Méthode Hamiltonienne

L.Jeanneret, T.Le Ber, D.Pernot

Résumé : on détermine par la méthode de Lagrange les équations de mouvement d'un mécanisme simple composé d'un bâti, d'une liaison et d'un solide, dont les caractéristiques sont connues.

Abstract : one determines by the method of Lagrange the motion equations of a simple linkage composed of a frame, a joint and a solid, whose characteristics are known.

■ Introduction

En mécanique il est important de pouvoir modéliser un mécanisme. Un mécanisme possède des entrées et des sorties et celles-ci obéissent à des règles appelées lois de comportement.

Deux méthodes sont à notre disposition pour établir ces équations : le Principe Fondamental de la Dynamique et les équations de Lagrange. C'est à cette dernière méthode que nous allons nous intéresser. Ces équations mettent donc en relation les paramètres de sortie avec les paramètres d'entrée, le système pouvant être composé de plusieurs solides, le calcul des lois de comportement devient lourd et l'intérêt d'un programme devient évident.

La méthode hamiltonienne permet de déterminer les équations de mouvement grâce aux énergies cinétiques et aux puissances développées par les actions extérieures au mécanisme ainsi que par les liaisons si celles-ci sont imparfaites. Nous devons donc être capable de calculer ces énergies et puissances en utilisant des outils de la mécanique générale tels que les torseurs.

■ Analyse du problème

Pour effectuer la programmation avec Mathematica, il nous faut déterminer une structure de données qui modélise le mécanisme étudié. Il doit nous être possible de représenter les solides, les liaisons et les efforts extérieurs appliqués. Nous devons également créer l'outil torseur qui intervient dans les calculs de puissance et d'énergie ; cet outil se présente sous différentes formes selon qu'il représente des efforts, des vitesses etc... Il faut également programmer les opérations qui lui sont associées : la somme, le comoment, le changement de repère, etc.

Dans un premier temps le programme devra déterminer l'équation de mouvement d'un mécanisme simple mais tout le programme doit être développé de telle sorte qu'on puisse ultérieurement considérer des mécanismes composés de plusieurs solides. De même, il doit être possible de considérer des champs virtuels compatibles ou non avec les liaisons du mécanisme et ainsi de laisser à l'utilisateur le choix du paramétrage et de pouvoir faire apparaître les efforts s'exerçant dans les liaisons même si celles-ci sont parfaites.

Une base de connaissances relative aux liaisons nous sera rendue accessible par un autre groupe, il s'agit donc de développer le programme en relation avec celui-ci tel qu'il soit compatible avec cette base de règles : nous devrons choisir un paramétrage commun.

Des hypothèses doivent être posées pour simplifier les calculs et rendre l'utilisation du programme plus aisée. Par exemple, la matrice d'inertie liée au solide doit être exprimée au centre de gravité de celui-ci.

Dans un premier temps, nous ne considérons qu'un système composé d'un bâti, d'une liaison et d'un solide dont tous les paramètres sont connus.

■ Conception du programme

■ Détermination des équations de mouvement

Premièrement, il nous est utile de définir l'outil torseur. Un torseur est composé d'une résultante et d'un moment, que l'on prend à l'origine. Voici comment nous l'avons programmé :

```
R[Torsor[r_, m_]] := r
M[Torsor[r_, m_]] := m
```

Durant notre étude, il nous sera nécessaire d'exprimer un ou plusieurs torseurs en d'autres points que celui de leur expression d'origine. Pour cela il nous suffit d'appliquer au moment du torseur la formule de changement de point car la résultante ne change pas.

$$\overrightarrow{\mathcal{M}}(B)=\overrightarrow{\mathcal{M}}(A)+\overrightarrow{BA}\wedge\overrightarrow{R}$$

Et voici comment nous avons traduit cela en Mathematica :

```
Torsor[r_, m_][pt_] := Torsor[r, m + Cross[pt, r]]
```

Changement de repère liant le repère du bâti et le repère du solide (ceci est un extrait de la base de connaissance) :

```
Movement [PivotY[Q_, t_: t]] :=
 {{Cos[Q[t]], 0 , Sin[Q[t]]}, {0, 1 , 0},
    {-Sin[Q[t]], 0, Cos[Q[t]]}}.# &
```

Pour réaliser tous les calculs il faut aussi exprimer les torseurs dans un même repère. Dans notre projet, tous les calculs seront réalisés dans le repère du bâti. Il a donc fallu créer une formule de changement de repère.

```
ChangeReferential[Torsor[r_, m_], ch_] :=
 Torsor[ch[r] - ch[{0, 0, 0}],
  ch[m] + Cross[ch[{0, 0, 0}], r] - ch[{0, 0, 0}]]
```

Nous allons ensuite introduire la somme de deux torseurs, ainsi que la multiplication externe :

```
Plus[Torsor[r1_, m1_], Torsor[r2_, m2_]] ^:=
 Torsor[r1 + r2, m1 + m2]
Times[k_, Torsor[r_, m_]] ^:= Torsor[k r, k m]
```

Une fois l'outil torseur défini ainsi que les différents changements de point et de repère, il nous faut faire apparaître une autre opération entre deux torseurs qui se nomme le comoment. Cette opération est utile lors de différents calculs de notre étude. Voici comment se définit ce comoment entre deux torseurs impérativement définis dans un même repère :

$$C=\{\mathcal{T}_1\}\otimes\{\mathcal{T}_2\}=\overrightarrow{R_1}.\overrightarrow{T_2(A)}+\overrightarrow{R_2}.\overrightarrow{T_1(A)}$$

C'est donc la somme du produit scalaire entre la résultante du premier torseur et le moment du second et du produit scalaire entre la résultante du second torseur et le moment du premier. A noter que le comoment ne dépend pas du point choisi. Voici comment nous l'avons défini ici :

```
Comoment[Torsor[r1_, m1_], Torsor[r2_, m2_]] :=
 Plus[Dot[r1, m2], Dot[r2, m1]]
```

Nous allons maintenant définir le torseur cinématique qui représente les vitesses de rotation et de translation d'un solide. Il est défini comme suit :

$$\{V^i{}_j\} = \begin{pmatrix} \overrightarrow{\Omega^i{}_j} \\ \overrightarrow{V^i{}_j(A)} \end{pmatrix}_A$$

où $\overrightarrow{\Omega^i{}_j}$ est la résultante du torseur cinématique qui représente la vitesse angulaire du solide et $\overrightarrow{V^i{}_j(A)}$ représente le vecteur vitesse du point A appartenant au solide (dans notre cas, l'origine). Sachant que pour l'instant on utilise une liaison pivot en guise d'exemple, nous avons défini le torseur cinématique pour cette liaison de la façon suivante :

```
KinematicTorsor[PivotY[Q_, t_: t]] :=
 Torsor[{0, D[Q[t], t], 0}, {0, 0, 0}]
```

Cependant un autre groupe qui travaille sur la modélisation des liaisons pourra fournir par la suite une base de règles que nous pourrons adjoindre à notre programme.

Pour faire les calculs, il nous faut modéliser le système mécanique étudié. Pour cela nous avons choisi de le définir sous forme de graphe avec des arcs représentant les liaisons entre les solides (type Edge) et des sommets qui sont les solides (type Vertex).

```
MechanicalPattern :=
 Graph [{ Vertex [S0_, Bati[]], Edge[{S0_, S1_}, link_],
   Vertex[s1_, Body[m_, J_, G_]], torsor_}]
```

Dans cette définition "torsor" représente le torseur des actions extérieures appliquées sur le système. "Vertex" représente un sommet du graphe c'est-à-dire un solide. Ici nous avons le solide étudié qui est "Body" avec toutes ses caractéristiques connues. Enfin "Edge" représente un arc du graphe (ici il n'y en a qu'un seul qui est la liaison entre le bâti et le solide).

Une fois notre système mécanique modélisé, nous allons introduire de nouveaux outils pour calculer les équations de mouvement. Le premier est le torseur cinétique qui servira ensuite à calculer l'énergie cinétique. Ce torseur s'écrit de la façon suivante :

$$\{C^i{}_j\} = \begin{pmatrix} \overrightarrow{\sigma^i{}_j} \\ \overrightarrow{\mu^i{}_j(A)} \end{pmatrix}_A$$

où $\overrightarrow{\sigma^i{}_j}$ est la résultante du torseur et $\overrightarrow{\mu^i{}_j(A)}$ est le moment cinétique du torseur. Ils se calculent avec les relations suivantes :

$$\overrightarrow{\sigma^i{}_j} = m\,\overrightarrow{V^i{}_j(G)}$$
$$\overrightarrow{\mu^i{}_j(A)} = \overline{\overline{I_{A,j}}}\,\overrightarrow{\Omega^i{}_j} + m\,\overrightarrow{AG} \wedge \overrightarrow{V^i{}_j(A)}$$

Dans notre programme, on exprimera ce torseur comme ceci :

```
KineticTorsor[MechanicalPattern] :=
 Torsor[Times [m, M[KinematicTorsor[link][G]]],
  J.R[KinematicTorsor[link]]]
```

Dans cette relation, "MechanicalPattern" représente par un filtre la structure du système mécanique où la liaison "Link" est définie. On peut donc maintenant définir l'énergie cinétique qui est aussi obtenue en fonction du système mécanique modélisé précédemment.

$$T^i{}_j = \frac{1}{2}\{V^i{}_j\} \otimes \{C^i{}_j\}$$

D'où la programmation :

```
KineticEnergy[MechanicalPattern] := Times[1 / 2,
  Comoment[Torsor[Times [m, M[KinematicTorsor[link][G]]],
    J.R[KinematicTorsor[link]]], KinematicTorsor[link][G]]]
```

Pour calculer les équations de Lagrange nous avons besoin d'un dernier outil qui est la puissance virtuelle définie comme suit :

$$P^{i}{}_{j}{}^{*}=\{F_{i\to j}\}\otimes\{V^{i}{}_{j}{}^{*}\}$$

où $\{F_{i\to j}\}$ est le torseur d'actions extérieures et $\{V^{i}{}_{j}{}^{*}\}$ est le torseur cinématique virtuel. Voici notre programme :

```
VirtualPower[MechanicalPattern] := Comoment[torsor,
  ChangeReferential[KinematicTorsor [link][G], Movement [link]]]
```

Après avoir défini et programmé tous les outils nécessaires, nous pouvons enfin exprimer la formule donnant les équations de mouvement par la méthode de Lagrange.

$$\sum_{i=1}^{n} q_i^{*}\left(\frac{d}{\mathrm{dt}}\frac{\partial}{\partial q_i}-\frac{\partial}{\partial q_i}\right)T_{\Sigma}^{0}=\sum_{i=1}^{n} Q_i\, q_i^{*}$$

Ce qui nous donne avec Mathematica :

```
LagrangeEquation[MechanicalPattern_, Q_, t_: t] :=
 D[D[KineticEnergy[MechanicalPattern], D[Q[t], t]], t] -
   D[KineticEnergy[MechanicalPattern], Q[t]] ==
  D[VirtualPower[MechanicalPattern], D[Q[t], t]]
```

Nous allons tester la première partie de notre programme. Pour cela nous allons prendre l'exemple du pendule d'axe y, de paramètre θ, de masse m, de tenseur d'inertie nul, d'une longueur de tige L. L'axe z du bâti est vertical ascendant.

```
MechanicalSystem =
  Graph [{ Vertex [0, Bati[]], Edge[{0, 1}, PivotY[θ, t]], Vertex[
     1, Body[m, {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}, {0, 0, -L}]],
    Torsor[{0, 0, m * g}, {0, 0, 0}]}];

KineticTorsor[MechanicalSystem]

Torsor[{L m θ'[t], 0, 0}, {0, 0, 0}]

KineticEnergy[MechanicalSystem]

1/2 L^2 m θ'[t]^2
```

Et enfin voici l'équation de mouvement du système :

```
Simplify[LagrangeEquation[MechanicalSystem, θ, t]]

L m (g Sin[θ[t]] + L θ''[t]) == 0
```

A ce stade, la première partie de l'étude, qui consiste à obtenir la ou les équations de mouvement du système mécanique considéré, est terminée. Nous allons maintenant nous intéresser à la détermination de différents efforts de liaisons.

- **Détermination des efforts de liaison**

Dans cette partie la conception du programme est similaire à celle vue précédemment cependant il est nécessaire d'introduire de nouveaux outils. D'abord il est indispensable de savoir quels sont les efforts que nous voulons obtenir à la fin de l'étude. C'est pourquoi nous devons avoir un torseur des efforts qui nous permet de choisir quel(s) effort(s) nous voulons déterminer. En voici sa définition :

```
EffortTorsor := Torsor[{0, 0, 0}, {1, 0, 1}]
```

Les "1" correspondent aux efforts voulus. L'utilisateur n'a donc plus qu'à choisir où placer les "1" pour obtenir ces efforts. Dans cet exemple les "1" sont placés de telle sorte que nous puissions obtenir les efforts X et Z. Après cela nous devons concevoir une nouvelle formule pour chacune des étapes de la détermination des efforts. Tout d'abord nous devons définir le torseur cinématique virtuel :

```
VirtualKinematicTorsor[t_: t] :=
 Torsor[{D[α[t], t], D[β[t], t], D[γ[t], t]} * R[EffortTorsor],
  {D[x[t], t], D[y[t], t], D[z[t], t]} * M[EffortTorsor]]
```

Le torseur virtuel des mouvements :

```
VirtualMovementTorsor :=
 Torsor[{α, β, γ} * R[EffortTorsor], {x, y, z} * M[EffortTorsor]]
```

On introduit aussi la fonction Auxiliary qui va nous servir à associer un couple de solides à chaque effort de liaison. Cela nous permettra de savoir en fin d'étude à quelle liaison correspond chaque effort.

```
Auxiliary [Torsor[{x_, y_, z_}, {l_, m_, n_}], c_] :=
 Torsor[{x[c], y[c], z[c]}, {l[c], m[c], n[c]}]

Effort := Torsor[{x, y, z}, {d, e, f}]
OutEffort[MechanicalPattern] := Auxiliary[Effort, S0 → S1]
```

Calcul des efforts virtuels :

```
theVirtualPower :=
 Plus @@ (R[OutEffort] * M[VirtualKinematicTorsor[t]]) +
  Plus @@ (M[OutEffort] * R[VirtualKinematicTorsor[t]])
```

La suite est identique à la partie précédente à savoir que nous calculons le torseur cinématique, le torseur cinétique, l'énergie cinétique, la puissance virtuelle et enfin les équations de Lagrange qui nous donnent les efforts souhaités ; tout ceci en tenant compte du torseur des efforts.

Torseur cinématique :

```
KinematicTorsorEffort[MechanicalPattern] :=
 Plus[ChangeReferential[KinematicTorsor[link][G],
   Movement [link]], VirtualKinematicTorsor[t]]
```

Torseur cinétique :

```
KineticTorsorEffort[MechanicalPattern] := Torsor[
  Times [m, M[Plus[ChangeReferential[KinematicTorsor[link][G],
      Movement [link]], VirtualKinematicTorsor[t]]]],
  J.R[Plus[ChangeReferential[KinematicTorsor[link][G],
      Movement [link]], VirtualKinematicTorsor[t]]]]
```

Energie cinétique :

```
KineticEnergyEffort[ms_] := Times[1 / 2,
  Comoment[KineticTorsorEffort[ms], KinematicTorsorEffort[ms]]]
```

Puissance virtuelle :

```
VirtualPowerEffort[MechanicalPattern] :=
 Comoment[torsor, Plus[ChangeReferential[
     KinematicTorsor[link][G], Movement [link]],
    VirtualKinematicTorsor[t]]] + theVirtualPower
```

Equation(s) d'Euler :

```
EulerEquation[MechanicalPattern_, Q_, t_: t] := Eliminate[
  {D[D[KineticEnergyEffort[MechanicalPattern], D[Q[t], t]], t] -
     D[KineticEnergyEffort[MechanicalPattern], Q[t]] ==
    D[VirtualPowerEffort[MechanicalPattern], D[Q[t], t]],
   Q''[t] == 0}, Q''[t]]
```

Dans cette expression nous utilisons la fonction "Eliminate" qui nous permet de supprimer les paramètres nuls lors de l'affichage des équations. En fait cela revient à utiliser les équations de liaison (par exemple `x'[t] = 0`).

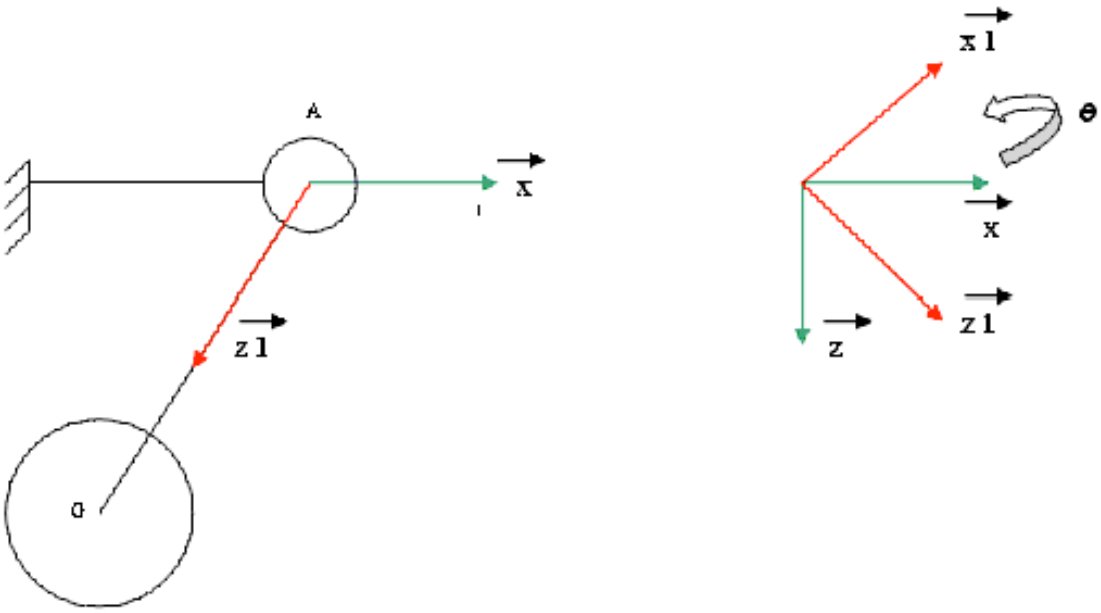
Nous introduisons aussi une nouvelle fonction "Parameters". Cependant elle ne sera pas utilisée dans notre programme mais elle servira de piste pour l'améliorer. Elle nous fournit le ou les paramètres d'une liaison.

```
Parameters [PivotY[Q_, t_: t]] := Q
Parameters [MechanicalPattern] := Parameters [link]
```

La seconde partie qui concerne la détermination des efforts de liaison est maintenant terminée et nous avons établi pour la suite un test du programme.

■ Test

Ce test a pour but de montrer que le programme fonctionne et qu'il nous fournit les bons résultats. Pour cela nous allons nous intéresser à la détermination de l'équation de mouvement d'un pendule simple. Ce mécanisme est composé d'un bâti, d'une liaison pivot d'axe y et du solide qui représente le pendule.




```
KinematicTorsor[PivotY[θ, t]]
```

```
Torsor[{0, θ'[t], 0}, {0, 0, 0}]
```

Changement de point : en G :

```
KinematicTorsor [PivotY[θ, t]][{0, 0, -L}]
```

```
Torsor[{0, θ'[t], 0}, {L θ'[t], 0, 0}]
```

Changement de base : base 0 (bâti) :

```
ChangeReferential[KinematicTorsor [PivotY[θ, t]][{0, 0, -L}],
 Movement[PivotY[θ, t]]]
```

```
Torsor[{0, θ'[t], 0}, {L Cos[θ[t]] θ'[t], 0, -L Sin[θ[t]] θ'[t]}]
```

Torseur cinétique en G (on reprend le système mécanique défini plus haut) :

```
MechanicalSystem
```

```
Graph[{Vertex[0, Bati[]], Edge[{0, 1}, PivotY[θ, t]],
  Vertex[1, Body[m, {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}, {0, 0, -L}]],
  Torsor[{0, 0, g m}, {0, 0, 0}]}]
```

```
KineticTorsor[MechanicalSystem]
```

```
Torsor[{L m θ'[t], 0, 0}, {0, 0, 0}]
```

Energie cinétique :

```
KineticEnergy[MechanicalSystem]
```

$$\frac{1}{2} L^2 m \theta'[t]^2$$

Puissance virtuelle :

```
VirtualPower[MechanicalSystem]
```

```
-g L m Sin[θ[t]] θ'[t]
```

Nous avons choisi de faire apparaître tous les efforts de liaison pour ce test.

```
EffortTorsor := Torsor[{1, 1, 1}, {1, 1, 1}]
TorsorToList[Torsor[{a_, b_, c_}, {d_, e_, f_}]] :=
 {a, b, c, d, e, f}
Psup = Intersection[TorsorToList[VirtualMovementTorsor],
  {α, β, γ, x, y, z}]
```

```
{x, y, z, α, β, γ}
```

```
KinematicTorsorEffort[MechanicalSystem]
```

```
Torsor[{α'[t], β'[t] + θ'[t], γ'[t]},
 {x'[t] + L Cos[θ[t]] θ'[t], y'[t], z'[t] - L Sin[θ[t]] θ'[t]}]
```

```
KineticTorsorEffort[MechanicalSystem]
```

```
Torsor[{m (x'[t] + L Cos[θ[t]] θ'[t]),
  m y'[t], m (z'[t] - L Sin[θ[t]] θ'[t])}, {0, 0, 0}]
```

```
KineticEnergyEffort[MechanicalSystem]
```

$$\frac{1}{2} \left(m\, y'[t]^2 + m \left(x'[t] + L \operatorname{Cos}[\theta[t]]\, \theta'[t]\right)^2 + m \left(z'[t] - L \operatorname{Sin}[\theta[t]]\, \theta'[t]\right)^2\right)$$

```
OutEffort = OutEffort[MechanicalSystem]
```

```
Torsor[{x[0 → 1], y[0 → 1], z[0 → 1]}, {d[0 → 1], e[0 → 1], f[0 → 1]}]
```

```
VirtualPowerEffort[MechanicalSystem]
```

```
x[0 → 1] x'[t] + y[0 → 1] y'[t] + z[0 → 1] z'[t] + d[0 → 1] α'[t] +
 e[0 → 1] β'[t] + f[0 → 1] γ'[t] + g m (z'[t] - L Sin[θ[t]] θ'[t])
```

Voici enfin l'équation de mouvement du système ainsi que les efforts de liaison souhaités :

```
Simplify[LagrangeEquation[MechanicalSystem, θ, t]]
```

```
L m (g Sin[θ[t]] + L θ''[t]) == 0
```

```
ColumnForm[
 Map [Simplify[EulerEquation[MechanicalSystem, #, t]] &, Psup]]
```

$$x[0 \to 1] + L\, m\, \text{Sin}[\theta[t]]\, \theta'[t]^2 == L\, m\, \text{Cos}[\theta[t]]\, \theta''[t]$$
$$y[0 \to 1] == 0$$
$$z[0 \to 1] + m\, (g + L\, \text{Cos}[\theta[t]]\, \theta'[t]^2 + L\, \text{Sin}[\theta[t]]\, \theta''[t]) == 0$$
$$d[0 \to 1] == 0$$
$$e[0 \to 1] == 0$$
$$f[0 \to 1] == 0$$

A noter que par le calcul à la main nous obtenons la même chose.

■ Perspectives

Ce programme est notre version finale ; cependant plusieurs perspectives sont encore possibles. En premier lieu on pourra s'intéresser à d'autres types de liaisons "simples" fournies par le groupe qui travaille sur le projet MG04 (liaison pivot glissant, glissière...) puis à des liaisons présentes dans des systèmes plus complexes (liaison élastique, visqueuse, etc...). En second lieu on étudiera des systèmes mécaniques comportant plusieurs solides ainsi que plusieurs liaisons.

■ Bibliographie

[1] Ouisse M. Cours de mécanique générale, polycopié ENSMM, 2006
[2] Barrère R. Calcul scientifique avec Mathematica, Micro-projets de conception-modélisation-simulation, ENSMM, 2001
[3] Barrère R. Mathematica, calcul formel et programmation symbolique pour l'informatique scientifique, ENSMM, 2000