique

Présentation générale XCos

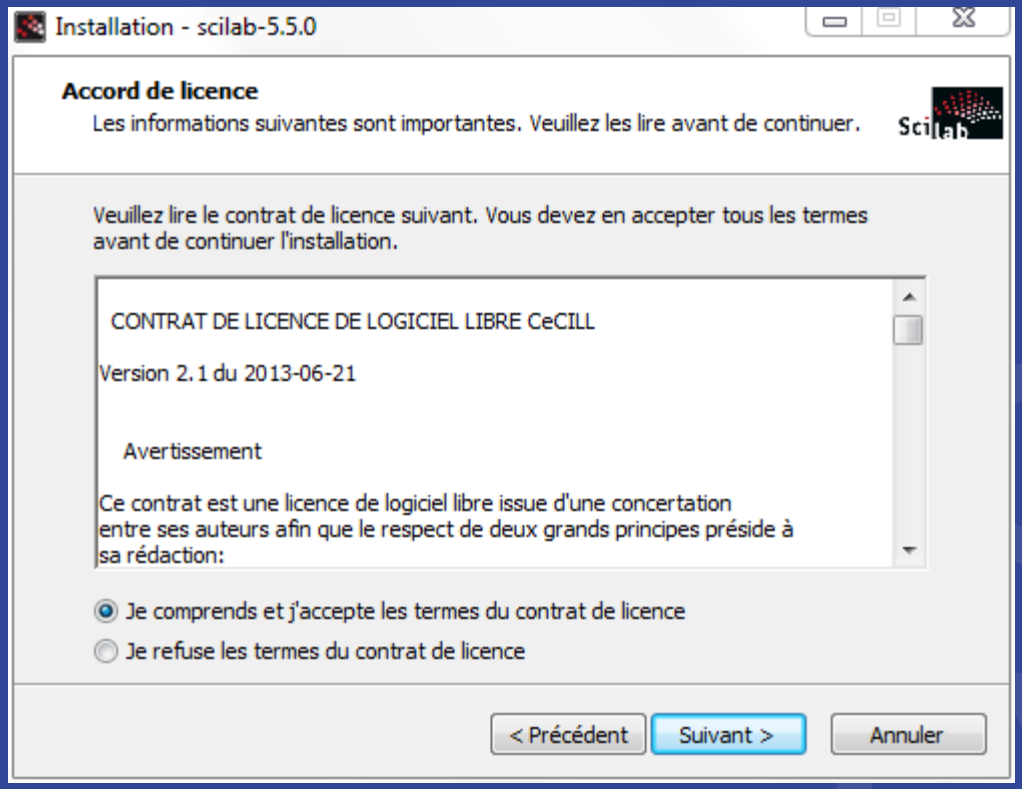
- Blocs acausaux:
  - Plusieurs ports d'accès:
    - Broche pour les composants électriques
    - Connecteur pour l'hydraulique
    - Point d'application pour une force ou centre de rotation pour un couple en mécanique cinématique
  - Expriment une relation entre variables flux et potentiels à leurs différents ports d'accès
  - Permettent éventuellement l'instrumentation ou l'actionnement de flux/potentiels
  - Possibilité de faire ses propres blocs (non-étudié dans cette formation)

Présentation générale XCos

- Quelques bibliothèques « métiers » en acausal:
  - Coselica (EN):
    - Electricité
    - Transfert de chaleur
    - Mécanique
    - Moteurs
  - SIMM (FR):
    - Coselica en français +
    - Signaux
  - CPGE:
    - Automatique
    - Systèmes (MaxPID…)

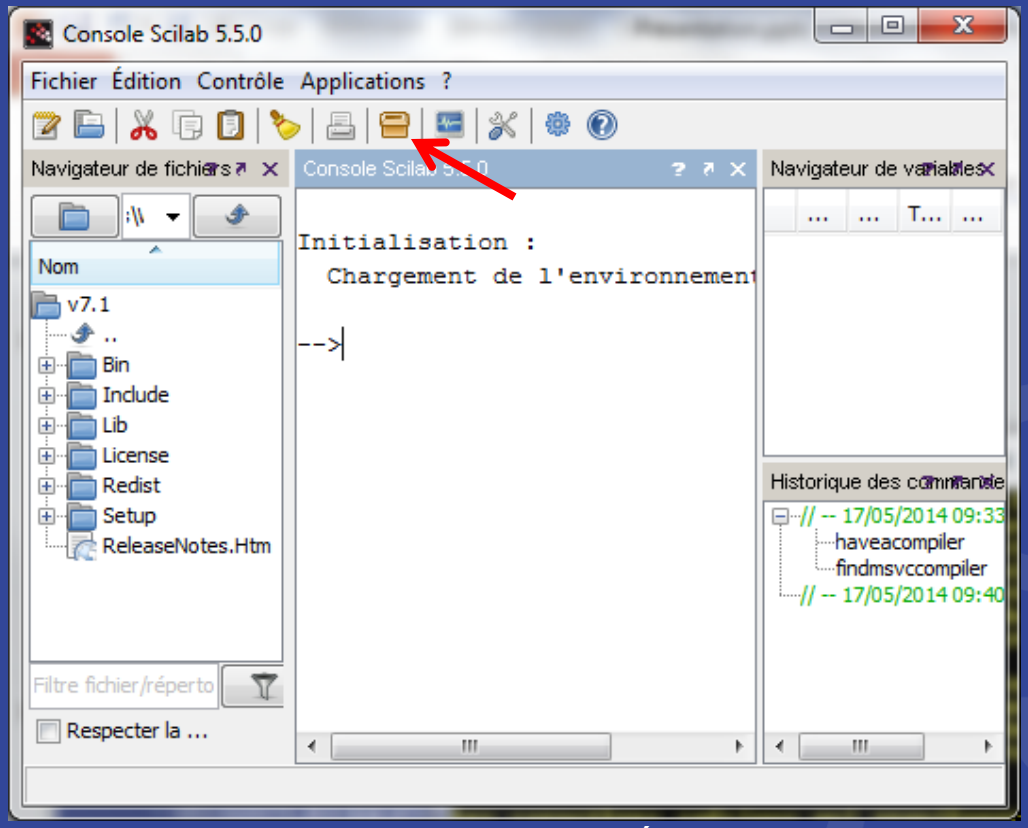
Présentation générale
Installation

- Installation de Scilab:
  - Programme principal depuis le site http://www.scilab.org/
  - Version actuelle 5.5.0, pas de réglage particulier



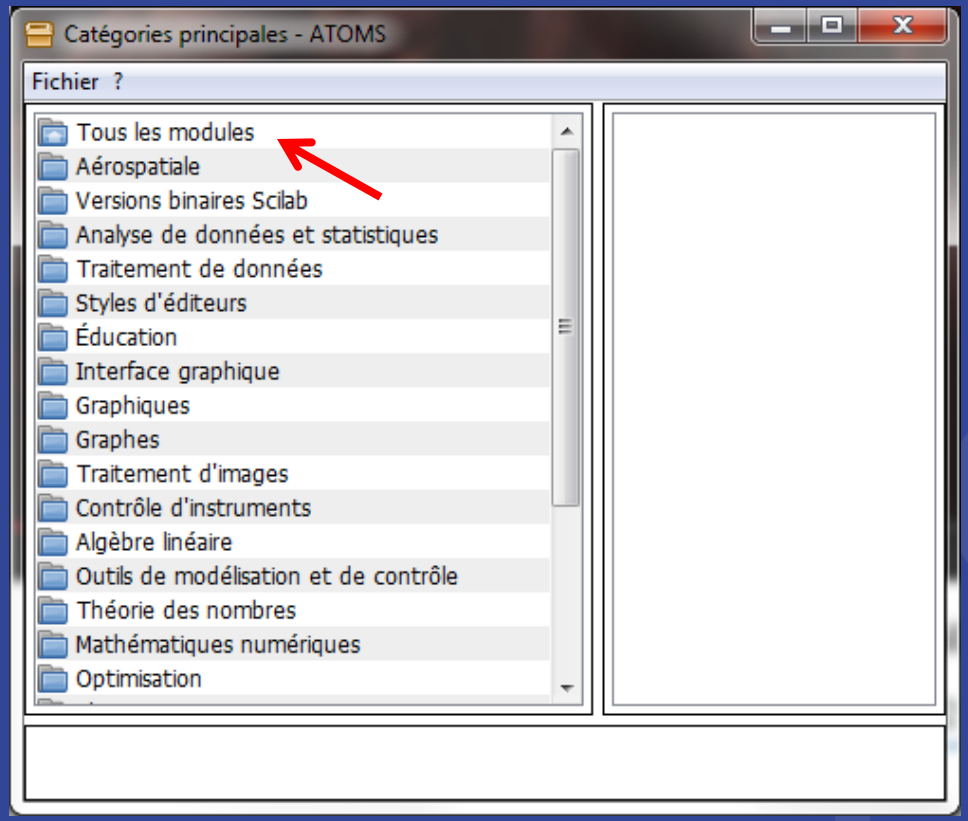
Présentation générale
Installation

- Installation des modules complémentaires:
  - gestionnaire de modules Atoms:



Présentation générale
Installation

- Installation des modules complémentaires:
  - gestionnaire de modules Atoms:



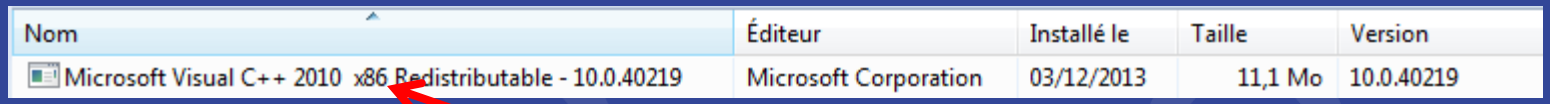
Présentation générale Installation

- Installation des modules complémentaires:
  - Coselica: blocs XCos pour la simulation de modèles acausaux
  - CPGE: blocs Xcos pour l'asservissement et la communication avec des systèmes (MaxPID)
  - NIDAQ: communication avec les boitiers National Instruments
  - (SIMM): quelques blocs utiles en plus de Coselica

Présentation générale
Installation

- Installation d'un compilateur:
  - Microsoft Windows SDK 7.1

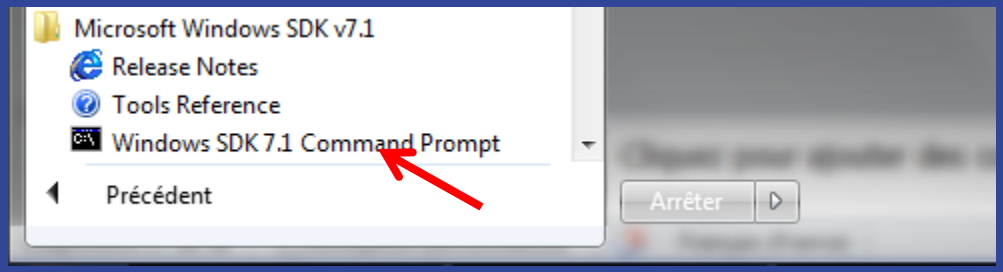
  http://www.microsoft.com/en-us/download/details.aspx?id=8279
  - Installation par défaut
  - Eventuellement décocher « .net » pour installer plus rapidement et réduire l'espace disque utilisé
  - Si l'installation échoue, il faut préalablement désinstaller



en version x86 et x64

Présentation générale
Lancement

- Pour que le compilateur soit détecté:

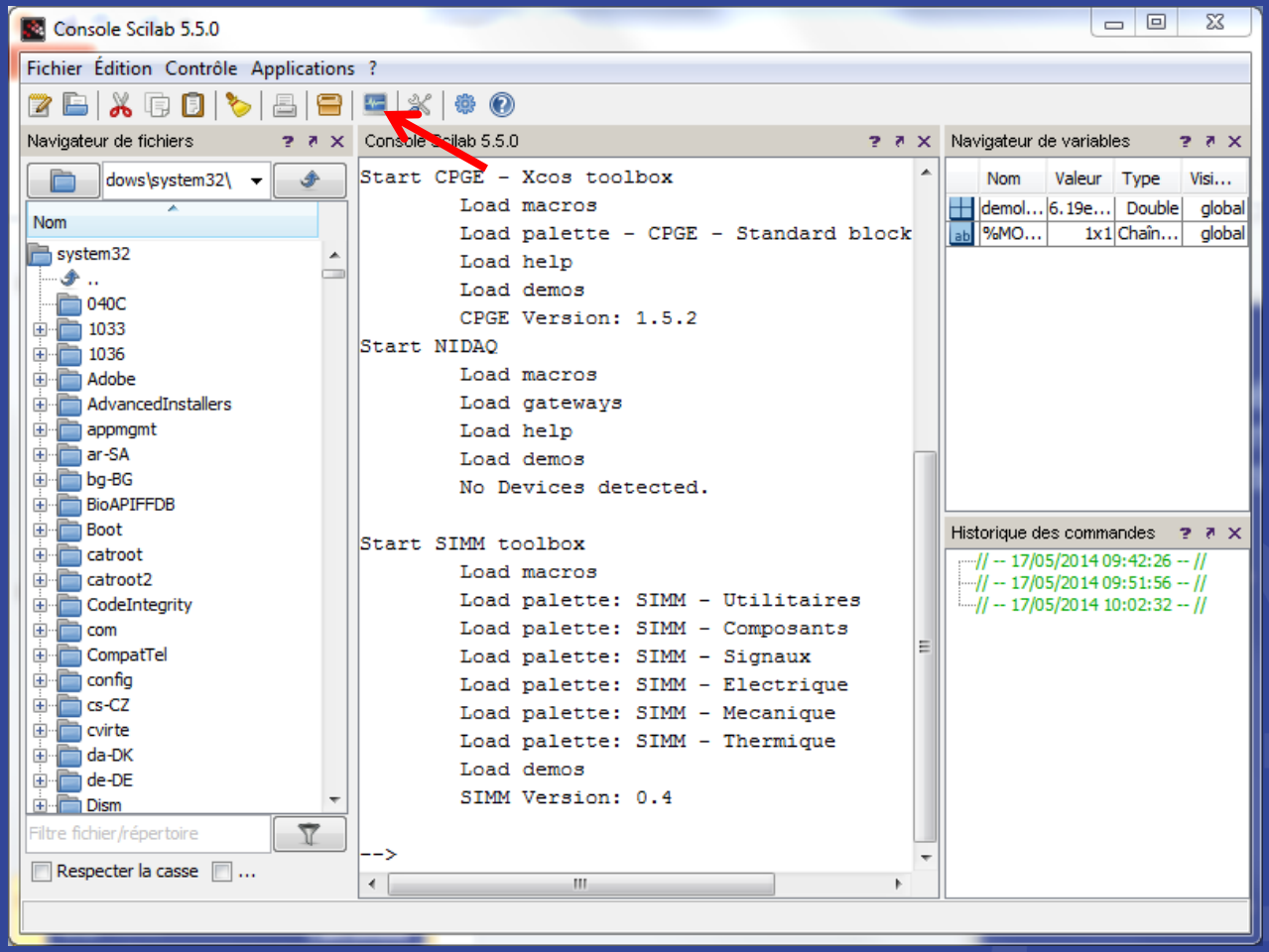


- Puis dans l'invite de commandes, exécuter:

"c:\Program Files\scilab-5.5.0\bin\WScilex.exe"

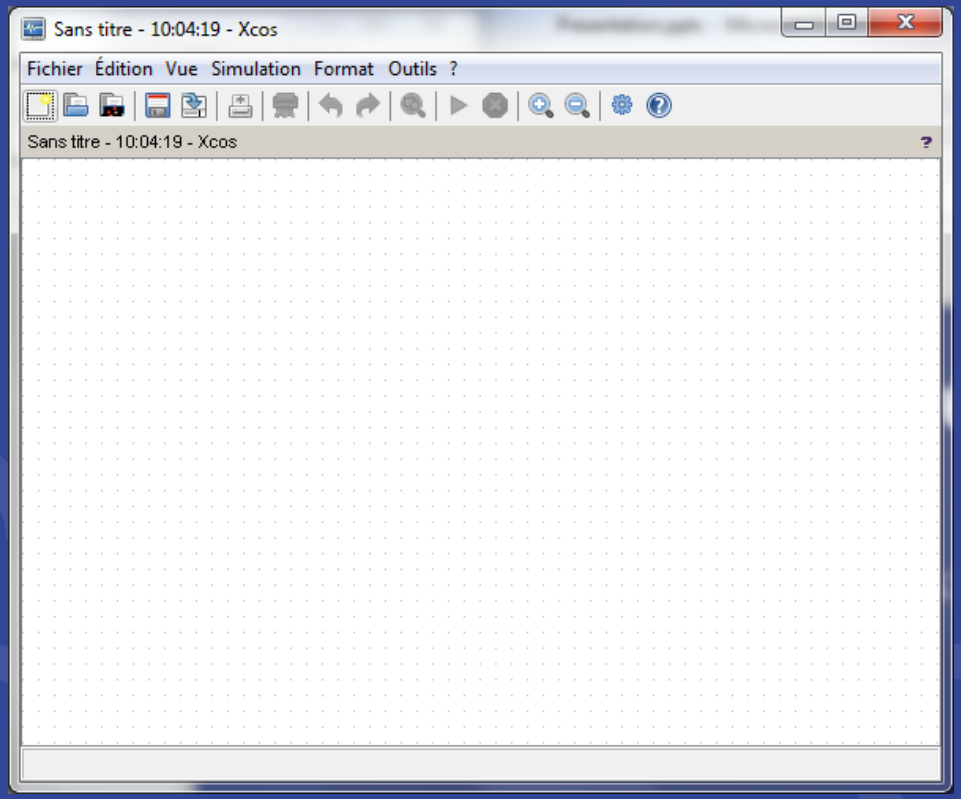
Présentation générale
Lancement

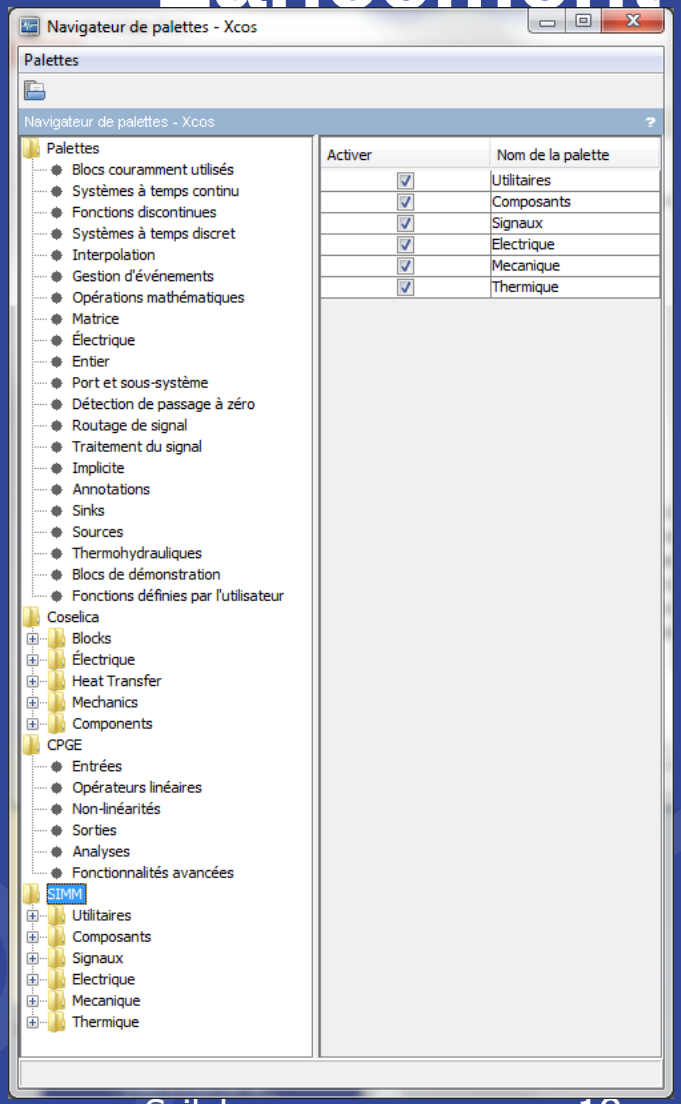
- Environnement Scilab:



Présentation générale
Lancement

- Environnement XCos:





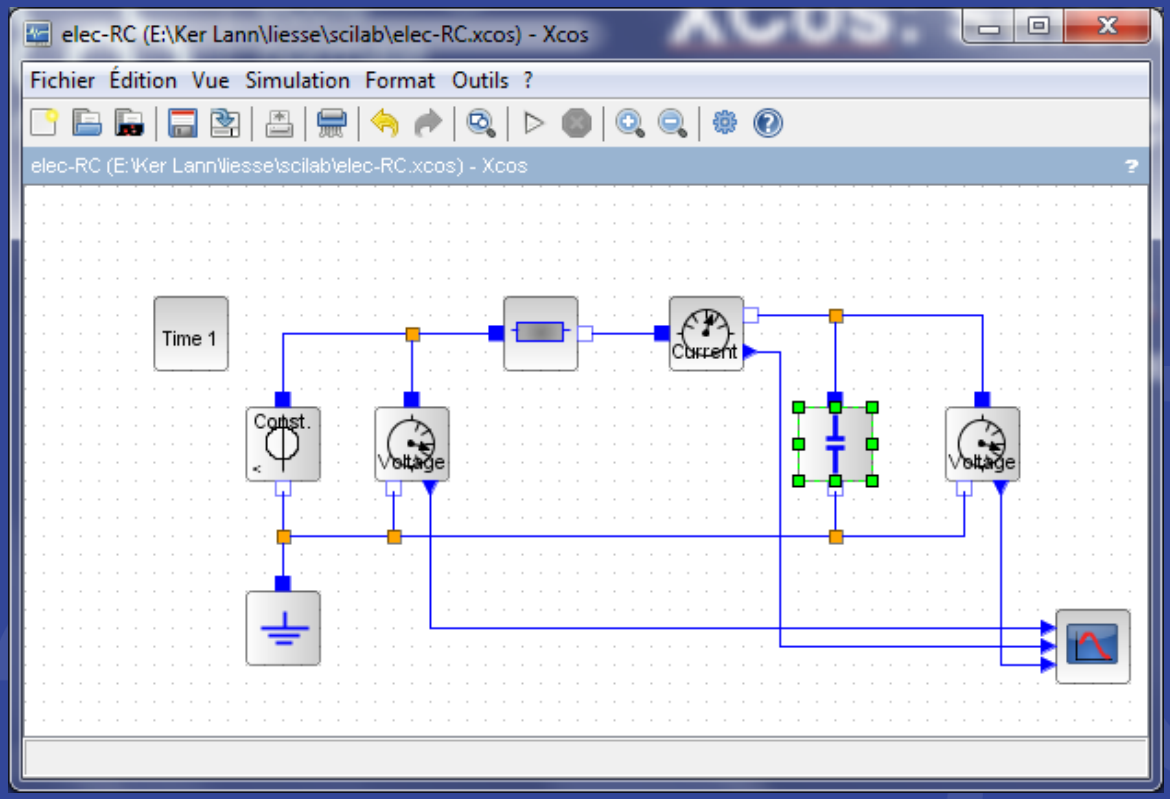
Présentation générale
Lancement

- Il existe plusieurs bibliothèques (incompatibles entre elles) proposant des blocs pour simuler des systèmes multiphysiques (mécanique, électrique, hydraulique, thermique…)

- Pour cette formation, nous utiliserons en priorité les composants de la bibliothèque SIMM (qui utilise les blocs de la bibliothèque Coselica avec une hiérarchie un peu plus adaptée).

XCos: saisie graphique des systèmes

- Premier exemple (elec-RC.xcos):
  - Utilisation de la boite à outils « élec »
  - Simulation de la charge d'une capacité

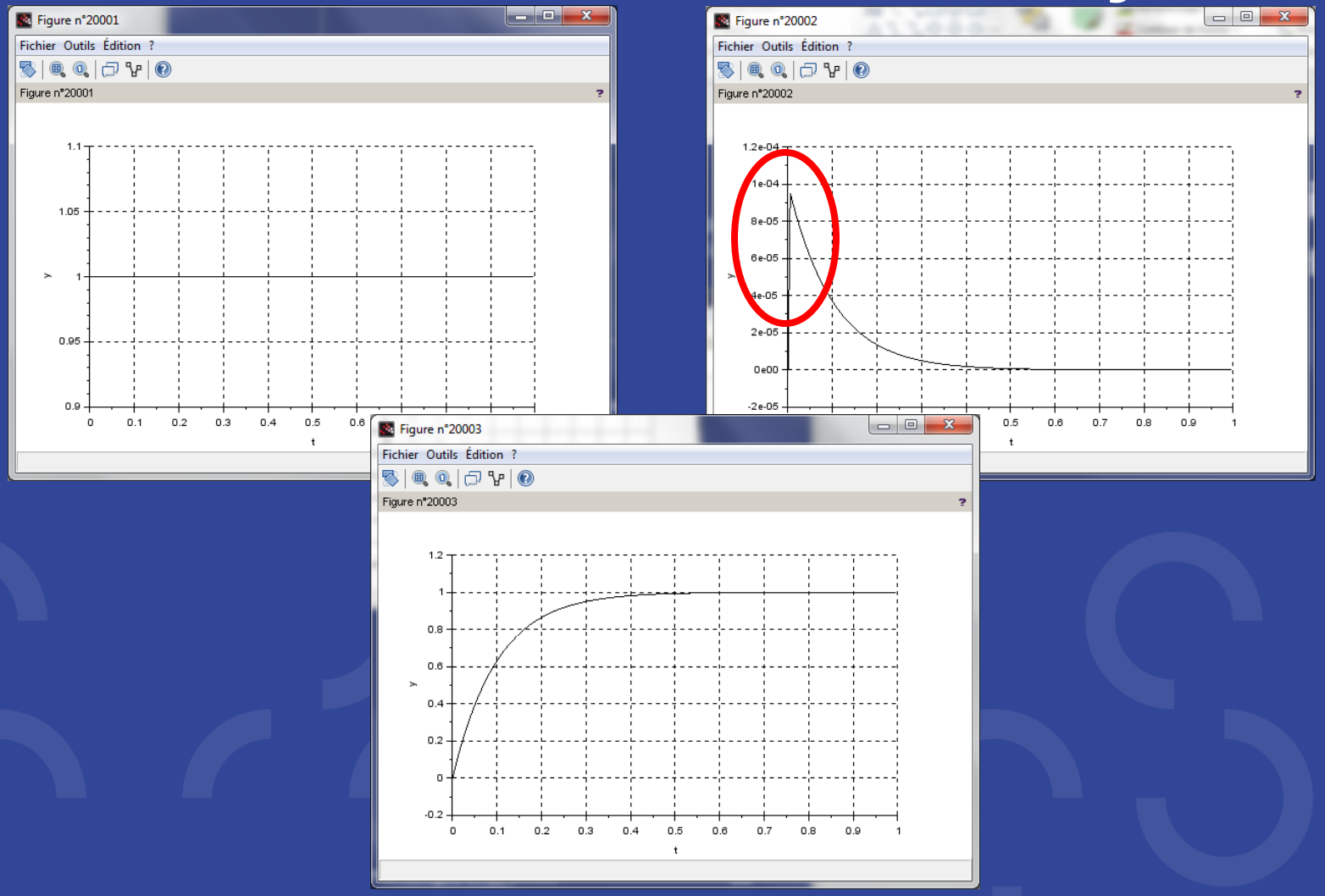


XCos: saisie graphique des systèmes

- Premier exemple (elec-RC.xcos):
  - Paramètres de simulation
  - Lancement de la simulation

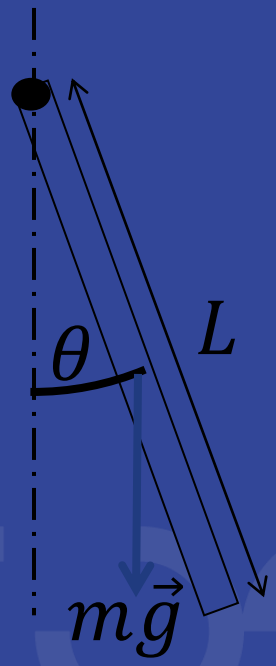
- Références à des variables Scilab dans le modèle:
  - Permet une mise à jour rapide des conditions sans avoir à modifier le modèle
  - Exécuter le script elec.sce situé dans le même répertoire

XCos: saisie graphique des systèmes



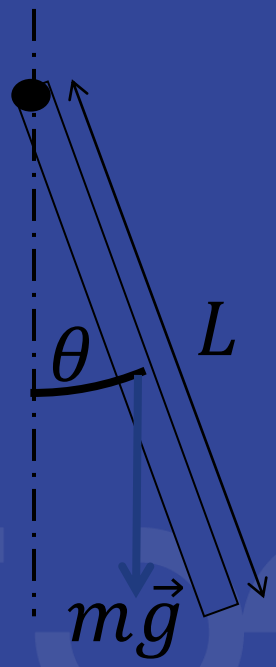
XCos: saisie graphique des systèmes

- Second exemple:
  - simulation d'un pendule
  - bibliothèque SIMM pour la mécanique générale (problème plan)



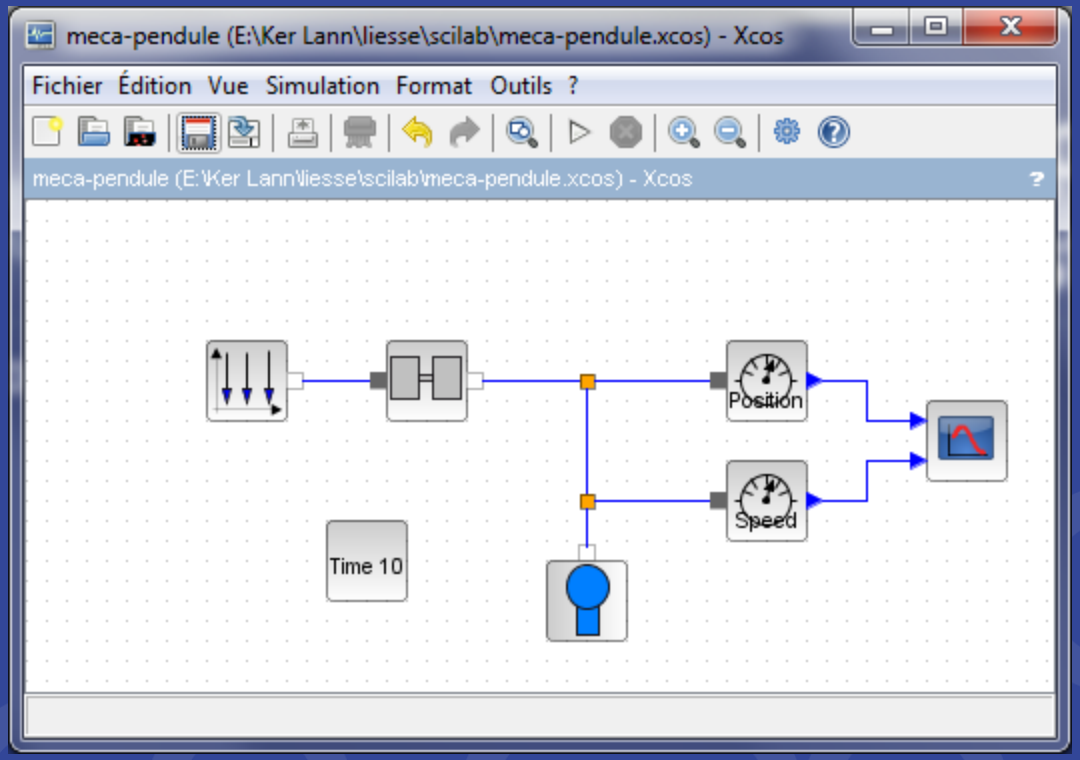
XCos: saisie graphique des systèmes

- Second exemple:
  - simulation d'un pendule, problème plan
  - bibliothèque SIMM pour la mécanique générale



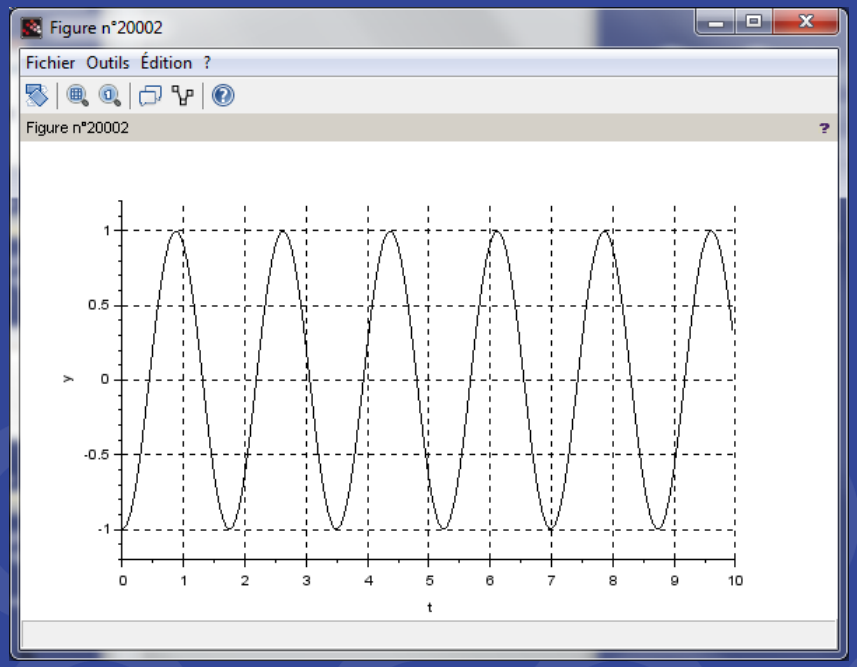
XCos: saisie graphique des systèmes

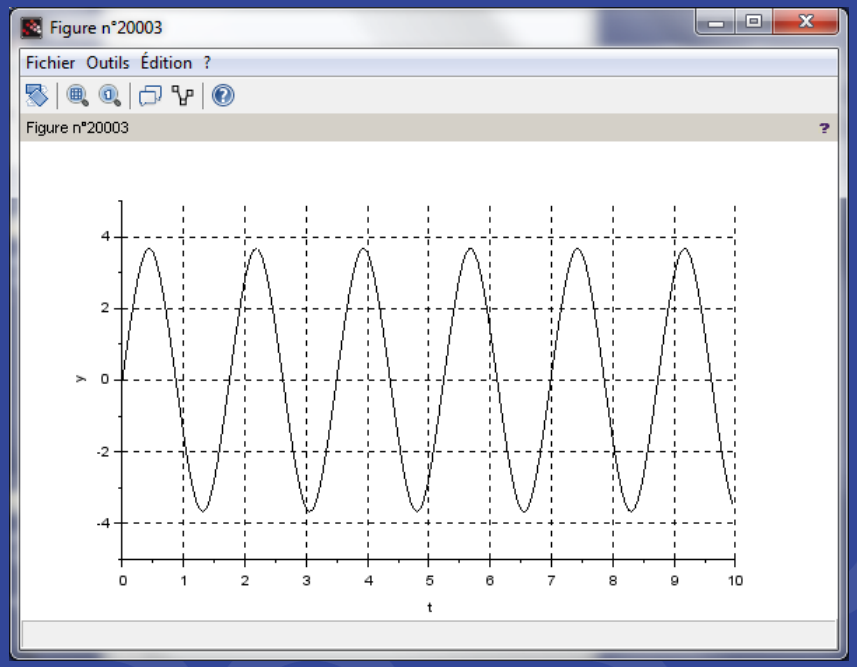
- Second exemple (meca-pendule.xcos):



XCos: saisie graphique des systèmes

- Second exemple (meca-pendule.xcos):



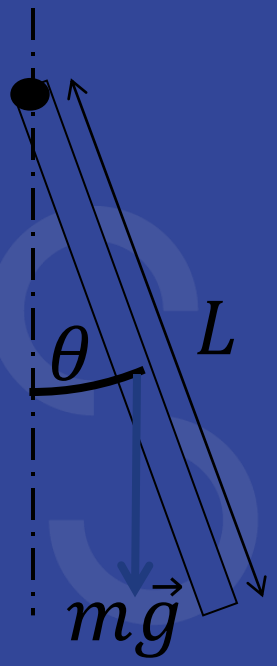


XCos: saisie graphique des systèmes

- Troisième exemple:
  - simulation d'un pendule
  - on possède un modèle de fonctionnement

- Equation différentielle:

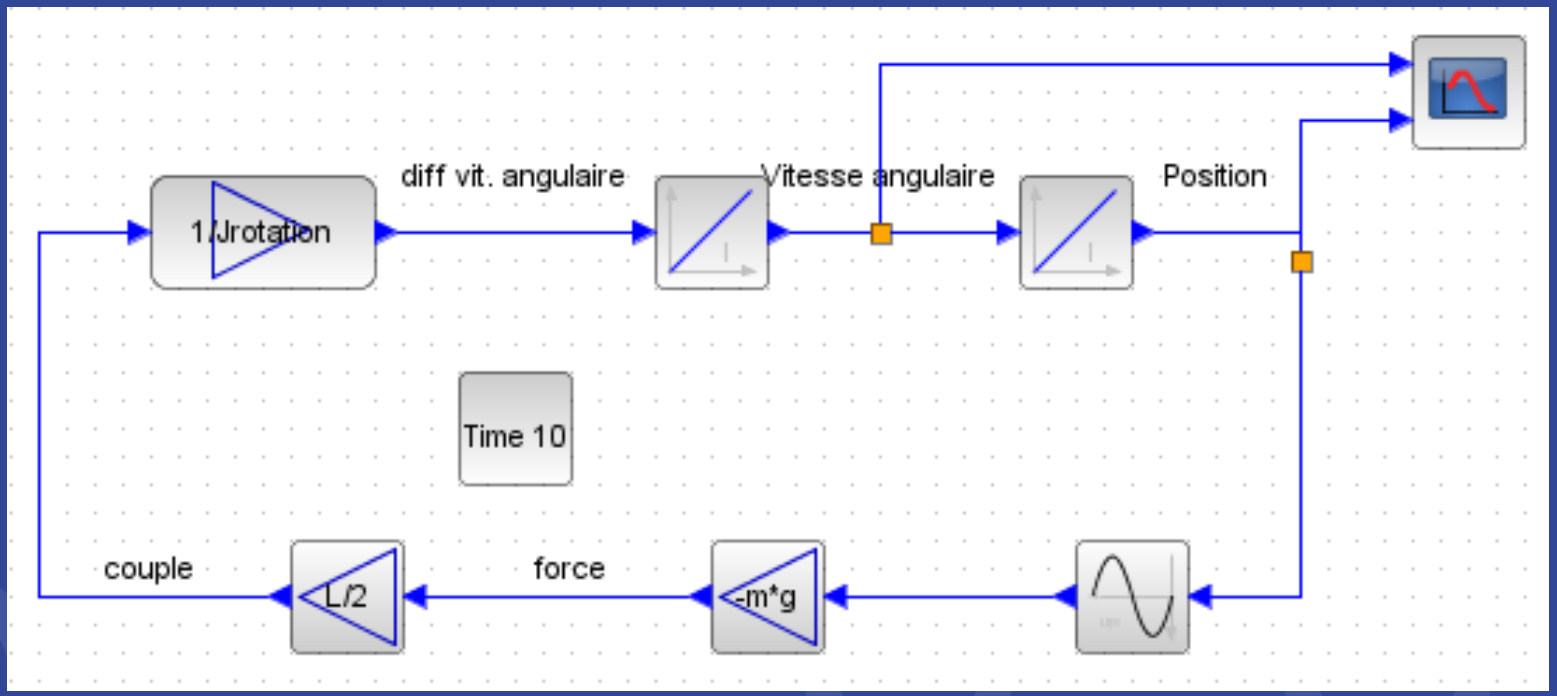
$$J\ddot{\theta} = -\frac{L}{2}mg\sin\theta$$



XCos: saisie graphique des systèmes

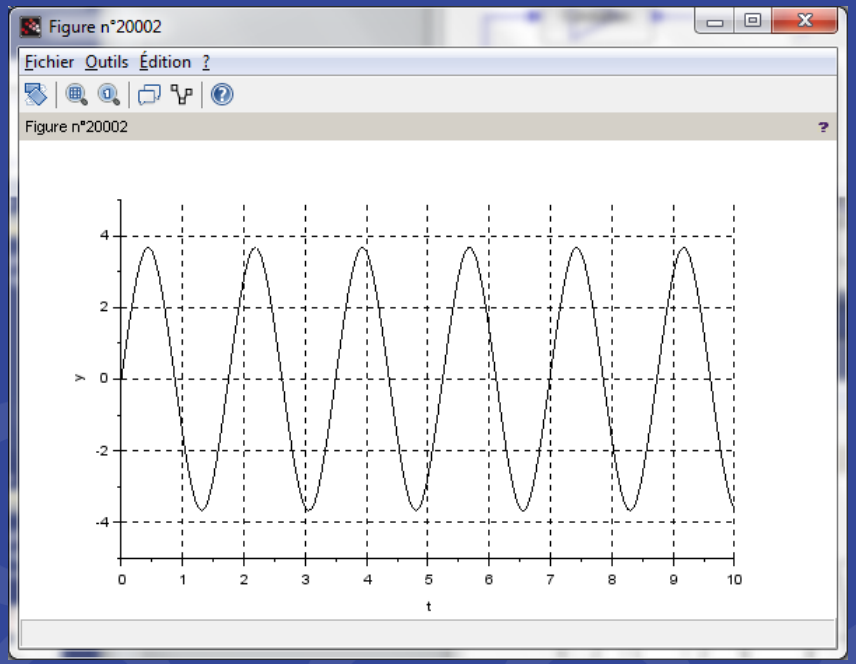
- Troisième exemple (pendule.xcos):

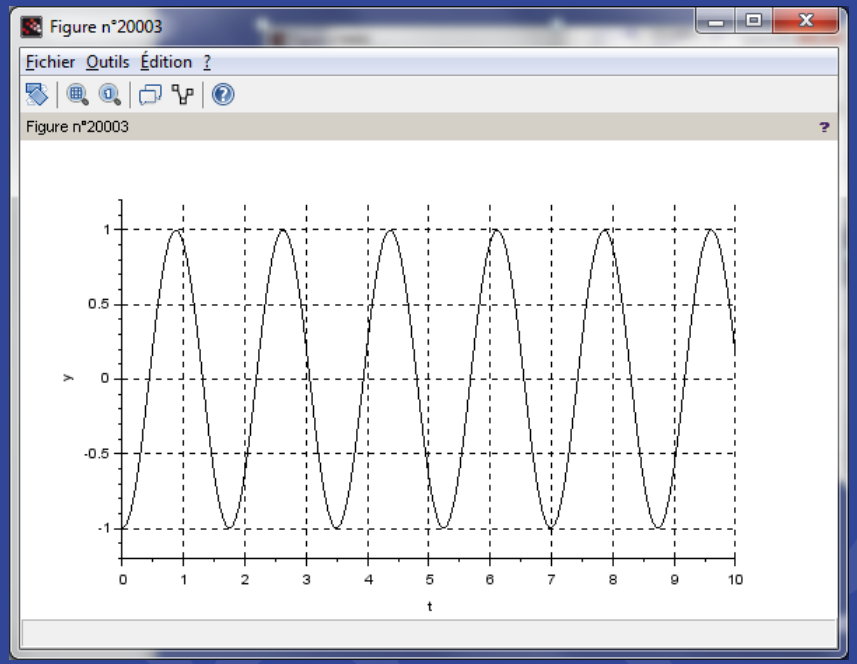
$$J\ddot{\theta} = -\frac{L}{2}mg\sin\theta$$



XCos: saisie graphique des systèmes

- Troisième exemple (pendule.xcos):

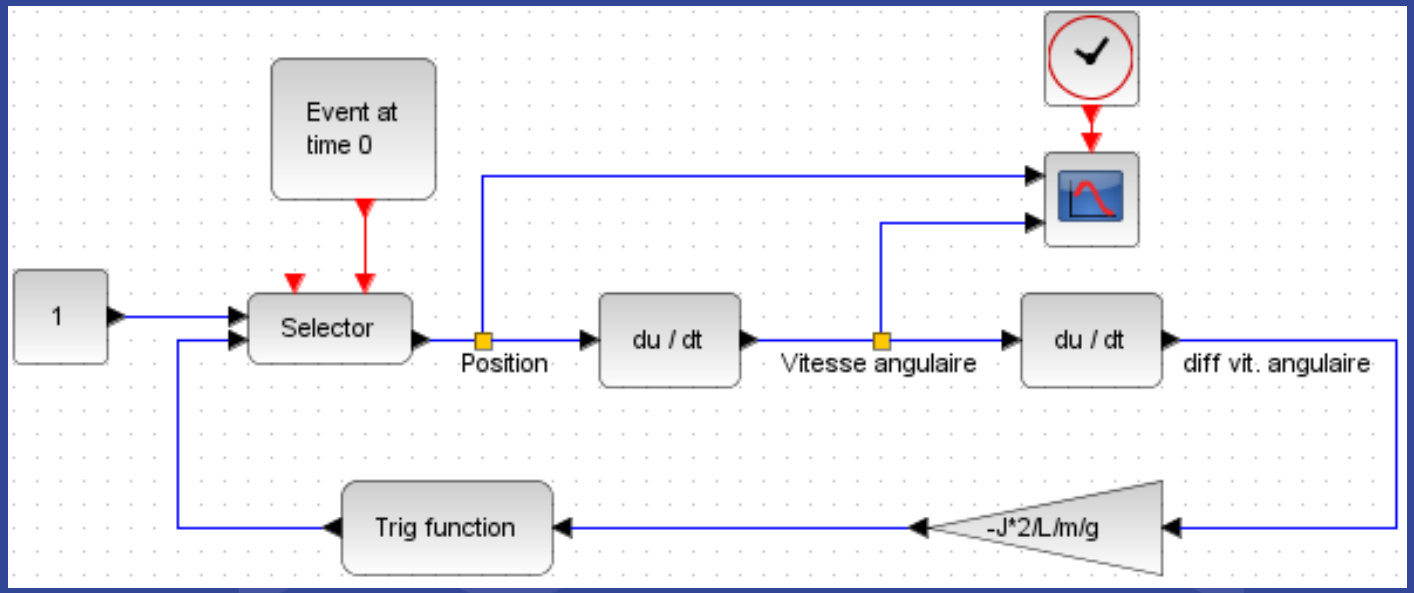




XCos: saisie graphique des systèmes

- Equation différentielle (pendule-deriv.xcos):

$$J\ddot{\theta} = -\frac{L}{2}mg\sin\theta$$

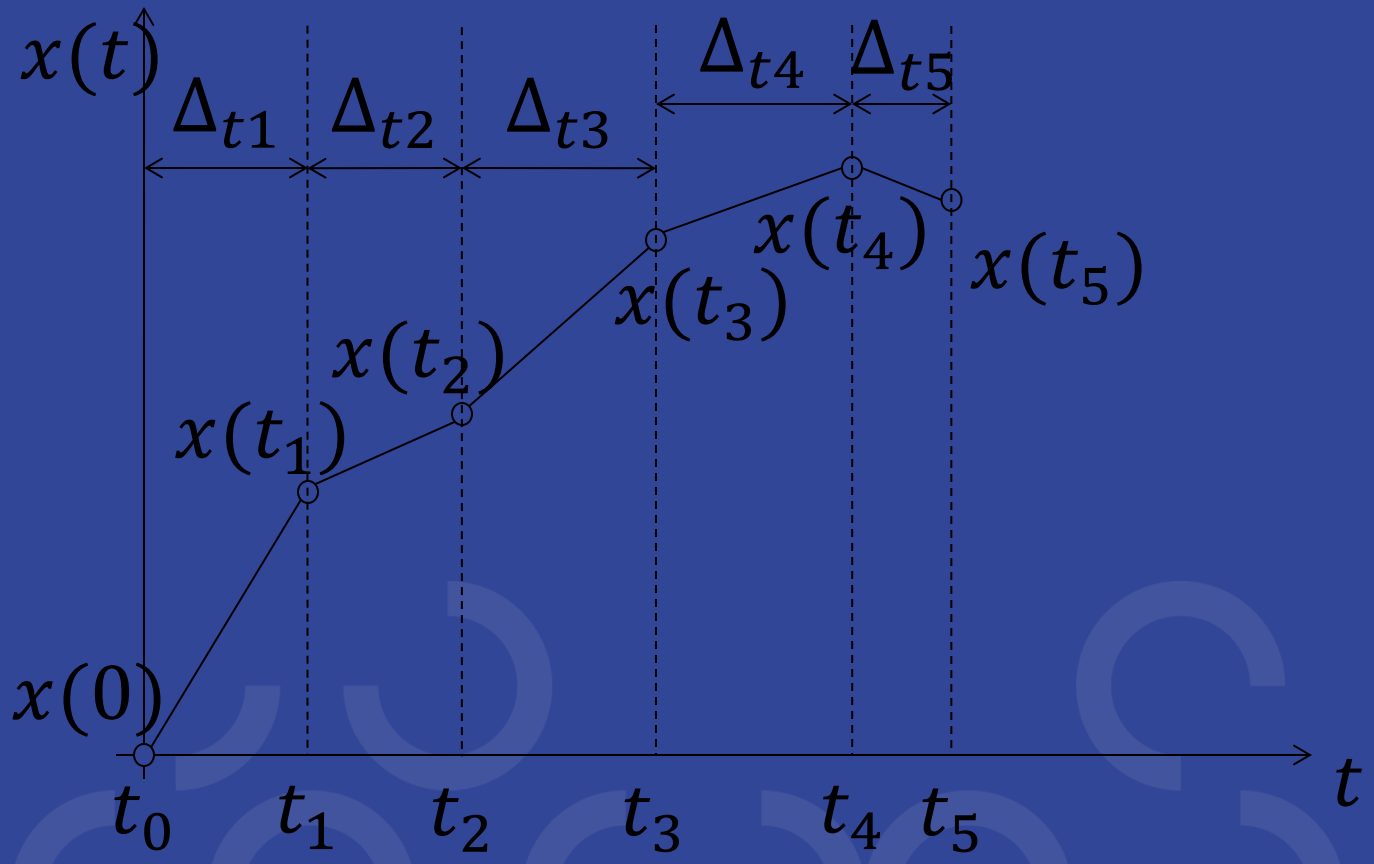


Ne peut pas être simulé! (boucle algébrique)

XCos: saisie graphique des systèmes

- Toujours utiliser des intégrateurs!
- Différentiateur à éviter, impossible de boucler la sortie sur l'entrée
- Limitation liée au mécanisme de simulation utilisé par Xcos
- Algorithme de simulation:
  - basé sur l'intégration des dérivées dans un système d'équations différentielles
  - Identique à l'algorithme de simulation SPICE
- Simulation numérique:
  - Précision des calculs
  - Les équations « des courbes » n'existent pas

XCos: saisie graphique des systèmes

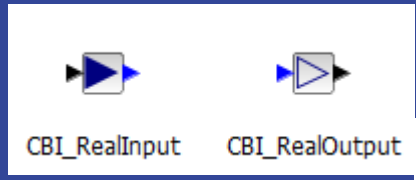


XCos: saisie graphique des systèmes

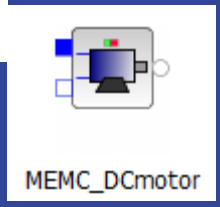
- Attention de ne pas mélanger les catégories incompatibles entre elles
- Type de la catégorie codé dans la forme et la couleur des entrées/sorties des blocs:
  - Flèches rouges = triggers
  - Flèches noires = information explicite causale
  - Flèches bleues = information implicite causale (image d'une grandeur physique acausale)
  - Carrés bleus/blancs = grandeur électrique acausale
  - Carrés rouges/blancs = grandeur thermique acausale
  - Cercles gris/blancs = grandeur mécanique en rotation
  - Carrés verts/blancs = grandeur mécanique en translation
  - Carrés gris/blancs = grandeur mécanique dans le plan

XCos: saisie graphique des systèmes

- Il existe quelques blocs de conversion:
  - SIMM/Utilitaires/Routage



  - SIMM/Composants/Actionneurs:



  - SIMM/Composants/Adaptateurs: poulies, différentiels, réducteurs, roues
  - SIMM/XXX/XXX/Sources: conversion d'une information acausale en grandeur physique acausale
  - SIMM/XXX/XXX/Mesures: conversion d'une grandeur physique acausale en information acausale

XCos: saisie graphique des systèmes

- Autres blocs de simulations:
  - Configuration et scopes dans SIMM/Utilitaires/Visualisation
  - Sources de stimuli dans SIMM/Signaux/Sources
  - Systèmes linéaires continus SIMM/Signaux/Continu
  - Opérations mathématiques SIMM/Signaux/Math
  - …

Manipulations proposées

- Modélisation d'un système masse-ressort en translation:
  - Masse ponctuelle $m = 150g$
  - Ressort raideur $k = 1N \cdot m^{-1}$
  - Position initiale $s = 0.1m$

  - Etude préalable du problème
  - Améliorations possibles:
    - Amortisseur $d = 0.1N \cdot m^{-1} \cdot s$
    - Gravité

Manipulations proposées

- Création d'un modèle de fonctionnement d'un moteur à courant continu:
  - $u(t) = e(t) + R \times i(t) + L\frac{di(t)}{dt}$
  - $J\frac{d\Omega(t)}{dt} = C_M(t) - C_R(t)$
  - $C_M(t) = K_I \times i(t)$
  - $e(t) = K_V \times \Omega(t)$

  - $e(t)$ dépend de $\Omega(t)$ qui est une variable d'état
  - $C_M(t)$ dépend de $i(t)$ qui est une variable d'état

Manipulations proposées

- Création d'un modèle de fonctionnement d'un moteur à courant continu:
  - $R = 1.2\ Ohm$
  - $L = 25\ mH$
  - $K_V = 0.05\ V \cdot rad^{-1} \cdot s$
  - $K_I = 0.04\ N \cdot m \cdot A^{-1}$
  - $J = 0.001\ kg \cdot m^2$

  - Alimentation électrique $U = 12\ V$

- Modèle à sauvegarder

Manipulations proposées

- Création d'un modèle de fonctionnement d'un moteur à courant continu:
  - Créer un « super-bloc » à partir de la MCC
  - Ajouter un réducteur de rapport 50 et de rendement 90%
  - Placer le moteur pour motoriser le pendule
  - Asservir le pendule en position

XCos: du modèle à l'équation différentielle

- Extraction automatique d'après le schéma
- Mise sous la forme « état »:

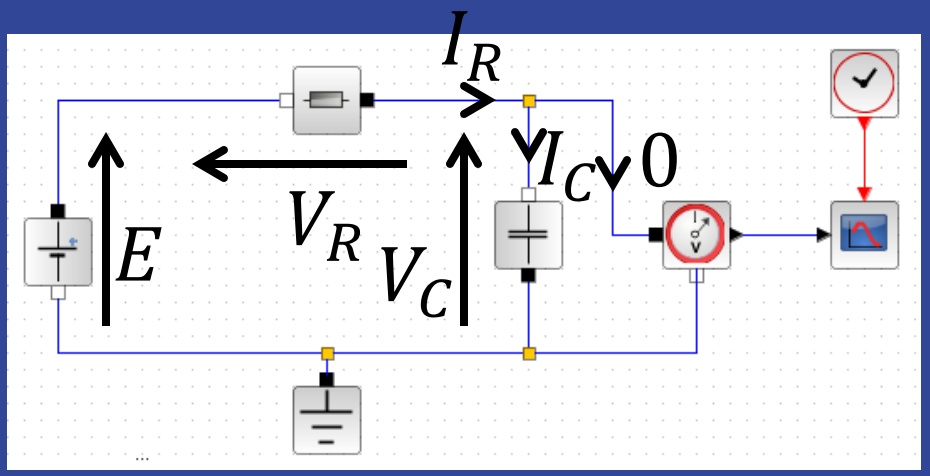
$$\begin{cases} \dot{X} = AX + BU \\ Y = CX + DU \end{cases}$$

- X état du système
- Y sortie du système
- U entrée du système

- A évolution propre de l'état
- B action des entrées sur l'état
- C action de l'état sur la sortie
- D action directe des entrées sur les sorties

XCos: du modèle à l'équation différentielle

- Exemple:



  - $E = V_C + V_R$
  - $V_R = RI_R$
  - $i_C = C\frac{dV_C}{dt} = C\dot{V}_C$
  - $i_R = i_C$

- $$\begin{cases} U = E \\ \dot{V}_C = \frac{-1}{RC}V_C + \frac{1}{RC}E \\ Y = V_C \\ V_c(0) = 0 \end{cases}$$

XCos: du modèle à l'équation différentielle

- Systèmes multi-entrées/multi-sorties/multi-états:

$$\begin{cases} \dot{X} = AX + BU \\ Y = CX + DU \end{cases}$$

- Vecteurs d'éléments:
  - X états du système (éléments x_1 à x_n)
  - Y sortie du système (y_1 à y_m)
  - U entrée du système (u_1 à u_r)
- Matrice d'actions:
  - A évolution propre de l'état (dimension $n \times n$)
  - B action des entrées sur l'état ($n \times r$)
  - C action de l'état sur la sortie ($m \times n$)
  - D action directe des entrées sur les sorties ($m \times r$)

Rappels de calculs matriciels

- Addition entre vecteurs:

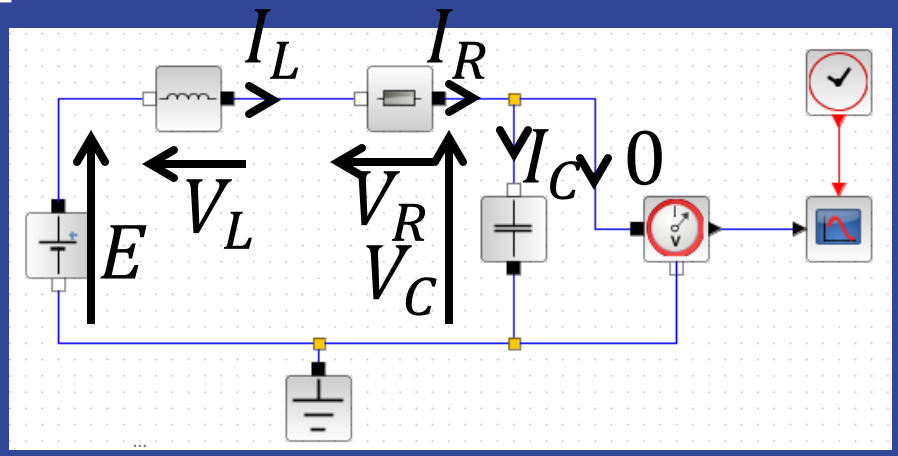
$$\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ \vdots \\ x_n + y_n \end{bmatrix}$$

- Multiplication d'une matrice par un vecteur:

$$\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,m} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} x_{1,i} y_i \\ \vdots \\ \sum_{i=1}^{m} x_{n,1} y_i \end{bmatrix}$$

Xcos: du modèle à l'équation différentielle

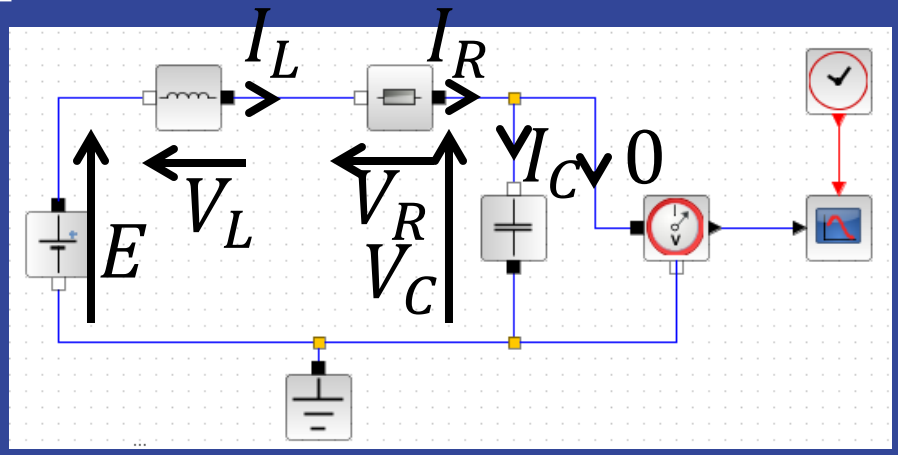
- Exemple:



  - $E = V_C + V_R + V_L$
  - $V_R = RI_R$
  - $V_L = L\frac{dI_L}{dt} = L\dot{I_L}$
  - $I_C = C\frac{dV_c}{dt} = C\dot{V_c}$
  - $I_R = I_c = I_L$
  - $$\begin{cases} \begin{bmatrix} \dot{I_L} \\ \dot{V_C} \end{bmatrix} = \begin{bmatrix} -R/L & -1/L \\ 1/C & 0 \end{bmatrix} \begin{bmatrix} I_L \\ V_C \end{bmatrix} + \begin{bmatrix} 1/L \\ 0 \end{bmatrix} [E] \\ [V_C] = [0 \quad 1] \begin{bmatrix} I_L \\ V_C \end{bmatrix} + [0][E] \end{cases}$$

Xcos: du modèle à l'équation différentielle

- Exemple:



  - 2 états ($n = 2$)
  - 1 sortie ($m = 1$)
  - 1 entrée ($r = 1$)
  - $X = \begin{bmatrix} I_L \\ V_C \end{bmatrix}$
  - $A = \begin{bmatrix} -{}^{R}/_{L} & {}^{-1}/_{L} \\ {}^{1}/_{C} & 0 \end{bmatrix}$
  - $B = \begin{bmatrix} {}^{1}/_{L} \\ 0 \end{bmatrix}$
  - $C = [0 \quad 1]$
  - $D = [0]$

XCos: formulaire Scilab

- Nécessité d'avoir des connaissances sur Scilab!
- Ecriture d'une variable: `x = 1`
- Vecteurs:
  - Création d'un vecteur ligne: `x = [2 3 4]`
  - Création d'un vecteur colonne: `x = [2;3;4]`
  - Lecture d'un élément d'un vecteur: `i = x(2)`
  - Ecriture dans un élément d'un vecteur: `x(3) = 2`
- Matrices:
  - Création d'une matrice: `x = [2 3 4;5 6 7]`
  - Lecture d'un élément d'une matrice: `i = x(2,3)`
  - Ecriture d'un élément d'une matrice: `x(2,3) = 4`

XCos: formulaire Scilab

- Matrices:
  - Récupération d'une colonne d'une matrice: `c=A(:,1)`
  - Récupération d'une ligne d'une matrice: `c=A(1,:)`
  - Affectation d'une colonne dans une matrice: `A(:,1)=[2 3 4]`
  - Affectation d'une ligne dans une matrice: `A(1,:)=[2 3 4]`
- Constantes:
  - 1.23e45 = 1.23×10^{45}
  - %pi, %i, %eps

XCos: formulaire Scilab

- Transformation d'un vecteur ligne en colonne: `x'`
- Les produits (attention aux dimensions):
  - Produit entre une matrice et un vecteur: `A*x`
  - Produit entre matrices: `A*B`
  - Produit scalaire entre vecteurs: `x*y`
  - Produit élément par élément: `x.*y`
- Les additions:
  - Additions élément par élément: `x+y`
- Les chaînes de caractères:
  - `'abc'` ou `"abc"`

XCos: formulaire Scilab

- Génération de valeurs équiréparties:
  `linspace(a,b,c)`

  `a:b`

- Le point-virgule à la fin d'une ligne indique de ne pas afficher le résultat dans la console

- Affichage de données: `plot(y,'r')`
- Affichage de données: `plot(x,y,'r')`
- Affichage cumulé de plusieurs courbes:
  `set(gca(),"auto_clear","off")`

- Aide d'une fonction: `help XXX`

XCos: formulaire Scilab

- `syslin`: crée un système linéaire invariant:
  - A temps continu ou discret ‘c’ ou ‘d’
  - A partir des matrices A, B, C & D
  - Renvoie un système linéaire invariant

- `csim`: simule un système linéaire invariant à temps continu ou discret:
  - Stimuli ‘step’, ‘impulse’ ou fonction entrées=f(t)
  - Instants de simulation
  - Système créé à partir de syslin
  - Valeurs initiales de l’état
  - Renvoie les sorties aux instants t (matrice)

Xcos: de l'équation diff. au résultat

- Systèmes linéaires continus invariants:

```
mat2_A = [-R/L -1/L;1/C 0];
mat2_B = [1/L;0];
mat2_C = [0 1];
mat2_D = [0];

sl2=syslin('c',mat2_A,mat2_B,mat2_C,mat2_D);

t=linspace(0,1,100);
y=csim("step",t,sl2);

plot(t,y);
```

Xcos: de l'équation diff. au résultat

- Systèmes linéaires continus invariants:

```
mat2_A = [-R/L -1/L;1/C 0];
mat2_B = [1/L;0];
mat2_C = [0 1];
mat2_D = [0];

sl2=syslin('c',mat2_A,mat2_B,mat2_C,mat2_D);

t=linspace(0,1,100);
y=csim("step",t,sl2);

plot(t,y);
```

Xcos: de l'équation diff. au résultat

- Systèmes non-linéaires?
- Systèmes non-continus?
- Systèmes non-invariants?

- Nécessité de recalculer les matrices A, B, C & D à chaque point de fonctionnement
- Eventuellement ne pas construire les matrices mais utiliser un modèle de fonctionnement

- ODE: « Ordinary Differential Equations »
  - Système de plusieurs équations différentielles
  - Toujours sous une forme d'état!
  - Passage par une fonction de calcul « perso »

Xcos: de l'équation diff. au résultat

- Problème non-linéaire précédent:

$$J\ddot{\theta} = \frac{L}{2} mg \sin\theta$$

- Utilisation de la fonction `ode(X0,0,t,f)`:

$$X_0 = \begin{bmatrix} \dot{\theta}(t=0) \\ \theta(t=0) \end{bmatrix}$$

```
t=linspace(0,10,1001);
```

Xcos: de l'équation diff. au résultat

- Problème non-linéaire précédent:

$$J\ddot{\theta} = \frac{L}{2} mg \sin\theta$$

- Utilisation de la fonction `ode(X0,0,t,f)`:

$$\begin{bmatrix} \ddot{\theta} \\ \dot{\theta} \end{bmatrix} = f\left(t, \begin{bmatrix} \dot{\theta} \\ \theta \end{bmatrix}\right)$$

```
function Xdot = f(t,X),
    Xdot(1) = -m*g*L/2/J*sin(X(2));
    Xdot(2) = X(1);
endfunction
```

Xcos: de l'équation diff. au résultat

- Problème non-linéaire précédent:

```
t=linspace(0,10,1001);
vitesse0=0;
position0=-1;
X0=[vitesse0;position0];
y=ode(X0,0,t,f);
vitesse=y(1,:);
position=y(2,:);
plot(t,position);
```

Xcos: de l'équation diff. au résultat

- Comment fonctionnent `csim` et `ode`?

- Calcul de Δt:
  - Basé sur l'activité des variables d'état
  - Bornes hautes et basses fixées globalement
  - Notion de précision sur chacune des variables d'état

- Calcul des variables d'état:
  - Intégration de la dérivée retournée par le modèle
  - Eventuelles contraintes

Xcos: de l'équation diff. au résultat

- Comment fonctionnent `csim` et `ode`?
- Problème biaisé:
  - Erreurs d'intégration
- `csim` est une surcouche d'`ode`
- Implémentation d'algorithmes faiblement biaisés:
  - Algorithmes itératifs
  - Convergence?
  - Adams/Newton-Gauss/Euler-Gauss/Runge-Kutta
  - …

Xcos: de l'équation diff. au résultat

- Intégration Runge-Kutta à l'ordre 1 (Euler):
  - $X(t_{n+1}) = X(t_n) + (t_{n+1} - t_n) \times \dot{X}(t_n)$
- Intégration Runge-Kutta à l'ordre 2 (prédicteur-correcteur):
  - $X\left(t_{n+\frac{1}{2}}\right) = X(t_n) + \frac{t_{n+1} - t_n}{2} \dot{X}(t_n)$
  - $X(t_{n+1}) = X(t_n) + (t_{n+1} - t_n) \times \dot{X}\left(t_{n+\frac{1}{2}}\right)$
- Intégration Runge-Kutta à l'ordre 4…

Manipulations proposées

- Appropriation des modèles existants
- Ecriture et simulation d'un modèle (MCC)
- Amélioration d'un modèle (MCC+pendule)
- Limitation des algorithmes
- Fitting?
- 4h30 de TP

Ecriture d'un problème (MCC)

- Moteur à courant-continu:
  - $\mathrm{E(t)} = U(t) - RI(t) - L\dot{I}(t)$
  - $E(t) = K\Omega(t)$
  - $C(t) = KI(t)$
  - $J\dot{\Omega} = \sum C$
- Modèle Xcos déjà fait
- Variables d'états = ?
- Relever Ω, I, θ

Amélioration d'un problème

- Placement du moteur sur l'axe du pendule
- Adaptation des schémas Xcos
- Variables d'états = ?
- Stimulations du système = ?

Amélioration d'un problème

- Placement du pendule+moteur dans un repère mobile:
  - Houlogénérateur
  - La position dépend des vagues (sinusoïde)
  - Récupération d'énergie (dans une résistance)

- Données à mesurer:
  - Position du pendule
  - Position du support
  - Puissance récupérée

Limitation des algorithmes

- Gravitation d'un objet autour de la terre
- Rappels:
  - $\vec{F} = G\frac{m_1 m_2}{\|u_{12}\|^2}\overrightarrow{u_{12}}$
  - Constante de gravitation $G = 6{,}6742 \times 10^{-11}\ N \cdot m^2 \cdot kg^{-2}$
  - Masse de la terre $m_1 = 5{,}972 \times 10^{24}\ kg$
  - Masse du satellite $m_2 = 10^3\ kg$
  - Vitesse de satellisation $V = \sqrt{\frac{Gm_1}{\|u_{12}\|^2}}$
- Variables d'états = ?
- Mise en équations du système
- Ecriture du système
  - Fonctions utiles: `sqrt atan`

Limitation des algorithmes

- Gravitation d'un objet autour de la terre
- Résultats:
  - Post-calculs de la distance et de la vitesse d'après les variables d'états
  - Influence de la précision de simulation sur les résultats

- Implémentation « perso » d'ode?

Fitting = régression

- Faire coller une équation sur des résultats
  - de simulation
  - expérimentaux

- Méthode simple pour les modèles polynomiaux:

```
function [p]=polyfit(x, y, n, s)
  // return coeff vector or poly if fourth string argument
given
  [lhs, rhs] = argn(0)
  x = x(:); y = y(:)
  m = length(x)
  if length(y) <> m, error('x and y must have same length'),
end
  v = ones(m,n+1)
  for i=2:n+1, v(:,i) = x.*v(:,i-1), end
  p = (v\y)'
  if rhs > 3, p = poly(p, s, 'coeff'), end
endfunction
```

Fitting = régression

- Méthode simple pour les modèles polynomiaux:

```
p=polyfit(x, y, n, s)
x,y = vecteurs définissant les points
n = ordre d'interpolation
s = nom de la variable, par exemple 'x'
```

- Exemple d'utilisation:

```
x=1:10
y=1+x+x^2
p=polyfit(x,y,2,"x")
c=coeff(p)
```

Fitting = régression

- Méthode générale pour tout modèle:

```
[ecart,modele_opt]=leastsq(fonction_cout, modele_init)
fonction_cout = fonction de calcul de coût
modele_init = modèle initial (vecteur de coefficients)
ecart = ecart pour le modèle identifié
modele_opt = modèle identifié
```

- Modèle à identifier:

```
courbe_x=1:10
courbe_y=10*(1-exp(-courbe_x/3))
```

- Exemple de fonction coût pour une exponentielle:

```
function cout = cout_exp(modele)
    exp_modele = modele(1)*(1-exp(courbe_x/modele(2)));
    diff = exp_modele - courbe_y;
    cout = sum(diff.*diff);
endfunction
```

Fitting = régression

- Méthode générale pour tout modèle:

```
[ecart,modele_opt]=leastsq(fonction_cout, modele_init)
fonction_cout = fonction de calcul de coût
modele_init = modèle initial (vecteur de coefficients)
ecart = ecart pour le modèle identifié
modele_opt = modèle identifié
```

- Exemple d'appel:

```
modele_init(1) = 8;
modele_init(2) = 2;
[ecart,modele_opt] = leastsq(cout_exp, modele_init)
```

- Exercice: fitting de la réponse sur la charge de capacité

Acquisitions avec Scilab et modules NI

- Utiliser les facultés de programmation et de traitement des données de Scilab pour interfacer du matériel
- Se passer de Labview

- Matériel requis: module National Instruments (quasiment tous sont supportés)

- Logiciel requis:
  - Scilab
  - module ATOMS NIDAQ
  - pilote de la carte, logiciel National Instruments « Measurement and Automation »

Acquisitions avec Scilab et modules NI

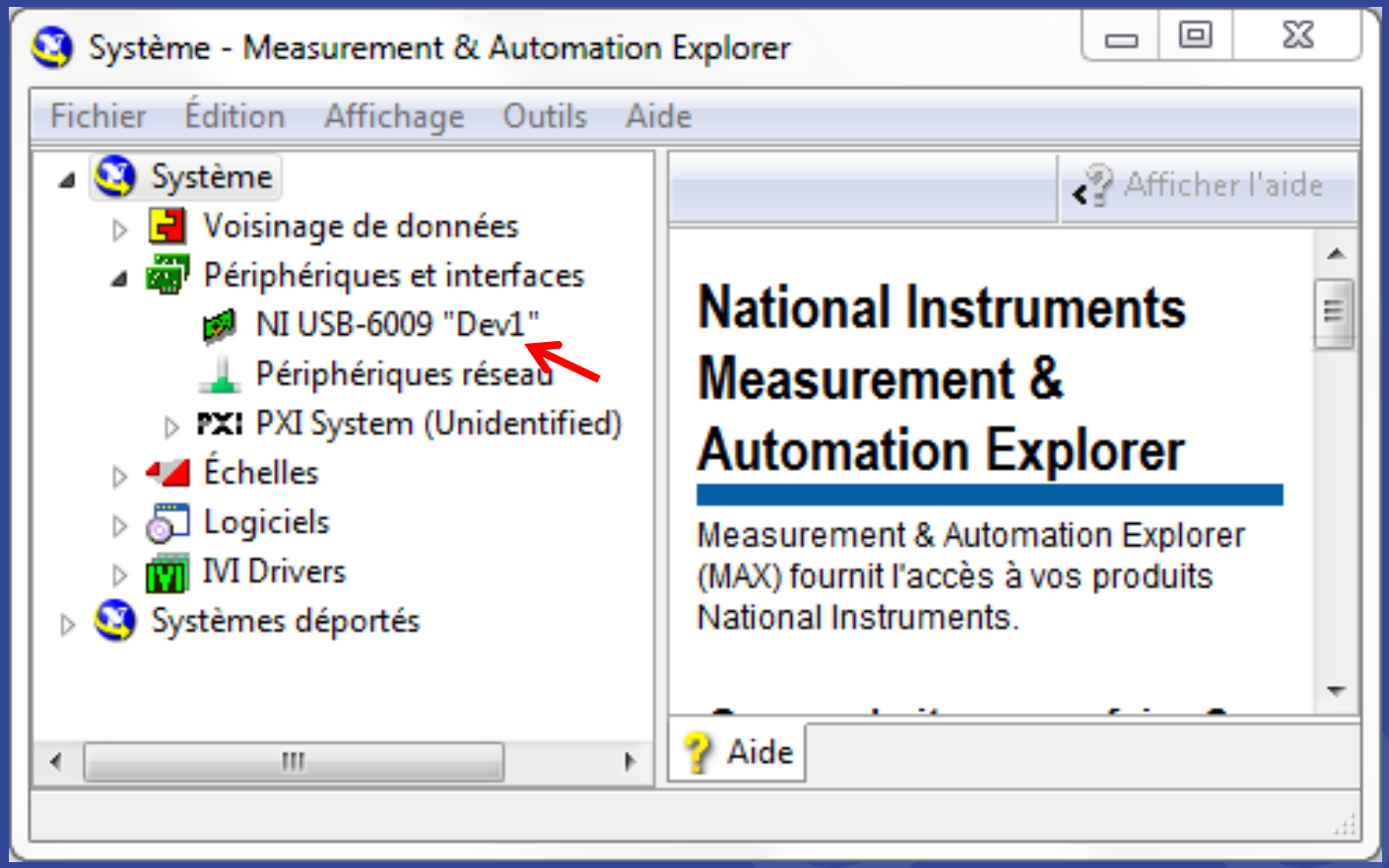
- Plusieurs étapes à réaliser:
  - Identifier le nom de la carte connectée
  - Créer la tâche d'acquisition
  - Ajouter les canaux d'entrées ou de sorties
  - Configurer l'horloge d'échantillonnage
  - Lancer la lecture ou l'écriture
  - Fermer la tâche d'acquisition

- Pour chaque action, vérifier la bonne exécution

- Finalement, assez peu de documentation sur internet (comme souvent)

Acquisitions avec Scilab et modules NI

- Identifier le nom de la carte connectée:



Acquisitions avec Scilab et modules NI

- Créer la tâche d'acquisition:

```
[task,err] = DAQ_CreateTask("");
DAQ_ErrChk(task,err);
```

- Possibilité de créer plusieurs tâches en parallèle:
  - Entrées ou sorties
  - exécutions successives

Acquisitions avec Scilab et modules NI

- Ajouter les canaux d'entrées:

```
err = DAQ_CreateAIVoltageChan(task,
        "Dev1/ai0",DAQ("Val_RSE"),-10,+10);
DAQ_ErrChk(task,err);
```

Vmin Vmax

- Spécification du couplage de l'entrée:
  - `Val_RSE` pour mode commun (potentiel référencé à la masse)
  - `Val_Diff` pour mode différentiel (différence de potentiels)
- Spécifications de la carte pour les plages supportées

Acquisitions avec Scilab et modules NI

- Ajouter les canaux de sorties:

```
err = DAQ_CreateAOVoltageChan(task,
      "Dev1/ao0",0,5);
DAQ_ErrChk(task,err);
```

Vmax

Vmin

- Spécifications de la carte pour les plages supportées

Acquisitions avec Scilab et modules NI

- Configurer l'horloge d'échantillonnage:

```
err = DAQ_CfgSampClkTiming(task,
        "OnboardClock",10,
        DAQ("Val_Rising"),
        DAQ("Val_FiniteSamps"),20);
DAQ_ErrChk(task,err);
```

Fe

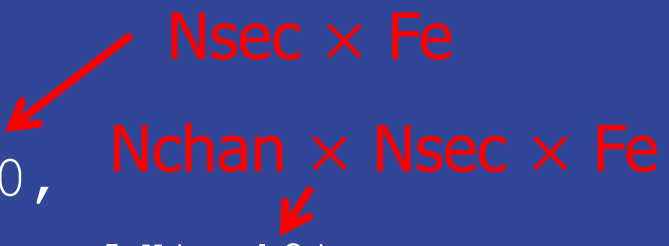
Nsec × Fe

- Utilisation de l'horloge interne à la carte
- Possibilité de synchroniser avec une horloge externe

Acquisitions avec Scilab et modules NI

- Lancer la lecture:

```
err = DAQ_StartTask(task);
DAQ_ErrChk(task,err);
[values,samples,err] =
  DAQ_ReadAnalogF64(task,20,
    -1,DAQ("Val_GroupByChannel"),40);
DAQ_ErrChk(task,err);
```



- Values = vecteur concaténant les échantillons de toutes les voies:
  - `Voie1 = values(1:samples);`
  - `Voie2 = values(samples+1:2*samples);`
  - `Voie3 = values((2*samples)+1:3*samples);`

Acquisitions avec Scilab et modules NI

- Lancer l'écriture:

```
err = DAQ_StartTask(task);
DAQ_ErrChk(task,err);
[samples,err] =
  DAQ_WriteAnalogF64(task,1,-1,
    DAQ("Val_GroupByChannel"),[1 2]);
DAQ_ErrChk(task,err);
```

Nech

Nech × Nchan

- Création du vecteur de valeur à envoyer:
  - `[Voie1 Voie2 Voie3]`

Acquisitions avec Scilab et modules NI

- Fermer la tâche d'acquisition:

```
err = DAQ_StopTask(task);
DAQ_ErrChk(task,err);
clear task;
```

Acquisitions avec Scilab et modules NI

- Comment faire un système bouclé (asservi):
  - Identifier la réponse impulsionnelle
  - Déterminer le correcteur

- Implémentation d'un correcteur
  - Lire la réponse du système (1 échantillon)
  - Ecrire la commande (1 échantillon)
  - Correcteur à implémenter

- Script « asservissement.sce »: création d'une base de temps « ad-hoc » avec les fonctions `tic()` et `toc()`

Acquisitions avec Scilab et modules NI

- Identifier la réponse impulsionnelle:
  - Méthode de Strejc ou de Broïda sur les SLCI 2nd ordre
  - Estimation du retard dans la chaîne numérique = Te

- Implémentation d'un correcteur proportionnel:
  - `commande = K_P * (consigne - reponse);`

- Implémentation d'un correcteur proportionnel-intégral:
  - $C(p) = \frac{U(p)}{\varepsilon(p)} = K_P\left(1 + K_I\frac{1}{P}\right)$
  - $C(z) = \frac{U(z)}{\varepsilon(z)} = K_P\left(1 + K_I\frac{T_e}{2}\frac{z+1}{z-1}\right)$

Liens utiles

- Scilab: http://www.scilab.org/
- Octave: http://www.gnu.org/software/octave/
- Matlab: http://www.mathworks.fr/products/matlab/
- Méthodes d'intégration numériques: http://sin-web.paris.ensam.fr/IMG/pdf/C5.pdf
- Fitting: http://wiki.scilab.org/Non%20linear%20optimization%20for%20parameter%20fitting%20example