SysTherLin

Outil de simulation de systèmes thermiques linéaires

Introduction

Le projet SysTherLin est un code Orienté Objet écrit en Python qui permet de simuler l'état au cours du temps d'un système thermique linéaire. Ce système thermique est supposé n'être composé que de multicouches 1D conducteurs (plans, ou à symétrie cylindrique ou sphérique) et de cavités.

Comme le calcul se fait en calculant la solution exacte du problème dans le domaine de Laplace puis en revenant dans le domaine temporel par transformée de Laplace réciproque numérique, on ne peut résoudre qu'un problème vérifiant les hypothèses suivantes :

1. L'état d'une couche conductrice (supposée homogène et sans source thermique à l'intérieur) est caractérisé par les champs de température $T(r,t)$ et de densité surfacique de flux de puissance $\phi(r,t)$, orienté selon les positions *r* croissantes, vérifiant les équations aux dérivées partielles linéaires suivantes :

$$\left\{\begin{array}{lcll} \partial_t T(r,t) & = & \dfrac{-1}{\rho\, c_p}\left(\partial_r \phi(r,t) + \dfrac{m}{r}\,\phi(r,t)\right) & \text{(conservation de l'énergie)} \\ \phi(r,t) & = & -k\,\partial_r T(r,t) & \text{(conduction)} \end{array}\right. \tag{1}$$

   avec les notations suivantes :

   - k est la conductivité thermique (en $\mathrm{J\cdot s^{-1}\cdot m^{-1}\cdot K^{-1}}$) ;
   - ρ est la masse volumique (en $\mathrm{kg\cdot m^{-3}}$) ;
   - c_p est la capacité calorifique massique (en $\mathrm{J\cdot K^{-1}\cdot kg^{-1}}$) ;
   - $\left|\begin{array}{lcll} m & = & 0 & : \text{ couche plane} \\ m & = & 1 & : \text{ couche à symétrie cylindrique} \\ m & = & 2 & : \text{ couche à symétrie sphérique} \end{array}\right.$; ; .

   Remarque importante : pour un multicouche, **l'axe des *r*** (distance au centre pour les symétries cylindrique et sphérique) **est toujours orienté de la gauche vers la droite**.

2. À l'interface entre deux couches conductrices (nécessairement de même nature géométrique) on suppose qu'il y a continuité de la température et du flux (pas de résistance thermique de contact).

3. Les conditions limites au bord d'un multicouche peuvent être de 3 sortes :

   - Température imposée (type Dirichlet, difficile à réaliser) ;
   - Densité surfacique de flux imposée (type Neumann) ;
   - Convection (vers un gaz ou un fluide « agité ») : la densité surfacique de flux de puissance thermique est proportionnelle à la différence de températures entre la paroi et le fluide :

$$\phi(r,t) = \pm\eta\ \left(T(r,t) - T_{\text{fluide}}(t)\right) \quad , \tag{2}$$

     où η désigne le *coefficient de convection* (en $\mathrm{J\cdot s^{-1}\cdot m^{-2}\cdot K^{-1}}$).

   Dans le cas particulier d'un multicouche cylindrique ou sphérique plein ($r = 0$ sur la frontière gauche du multicouche), il faut rajouter une *condition de symétrie*, ce qui est fait automatiquement dans le code.

4. Une cavité (supposée contenir un gaz ou un fluide « agité ») est caractérisée par :
   - son volume $\mathcal{V}$ (en m^3) ;
   - sa masse volumique ρ (en $\mathrm{kg}\cdot\mathrm{m}^{-3}$) ;
   - sa capacité calorifique massique c_p (en $\mathrm{J}\cdot\mathrm{K}^{-1}\cdot\mathrm{kg}^{-1}$) ;

   $\rightarrow$ soit une capacité calorifique globale $\rho\, c_p\, \mathcal{V}$ (en $\mathrm{J}\cdot\mathrm{K}^{-1}$) ;
   - ses parois, chacune étant le bord d'un multicouche de surface $\mathcal{S}_i$ avec une condition de convection vers la cavité de coefficient η_i ;

   $\rightarrow$ l'équation de conservation d'énergie dans chaque cavité s'écrit :

$$T'_{\text{cavité}}(t) = \frac{1}{\rho\, c_p\, \mathcal{V}} \sum_i \mathcal{S}_i\, \varphi_i(t) \quad \text{avec} \quad \varphi_i(t) = \eta_i \left(T_i(t) - T_{\text{cavité}}(t)\right) \ , \tag{3}$$

   la température $T_i(t)$ de la paroi i étant supposée uniforme.

5. Depuis la version 1.2, **chaque élément** (couche conductrice ou cavité), **peut avoir une température initiale différente** (température supposée unuiforme sur tout l'élément). Il s'agit du seul argument optionnel **`Tinit`**, qui vaut 0.0 par défaut.

Pour utiliser cet outil de simulation, il faudra respecter l'ordre décrit ci-dessous.

Des extraits des deux fichiers d'exemples **`SysTherLin_Exemple_1_une_cavite.py`** et **`SysTherLin_Exemple_2_deux_cavites.py`** sont donnés pour illustrer la démarche.

1 Définition des couches conductrices

On commence par définir chacune des couches conductrices présentes dans le système. Pour ce faire, on instancie la classe **`CoucheConductrice`** (plane), dont le constructeur a comme arguments, dans l'ordre, la conductivité k, la masse volumique ρ, la capacité thermique massique c_p, l'épaisseur e et la température initiale T_0 (valeur par défaut nulle pour rester compatible avec les programmes utilisant les versions antérieures à 1.2), ou l'une de ses classes dérivées **`CoucheConductriceCylindrique`** ou **`CoucheConductriceSpherique`**, dans les constructeurs desquelles l'épaisseur e est remplacée par les rayons $R_{\min}$ et $R_{\max}$. Ces 3 classes sont définies dans le fichier **`Couches_conductrices.py`**.

```
from Couches_conductrices import CoucheConductrice, Multicouche
## 1 - Définition des couches conductrices
T0 = 15.0 # Température initiale commune ici à tous les éléments
inox = CoucheConductrice(16.5, 8000.0, 500.0, 3.0e-3, Tinit=T0)
verre_socle =  CoucheConductrice(1.0, 2800.0, 1000.0, 4.0e-3, Tinit=T0)
verre_coque =  CoucheConductrice(1.0, 2800.0, 1000.0, 8.0e-3, Tinit=T0)
```

Lignes de codes de l'exemple n°1 créant des couches conductrices.

```
from Couches_conductrices import (CoucheConductriceCylindrique,
                                  CoucheConductriceSpherique)
## 1.1 - Couches sphériques de la cloche
inox1 = CoucheConductriceSpherique(16.5, 8000.0, 500.0,
                                   0.150, 0.152, T0)
pstyr = CoucheConductriceSpherique(0.04, 18.0, 1450.0,
                                   0.152, 0.157, T0)
inox2 = CoucheConductriceSpherique(16.5, 8000.0, 500.0,
                                   0.157, 0.159, T0)
## 1.2 - Couches cylindriques du tube
# Polypropylène avec 25% de fibres de verre.
pp1 = CoucheConductriceCylindrique(0.22, 910.0, 1800.0,
                                   25.0e-3, 26.5e-3, T0)
air = CoucheConductriceCylindrique(0.026, 1.2, 1000.0,
                                   26.5e-3, 28.5e-3, T0)
pp2 = CoucheConductriceCylindrique(0.22, 910.0, 1800.0,
                                   28.5e-3, 30.0e-3, T0)
```

Lignes de codes de l'exemple n°2 créant des couches conductrices cylindriques et sphériques.

2 Définition des multicouches

On assemble ensuite les couches pour former des multicouches, instances de la classe **`MultiCouche`** définie également dans le fichier **`Couches_conductrices.py`**. Ces **couches** doivent être **de même type** (plan, cylindrique, ou sphérique). Elles sont **ordonnées selon les valeurs croissantes des positions x (ou r)**. Si les couches ne sont pas planes, il doit y avoir correspondance des rayons aux interfaces.

```
## 2.1 - Socle multicouche, en spécifiant les conditions aux bords
socle = Multicouche([inox, verre_socle])
socle.definir_CL("G", "Neumann") # Côté chauffage
socle.definir_CL("D", "Convection",200.0) # Côté eau (cavité)
## 2.2 - Ensemble des autres parois (monocouche)
coque = Multicouche([verre_coque])
coque.definir_CL("G", "Convection", 200.0) # Côté eau (cavité)
coque.definir_CL("D", "Convection", 10.0) # Côté air extérieur
```

Lignes de codes de l'exemple n°1 créant un bicouche et un monocouche, puis spécifiant les types de conditions aux bords.

```
## 2.1 Cloche hémisphérique
coque = Multicouche([inox1, pstyr, inox2])
coque.definir_CL("G", "Convection", 10.0) # Côté intérieur
coque.definir_CL("D", "Convection", 10.0) # Côté extérieur
## 2.2 Tube cylindrique
tube = Multicouche([pp1, air, pp2])
tube.definir_CL("G", "Convection", 200.0) # Côté intérieur
tube.definir_CL("D", "Convection", 10.0) # Côté extérieur
```

Lignes de codes de l'exemple n°2 créant 2 tricouches, le premier sphérique et le second cylindrique.

Il est conseillé de spécifier les conditions aux bords juste après la définition du multicouche, comme sur les exemples. Les **multicouches** sont **orientés de la gauche vers la droite**. Le côté gauche correspond toujours à la valeur minimale de la position et le côté droit à la valeur maximale. Les types de conditions aux bords sont **`"Dirichlet"`**, **`"Neumann"`** et **`"Convection"`** (cette dernière condition est obligatoire sur un côté donnant sur une cavité). En $r = 0$ pour les cas cylindrique –cylindre plein– et sphérique –boule pleine–, la condition au centre est automatiquement **`"(Symétrie)"`**.

3 Définition des cavités

Un système linéaire peut soit contenir un multicouche seul, soit une seule cavité avec des parois multicouches, soit plusieurs cavités avec des parois multicouches. Dans le cas à plusieurs cavités, deux cavités soit ont une paroi en commun, nécessairement multicouche, soit n'ont pas de paroi commune. Deux cavités ne peuvent pas être en contact directement. Si l'on veux simuler ce dernier cas, il faut introduire une paroi fictive d'épaisseur très petite.

Une cavité est définie par son volume, la masse volumique et la capacité thermique massique du fluide « agité » qu'elle contient, ses parois, et la température initiale moyennée du fluide (valant 0 par défaut pour garantir la compatibitité avec les versions antérieures à 1.2). C'est une instance de la classe **`Cavite`** définie dans le fichier **`Systemes_thermiques_lineaires.py`**. Chaque paroi est définie par un triplet **`(mc,cote,S)`**, où **`mc`** désigne un multicouche, **`cote`** (**`"G"`** ou **`"D"`**) le côté du multicouche donnant sur la cavité, et **`S`** la surface de la paroi de ce côté-là.

```
from Systemes_thermiques_lineaires import Cavite
## 3 - Définition de la cavité
cavite = Cavite(0.2, 1000.0, 4200.0,
                [(socle,"D",0.4), (coque,"G",1.8)], T0)
```

Lignes de codes de l'exemple n°1 créant une cavité.

```
## 3.1 - Cavité dans un tube
eau = Cavite(V_tube, 1000.0, 4200.0, [[tube, "G", S_tube_int]], T0)
## 3.2 - Cavité entre un plateau et une cloche, contenant un tube
air = Cavite(V_air, 1.2, 1000.0, [ [coque, "G", S_coque_int],
                                   [socle, "D", S_socle_sup],
                                   [tube, "D", S_tube_int]
                                 ], T0)
```

Lignes de codes de l'exemple n°2 créant 2 cavites, les paramètres **`V_tube`**, **`V_air`**, **`S_tube_int`**, **`S_tube_ext`**, **`S_socle_sup`** et **`S_coque_int`** ayant été définis préalablement.

4 Définition du système linéaire

Il n'y a ensuite plus qu'à assembler pour définir le système thermique linéaire à simuler, instance de la classe **`SystemeThermiqueLineaire`**, en spécifiant dans l'ordre : la durée d de simulation et le pas de temps T_s auxquels on veut connaître la solution, puis le système : soit un multicouche seul, soit une cavité seule, soit une liste de cavités. Cette classe est définie dans le fichier **`Systemes_thermiques_lineaires.py`**.

```
from Systemes_thermiques_lineaires import SystemeThermiqueLineaire
## 4 - Définition du systeme global, à une seule cavité
jour = 3600.0 * 24.0
STL = SystemeThermiqueLineaire(4*jour, 10.0, cavite)
```

Lignes de codes de l'exemple n°1 créant le système global à simuler, sur une durée de 4 jours par pas de 10 secondes.

```
## 4 - Système du système global, à 2 cavités
systeme = SystemeThermiqueLineaire(2*3600, 1.0, [air,eau])
```

Lignes de codes de l'exemple n°2 créant le système global à simuler, sur une durée de 2 heures par pas de 1 seconde.

5 Définition des signaux et résolution

Il reste à définir tous les signaux extérieurs (flux imposé, température imposée, ou température extérieure du fluide vers lequel il y a un flux de convection), à l'aide de la méthode **`Multicouche.definir_signal`**. Cette méthode admet soit le vecteur des valeurs du signal échantillonné aux instants positifs ou nuls (solution conseillée si on ne maîtrise pas la transformation de LAPLACE), soit une fonction vectorisée représentant la transformée de LAPLACE du signal. Dans ce dernier cas, nous rappelons que pour obtenir une constante **`C`**, la transformée de LAPLACE peut être définie par : « **`lambda s : C/s`** ».

```
## 5.1 Définition du signal de chauffage
def chauf(t) : # Fonction vectorisée
    h = (t/3600.0)%24.0  # Heure de la journée
    return ((h>23)|(h<4))*500.0 # 500 W/m**2 s'il est allumé
instants_positifs = STL.positiveTimeValues
socle.definir_signal("G", chauf(instants_positifs))
## 5.2 Définition de la température extérieure
def T_ext(t, Tinit=T0) :
    # Constantes :
    m,a,j,dp = 5.0, 0.15, 3600.0*24.0, 2.0*np.pi
    # Signal renvoyé
    return Tinit + m*(1-np.cos(dp*t/j-a))
coque.definir_signal("D", T_ext(instants_positifs))
## 5.3 Calcul de la solution
STL.calculer_maintenant()
```

Lignes de codes de l'exemple n°1 spécifiant les signaux aux bords, puis lançant le calcul de la solution.

```
## 5 - Calcul et visualisation des données
# Température extérieure constante de 10°C: transformée de Laplace 10/s
socle.definir_signal("G", lambda s : 10.0/s)
coque.definir_signal("D", lambda s : 10.0/s)
systeme.calculer_maintenant()
```

Lignes de codes de l'exemple n°2 spécifiant les signaux aux bords, puis lançant le calcul de la solution.

Une fois que tout est bien défini, il suffit de lancer le calcul l'instruction :

« **`SystemeThermiqueLineaire.calculer_maintenant()`** ».

Vous aurez alors dans la console soit un message d'erreur vous indiquant les données manquantes, soit le message « **`... Calcul effectué.`** ».

6 Visualisation de la solution

Pour accéder aux températures dans les cavités, il suffit de récupérer les signaux dans la liste donnée par l'attribut **`SystemeThermiqueLineaire.T_cavites`**.

Pour accéder aux températures et aux densités surfaciques de flux à l'intérieur d'un multicouche, il suffit d'appeler la méthode **`Multicouche.T_Phi`** en spécifiant la position x (ou le rayon r) à laquelle on veut l'information. Par commodité, cette fonction renvoie aussi le vecteur des instants d'échantillonnage. Un usage répandu en Python consiste à nommer « _ » tout ce qui est renvoyé par une fonction et dont on n'a pas besoin. Par exemple, on écrit :

« **`_,T,_ = mc.T_Phi(x0)`** » si on ne veut que le signal de température en x_0,

et « **`_,_,phi = mc.T_Phi(x1)`** » si on ne veut que le signal de densité surfacique flux en x_1.

```
instants = STL.timeValues # en secondes
t_en_jours = instants / jour
## 6.1 - Température de l'eau dans la cavité
T_cav = STL.T_cavites[0]
plt.plot(t_en_jours, T_cav, <options>, label = "T° eau")
## 6.2 -  Température au coeur de l'inox du socle
_,T_inox,_ = socle.T_phi(1.5e-3) # milieu de l'inox
plt.plot(t_en_jours, T_inox, <options>, label = "T° inox")
## 6.3 -  Température des parois de verre
# coque.X donne la liste des positions des interfaces
_,T_verre_int,_ = coque.T_phi(coque.X[0]) # intérieur
_,T_verre_ext,_ = coque.T_phi(coque.X[-1]) # extérieur
plt.plot(t_en_jours, T_verre_int, <options>, label = "T° v.int.")
plt.plot(t_en_jours, T_verre_ext, <options>, label = "T° v.ext.")
## 6.4 - Finalisation du tracé
plt.legend(loc="best", fontsize=10) ; plt.show()
```

Lignes de codes de l'exemple n°1 permettant d'extraire quelques signaux de température pour les afficher sur le même graphique. `<options>` correspond aux directives de tracé (style, couleur et épaisseur de trait).

Annexe : Diagrammes de classes

Classe CoucheConductrice

« *attributs* » en lecture seule

+ **k** : float	*Conductivité thermique k (en $\mathrm{J\cdot s^{-1}\cdot m^{-1}\cdot K^{-1}}$)*
+ **rho** : float	*Masse volumique ρ (en $\mathrm{kg\cdot m^{-3}}$)*
+ **Cp** : float	*Capacité calorifique massique c_p (en $\mathrm{J\cdot K^{-1}\cdot kg^{-1}}$)*
+ **e** : float	*Épaisseur e (en m)*
+ **un_sur_alpha** : float	$\frac{1}{\alpha}=\frac{\rho\,c_p}{k}\left(\alpha=\frac{k}{\rho\,c_p}\right)$
+ **tau** : float	*Constante de temps* $\tau=\frac{e^2}{2\,\alpha}$
+ **Tinit** : float	*Température initiale (uniforme) (en °C)*

Méthodes publiques

+ **CoucheConductrice**(**k** : float, **rho** : float, **Cp** : float, **e** : float, **Tinit** : float)

+ **P**(**s** : vector of floats, **x** : float) : ndarray

Tableau $n_s\times 2\times 2$ permettant de déterminer dans le domaine de Laplace *les champs de température et de densité surfacique de flux à n'importe quelle position x*

+ **copy**() *Duplication de la couche*

Classe CoucheConductriceCylindrique(CoucheConductrice)

« *attributs* » en lecture seule

+ **Rmin** : float	*Rayon intérieur $r_{\min}$ (en m)*
+ **Rmax** : float	*Rayon extérieur $r_{\max}$ (en m)*
+ **tau** : float	*Constante de temps, fonction de e, α et du rapport $r_{\min}/r_{\max}$, allant de $\frac{e^2}{4\,\alpha}$ pour un cylindre plein à $\frac{e^2}{2\,\alpha}$ lorsque $\frac{r_{\min}}{r_{\max}}\underset{<}{\to}1$.*

Méthodes publiques

+ **CoucheConductriceCylindrique**(**k** : float, **rho** : float, **Cp** : float, **Rmin** : float, **Rmax** : float, **Tinit** : float)

+ **P**(**s** : vector of floats, **r** : float) : ndarray

Redéfinition de la méthode pour tout rayon r

Classe CoucheConductriceSpherique(CoucheConductrice) : idem.

- La constante de temps vaut pour une couche sphérique : $\tau=\frac{e^2}{6\,\alpha}\left(1+2\,\frac{r_{\min}}{r_{\max}}\right)$.

Classe **Multicouche**

« *attributs* » en lecture seule

+ **nb** : int	*Nombre de couches*
+ **couches** : list of CoucheConductrice	*Liste des couches le constituant*
+ **X** : list of floats	*Positions des interfaces, en partant de* 0 *(cas plan) ou de* $r_{\min}$ *(autres cas)*
+ **geometrie** : str	***"plane"***, ***"cylindrique"*** *ou* ***"sphérique"***
+ **timeValues** : vector of floats	*Vecteur* **t** *des instants d'échantillonnage auxquels on connaîtra les champs recherchés*
+ **positiveTimeValues** : vector of floats	*Vecteur des instants positifs ou nuls*
+ **s** : float	*Vecteur* **s** *des valeurs complexes correspondantes, dans le domaine de* LAPLACE
+ **CLG** : tuple	*Type de condition limite à gauche :* ***"Dirichlet"*** *ou* ***"Neumann"*** *suivi de la transformée de* LAPLACE *numérique du signal extérieur, ou* ***"Convection"*** *suivi du coefficient de convection puis de la T.L. numérique du signal extérieur, ou* ***"[Symétrie]"*** *si* $r = 0$ *(cylindrique ou sphérique)*
+ **CLD** : tuple	*Idem, du côté droit et sans condition de symétrie possible*

Méthodes publiques

+ **Multicouche**(**couches** : list of CoucheConductrice)

+ **definir_temps**(**duree** : float, **Ts** : float)
Sert à définir la durée et la période d'échantillonnage

+ **TLdir**(**signal** : vector of floats) : vector of complexes
Renvoie la T.L. numérique du signal ; comportant autant de valeurs que d'instants d'échantillonnage positifs ou nuls

+ **TLrec**(**U** : vector of complexes) : vector of floats
Renvoie le signal obtenu par T.L. numérique réciproque, U comportant autant de valeurs que le vecteur **s**

+ **signal**(**cote** : str) : vector of floats
Renvoie le signal extérieur, du côté ***"G"*** *ou* ***"D"***

+ **TLsig**(**cote** : str) : vector of complexes
Renvoie la T.L. du sig. ext., du côté ***"G"*** *ou* ***"D"***

+ **tout_est_defini**(**signals** : tuple of bool) : bool
Renvoie un booléen indiquant si les conditions aux bords sont bien définies ; ***signals=(bG,bD)*** *où ces booléens indiquent s'il faut vérifier la définition d'un signal extérieur sur le côté gauche/droit, ou pas*

+ **set_AB**(**newAB** : array of complexes)
On donne le tableau $n_s \times 2nb$ *des coefficients qui permettent de définir les champs partout*

+ **M_matrix**(<*options*>) : array of complexes
Renvoie le tableau $n_s \times 2nb \times 2nb$ *des matrices qui permettent de trouver le tableau* AB *des coefficients*

+ **T_phi**(**x** : float) : list of vectors of floats
Renvoie les vecteurs : **t** *des instants,* **T** *des températures et* **Φ** *des densités surfaciques de flux*

Classe **Cavite**

« *attributs* » en lecture seule

+ **volume** : float	*Volume $\mathcal{V}$ de la cavité (en m^3)*
+ **rho** : float	*Masse volumique ρ (en $\mathrm{kg{\cdot}m^{-3}}$)*
+ **Cp** : float	*Capacité calorifique massique c_p (en $\mathrm{J{\cdot}K^{-1}{\cdot}kg^{-1}}$)*
+ **a** : float	$a = \dfrac{1}{\rho\, c_p\, \mathcal{V}}$ *(en $\mathrm{K{\cdot}J^{-1}}$)*
+ **parois** : list of Multicouche	*Liste des parois multicouches*
+ **cotes** : list of int	*Côtés de chaque paroi : 0 pour* **"G"** *et -1 pour* **"D"**
+ **surfaces** : list of floats	*Liste des surfaces des parois (en m^2)*
+ **Tinit** : float	*Température initiale (en °C)*

Méthode publique

+ **Cavite(volume** : float, **rho** : float, **Cp** : float , **parois** : tuple, **Tinit** : float **)**

parois *est une liste de triplets* ***(mc, cote, S)***, *où* ***mc*** *est le multicouche constituant la paroi,* ***cote*** *est* **"G"** *ou* **"D"** *spécifiant le côté en contact avec la cavité et* ***S*** *la surface de contact.*

Classe **SystemeThermiqueLineaire**

« *attributs* » en lecture seule

+ **timeValues** : vector of floats	*Vecteur* **t** *des instants*
+ **positiveTimeValues** : vector of floats	*Vecteur des instants positifs ou nuls*
+ **cavites** : list of Cavite	*Liste des cavités (peut être vide)*
+ **T_cavites** : list of vectors of floats	*Liste des signaux de température dans les cavités (disponible uniquement lorque le calcul a été effectué)*
+ **multicouches** : list of Multicouche	*Liste des multicouches présents dans le système*

Méthodes publiques

+ **SystemeThermiqueLineaire(duree** : float, **dt** : float , **elements** : plusieurs types **)**

duree *est la duree sur laquelle on veut calculer la solution, avec un pas de temps* ***dt***. ***elements*** *est soit un multicouche seul, soit une cavité seule, soit une liste de cavités.*

+ **calculer_maintenant( )** *Sert à lancer le calcul, qui se fait uniquement si les données fournies sont complètes, et sinon affiche dans la console la nature du ou des problèmes.*