

# Modelica

Un langage pour modéliser et simuler  
des systèmes dynamiques hybrides.  
Des systèmes cyber-physiques...

`http://www.lirmm.fr/~reitz/Modelica`

Philippe.Reitz@LIRMM.fr

Équipe MAREL

novembre 2017

# Sommaire

## 1. Présentation

1a. Les systèmes dynamiques

1b. Les automates hybrides

1c. Modelica : grandes lignes

1d. Un exemple de A à Z : le circuit RC

## 2. Le langage Modelica

2a. Les classes

2a1. Différentes sortes

2a2. Les types de base

2a3. Les variables

2a4. Modèles et connecteurs

2b. La dynamique

2b1. Les équations

2b2. Les algorithmes

2b3. Les fonctions

2c. L'héritage

2d. Notions non étudiées ici

## 3. OpenModelica

3a. Les outils

3b. Exercices

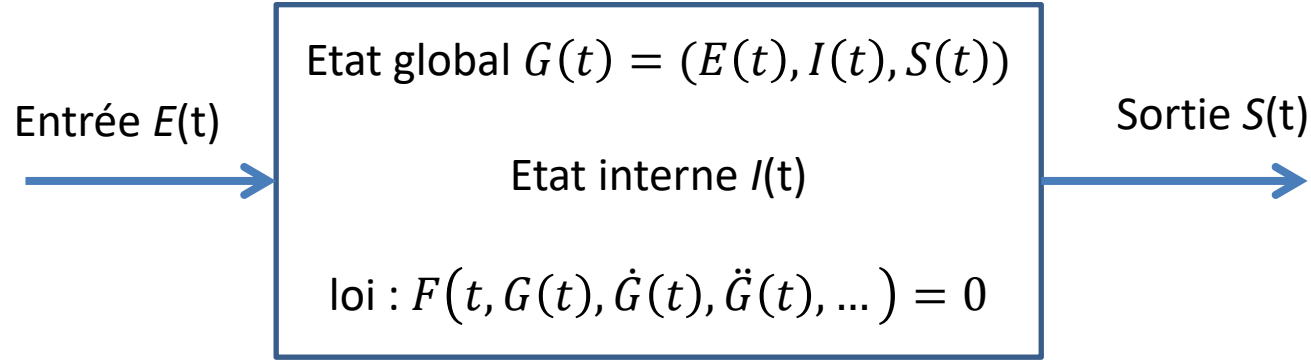
# 1a. Un exemple

- Un bâtiment intelligent (smart building)
    - Gestion au mieux des fluides
      - Air (conditionnement, filtrage)
      - Lumière (naturelle, artificielle)
      - Température (régulation climatisation, chauffage)
      - Electricité (régulation puissance, sources)
      - Numérique
      - Eau, gaz, ...
    - Gestion des accès
  - Caractéristiques
    - De nombreux capteurs
    - De nombreux actionneurs
    - Un pilotage entièrement informatisé
- } Domaines de la Physique multiples

# 1a. Contexte

- Modéliser et simuler des systèmes complexes
  - La notion de système (au sens systémique, théorie des ...)
    - Systèmes dynamiques classiques (cadre continu)
    - Systèmes informatiques (cadre discret)
    - Systèmes dynamiques hybrides (cadre continu et discret)
    - Systèmes cyber-physiques
      - Contrôle/pilotage informatisé au maximum
      - Interaction forte avec notre monde (capteurs, actionneurs)
        - » implique plusieurs champs disciplinaires de la physique
  - Modélisation multi-domaines
- Langages informatiques dédiés (*DSL*)
  - Modéliser à partir de composants prédéfinis
  - Modéliser en écrivant des équations (*EBP*)
    - Simuler en les résolvant dans un contexte donné

# 1a. Les systèmes dynamiques classiques (1/2)



- Modélisation dans un cadre continu (1947 N. Wiener)
  - Cadre classique (théorie générale des systèmes/systemique) :
    - $\Sigma$  = espace des *états* (ou des *phases*) [continu]
      - Variables d'entrée
      - Variables de sortie
      - Variables internes
      - Variables paramètres (réglages)
    - $t$  = temps [continu]
    - $f$  = fonction de transfert ; loi de comportement :  $S = f(E, t)$ 
      - Expression des contraintes entre états et temps
        - » en général, un système d'équations différentielles  $F$

# 1a. Les systèmes dynamiques classiques (2/2)

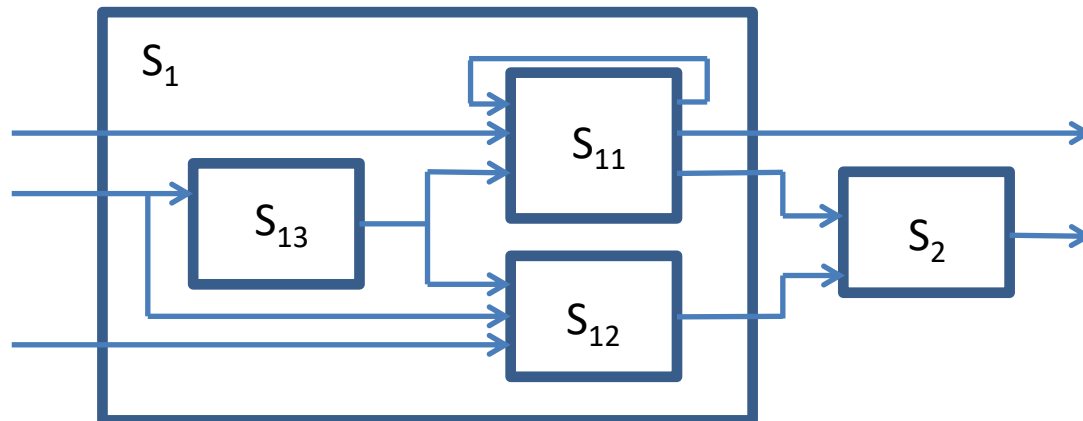
- spatialisation ou non du système
  - pas d'espace explicite :
    - Modèle = équation différentielle ordinaire (EDO [ODE])
      - Théorie des systèmes  $\Rightarrow$  fonctions de transfert
        - » Cadre usuel de l'automatique (électronique)
    - Modèle = équation différentielle et algébrique (EDA [DAE])
      - EDA = EDO + contraintes algébriques (inéquations, ...)
  - espace explicite [continu]
    - Modèle = équation aux dérivées partielles (EDP [PDE])  $\Rightarrow$  différentiation par rapport à toutes les dimensions : temps ( $t$ ) et espace ( $x_1, x_2, \dots$ ).
      - Cadre usuel de tout domaine lié à la physique

# 1a. Les systèmes dynamiques discrets

- Modélisation dans un cadre discret
  - Dimensions toutes discrétisées
    - $\Sigma$  = espace des *états* (ou *phases*) [discret]
    - $t$  = *temps* [discret]
    - $S$  = *espace* [discret]
  - Modèle = automate (fonction calculable  $f$ ) calculant l'état suivant connaissant l'état courant :
$$G_{n+1} = f(G_n)$$
    - ou plus généralement une version stochastique (déterminisme pas nécessaire, type *automates de Markov*)

# 1a. Les systèmes dynamiques

- Approche compositionnelle naturelle
  - Interconnexion des systèmes
    - Le monde extérieur est un système de loi inconnue
    - Plusieurs entrées peuvent être reliées ensemble
    - Deux sorties ne peuvent être reliées ensemble
    - Toute entrée doit être reliée à une sortie
    - Connectabilité contrôlable par des étiquettes (*prises*)
      - Chaque variable d'entrée ou sortie est associée à une étiquette
      - Deux variables sont connectables si leurs étiquettes coïncident
  - Décomposition en sous-systèmes





# 1a. Quid de modéliser – simuler ?

- Modéliser =
  - spécifier entièrement un système dynamique
  - spécifier ne signifie pas nécessairement que la loi de comportement est explicitable
    - Une équation différentielle n'est qu'une contrainte devant être satisfaite par cette loi
- Simuler =
  - placer le système dans un état initial, et observer comment il évolue (trajectoire dans l'espace des états/phases)  $\Rightarrow$  notion de *processus*

# 1a. Les systèmes dynamiques :

## une vision *systemique* de la programmation

- Programmer = modéliser
  - C'est construire explicitement un espace d'état et une loi de comportement (trajectoires conformes à une spécification)
  - Tout programme qui produit un résultat doit se terminer  
⇒ la trajectoire d'état converge vers un point fixe – notion d'*attracteur* dans les systèmes dynamiques
- Exécuter un programme = simuler
  - C'est placer le système dans un état initial (spécification des données d'entrée)
  - C'est laisser la trajectoire des états se dérouler (*processus*), jusqu'à obtention du point fixe (extraction des données de sortie)

# 1a. Les systèmes dynamiques hybrides

- Comment mixer ces deux approches de modélisation :
  - systèmes purement continus (systémique)
  - systèmes purement discrets (informatique)en faisant en sorte que le formalisme permette :
  - de répondre aux mêmes questions théoriques que les formalismes existants
  - de produire un programme exécutable
- Réponse = les *systèmes dynamiques hybrides*
  - formalisme 1 : les automates hybrides
  - formalisme 2 : les EDA hybrides (Modelica)

# 1a. Synthèse des approches de modélisation de systèmes (spécialisés ou pas)

Début	Etat	Temps	Espace	Modèles associés
1940	Continu	Continu	-	Systèmes dynamiques : EDO (équations différentielles ordinaires) ; calculateurs analogiques ; ...
1950	Continu	Continu	Continu	Systèmes dynamiques spécialisés : EDP (équation aux dérivées partielles) ; ...
1990	Continu	Discret	-	Systèmes dynamiques à temps discret ; équations aux différences ; automates à alphabet continu (machine de Blum, Shub et Smale [BSS]) ; ...
1950	Continu	Discret	Discret	Eléments finis, ...
1980	Discret	Continu	-	Modélisation à événements discrets, ...
1940	Discret	Discret	-	Modèles de calcul numérique (Turing, $\lambda$ -calcul, ...)
1970	Discret	Discret	Continu	Gaz sur réseau, ...
1950	Discret	Discret	Discret	Automates cellulaires, calcul numérique spécialisé, modèles bio-inspirés, chimio-inspirés, ...
1995	Continu + Discret	Continu + Discret	-	Automates hybrides ; EDA (équations différentielles et algébriques hybrides) ; ...

# 1b. Les automates hybrides [Henzinger'96]

- Présentation rapide  
*(quelques détails juste après)*
  - Automate hybride = automate caractérisé par :
    - son état
      - état discret + état continu
    - sa loi de transition
      - l'état discret évolue selon un automate
      - l'état continu évolue selon une EDO
  - L'état évolue selon la loi de transition en s'appuyant sur un temps continu
    - état initial donné
    - alternance de phases continues et de transitions d'état instantanées.

# 1b. Les automates hybrides (1/2)

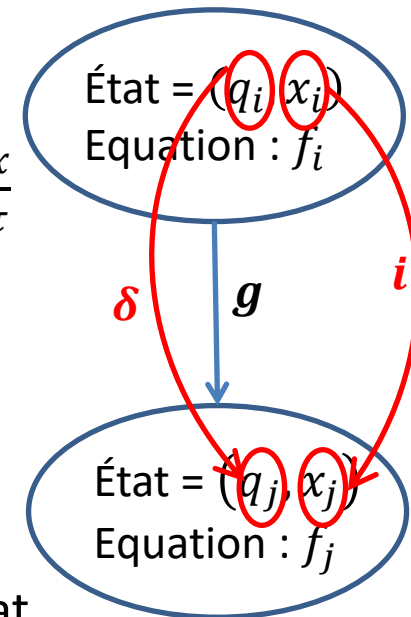
- Le modèle :

- état =  $(q, x, \tau)$  avec :

- temps  $\tau$  (pris dans un ensemble continu  $T$ )
- état discret  $q$  (pris dans un ensemble discret fini  $Q$ )
- état continu  $x$  (pris dans un ensemble continu  $E$ ) ;  $\dot{x} = \frac{dx}{d\tau}$

- loi de transition, décomposée en 4 fonctions :

- $f(q, x, \dot{x}, \tau) = 0$  : équation différentielle régissant l'évolution de l'état continu en phase stable
- $g(q, x, \dot{x}, \tau) \in \{0, 1\}$  : prédicat détectant une transition d'état
- $\delta(q, x, \dot{x}, \tau) \in Q$  : fonction régissant la transition de l'état discret
- $i(q, x, \dot{x}, \tau) \in E$  : fonction régissant la transition de l'état continu (réinitialisation)



# 1b. Les automates hybrides (2/2)

- Sémantique

- état courant =  $(q, x, \tau)$

- phase stable

- phase stable durant  $(\tau, \tau')$

$$\Leftrightarrow \forall t \in [\tau, \tau'[, g(q, x, \dot{x}, t) = 0$$

- durant toute cette phase, à chaque instant  $t$ ,

- l'état discret de l'automate reste à  $q$

- l'état continu  $x$  évolue selon l'équation différentielle  $f$

$\Rightarrow$  *système continu pur*

- phase de transition

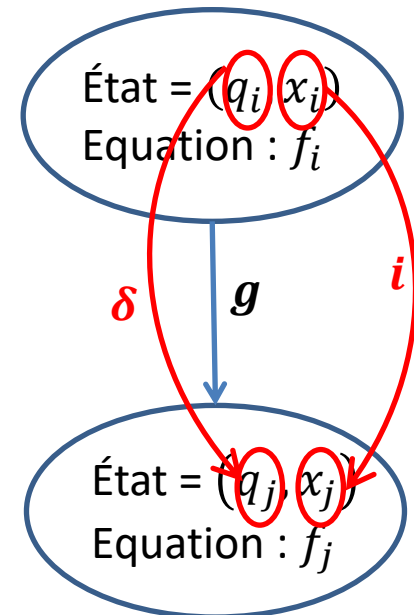
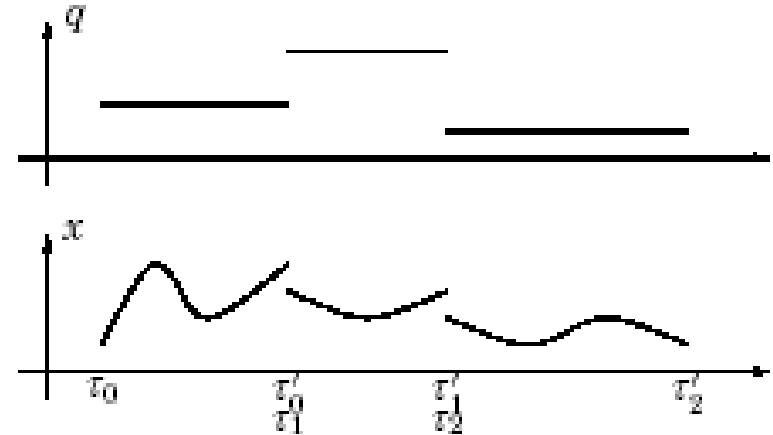
- transition de phase à l'instant  $t \Leftrightarrow g(q, x, \dot{x}, t) = 1$

- lorsqu'une transition de phase est détectée, alors :

- l'état discret de l'automate passe à  $\delta(q, x, \dot{x}, t)$

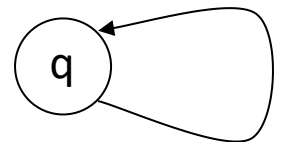
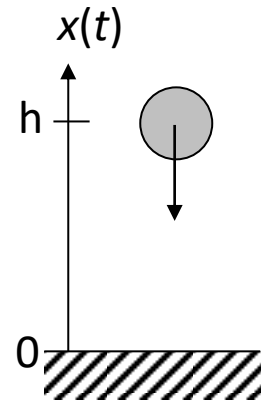
- l'état continu  $x$  est réinitialisé à  $i(q, x, \dot{x}, t)$

$\Rightarrow$  *système discret pur*



# 1b. Les automates hybrides – exemple 1

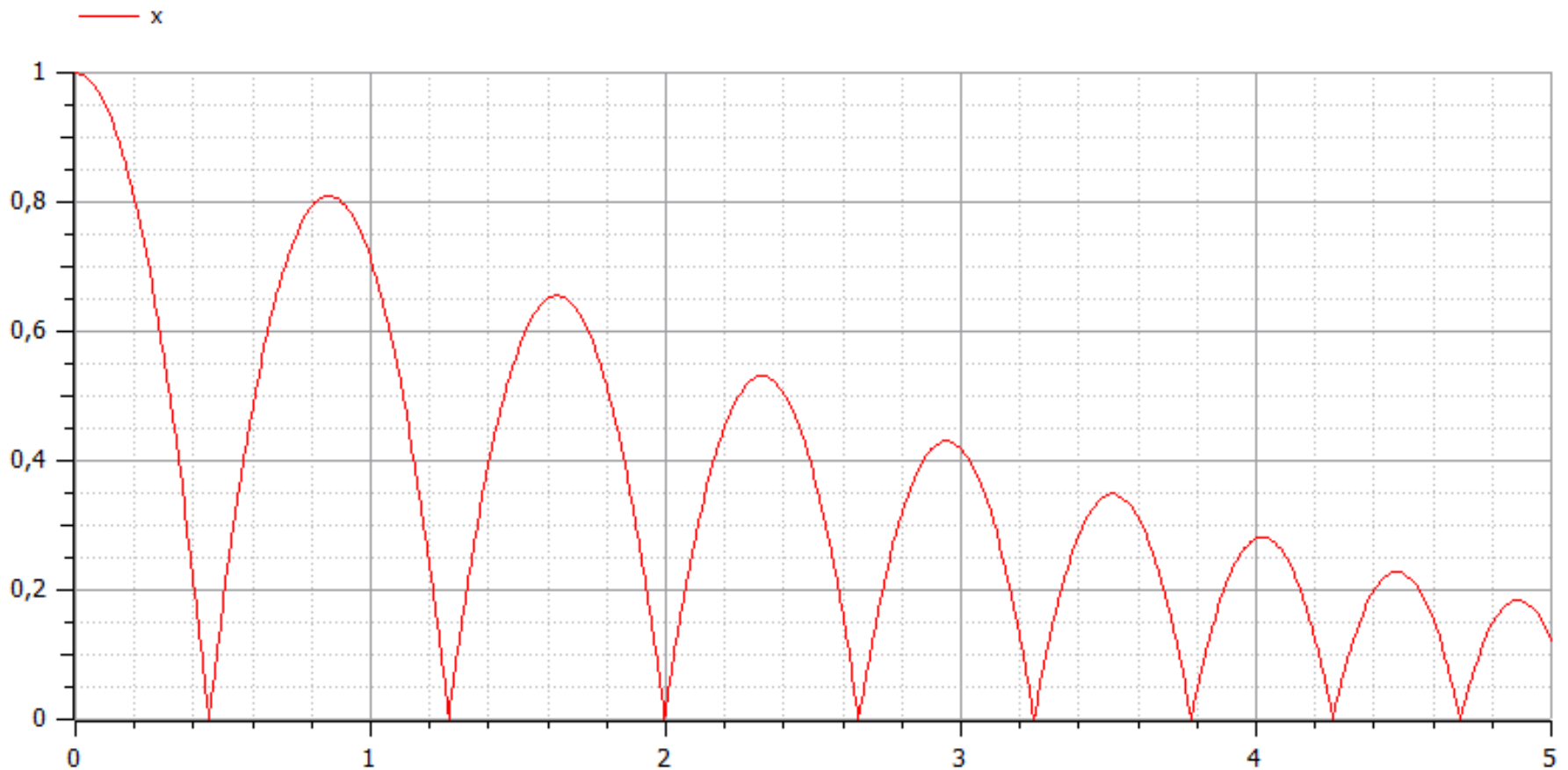
- Une balle rebondissante : le modèle
  - Position verticale de la balle :  $x$ 
    - $\Rightarrow$  état continu :  $(x, \dot{x}) = q$
  - Equation de la dynamique :  $\begin{cases} v = \dot{x} \\ \dot{v} = -g \end{cases}$ 
    - accélération = pesanteur  $g > 0$
  - Conditions initiales :  $x(0) = h$  et  $v(0) = 0$
  - Un seul état, une seule transition :
    - Condition de la transition :  $x = 0$  et  $v < 0$
    - Réinitialisation si transition :  $v := -c \cdot v$ 
      - coefficient d'amortissement au rebond :  $c \in [0,1]$





# 1b. Les automates hybrides – exemple 1

- Une balle rebondissante : une simulation ( $h=1$ )



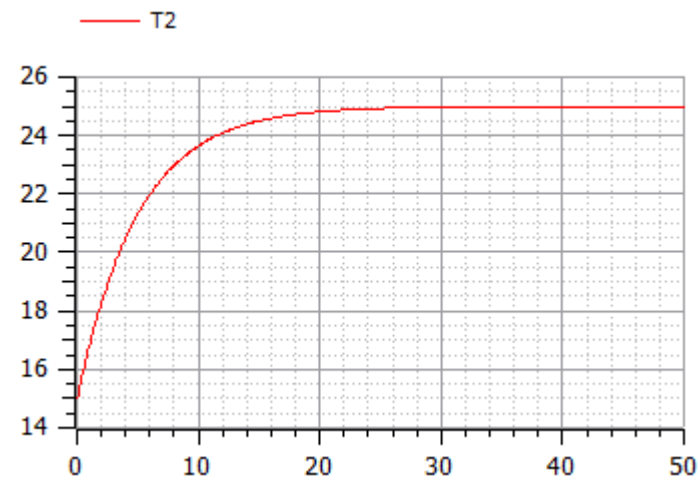
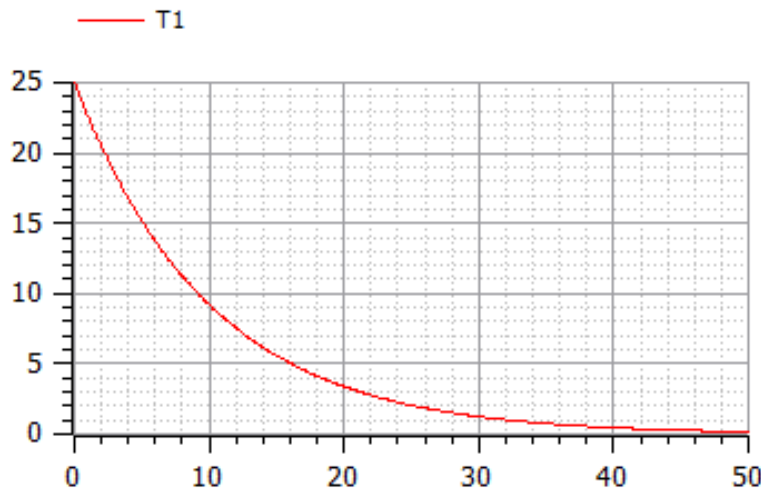
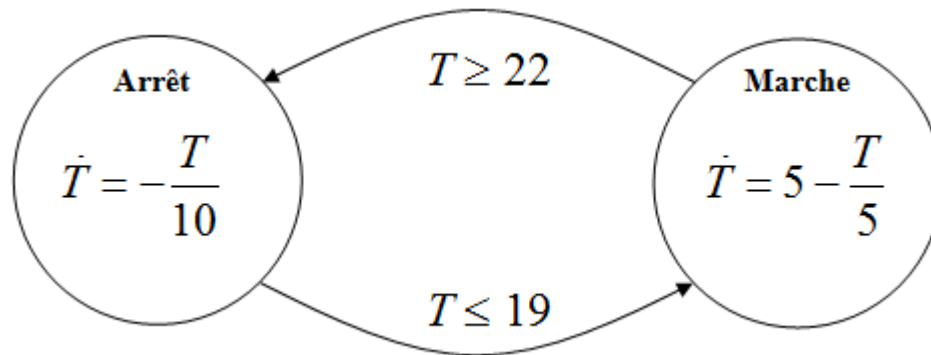
# 1b. Les automates hybrides – exemple 1

- Une balle rebondissante : le code Modelica

```
1 model balleRebondissanteBasique
2   constant Real g = 9.81 ;
3   parameter Real c = 0.9 ;
4   parameter Real h0 = 1;
5   Real x (start = h0) ;
6   Real v (start = 0) ;
7 equation
8   der( x ) = v ;
9   der( v ) = -g ;
10  when x <= 0 and v<0 then
11    reinit ( v , -c*pre(v) ) ;
12    reinit ( x , 0 ) ;
13  end when ;
14 end balleRebondissanteBasique;
```

# 1b. Les automates hybrides – exemple 2

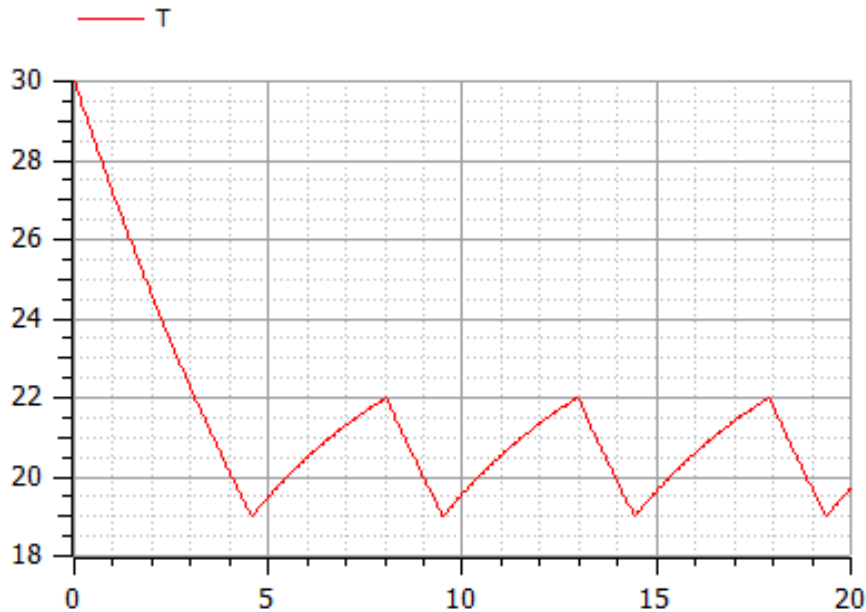
- Un thermostat simple : le modèle
  - Température de la pièce :  $T$



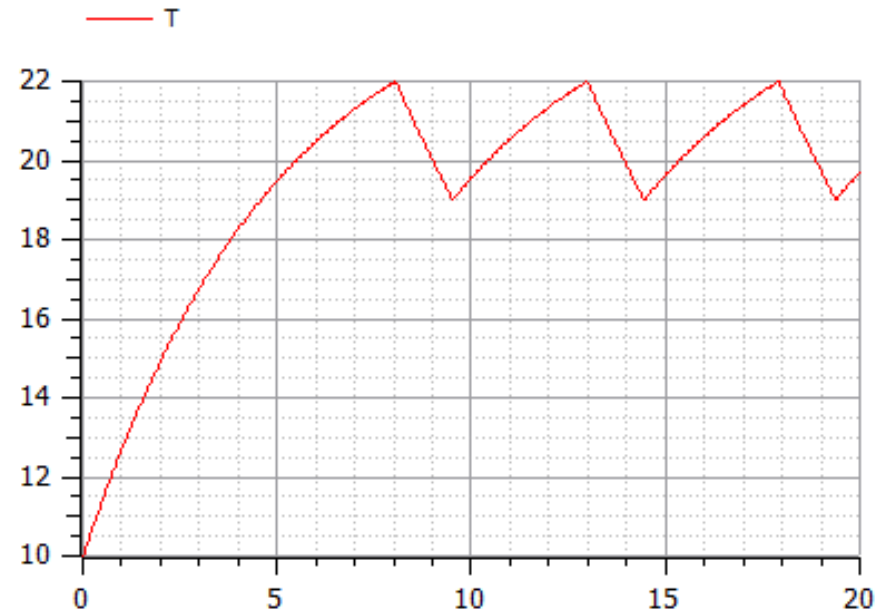
# 1b. Les automates hybrides – exemple 2

- Un thermostat simple : quelques simulations

$T(0) = 30$



$T(0) = 10$



# 1b. Les automates hybrides - bilan

- Principaux résultats théoriques
  - dans le cas général, ils sont tous dramatiques (questions indécidables, au mieux NP)
    - Atteignabilité : quelle entrée  $e$  pour obtenir une sortie  $s$
    - Contrôlabilité : atteignabilité en temps borné
    - Stabilité : si  $e$  change, comment change  $s$

*Exposants de Lyapunov, système chaotique (effet papillon)*
  - dans quelques cas particuliers (automates hybrides linéaires), les automaticiens ont réussi à trouver des algorithmes *raisonnables* pour résoudre leurs problèmes (cf. vérification).
    - système HYTECH d'Henzinger.
    - système TEMPO de N. Lynch

# 1c. Modelica (1/2)

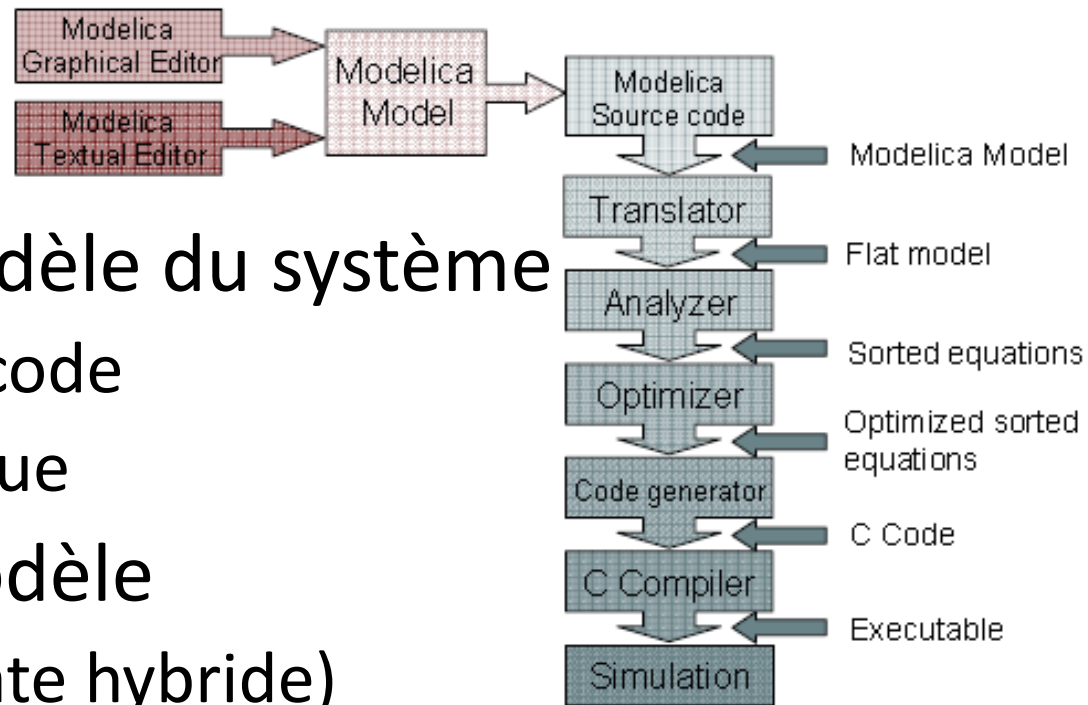
- Un langage créé vers 1995 pour modéliser/simuler des systèmes dynamiques hybrides
  - issu d'une communauté de modélisateurs, familière de l'approche objet (école scandinave, à l'origine de **Simula** [1967])
- Un langage déclaratif (orienté équations), orienté objet/composant
  - un comité de normalisation
    - Site de référence : <http://modelica.org>
  - devenu un langage d'échange de modèles entre la majorité des outils de modélisation/simulation existants

# 1c. Modelica (2/2)

- langage d'expression d'un modèle ou d'une simulation
  - compréhensible par le modélisateur (EDO, EDP)
  - traitable informatiquement
- langage spécialement adapté à la modélisation de systèmes
  - applicable à tous les domaines (multi-domaines)
    - Robotique, Mécatronique, Electronique, Systèmes de puissance, Hydraulique, Aérodynamique, Thermique, Biologie, Chimie, ...
- définition *propre* de tous les objets manipulés
  - sémantique basée sur le formalisme des EDA hybrides, équivalent à celui des automates hybrides

# 1c. Modéliser / simuler avec Modelica

## : processus général (1/3)



### 1. définition d'un modèle du système

- Écriture directe du code
- Conception graphique

### 2. compilation du modèle

- Mise à plat (automate hybride)
- Génération du code C équivalent

### 3. définition des paramètres d'une simulation

### 4. exécution du programme de simulation



# 1c. Modéliser / simuler avec Modelica

## : processus général<sup>(2/3)</sup>

### 1. Modéliser = définition d'un modèle du système

- langage orienté composant
  - variables d'état discrètes et/ou continues
  - dynamique (temps continu) décrite par
    - équations différentielles et algébriques
    - programmes
  - héritage entre composants
    - relation d'héritage : modifier/compléter/étendre des composants existants
    - liaison dynamique de code, avec quelques limitations
  - composants réutilisables multi-domaines (bibliothèques de composants)
- modèle = assemblage de composants
  - éditeurs graphiques

# 1c. Modéliser / simuler avec Modelica : processus général (3/3)

## 2. Simuler =

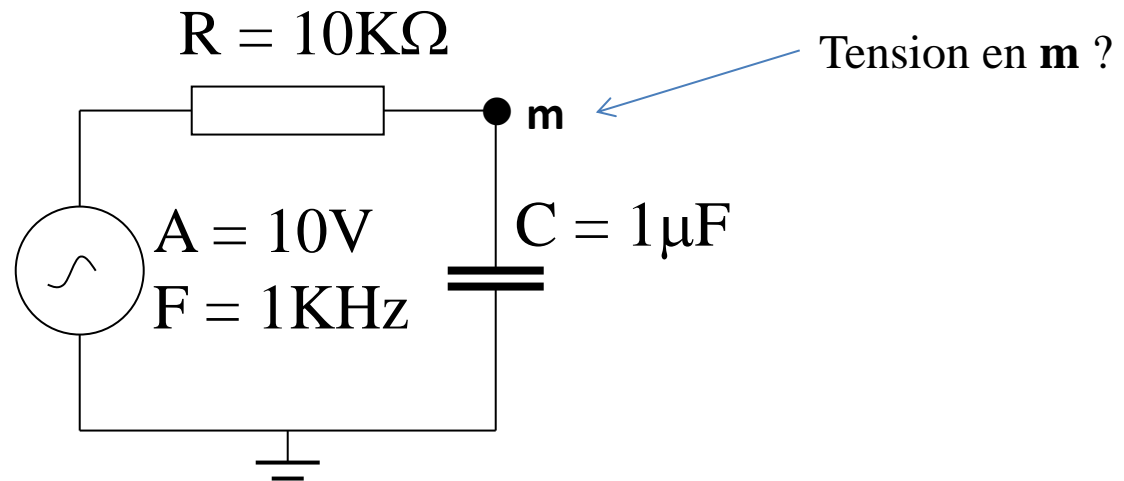
- a. compilation du modèle paramétré
  - mise à plat du modèle objet
  - production d'un système d'EDA hybride
- b. définition des paramètres d'une simulation
  - paramètre = variable dont la valeur
    - est librement choisie avant de lancer la simulation
    - restera invariante durant la simulation
- c. exécution du programme de simulation
  - exploitation de solveurs d'EDA hybrides adaptés au problème

# 1c. Programmer en Modelica

- Solutions libres :
  - **OpenModelica**  $\Rightarrow$  solution adoptée dans ce cours
  - *JModelica*
  - *Scilab/Scicos/Xcos* (Inria)
- Solutions commerciales :
  - *Matlab/Simulink* de MathWorks (pas compatible *Octave*)
  - *Maple/Maplesim* de MapleSoft
  - *Mathematica/SystemModeler* de Wolfram Research
  - *LabView* de National Instruments
  - *Catia/Dymola* de Dassault Systems

# 1d. Modelica sur un exemple

- Les concepts clés sur un exemple électronique simple : le circuit RC (filtre passe-bas)



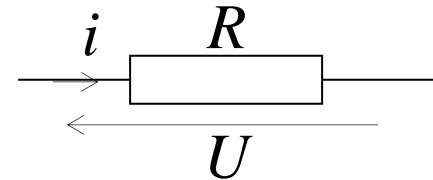
- Modélisation classique
- Modélisation en Modelica à partir de rien
- Modélisation en Modelica à partir de composants prédéfinis

# 1d. Le circuit RC - modélisation

- Rappels (?) d'électronique :

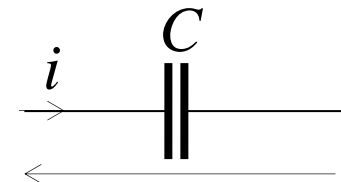
- Résistance :

$$U(t) = R i(t)$$



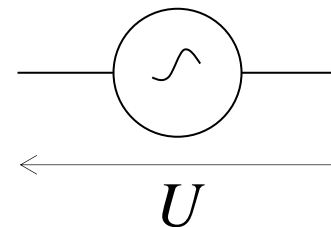
- Condensateur :

$$i(t) = C \frac{d U(t)}{d t} \text{ ou } U(t) = U(0) + \frac{1}{C} \int_0^t i(\tau) d \tau$$



- Source de tension :

$$U(t) = A \sin 2 \pi F t$$



# 1d. Le circuit RC – modélisation

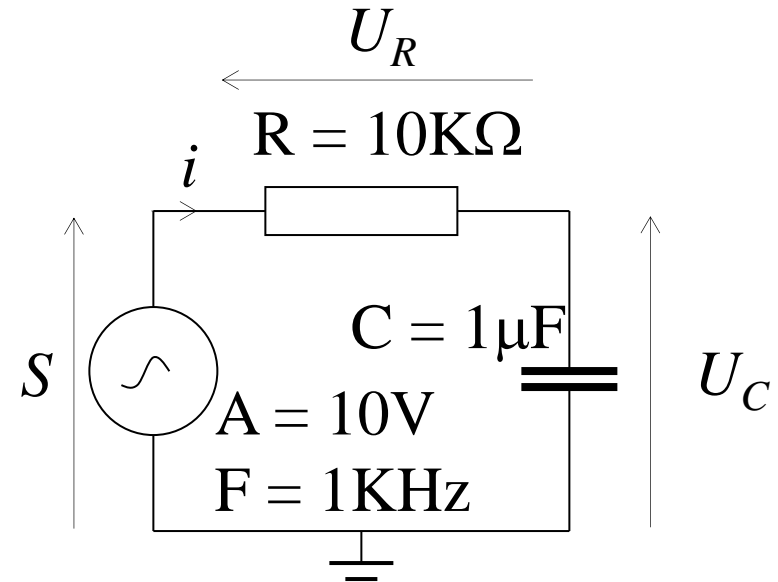
- Explicitation des équations

$$U_R = R i$$

$$i = C \frac{dU_C}{dt}$$

$$S = A \sin 2 \pi F t$$

$$U_R + U_C = S$$



- Remise en forme finale en une seule équation (d'inconnue  $U_C$ , la tension à étudier) :

$$R C \frac{dU_C}{dt} + U_C = A \sin 2 \pi F t$$

# 1d1. Le circuit RC – résolution directe

- L'équation différentielle :

$$R C \frac{dU_C}{dt} + U_C = A \sin 2 \pi F t$$

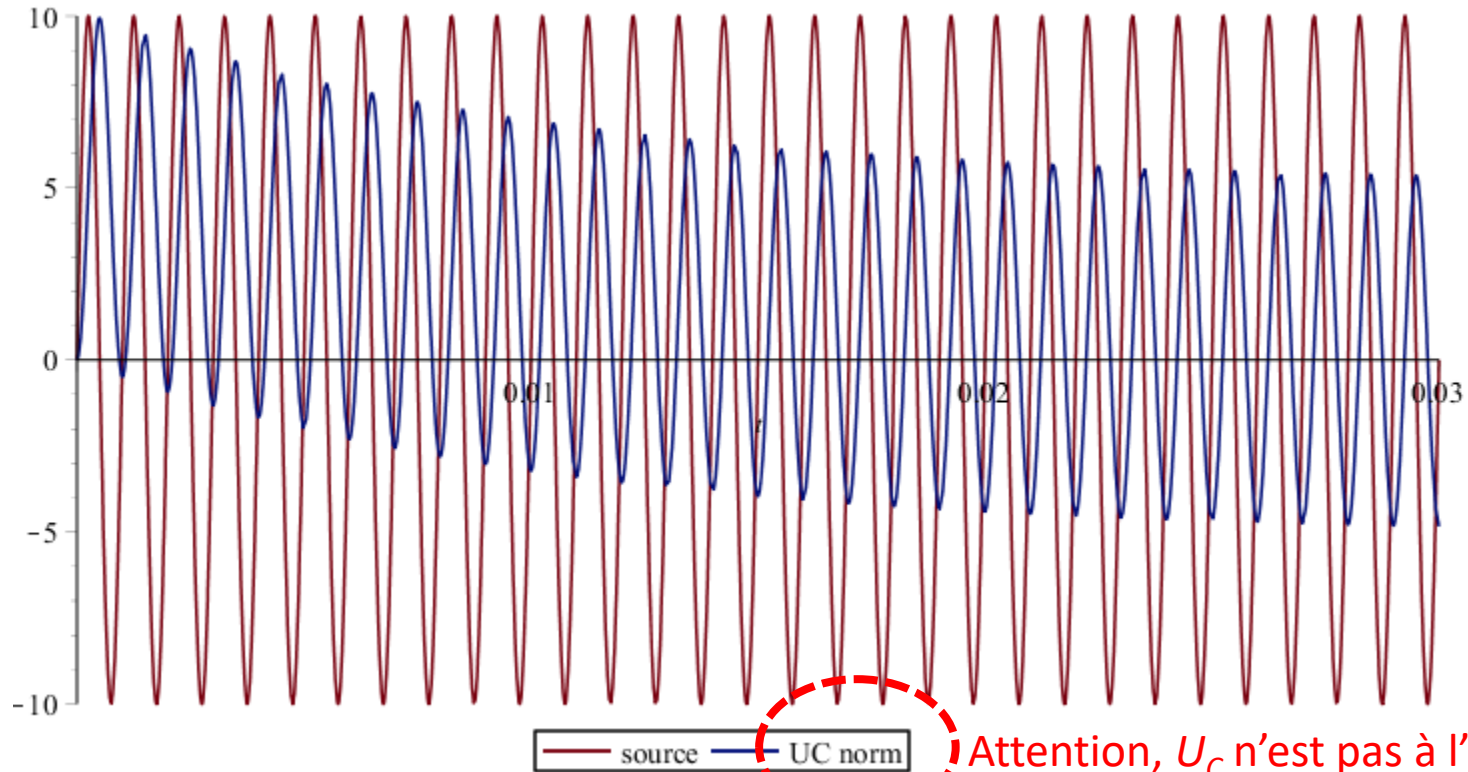
a pour solution générale (en posant  $\omega = 2 \pi F$ , et  $k$  une constante) :

$$U_C(t) = \frac{k e^{\frac{-t}{RC}} - A(R C \omega \cos \omega t - \sin \omega t)}{1 + (R C \omega)^2}$$

La valeur de  $k$  peut-être déduite à partir des conditions initiales, par exemple en posant  $U_C(0) = 0$  (condensateur initialement déchargé) ; on trouve alors  $k = A R C \omega$

# 1d1. Le circuit RC - simulation

- Evolution de la tension  $U_C$  dans le temps :



$$U_C(t) = A \frac{R C \omega \left( e^{-\frac{t}{RC}} - \cos \omega t \right) + \sin \omega t}{1 + (R C \omega)^2}$$



# 1d1. Le circuit RC – premier bilan

- La résolution directe exacte faite précédemment suppose que l'on sache résoudre une telle équation différentielle :
  - S'y attaquer vaillamment
  - Transformer l'équation en se plaçant dans le domaine de Laplace, de Fourier, ...
  - Exploiter un outil de calcul symbolique (*Maple, Xcas, Mathematica, ...*)
- Résoudre de telles équations n'est pas toujours possible
  - Tenter de trouver une solution approchée



passer

## 1d2. Le circuit RC – transformée de Laplace

a. Transformée de Laplace :

$$\begin{array}{ccc} & & \text{domaine de Laplace} \\ \text{domaine temporel} & \xrightarrow{\mathbf{L}} & \\ \widehat{u(t)} & \rightarrow & \widehat{U(p)} = \int_0^{\infty} e^{-pt} u(t) dt \end{array}$$

Notre équation :

$$R C \frac{du_C}{dt} + u_C = A \sin \omega t$$

est transformée en :

$$R C (p U - u_C(0)) + U = A \frac{\omega}{\omega^2 + p^2}$$

$$\text{soit } (R C p + 1)U = A \frac{\omega}{\omega^2 + p^2}$$

## 1d2. Le circuit RC – transformée de Laplace

b. Résolution dans le domaine de Laplace :

$$U = A \frac{\omega}{(\omega^2 + p^2)(RCp + 1)}$$

c. Transformée inverse :

$$\begin{array}{ccc} & & \text{domaine temporel} \\ \text{domaine de Laplace} & \xrightarrow{\mathbf{L}^{-1}} & \\ \underbrace{U(p)} & \longrightarrow & \underbrace{u(t)} = \frac{1}{2\pi i} \lim_{q \rightarrow \infty} \int_{\alpha - iq}^{\alpha + iq} e^{pt} U(p) dp \end{array}$$

Avec l'équation du b., la transformée inverse donne :

$$u(t) = A \frac{RC\omega \left( e^{\frac{-t}{RC}} - \cos \omega t \right) + \sin \omega t}{1 + (RC\omega)^2}$$

# 1d2. Le circuit RC – résolution approchée

- Recherche d'une solution approchée (version simple : schéma d'Euler en avant) :
  - discrétisation du temps :  $\tau_k = t_0 + k h$  avec  $k \in \mathbb{N}$ ,  $h > 0$  constant = pas de temps, et  $t_0 =$  temps initial
  - approximation de la dérivée :
$$\frac{dU}{dt}(\tau) = \lim_{\varepsilon \rightarrow 0} \frac{U(\tau + \varepsilon) - U(\tau)}{\varepsilon} \approx \frac{U(\tau + h) - U(\tau)}{h}$$
  - on pose  $U_k = U(\tau_k)$  ; l'approximation donne :
$$\frac{U_{k+1} - U_k}{h} \approx \frac{dU}{dt}(\tau_k)$$

# 1d2. Le circuit RC – résolution approchée

– on substitue dans l'équation originale :

$$R C \frac{U_{k+1} - U_k}{h} + U_k = A \sin 2 \pi F k h$$

– on en déduit la relation de récurrence :

$$U_{k+1} = U_k + \frac{h}{R C} (A \sin 2 \pi F k h - U_k)$$



– avec la condition initiale :

$$U_0 = 0$$

- Le schéma de résolution est naïf (Euler en avant) ; il existe des algorithmes/solveurs beaucoup plus précis (Runge-Kutta, ...), mais plus gourmands en calculs
  - Cf les algorithmes d'*analyse numérique*

# 1d. Les solveurs d'EDO

- Une telle équation est appelée *équation différentielle ordinaire* (ODE en anglais)
  - Les dérivées impliquent toutes une même variable (souvent le temps  $t$ )
  - Les dérivées peuvent être de n'importe quel ordre :  $\frac{d}{dt}$   $\frac{d^2}{dt^2}$   $\dots$   $\frac{d^n}{dt^n}$
  - L'équation peut être vectorielle (système d'équations)
- Un *solveur* d'EDO :
  - prend en entrée :
    - une équation différentielle
    - des conditions initiales
    - les bornes du domaine sur lesquelles réaliser l'approximation
    - des paramètres qui lui sont propres
      - par exemple le pas de discrétisation, ou le nombre de points minimum
  - retourne une solution approchée  $f$ 
    - sous forme de graphe d'une fonction, i.e. un tableau de couples  $(t_i, f_i)$  avec  $f_i = f(t_i)$

# 1d3. Le circuit RC en Modelica – version 1

- Version 1 : écriture directe des équations
  - Profitons du solveur d'EDO intégré !

```
1 model circuitRC
2   import Modelica.Constants.pi;
3   parameter Real R = 1e4 "résistance";
4   parameter Real C = 1e-6 "condensateur";
5   parameter Real F = 1e3 "fréquence";
6   parameter Real A = 10 "amplitude";
7   Real U_C      "tension aux bornes du condensateur";
8   Real S        "tension aux bornes de la source";
9   initial equation
10    U_C = 0 "conditions initiales";
11   equation
12    S = A * sin(2*pi*F*time) "équation de la source de tension";
13    R * C * der(U_C) + U_C = S "équation différentielle à résoudre";
14 end circuitRC;
```

# 1d3. Le circuit RC en Modelica – version 1

- Le compilateur OpenModelica indique :

```
Checking: model circuitRC... 0.6350000000002183 seconds -> OK
Check of circuitRC completed successfully.
Class circuitRC has 2 equation(s) and 2 variable(s).
1 of these are trivial equation(s).
```

- Génération de fichiers C, compilés avec GNU gcc

- Simulation :

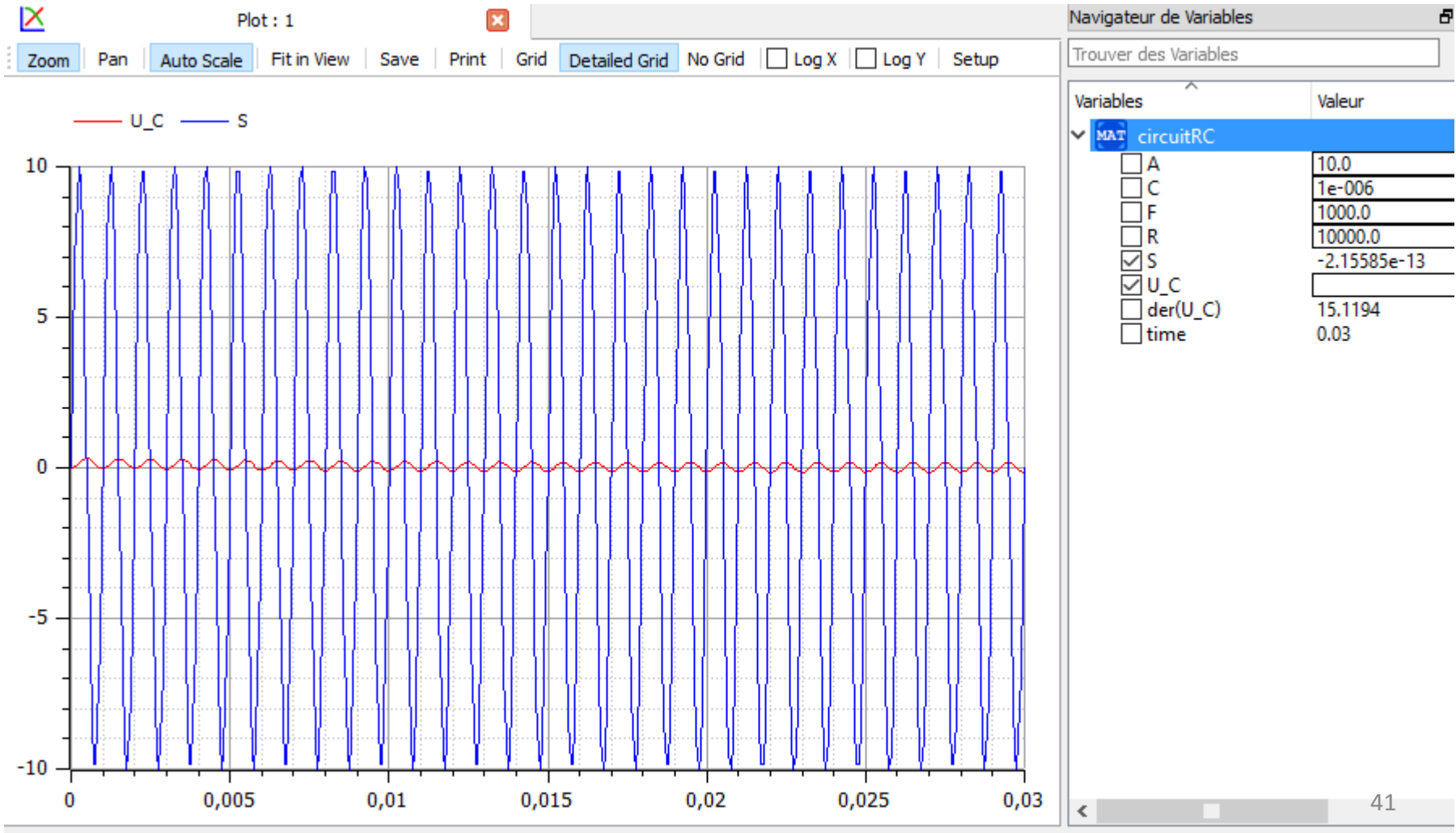
- exécutable nécessitant les paramètres de résolution :

- bornes de simulation
- solveur
- paramètres du modèle Modelica
- ...



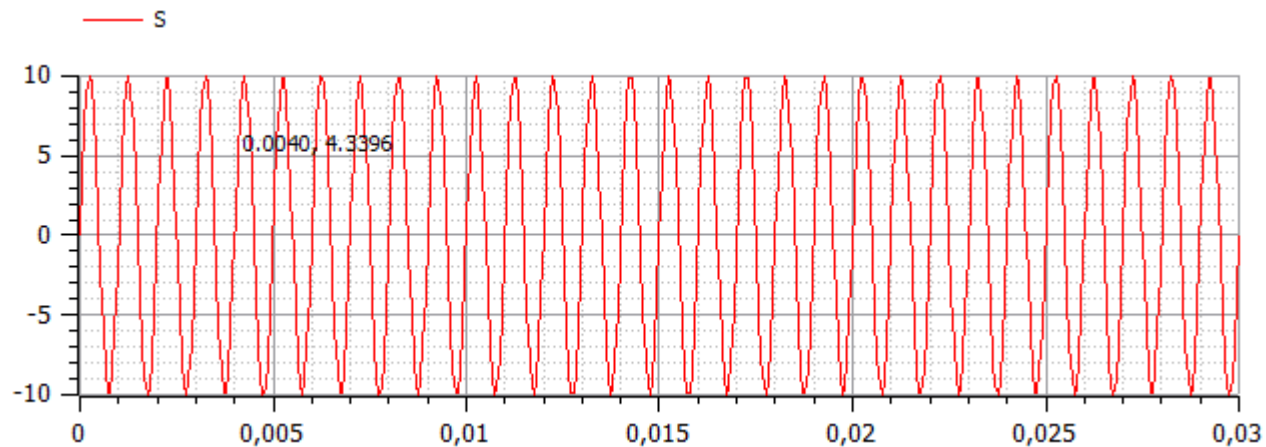
# 1d3. Le circuit RC en Modelica – version 1

- Résultats graphiques

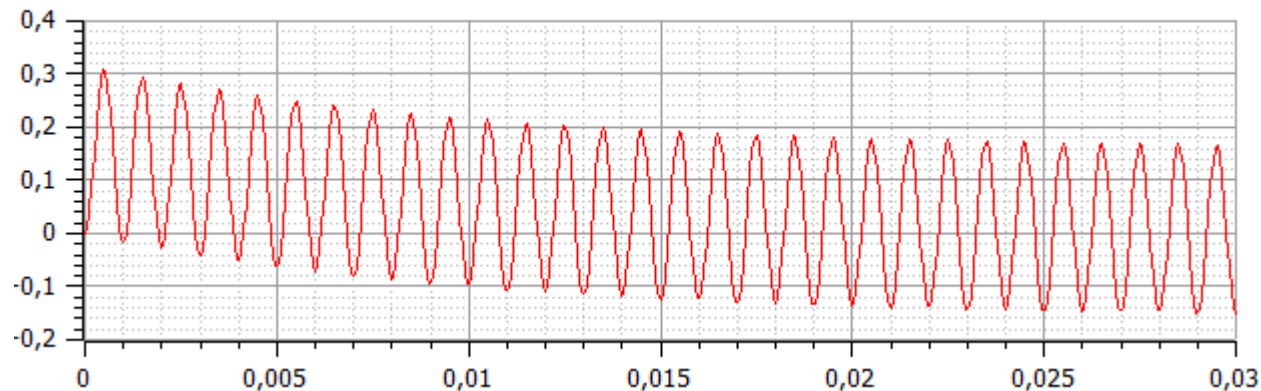


# 1d3. Le circuit RC en Modelica – version 1

- Tension  $S$

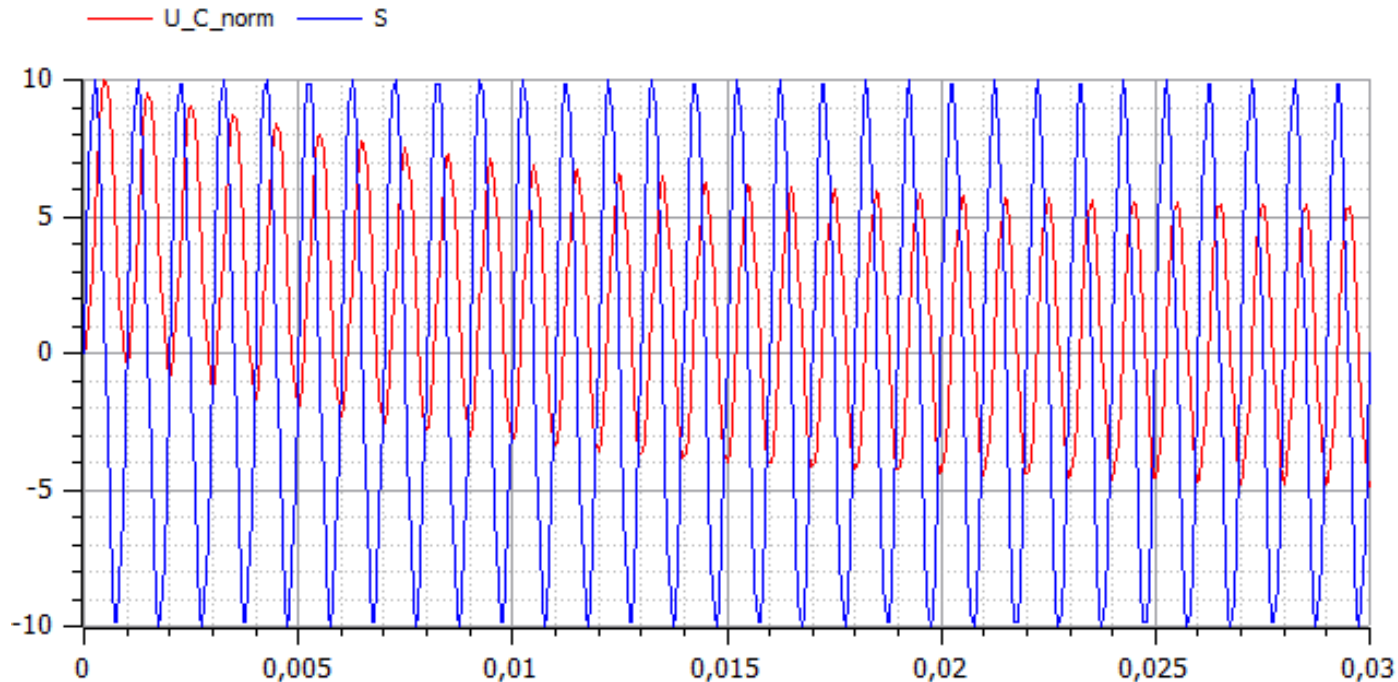


- Tension  $U_c$



# 1d3. Le circuit RC en Modelica – version 1 bis

- Les tensions  $S$  et  $U_C$  n'ont pas le même ordre de grandeur :
  - on recalibre  $U_C$  en  $U_{C\_norm} = K U_C$  avec  $K$  un coefficient judicieusement choisi 😊 (ici  $K \approx 30$ )



# 1d3. Le circuit RC en Modelica – version 1 bis

- Le code Modelica :

```
1 model circuitRCnormalise
2   import Modelica.Constants.pi;
3   parameter Real R = 1e4 "résistance";
4   parameter Real C = 1e-6 "condensateur";
5   parameter Real F = 1e3 "fréquence";
6   parameter Real A = 10 "amplitude";
7   Real U_C "tension aux bornes du condensateur";
8   Real S "tension aux bornes de la source";
9   Real U_C_norm "U_C ramené dans l'intervalle -A..A";
10 protected
11   final Real RCw = R * C * 2 * pi * F;
12   final Real coef = (1 + RCw ^ 2) / RCw / (1+exp(-pi/RCw));
13 initial equation
14   U_C = 0 "conditions initiales";
15 equation
16   R * C * der(U_C) + U_C = S "équation différentielle à résoudre";
17   S = A * sin(2 * pi * F * time) "équation de la source de tension";
18   U_C_norm = U_C * coef;
19 end circuitRCnormalise;
```

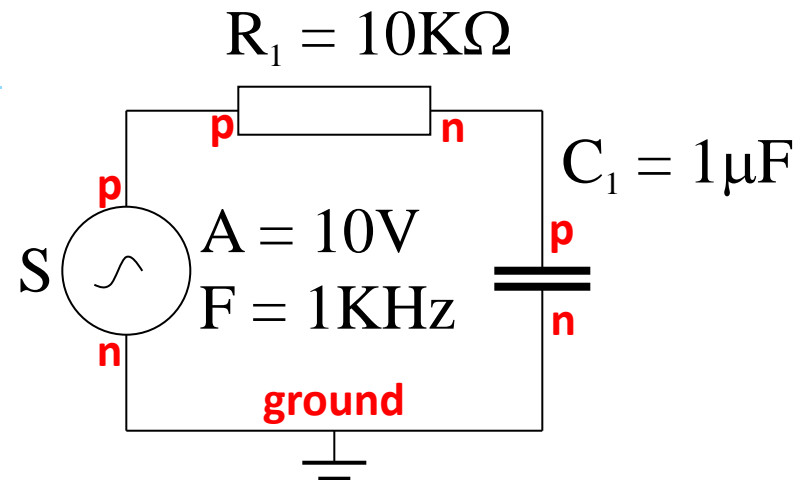
# 1d4. Le circuit RC – version 2

- Dans la version 1 précédente (2 variantes) :
  - Les composants électroniques ne sont pas explicites
  - Tout est à refaire si le schéma est modifié
- Dans cette version 2 :
  - Chaque composant est modélisé en tant que tel
    - Début d'une bibliothèque réutilisable
  - Les composants sont *connectés* pour former le circuit final

# 1d4. Le circuit RC – version 2

- Objectif en Modelica :
  - Les composants ont des bornes que l'on relie

```
model circuitRC_v2
  Resistor      R1 (R=1e4);
  Capacitor     C1 (C=1e-6);
  SinusSource   S (A=10, F=1000);
  Ground        G;
equation
  connect (S.p, R1.p);
  connect (R1.n, C1.p);
  connect (S.n, G.ground);
  connect (C1.n, G.ground);
end circuitRC_v2;
```



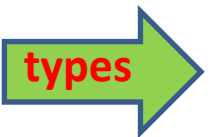
# 1d4. Le circuit RC – version 2

- Définition de types de base
  - types Voltage, Current, Frequency, ...
- Définition des points de connexion (bornes)
  - connecteur Pin
- Définition des composants électroniques
  - modèles Resistor, Capacitor, SinusSource, Ground
- Définition du circuit exemple

# 1d4. Le circuit RC v2 - types

- Définition des types de base (grandeurs électriques) :

```
type Frequency = Real(min=0, quantity="Frequency", unit="Hz");  
type Current   = Real(quantity="Current", unit="A");  
type Voltage   = Real(quantity="Voltage", unit="V");  
  
type Resistance = Real(min=0, quantity="Resistance", unit="Ohm");  
type Capacitance = Real(min=0, quantity="Capacitance", unit="F");
```





# 1d4. Le circuit RC v2 - connecteurs

- Définition des points de connexion `Pin` :

```
connector Pin
  Voltage v;
  flow Current i;
end Pin;
```

– en tout point de connexion  $P$

- Tension  $v(P)$  (*plus précisément un potentiel*)
- Courant  $i(P)$

– si des  $P_i$  sont connectés ensemble :

- Tous ont la même tension

$$\forall j \forall k \ v(P_j) = v(P_k)$$

- La somme de leurs courants est nulle

$$\sum_k i(P_k) = 0$$



# 1d4. Le circuit RC v2 – composant abstrait à 2 bornes

- Un composant ayant deux bornes `TwoPins` :

```
partial model TwoPins
  Pin      p "borne positive",
          n "borne négative";
  Voltage u "tension entre p et n";
  Current i "courant traversant : entre en p, sort en n";
equation
  0 = p.i + n.i;
  u = p.v - n.v;
  i = p.i;
end TwoPins;
```

- Possède 2 points de connexion (ses *bornes*) :
  - $p$  (borne positive)
  - $n$  (borne négative)
- Se caractérise par :
  - $v$  : tension entre  $p$  et  $n$  (différence des potentiels)
  - $i$  : courant le traversant



## 1d4. Le circuit RC v2 – résistance et capacité

- Un composant Resistor :

```
model Resistor
  extends TwoPins;
  parameter Resistance R;
equation
  u = R*i;
end Resistor;
```

- Un composant Capacitor :

```
model Capacitor
  extends TwoPins;
  parameter Capacitance C;
equation
  C*der(u) = i;
end Capacitor;
```

## 1d4. Le circuit RC v2 – source de tension sinusoïdale

- Un composant SinusSource :

```
model SinusSource
  extends TwoPins;
  parameter Frequency F;
  parameter Voltage A;
protected
  constant Real pi = 3.14159265358979;
equation
  u = A * sin(time * 2*pi*F);
end SinusSource;
```

## 1d4. Le circuit RC v2 – prise de terre

- Un composant Ground :

```
model Ground
  Pin ground;
equation
  ground.v = 0;
end Ground;
```

Et pourquoi pas une spécialisation de Pin ?

```
connector Ground
  extends Pin;
equation
  v = 0;
end Ground;
```

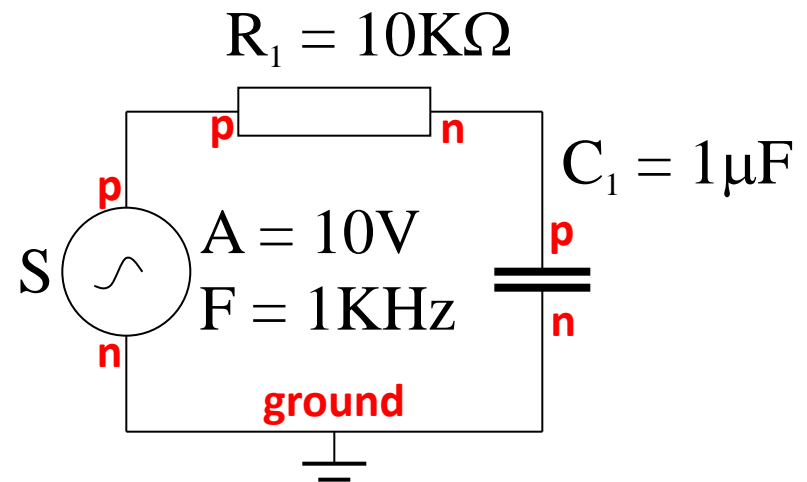
```
connector Pin
  Voltage v;
  flow Current i;
end Pin;
```

R : Un connecteur ne peut pas porter d'équation ☹️

# 1d4. Le circuit RC v2 – le circuit de l'exemple

- Objectif en Modelica atteint :

```
model circuitRC_v2
  Resistor      R1 (R=1e4);
  Capacitor     C1 (C=1e-6);
  SinusSource   S (A=10, F=1000);
  Ground        G;
equation
  connect (S.p, R1.p);
  connect (R1.n, C1.p);
  connect (S.n, G.ground);
  connect (C1.n, G.ground);
end circuitRC_v2;
```



# 1d4. Le circuit RC v2 – compilation

...  
c<sup>é</sup>quation

```
R1.u = R1.R * R1.i;  
0.0 = R1.p.i + R1.n.i;  
R1.u = R1.p.v - R1.n.v;  
R1.i = R1.p.i;  
C1.C * der(C1.u) = C1.i;  
0.0 = C1.p.i + C1.n.i;  
C1.u = C1.p.v - C1.n.v;  
C1.i = C1.p.i;  
S.u = S.A * sin(6.28318530717958 * time * S.F);  
0.0 = S.p.i + S.n.i;  
S.u = S.p.v - S.n.v;  
S.i = S.p.i;  
G.ground.v = 0.0;  
R1.p.i + S.p.i = 0.0;  
R1.n.i + C1.p.i = 0.0;  
C1.n.i + G.ground.i + S.n.i = 0.0;  
R1.p.v = S.p.v;  
C1.p.v = R1.n.v;  
C1.n.v = G.ground.v;  
C1.n.v = S.n.v;  
end circuitRC_v2;
```

```
n";  
: entre en p, sort en n";  
= 0.0) = 10000.0;
```

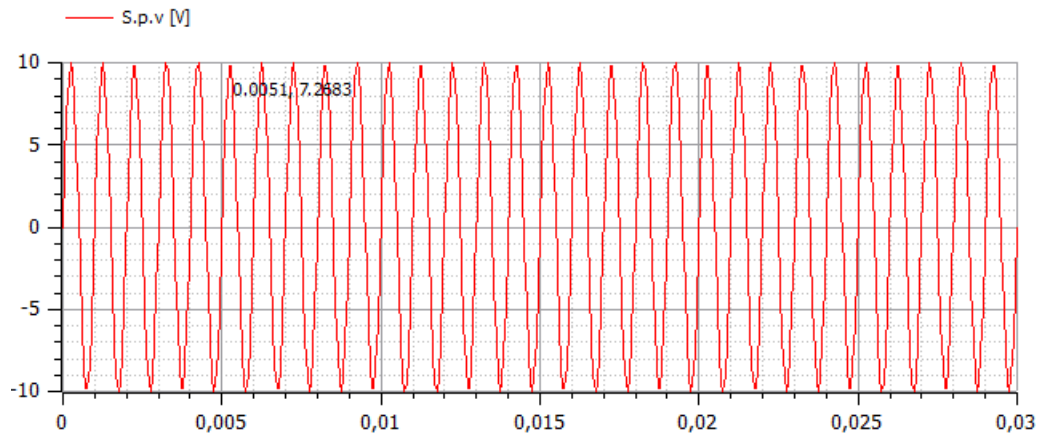
```
n";  
: entre en p, sort en n";  
0.0) = 1e-006;
```

```
n";  
: entre en p, sort en n";  
.0) = 1000.0;
```

# 1d4. Le circuit RC v2 – simulation

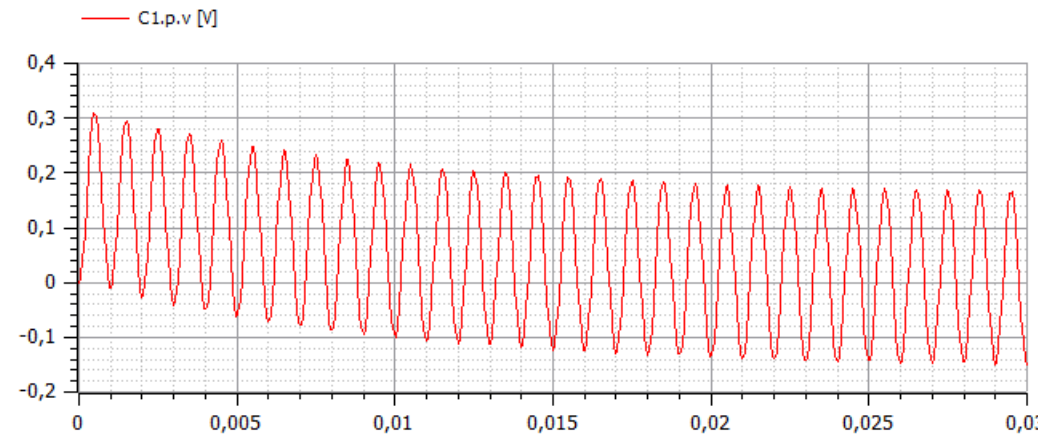
- Nous retrouvons les mêmes courbes (encore heureux 😊) :

– Tension de la source : S.p.v



Variables	Valeur
<input checked="" type="checkbox"/> MAT circuitRC_v2	
> C1	
> G	
> R1	
▼ S	
<input type="checkbox"/> A	10.0
<input type="checkbox"/> F	1000.0
<input type="checkbox"/> i	-1.51192e-05
> n	
▼ p	
<input type="checkbox"/> i	-1.51192e-05
<input checked="" type="checkbox"/> v	-1.92089e-12
<input type="checkbox"/> u	-1.92089e-12
<input type="checkbox"/> time	0.03

– Tension du condensateur : C1.p.v



Variables	Valeur
<input checked="" type="checkbox"/> MAT circuitRC_v2	
▼ C1	
<input type="checkbox"/> C	1e-006
<input type="checkbox"/> der(u)	15.1192
<input type="checkbox"/> i	1.51192e-05
> n	
▼ p	
<input type="checkbox"/> i	1.51192e-05
<input checked="" type="checkbox"/> v	-0.151192
<input type="checkbox"/> u	
> G	
> R1	
> S	
<input type="checkbox"/> time	0.03



# 1d5. Le circuit RC – version 3

- Les composants de la version 2 existent déjà dans la bibliothèque standard 😊 :
  - Réutilisation directe en écrivant le code Modelica
  - Edition purement graphique du circuit
    - Choix des composants
    - Interconnexion des composants

# 1d5. Le circuit RC – version 3

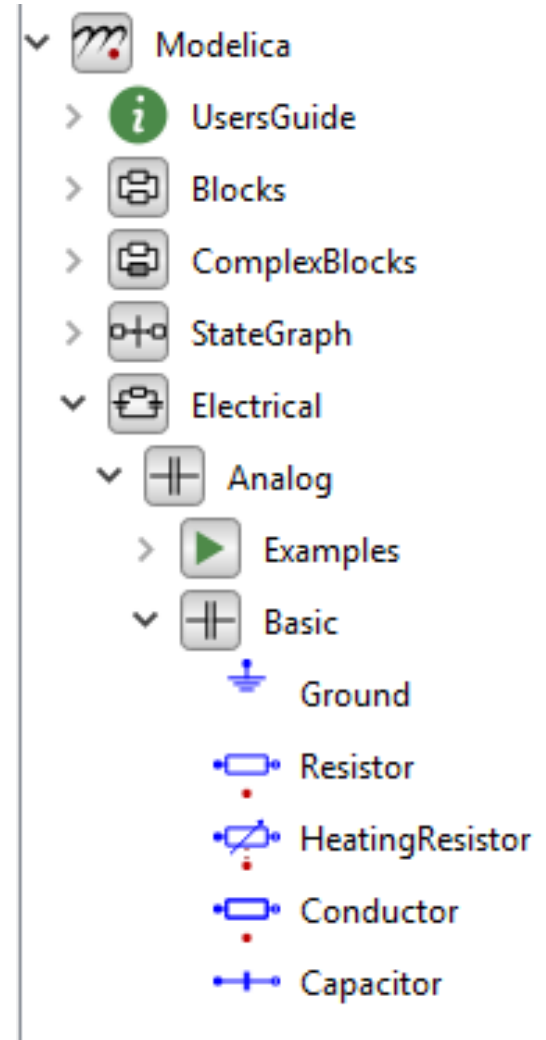
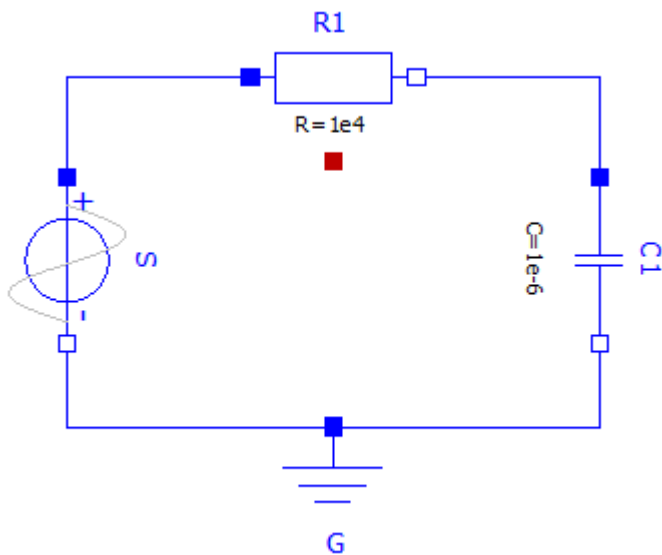
- Le nouveau code Modelica du circuit RC :

```
1 model circuitRC_v3
2   Modelica.Electrical.Analog.Basic.Resistor R1(R = 1e4);
3   Modelica.Electrical.Analog.Basic.Capacitor C1(C = 1e-6);
4   Modelica.Electrical.Analog.Sources.SineVoltage S(V = 10, freqHz = 1000);
5   Modelica.Electrical.Analog.Basic.Ground G;
6 equation
7   connect(R1.p, S.p);
8   connect(R1.n, C1.p);
9   connect(G.p, S.n);
10  connect(G.p, C1.n);
11
12  annotation(experiment(stopTime=0.03));
13 end circuitRC_v3;
```

- Conséquences :
  - Passage de 20 à 24 variables (et autant d'équations)

# 1d5. Le circuit RC – version 3 bis

- Conception visuelle du code :



# 1d5. Le circuit RC – version 3 bis

- Conception visuelle :

- le prix à payer : code pollué par les annotations de dessin 😞

```
1 model circuitRC_v3View
2   import Modelica.Electrical.Analog.Basic;
3   import Modelica.Electrical.Analog.Sources;
4   Modelica.Electrical.Analog.Basic.Resistor R1(R = 1e4) annotation(Placement(visi
5   Modelica.Electrical.Analog.Basic.Capacitor C1(C = 1e-6) annotation(Placement(vi
6   Modelica.Electrical.Analog.Sources.SineVoltage S(V = 10, freqHz = 1000) annotat
7   Modelica.Electrical.Analog.Basic.Ground G annotation(Placement(visible = true,
8 equation
9   connect(R1.n, C1.p) annotation(Line(points = {{10, 12}, {32, 12}, {32, 0}}, col
10  connect(C1.n, G.p) annotation(Line(points = {{32, -20}, {32, -30}, {0, -30}}, c
11  connect(R1.p, S.p) annotation(Line(points = {{-10, 12}, {-32, 12}, {-32, 0}}, c
12  connect(S.n, G.p) annotation(Line(points = {{-32, -20}, {-32, -30}, {0, -30}},
13  annotation(Diagram, Icon, version = "", uses(Modelica(version = "3.2.1")));
14 end circuitRC v3View;
```

## 2. Le langage Modelica

- Concepts clés : modéliser des objets avec plusieurs sortes de classes
  - les classes au sens général (**class**)
    - définitions de propriétés
      - classes imbriquées (notation pointée classique)
      - structure : variables (attributs)
      - dynamique : équations ou algorithmes
      - annotations
    - héritage multiple
    - protection d'accès (**public**, **protected**)
  - les types (**type**) élémentaires ou construits
    - primitifs : `Real`, `Integer`, `String`, `Boolean`
    - les énumérations (**enumeration**)
    - les tableaux (**array**)
  - les modèles (**model**)
  - les connecteurs (**connector**)
  - les blocs (**block**) et les fonctions (**function**)
  - les paquets (**package**)

## 2. Modelica : le langage

- Concepts clés : modéliser une dynamique
  - le temps évolue continûment (variable globale **time**), partagé par tous les objets d'un modèle
  - des événements discrets peuvent survenir
    - expression de comparaison changeant de valeur  
Exemple :  $x > 0$  qui jusqu'ici était faux, qui devient vrai, avec `Real x` et  $x$  variant continument selon une équation spécifiée
    - opérateurs spécifiques (liste non exhaustive) :
      - pre** ( $\nabla$ ) (valeur au pas de temps précédent)
      - edge** ( $\nabla$ ) (détection d'un front montant)
  - la dynamique est exprimée :
    - soit sous forme d'équations (différentielles ou pas)
      - opérateur **der**
    - soit sous forme algorithmique classique



# 2a1. Sémantique des classes



- Une classe :
  - porte un nom
  - est placée dans une arborescence de paquets
  - hérite éventuellement d'autres classes (cf. *héritage*)
  - encapsule des définitions d'*éléments*
    - de différentes natures
      - des variables (nom, classe, valeur)
        - » Pas de distinction explicite variable de classe / variable d'instance
        - » Plusieurs altérations possibles : **final**, **constant**, **parameter**, **input**
        - ...
      - des classes
      - pourvues de droits d'accès : **public** ou **protected**
    - des équations ou algorithmes
  - est instanciable (concrète) ou pas (**partial** = abstraite)
- Contrairement aux LOO classiques :
  - Pas de méthodes (en particulier de constructeurs)
    - Méthode = fonction avec un paramètre implicite : le receveur du message
    - Attention, il y a bien toutefois les notions de fonction et d'héritage...



## 2a1. Sémantique des classes - exemples



- Définitions de classes  
(sans héritage, mais avec composition) :

```
class A
  Real x;
  Integer i = 2;
  final String s = "classe A";
end A;
```

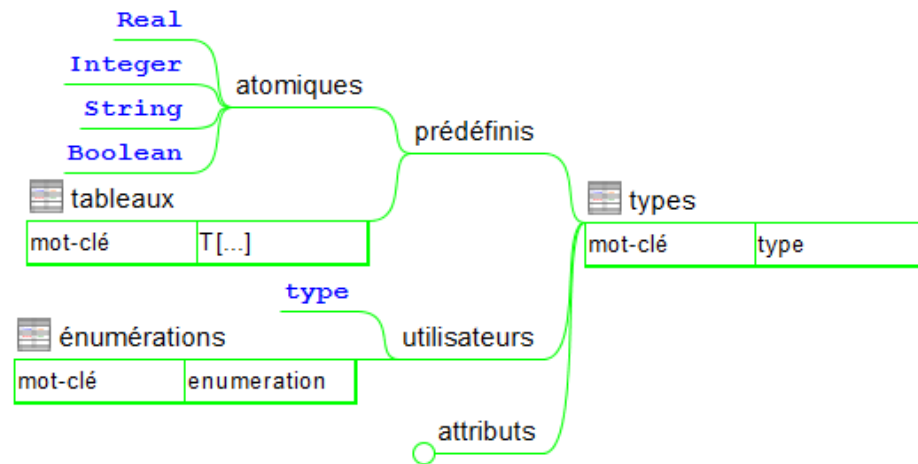
```
class B
  Boolean b;
  A      a(x=2, final i=3);
end B;
```

- Quelques instanciations

```
// quelques instanciations
A oa1(x=1),
  oa2(i=3, x=2),
  oa3(x=3, s="aïe"); // err : violation 'final' sur A.s
B ob1(b=true, a(x=4)),
  ob2(b=false, a(i=4)), // err : violation 'final' sur B.a.i
  ob3(b=ob1.b, a=oa1); // err : trop d'équations
```



# 2a2. Sémantique des types



- Un type est une classe particulière
  - pas de définition de classes imbriquées
  - pas de dynamique (équation ou algorithme)
- Type utilisateur nommé
  - Type énuméré
  - Sous-type d'un type
- Instanciation = résultat d'une expression

# 2a2. Sémantique des types - expressions

- Les littéraux :

- ceux des types primitifs, usuels :

- 1.0 (Real) 1 (Integer) **true** (Boolean) "1" (String)

- les tableaux

- array** (  $e_1, \dots, e_n$  ) pour un tableau quelconque

- $\{ e_1, \dots, e_n \}$  pour un vecteur (tableau 1D)

- $[ e_{11}, \dots, e_{1n} ; e_{21}, \dots, e_{2n} ; \dots, e_{mn} ]$  pour une matrice

- $\{ e \text{ for } i \text{ in } \dots \}$  pour définition en compréhension

- Les opérateurs :

arithmétiques	comparateurs	logiques
+ - * / ^	== < > <= >= <>	<b>and or not</b>

- Les appels (fonctions ayant un résultat unique) :

- avec arguments positionnés :  $nom (arg_1, \dots, arg_n)$

- avec arguments nommés :  $nom (par_1=exp_1, \dots, par_n=exp_n)$



# 2a2. Sémantique des types - attributs

- Attributs prédéfinis des types

Usage	Attribut	Type	Real	Integer	Boolean	String	énuméré	Commentaire
en simulation	value	T	✓	✓	✓	✓	✓	implicite si non précisé (ne peut être explicité) ; valeur courante dans une simulation
	start	T	✓	✓	✓	✓	✓	valeur en début de simulation
	fixed	Boolean	✓	✓	✓		✓	indique si la valeur sera figée ou pas durant une simulation
	nominal	T	✓					valeur nominale attendue durant une simulation
	stateSelect	StateSelect	✓					usage interne
définition	min	T	✓	✓	?		✓	plus petite valeur du type
	max	T	✓	✓	?		✓	plus grande valeur du type
	unit	String	✓					unité de mesure exploitée dans les calculs de dimensions
affichage	displayUnit	String	✓					unité de mesure affichée
	quantity	String	✓	✓	✓	✓	✓	libellé de la quantité représentée



## 2a2. Sémantique des types - exemples

- Pseudo-définition de `Real` :

```
type Real
  RealType value; // Accessed without dot-notation
  parameter StringType quantity = "";
  parameter StringType unit = "" "Unit used in equations";
  parameter StringType displayUnit = "" "Default display unit";
  parameter RealType min=-Inf, max=+Inf; // Inf denotes a large value
  parameter RealType start = 0; // Initial value
  parameter BooleanType fixed // = true; default for parameter/constant;
                               ; // = false; default for other variables

  parameter RealType nominal; // Nominal value
  parameter StateSelect stateSelect = StateSelect.default;
equation
  assert(value >= min and value <= max, "Variable value out of limit");
  assert(nominal >= min and nominal <= max, "Nominal value out of limit");
end Real;
```



## 2a2. Sémantique des types - exemples

- Quelques définitions de types utilisateur :

```
// notation concise
type PositiveReal = Real(min=0);

// notation standard - héritage pour modifier
type PositiveReal_2
  extends Real(min=0);
end PositiveReal_2;

// un point en 3 dimensions
type Point3D = Real[3];

// une transformation affine en 3D en coordonnées homogènes
type Transformation3D = Real[4,4];

type RGB = enumeration(rouge, vert, bleu);

type ImageRGB = Integer[1024, 768, RGB];
```



## 2a2. Sémantique des types - exemples



- Instanciation et expressions :

```
PositiveReal x=1, y=x+2;
```

```
Point3D P = { 0, 0, 1 };
```

```
RGB couleur = RGB.rouge;
```

```
Transformation3D
```

```
id1 = [1, 0, 0, 0 ; 0, 1, 0, 0 ; 0, 0, 1, 0 ; 0, 0, 0, 1],  
id2 = { {if i==j then 1 else 0 for i in 1:4} for j in 1:4 };
```

# 2a3. Sémantique des variables



- Variable = (nom, classe, valeur)
  - Niveau classe : définit nom, classe, valeur par défaut, ...
  - Niveau instance : associe une valeur à un nom
- Les altérations d'une variable sont liées :
  - à sa variabilité :  
**constant parameter discrete**
  - à sa causalité :  
**input output**
  - à sa connectabilité :  
**flow stream**
  - à sa partageabilité :  
**inner outer**
  - à sa redéfinissabilité :  
**final redeclare replaceable**

# 2a3. Sémantique des variables



- Notion de *variabilité* applicable à :  
**model block function record**
- Exprime une propriété liée aux simulations
  - si **constant**
    - valeur fixée *en dur* dans le modèle
    - invariable en simulation
  - si **parameter**
    - valeur libre dans le modèle
    - à définir en début de simulation mais qui restera invariable
  - si **discrete**
    - valeur libre dans le modèle
    - ne changera en simulation que lors des transitions discrètes
  - si non précisée
    - valeur libre dans le modèle
    - peut changer à tout instant de la simulation





## 2a3. Sémantique des variables

- Notion de *causalité* applicable aux classes :  
**block function**
- Exprime une propriété liée aux connexions
  - si **input**
    - si **function**, sera lié à un argument d'un appel
    - si **block**, sera lié à une variable lors d'une instantiation
  - si **output**
    - si **function**, sera lié à un résultat d'un appel

## 2a4. Sémantique des modèles et connecteurs ☑

- Les mots-clés **class** et **model** sont interchangeables
- Un connecteur est une classe telle que :
  - Ne porte pas de dynamique (équation ou algorithme)
  - Ses instances peuvent être connectées (équation **connect**)
  - Ses variables ne peuvent pas être **protected** ni partageables (**inner** et **outer**), et seulement de classe **type**, **connector** ou **record**
  - Ses variables sont altérables selon leur connectabilité
    - variable potentiel : rien (contrainte d'égalité)
    - variable flux : **flow** (contrainte de somme nulle)
    - variable canal : **stream** (forme spéciale de flux, avec contraintes sur les dérivées)

## 2a4. Sémantique des modèles et connecteurs

- Distinction entre variable de *potentiel* (ou d'*effort*) et de *flux* ; issue de différentes techniques de modélisation (indépendantes du domaine) :
  - Graphes de liaisons (*bond graph*)
  - Modélisation par *blocs* (à la Simulink ou Scicos)
- Lois de Kirchhoff (conservation de l'énergie)
  - Graphe de connexion
    - Nœud : caractérisé par ses potentiels et ses flux
    - Arc entre deux nœuds : transfert d'énergie
  - Loi des nœuds :
    - somme des flux nulle en un nœud
    - potentiels tous égaux en un nœud
  - Loi des mailles :
    - somme des potentiels nulle sur un circuit du graphe (chemin fermé)
- Dans Modelica, une équation de connexion :
  - porte sur deux objets d'une même classe
  - se transforme en une série d'équations d'égalité ; s'appuie sur :
    - les noms des variables
    - leur propriété de connectabilité



# 2a4. Sémantique des connecteurs - exemple



```
model testConnect1
```

```
connector C  
  Real x;  
  flow Real y;  
end C;
```

```
C a, b, c, d, e;
```

```
equation  
  connect (a, b);  
  connect (c, d);  
  connect (d, e);  
end testConnect1;
```

L'ajout redondant de :  
connect (c, e)  
ne pose aucun pb...

- Le code suivant :

- induit les équations :

```
a.x = b.x;  
(-a.y) + (-b.y) = 0.0;  
c.x = d.x;  
c.x = e.x;  
(-e.y) + (-d.y) + (-c.y) = 0.0;
```

## 2a4. La partageabilité

- La partageabilité, une façon d'exprimer des connexions implicites
    - soit  $C$  une classe portant une variable de nom  $n$  de classe  $T$  partagée en tant que *source* :  
**class**  $C$  ... **inner**  $T n$ ; ... **end**  $C$ ;
    - Toute instance  $i$  d'une classe  $D$  ayant une variable de nom  $n$  de classe  $T$  partagée en tant que *puits* :  
**class**  $D$  ... **outer**  $T n$ ; ... **end**  $D$ ;ayant dans l'ascendance de son graphe de composition un objet  $o$  de classe  $C$  vérifie l'équation :
- $$o.n = i.n$$
- Permet de définir une sorte de variable globale, partagée par un ensemble d'objets

## 2a4. La partageabilité - exemple

```
class A
  outer Real p; // p est un puits
  Real a = -1;
end A;
```

```
class B
  Real u = 1;
  A v(a=2);
end B;
```

```
class C
  inner Real p; // p est une source
  A x;
  B y(v(a=3));
end C;
```

```
class D
  C o1(p=4), // o1.p = 4 = o1.x.p = o1.y.v.p
  o2(p=5); // o2.p = 5 = o2.x.p = o2.y.v.p
end D;
```

## 2b. Expression de la dynamique

- La dynamique est exprimée selon :
  - Des équations
  - Des algorithmes
- La sémantique de cette dynamique diffère selon la nature de la classe qui la porte
  - Un modèle porte équations et algorithmes
  - Une fonction ne porte qu'un seul algorithme

# 2b1. Sémantique des équations

- Notion d'*équation* applicable aux classes :  
**class model block**
- Sémantique : celle des automates hybrides, après mise à plat du modèle
  - le temps continu se déroule : **time**
    - Evaluation des conditions de transition (clauses **when**) à chaque progression de **time**
  - en phase stable : caractérisent la dynamique continue
    - Le solveur fait évoluer les variables d'état
  - en phase de transition
    - Exécution des réinitialisations
    - Exécution de tous les algorithmes des composants associés (cf ci-après)



# 2b1. Les équations – formes possibles

- liste d'équations

*éq<sub>1</sub> ; ... ; éq<sub>n</sub> ;*

- équation d'égalité

*expr\_simple = expression*

- équation de connexion

**connect** (*expr<sub>1</sub>, expr<sub>2</sub>*)

- équation de réinitialisation

**reinit** (*nom, expression*)

- équation conditionnée

**when** *cond<sub>1</sub>* **then** *éqs<sub>1</sub>* **elsewhen** *cond<sub>2</sub>* ... **else** *éqs<sub>n+1</sub>* **end when**

- équation itérée

○ Domaine explicite : **for** *nom* **in** *domaine* **loop** *éqs* **end for**

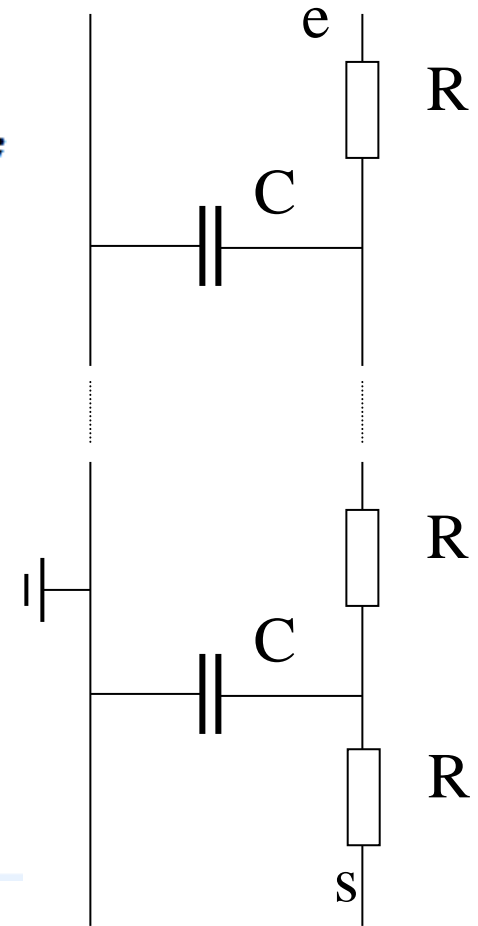
○ Domaine implicite : **for** *nom* **loop** *éqs* **end for**

# 2b1. Les équations – exemple

un nombre arbitraire de composants

- Etage atténuateur (filtres RC en cascade) :

```
model etageAttenuateur
  import Modelica.Electrical.Analog.Basic;
  import Modelica.Electrical.Analog.Interfaces;
  parameter Integer N;
  Interfaces.Pin      e, s;
  Basic.Resistor[N+1] R;
  Basic.Capacitor[N]  C;
  Basic.Ground        G;
equation
  connect(e, R[1].p);
  for i in 1:N loop
    connect(R[i].n, R[i+1].p);
    connect(R[i].n, C[i].p);
    connect(C[i].p, G.p);
  end for;
  connect(R[N+1].n, s);
end etageAttenuateur;
```



## 2b2. Sémantique des algorithmes

- Notion d'*algorithme* applicable aux classes :
  - un ou plusieurs = cas 1 : **class model block**
  - un seul = cas 2 : **function**
- Sémantique :
  - cas 1 : algorithmes invoqués implicitement (aucun appel explicite possible) à chaque pas de progression de **time**
    - si o composant de classe C ayant des sous-composants s
      - a. les algorithmes de chaque sous-composants s sont invoqués
      - b. ceux de C le sont sur o
    - si o composant de classe C héritant de plusieurs classes A et B, avec B héritant elle-même de A
      - a. les algorithmes de l'ascendance sont invoqués sur o, ceux de B avant A
      - b. ceux de C le sont sur o
  - cas 2 : algorithme invoqué explicitement lors d'un appel

# 2b2. Les algorithmes - instructions

- séquence

*instr<sub>1</sub> ; ... ; instr<sub>n</sub> ;*

- affectation

*nom := expression*

*(nom<sub>1</sub>, ..., nom<sub>n</sub>) := appel (...)*

- conditionnelle

**if** *cond<sub>1</sub>* **then** *instr<sub>1</sub>* **elseif** *cond<sub>2</sub>* ... **else** *instr<sub>n+1</sub>* **end if**

- boucles

**while** *cond* **loop** *instr* **end while**

**for** *nom* **in** *domaine* **loop** *instr* **end for**

○ *domaine = début : fin* ou *début : pas : fin*

○ *domaine = vecteur* ou *type énuméré*

- appel d'une fonction

*appel (...)*

# 2b2. Algorithmes et fonctions

```
type PositiveInt = Integer(min=0);
```

```
function quotient
  input PositiveInt a, b;
  output PositiveInt q;
algorithm
  q := if b>a then 0 else 1+quotient(a-b, b)
end quotient;
```

```
function reste
  input PositiveInt a, b;
  output PositiveInt r;
algorithm
  r := if b>a then a else reste(a-b, b);
end reste;
```

```
function quotient_et_reste
  input PositiveInt a, b;
  output PositiveInt q, r;
algorithm
  if b>a then
    q := 0;
    r := a;
  else
    (q, r) := quotient_et_reste(a-b, b);
    q := q+1;
  end if;
end quotient_et_reste;
```

```
model testFunc2
  Integer x, y;
algorithm
  x := quotient(15, 6);
  y := reste(13, 6);
  //
  (x, y) := quotient_et_reste(15, 6);
end testFunc2;
```

## 2c. Sémantique de l'héritage

- La relation d'héritage **extends**  $\mathbb{T}$  sert à :
  - reprendre une définition existante de  $\mathbb{T}$  :
    - telle quelle
    - en l'altérant et en laissant libres les sous-classes de l'altérer aussi
      - changer la valeur/classe associée à une variable  $n$  héritée :  $\mathbb{T} (n=v, \dots)$
      - changer la classe associée à une variable de classe
        - » généricité de type : **replace**, **replaceable**, **redeclare**
    - en l'altérant mais en empêchant les sous-classes de l'altérer : **final**
  - ajouter de nouvelles définitions propres à la classe

## 2d. Notions non étudiées ici

- La généricité (classes paramétrées)
- Les blocs : **block**
- Les enregistrements : **record**
- La surcharge des opérateurs : **operator function** et **operator record**
- Les paquets : **package**
  - Les classes encapsulées : **encapsulated**  $\mathbb{T}$
- Les variables **stream** dans les connecteurs
- Les outils utiles à l'expression d'une dynamique purement discrète
- Les annotations
- et d'autres encore ...

# 3. L'environnement OpenModelica

- Points positifs
  - Libre et open-source
    - Ecrit en bonne partie en Modelica
    - Quelques outils écrits en Java ou Python (livrés en binaire exécutable et autonome)
  - Multiplateformes : Windows, Linux, MacOS, ...
- Point négatif
  - Bogué :
    - plantages de temps en temps
    - Reconstruction des binaires mal contrôlée (dépendances ?) ⇒ effacer tous les fichiers binaires produits et relancer une compilation
- Prérequis
  - La chaîne de compilation GNU gcc

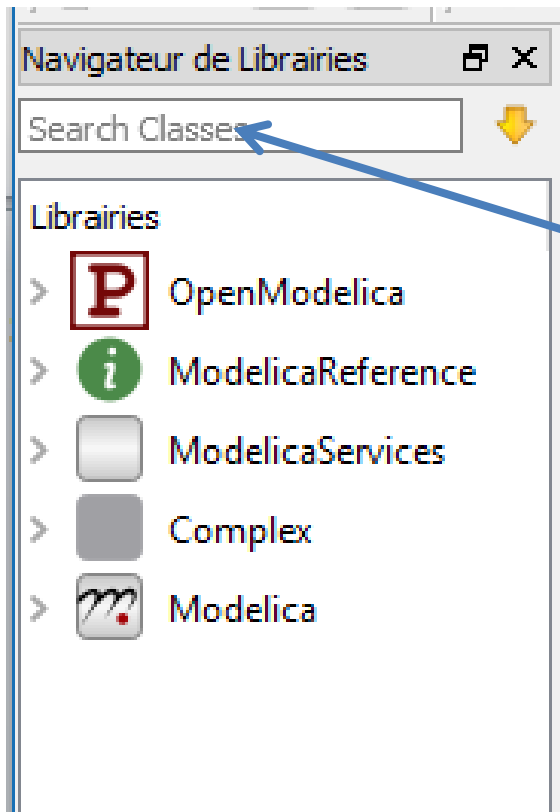


# 3a. OpenModelica : les outils

- Un environnement de développement complet
  - **OMEdit** (*OpenModelica Connection Editor*)
  - greffon pour Eclipse existant
- Un carnet de notes (*note-book*) : **OMNotebook**
  - Texte HTML et LaTeX
  - Graphiques
  - Appels à l'interpréteur **OMShell**
- Interpréteurs de scripts :
  - **OMShell** : fichiers `.mos` ; en fait du code Modelica
  - Autre langage de script : Python
- Outils divers :
  - Un solveur pour exprimer des problèmes d'optimisation : **OMOptim** (*OpenModelica Optimization Editor*)
  - Un générateur de courbes : **OMPlot**

# 3a1. L'IDE OMEdit

- Toute la documentation en ligne
  - OpenModelica : langage de script **OMShell**
  - Modelica : manuel de référence
  - Modelica : bibliothèque standard



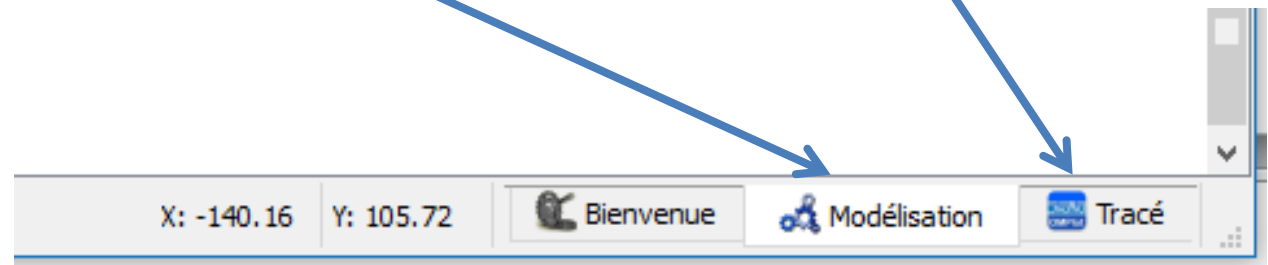
Saisir ici un fragment de texte de la notion cherchée : l'arborescence se focalise sur les classes trouvées

# 3a1. L'IDE OMEdit

- Les deux perspectives principales

Travail en mode modélisation

Travail en mode simulation



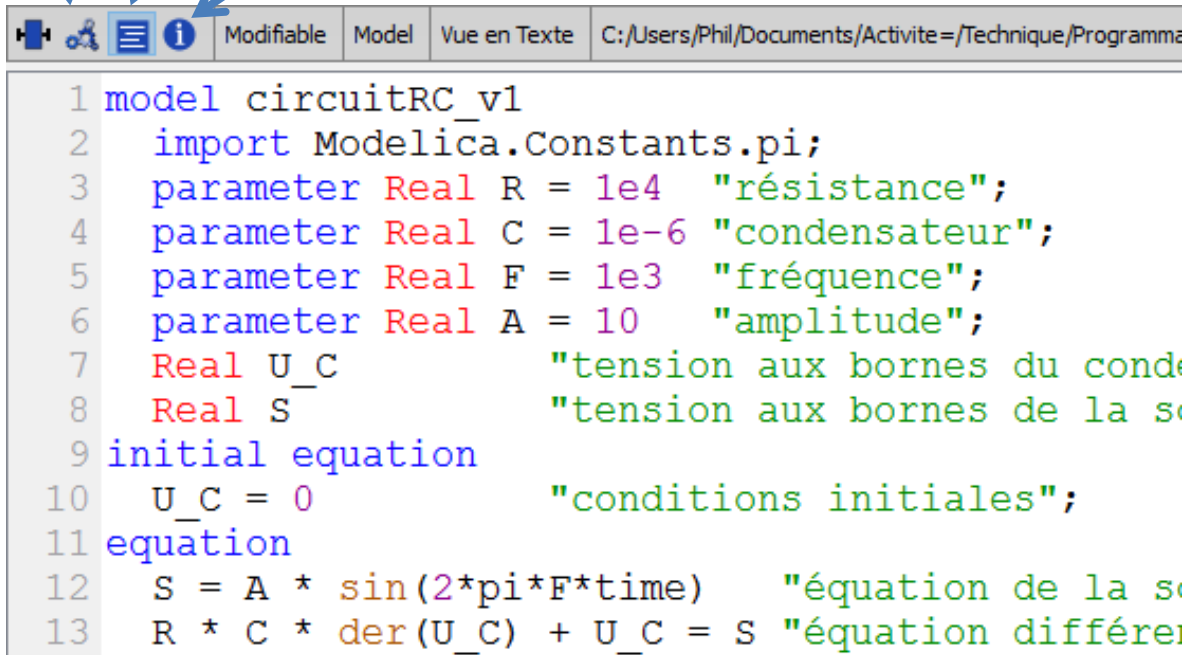
# 3a1. L'IDE OMEdit

- Perspective de modélisation

Voir le modèle en mode graphique (édition des connexions)

Voir le modèle sous forme de code source

Générer la documentation (si exploitée)



```
1 model circuitRC_v1
2   import Modelica.Constants.pi;
3   parameter Real R = 1e4 "résistance";
4   parameter Real C = 1e-6 "condensateur";
5   parameter Real F = 1e3 "fréquence";
6   parameter Real A = 10 "amplitude";
7   Real U_C "tension aux bornes du cond
8   Real S "tension aux bornes de la s
9   initial equation
10    U_C = 0 "conditions initiales";
11   equation
12    S = A * sin(2*pi*F*time) "équation de la s
13    R * C * der(U_C) + U_C = S "équation différen
```

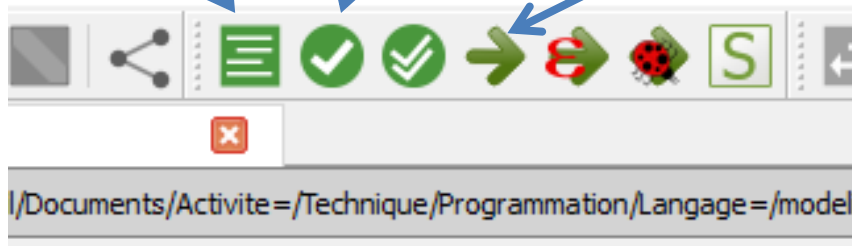
# 3a1. L'IDE OMEdit

- Perspective de modélisation : compilation

Mise à plat du modèle (liste variables et équations)

Vérification du modèle

Lancer la simulation



```
.pi;  
'résistance";  
'condensateur".
```

# 3a1. L'IDE OMEdit

- Perspective de simulation

Relancer la simulation

Modifier les paramètres de simulation

Sélectionner/changer des variables

The screenshot displays the OMEdit simulation environment. At the top, a toolbar contains various icons for simulation control. Below the toolbar, a window titled 'circuitRC\_v1' is open. The main area shows a plot of a red waveform over time, with a grid and a cursor at coordinates (0.0271, 0.2898). The x-axis is labeled with values 0,015, 0,02, 0,025, and 0,03. On the right side, there are two panels: 'Navigateur de Documentation' and 'Navigateur de Variables'. The 'Navigateur de Variables' panel shows a list of variables for the circuit, with their current values.

Variables	Valeur
<input checked="" type="checkbox"/> MAT circuitRC_v1	
<input type="checkbox"/> A	10.0
<input type="checkbox"/> C	1e-006
<input type="checkbox"/> F	1000.0
<input type="checkbox"/> R	10000.0
<input type="checkbox"/> S	-2.15585e-13
<input checked="" type="checkbox"/> U_C	
<input type="checkbox"/> der(U_C)	15.1228
<input type="checkbox"/> time	0.03 94

## 3a2. Documentations utiles

- Une référence compacte (*quick reference card*) sur Modelica :

<http://modref.xogeny.com/>

- Le site de référence :

<http://modelica.org>

- La spécification du langage
- Des actes de conférences
- Des tutoriels
- L'environnement de développement exploité :

<http://openmodelica.org>

## 3b. Exercices

- Exercice 1 : installation et test
  - Installer OpenModelica
  - Tester la version 1 du circuit RC
    - Visualiser la tension aux bornes de la source S
    - Visualiser la tension  $U_C$  aux bornes du condensateur C
  - Tester différentes valeurs de la fréquence de la source :  $F = 1\text{Hz}, 10\text{Hz}, 100\text{Hz}, 1\text{KHz}$  ; que remarque-t-on pour la tension  $U_C$  ?



# 3b. Exercices

- Exercice 2 : le pendule simple

- Ecrire le modèle permettant d'étudier un pendule simple, en exploitant les types physiques prédéfinis de Modelica (`Modelica.SIunits.*`)

```
import SI = Modelica.SIunits;  
permet d'écrire ensuite le type : SI.Length
```

- Dans un premier temps, supposer qu'il n'y a pas d'amortissement ; l'équation à résoudre est alors :

$$mL^2\ddot{\theta} + mLg \sin \theta = 0$$

- Prendre ensuite en compte un facteur d'amortissement  $\beta$ , l'équation devenant :

$$mL^2\ddot{\theta} + mLg \sin \theta + \beta L^2\dot{\theta} = 0$$

Données :  $L = 0,5\text{m}$ ,  $m = 0,2\text{Kg}$ ,  $\beta = 1$ ,  $\theta_0 = 75^\circ$

