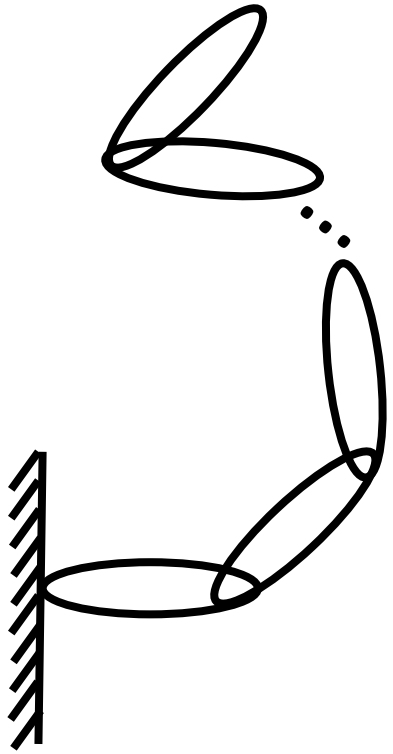
Simulation des systèmes de solides rigides polyarticulés

4- Résolution d'équations différentielles

Charles Pontonnier

Comportement dynamique d'un système de solides rigides polyarticulés



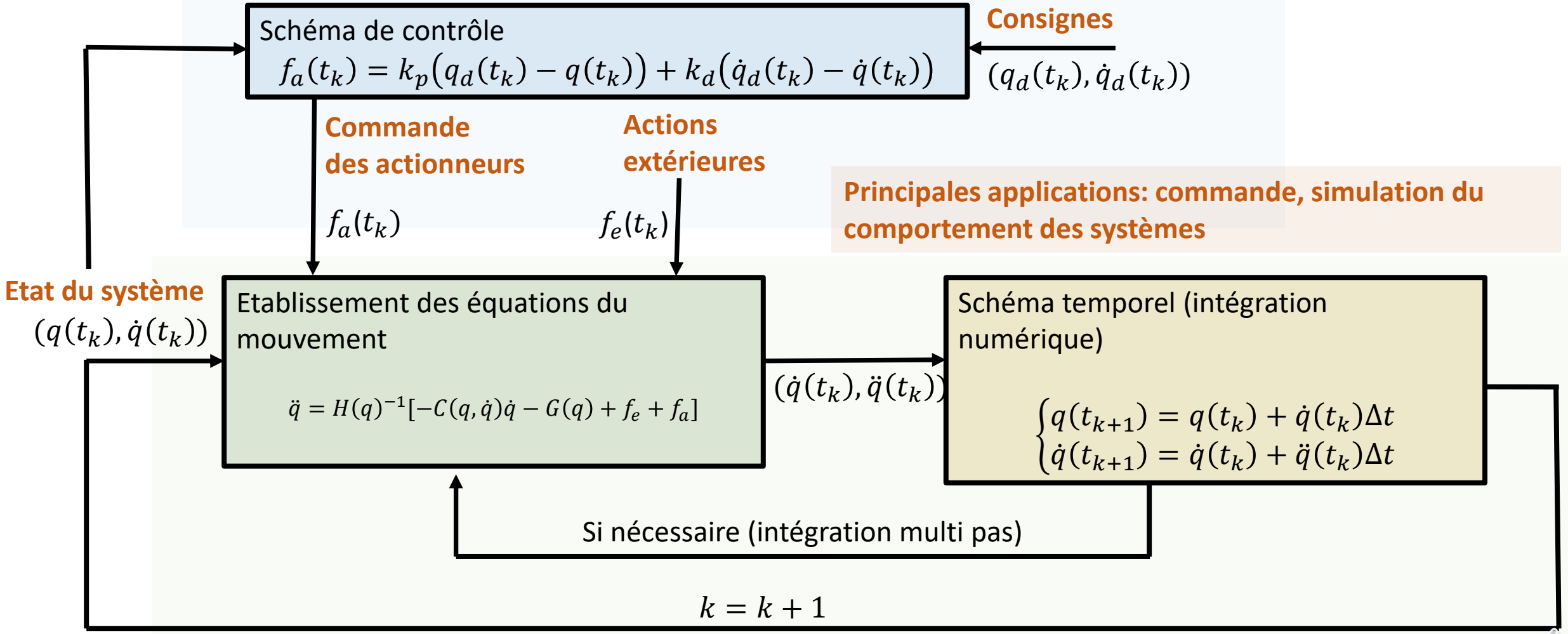
Soit un système de n_b
solides polyarticulés avec n_q liaisons

$$H(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + G(\boldsymbol{q}) = \boldsymbol{f_e} + \boldsymbol{f_a}$$

- $H(\boldsymbol{q})\ddot{\boldsymbol{q}}$: Matrice de masse
- $C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$: Coriolis/centrifuge
- $G(\boldsymbol{q})$: Gravité
- $\boldsymbol{f_e}$: Efforts extérieurs
- $\boldsymbol{f_a}$: Efforts des actionneurs

Les n_q équations peuvent prendre la forme suivante

Dynamique directe



Problématique: Intégration d'équations différentielles

x état du système à un instant t,
u commande de ce système
y sortie observée du système (les variables que l'on veut contrôler en d'autres termes).

$$\begin{cases} \dot{x} = f(x) + g(x)u \\ y = h(x) \end{cases}$$

→ 2 « challenges »: établissement des équations régissant l'évolution du système
→ Résolution de ces équations (schéma d'intégration temporelle)

Application à la dynamique des solides rigides

Nécessité d'intégrer $\ddot{\boldsymbol{q}}_{\boldsymbol{i}}(\boldsymbol{t}_{\boldsymbol{k}})$ afin de mettre à jour l'état du système mécanique $[\dot{q}(t_k), q(t_k)]$

Application à la dynamique des solides rigides

Les équations de la dynamique nous donnent un vecteur de n_q accélérations articulaires:

$$\ddot{q} = H(q)^{-1}[-C(q,\dot{q})\dot{q} - G(q) + f_e + f_a]$$ **Problème: équation du 2nd ordre !**

Ce qui peut être mis sous la forme d'un système d'équation différentielles du premier ordre en exploitant les dérivées successives de q, soit

$\boldsymbol{x = \begin{pmatrix} q \\ \dot{q} \end{pmatrix}}$ **vecteur de longueur** $\boldsymbol{2n_q}$

L'état du système peut être décrit à tout instant t_k par $\mathbf{x}(t_k) = [q(t_k), \dot{q}(t_k)]$

$$\begin{cases} \dot{x} = f(x) + g(x)u \\ y = h(x) \end{cases}$$

$$\dot{x} = \begin{pmatrix} \dot{q} \\ \ddot{q} \end{pmatrix} = \begin{bmatrix} x(n_{q+1}:2n_q) \\ \underbrace{H(q)^{-1}[-C(q,\dot{q})\dot{q} - G(q)]}_{f(x)} + \underbrace{H(q)^{-1}[f_e + f_a]}_{g(x)u} \end{bmatrix}$$

Méthodes d'euler

$$\begin{cases} \dot{x}_{t_k} = \frac{x_{t_{k+1}} - x_{t_k}}{\Delta t} + \partial(\Delta t) & \text{Différence avant d'ordre 1} \\ \dot{x}_{t_k} = \frac{x_{t_k} - x_{t_{k-1}}}{\Delta t} + \partial(\Delta t) & \text{Différence arrière d'ordre 1} \\ \dot{x}_{t_k} = \frac{x_{t_{k+1}} - x_{t_{k-1}}}{2\Delta t} + \partial(\Delta t^2) & \text{Différence centrée d'ordre 2} \end{cases}$$

Différences finies

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u}$$

$$x_{t_{k+1}} = x_{t_k} + \Delta t\left[f\left(x_{t_k}\right) + g\left(x_{t_k}\right)u_{t_k}\right] + \partial(\Delta t^2)$$ Euler explicite (+ très simple, - très instable)

$$x_{t_{k+1}} = x_{t_k} + \Delta t\left[f\left(x_{t_{k+1}}\right) + g\left(x_{t_{k+1}}\right)u_{t_{k+1}}\right] + \partial(\Delta t^2)$$

Euler implicite (+ inconditionnellement stable,
- compliquée à résoudre)

Peu adapté aux équations de la dynamique (principalement en raison de l'instabilité de l'Euler explicite qui supporte mal les grandes accélérations)

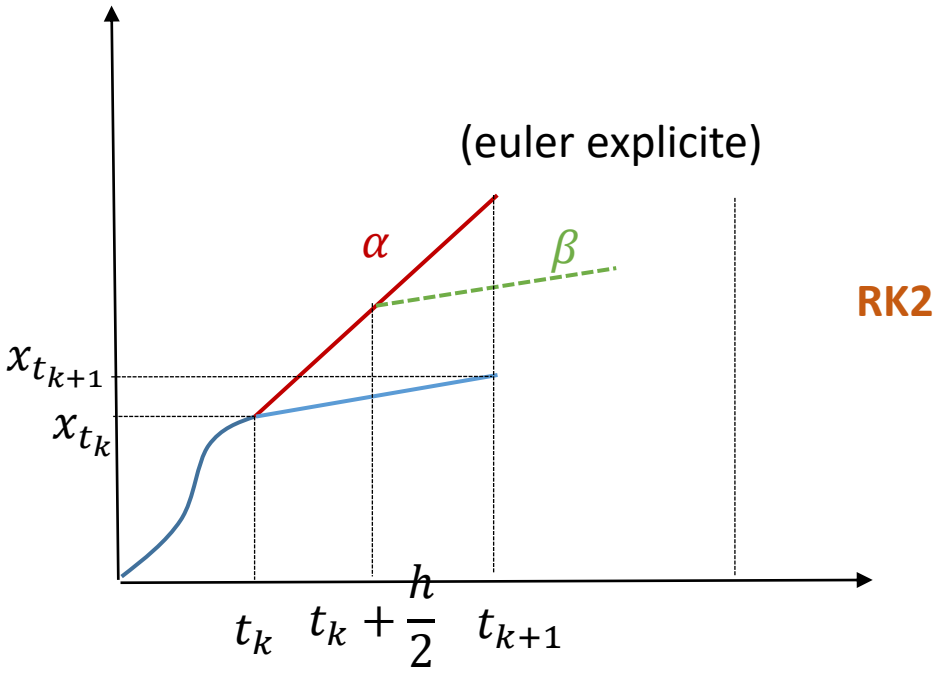
Méthodes à pas intermédiaire

La plus répandue: méthode de Runge-Kutta (d'ordre 2 ou 4)

Basée sur les estimées intermédiaires successives obtenues par itération

$$\dot{x} = f(x) + g(x)u$$

$$\begin{cases} \alpha = \Delta t\left[f\left(x_{t_k}\right) + g\left(x_{t_k}\right)u_{t_k}\right] \\ \beta = \Delta t\left[f\left(x_{t_k} + \alpha/2\right) + g\left(x_{t_k} + \alpha/2\right)u_{t_k+\Delta t/2}\right] \\ x_{t_{k+1}} = x_{t_k} + \beta + \partial(\Delta t^3) \end{cases}$$



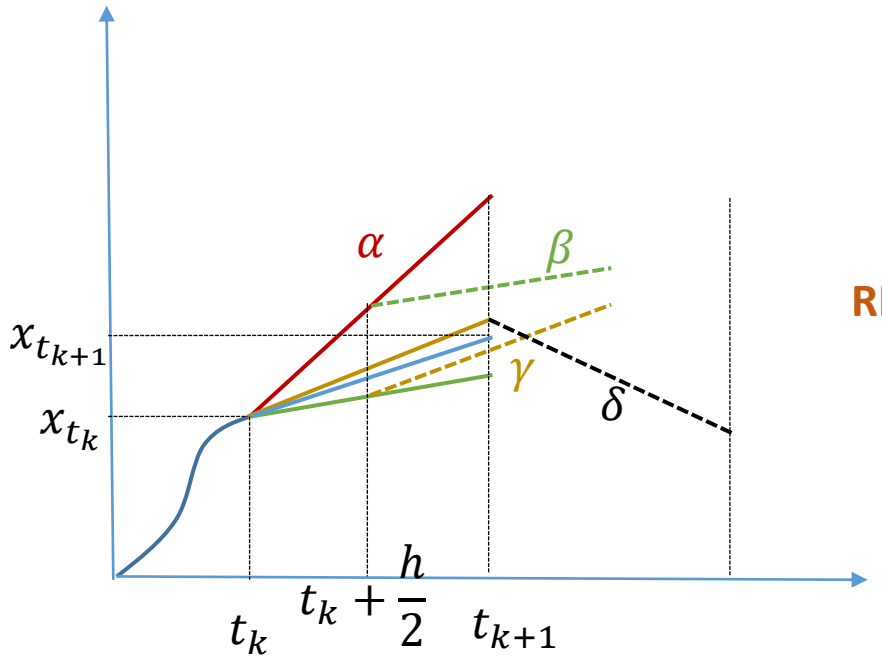
RK2

Méthodes à pas intermédiaire

La plus répandue: **méthode de Runge-Kutta** (d'ordre 2 ou 4)
Basée sur les estimées intermédiaires successives obtenues par itération

$$\dot{x} = f(x) + g(x)u$$

$$\begin{cases} \alpha = \Delta t\left[f\left(x_{t_k}\right) + g\left(x_{t_k}\right)u_{t_k}\right] \\ \beta = \Delta t\left[f\left(x_{t_k} + \alpha/2\right) + g\left(x_{t_k} + \alpha/2\right)u_{t_k+\Delta t/2}\right] \\ \gamma = \Delta t\left[f\left(x_{t_k} + \beta/2\right) + g\left(x_{t_k} + \beta/2\right)u_{t_k+\Delta t/2}\right] \\ \delta = \Delta t\left[f\left(x_{t_k} + \gamma\right) + g\left(x_{t_k} + \gamma\right)u_{t_k+\Delta t}\right] \\ x_{t_{k+1}} = x_{t_k} + \frac{1}{6}(\alpha + 2\beta + 2\gamma + \delta) + \partial(\Delta t^5) \end{cases}$$



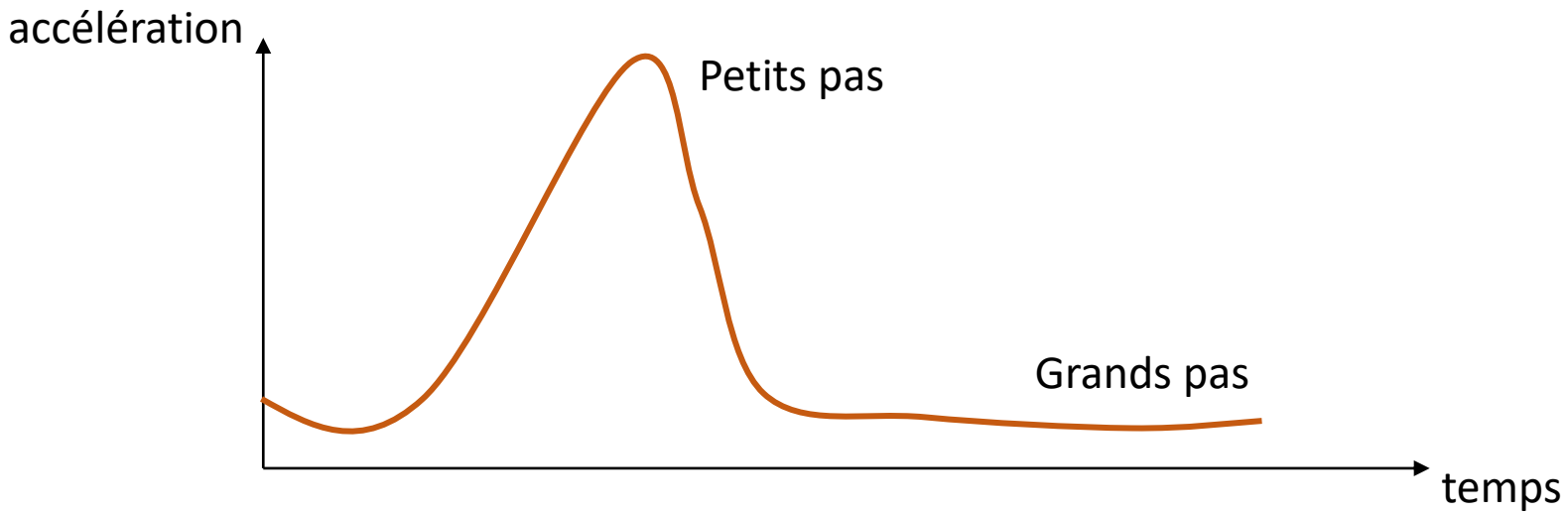
RK4

Beaucoup plus stable qu'Euler, mais nécessite un pas d'intégration petit pour être « bon » sur des systèmes « raides »

Méthodes à pas variable

Dynamique avec des accélérations très variables → intégrateurs à pas variable type “ode45” (Dormand-Prince)

Dynamique avec des systèmes raides → intégrateurs à pas variable et interpolation du pas de temps (multistep) type « ode15s » (intégrateur basé différences finies d’ordre 1 à 5)



Dormand-Prince est la méthode implémentée dans matlab sous le nom “ode45”

```
>> help ode45
 ODE45  Solve non-stiff differential equations, medium order method.
    [TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL]
    integrates the system of differential equations y' = f(t,y) from
    time T0 to TFINAL with initial conditions Y0.

    ODEFUN is a function handle. For a scalar T and a vector Y,
    ODEFUN(T,Y) must return a column vector corresponding to f(t,y).

    Each row in the solution array YOUT corresponds to a time
    returned in the column vector TOUT.
```

Un exemple

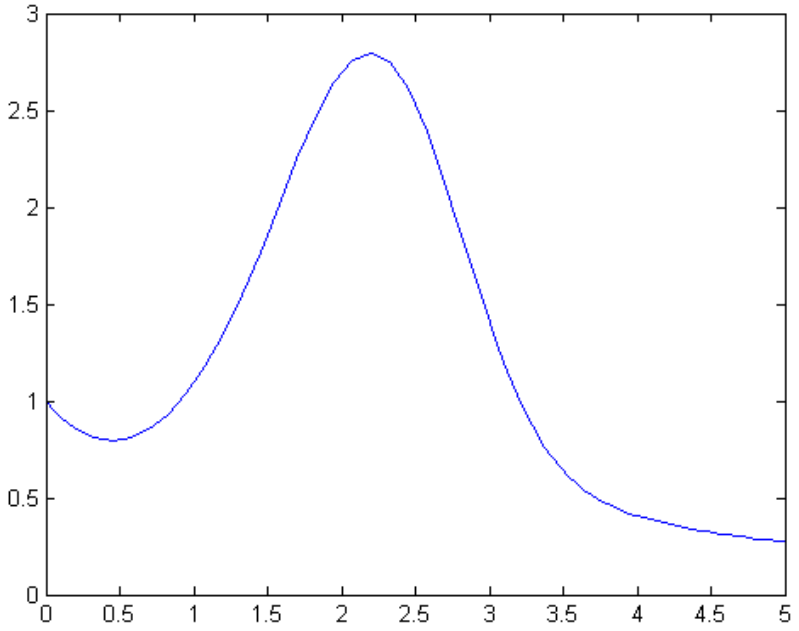
Pour l'equation différentielle $y^{(1)}(t) = y(t)(2-t)t + t - 1$

$y(0) = 1$

On peut appeler

```
function [dy] = f3a( t, y )
        dy = y*(2 - t)*t + t - 1;
end
```

```
>> [t3a, y3a] = ode45( @f3a, [0, 5], 1 );
>> plot( t3a, y3a );
```



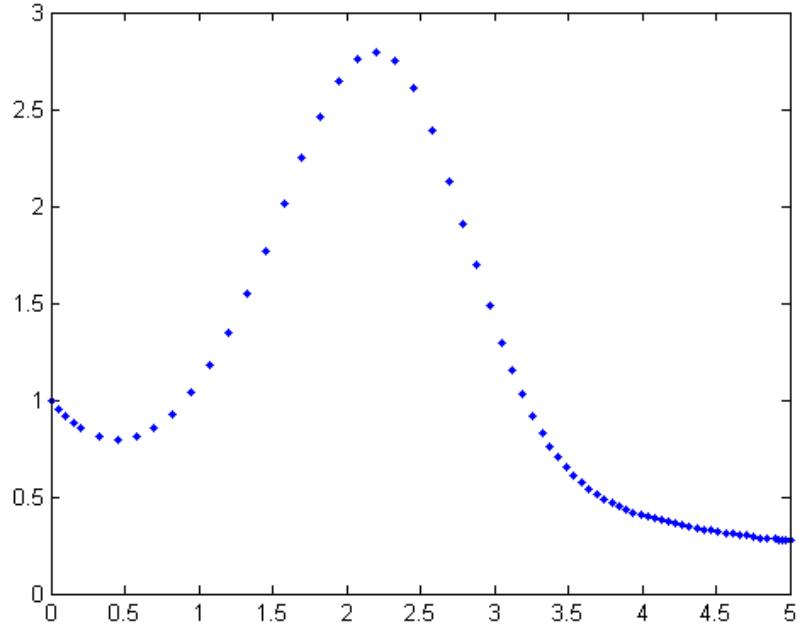
Intéressant

Si l'on trace la courbe solution

```
>> plot( t3a, y3a, '.' );
```

Les points sont plus serrés à droite

- Dormand-Prince est adaptative – elle adapte le pas d'integration à la fonction à intégrer



Méthodes adaptatives (à pas adaptatif)

Comment sait-on qu'il faut changer le pas d'intégration ?

On suppose que l'on a deux algorithmes (deux schémas d'intégration), dont l'un est supposé plus performant que l'autre

- Par exemple, Euler et runge Kutta 2
- Pour un point (t_k, y_k), on utilise les deux pour calculer le point suivant:

$$K_1 = f\left(t_k, y_k\right)$$

$$K_2 = f\left(t_k + h, y_k + hK_1\right)$$

$$y_{\text{tmp}} = y_k + hK_1$$ ← Euler

$$z_{\text{tmp}} = y_k + h\frac{K_1 + K_2}{2}$$ ← RK2

Exemple

Par exemple, pour la fonction suivante:

$$y^{(1)}(t) = -y(t)$$
$$y(0) = 1$$

On suppose que l'on veut approximer $y(0.1)$. On part avec un pas d'intégration initial de $h = 0.1$ et on veut que l'erreur finale sur l'intégration soie inférieure à $e_{abs} = 0.001$

$$K_1 = f(0,1) = -1$$
$$K_2 = f(0.1, 1 + 0.1K_1) = -0.9$$
$$y_{tmp} = 1 + 0.1(-1) = 0.9$$ ← Euler
$$z_{tmp} = 1 + 0.1\frac{(-1) + (-0.9)}{2} = 0.905$$ ← RK2

Exemple

Nous avons donc deux approximations

- La première est bonne, la deuxième est meilleure

$$y_{\text{tmp}} = 0.9$$
$$z_{\text{tmp}} = 0.905$$

La valeur théorique est de $e^{-0.1} = 0.9048374180\cdots$

- L'erreur théorique sur y_{tmp} est

$$\left|y_{\text{tmp}} - e^{-0.1}\right| = \left|0.9 - 0.9048374180\right| = 0.0048374180$$

- En utilisant z_{k+1}, l'erreur approximée est de

$$\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = \left|0.9 - 0.905\right| = 0.005$$

Exemple

On peut donc sans doute utiliser l'erreur $\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = 0.005$ comme une approximation de l'erreur théorique sur y_{tmp}

Cette erreur est plus grande que celle voulue e_{abs} = 0.001

→ on doit donc utiliser une valeur de h plus faible

Exemple

On sait que Euler a une erreur quadratique, de l'ordre O(h^2), soit

$$\left| y_{\text{tmp}} - z_{\text{tmp}} \right| = Ch^2$$

pour une certaine valeur de C

Si on diminue h à l'aide d'un facteur d'echelle s, l'erreur sera $C(sh)^2$:

$$C\left(sh\right)^2 < \varepsilon_{\text{abs}}$$

Exemple

Seulement, on veut que l'erreur e_{abs} soit obtenue à 0.1

Donc la contribution du pas *k* sur l'erreur doit être relative à la taille du pas par rapport à la longueur totale de l'intervalle d'intégration.

$$C(sh)^2 < \varepsilon_{\text{abs}} \frac{sh}{t_f - t_0}$$

- Pour être sur d'avoir une erreur maitrisée, on fait le choix

$$C(sh)^2 = \frac{1}{2}\varepsilon_{\text{abs}} \frac{sh}{t_f - t_0} = \frac{\varepsilon_{\text{abs}} sh}{2(t_f - t_0)}$$

Exemple

A partir des deux équations:

$$\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = Ch^2 \qquad C\left(sh\right)^2 = \frac{\varepsilon_{\text{abs}} sh}{2\left(t_f - t_0\right)}$$

On obtient alors :

$$Cs^2h^2 = \frac{\varepsilon_{\text{abs}} sh}{2\left(t_f - t_0\right)}$$

$$s\left(Ch^2\right) = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)}$$

Ce qui donne en substituant Ch^2:

$$s\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)}$$

Exemple

A partir de là:

$$s\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)}$$

On obtient alors le facteur d'échelle s

$$s = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)\left|y_{\text{tmp}} - z_{\text{tmp}}\right|}$$

Exemple

Dans cet exemple

$$h = 0.1$$
$$\varepsilon_{\text{abs}} = 0.001$$
$$\left| y_{\text{tmp}} - z_{\text{tmp}} \right| = 0.005$$
$$[t_0, t_f\,] = [0, 0.1]$$

Ce qui donne $$s = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)\left| y_{\text{tmp}} - z_{\text{tmp}} \right|} = \frac{0.001 \cdot 0.1}{2 \cdot 0.1 \cdot 0.005} = 0.1$$

Pour obtenir le bon résultat, il faut donc réduire le pas d'un facteur 10

Exemple

A présent, en utilisant $h = 0.01$, on a

$$K_1 = f(0,1) = -1$$

$$K_2 = f(0.01, 1 + 0.01 \cdot K_1) = -0.99$$

$$y_{\text{tmp}} = 1 + 0.01(-1) = 0.99 \quad \leftarrow \text{Euler}$$

$$z_{\text{tmp}} = 1 + 0.01\frac{(-1) + (-0.99)}{2} = 0.99005 \quad \leftarrow \text{RK2}$$

La vraie valeur en ce point est $e^{-0.031} = 0.9900498337\cdots$

- L'erreur absolue avec la méthode d'Euler est alors $0.0000498337\cdots$ ce qui est du même ordre que

$$\frac{\varepsilon_{\text{abs}} h}{2(t_f - t_0)} = 0.00005$$

- Utiliser y_{tmp} comme approximation de y_{k+1}

Garde-fou

Ce n'est pas toujours une bonne idée d'adapter le pas de temps

h = s*h;

Car s peut être très grand ou très petit.

Il est plus adapté d'être un peu conservatif, c'est à dire d'imaginer que h ne change pas trop à court terme.

- Si $s \geq 2$, doubler la valeur *h*
- If $1 \leq s < 2$, garder *h* inchangé, et
- If $s < 1$, diviser par deux la valeur de *h* et recalculer le point suivant

Dormand-Prince

Dormand-Prince calcule 7 pentes différentes

$$K_1, K_2, K_3, K_4, K_5, K_6, \text{ et } K_7$$

Ces pentes sont eploitées dans deux combinaisons linéaires différentes permettant d'approximer le point suivant

- La première est O(h^4) et la seconde O(h^5)
- Les coefficients de la solution O(h^5) sont choisis pour minimiser son erreur
- On exploite alors ces deux approximations pour estimer *s*:

$$s = \sqrt[4]{\frac{h\varepsilon_{\text{abs}}}{2\left(t_f - t_0\right)\left|y_{\text{tmp}} - z_{\text{tmp}}\right|}}$$

Tableau de Butcher

$\mathbf{c}$	$\mathbf{A}$
	$\mathbf{b}$

Représentation générique d'un schema d'integration multipas (le Coeur de l'algorithme est alors inchangé)

RK2

0		
1	1	
	$\frac{1}{2}$	$\frac{1}{2}$

RK4

0				
$\frac{1}{2}$	1			
$\frac{1}{2}$	0	1		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Implémentation RK4 avec le tableau de Butcher

```
A = [0 0 0 0
     1 0 0 0
     0 1 0 0
     0 0 1 0]';
c = [0 1/2 1/2 1]';
b = [1/6 1/3 1/3 1/6]';
```

$$K_1 = f\left(t_k, y_k\right)$$
$$K_2 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_1\right)$$
$$K_3 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_2\right)$$
$$K_4 = f\left(t_k + h, y_k + h \cdot K_3\right)$$

```
t0 = t_rng(1);
tf = t_rng(2);
h = (tf - t0)/(n - 1);
t_out = linspace( t0, tf, n );
y_out = zeros( 1, n );
y_out(1) = y0;
k_n = 4;

for k = 1:(n - 1)
    K = zeros( 1, k_n );

    for m = 1:k_n
        K(m) = f( t_out(k) + h*c(m), ...
                  y_out(k) + h*c(m)*K*A(:,m) );
    end

    y_out(k + 1) = y_out(k) + h*K*b;
end
```

$$y_{k+1} = y_k + h\left(\tfrac{1}{6}K_1 + \tfrac{1}{3}K_2 + \tfrac{1}{3}K_3 + \tfrac{1}{6}K_4\right)$$

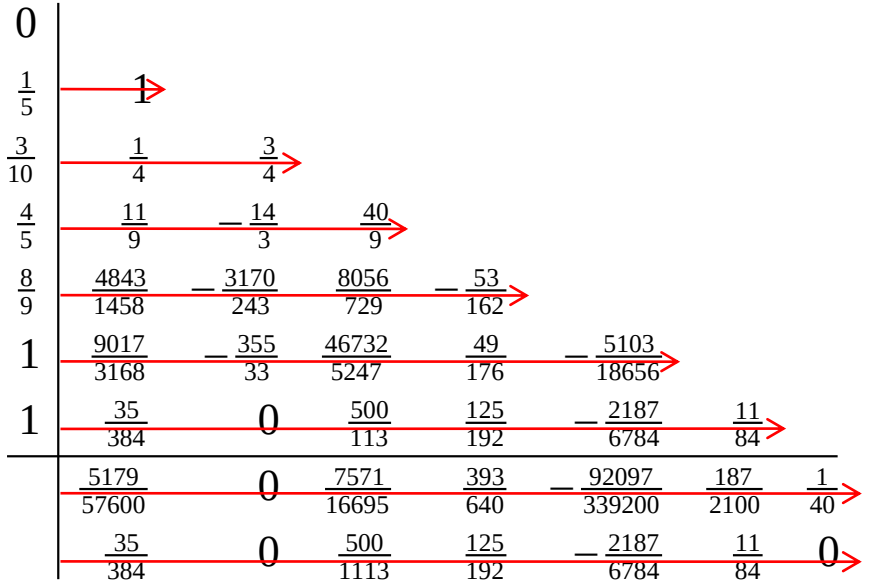
Le tableau de Butcher modifié de Dormand-Prince

0							
$\frac{1}{5}$	1						
$\frac{3}{10}$	$\frac{1}{4}$	$\frac{3}{4}$					
$\frac{4}{5}$	$\frac{11}{9}$	$-\frac{14}{3}$	$\frac{40}{9}$				
$\frac{8}{9}$	$\frac{4843}{1458}$	$-\frac{3170}{243}$	$\frac{8056}{729}$	$-\frac{53}{162}$			
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0

$O(h^4)$

$O(h^5)$

Chaque ligne donne 1



En Matlab…

```
A = [ 0           0          0         0          0         0   0;
      1           0          0         0          0         0   0;
     1/4        3/4          0         0          0         0   0;
    11/9      -14/3       40/9         0          0         0   0;
  4843/1458 -3170/243  8056/729  -53/162          0         0   0;
  9017/3168  -355/33  46732/5247  49/176 -5103/18656    0   0;
    35/384         0     500/1113 125/192 -2187/6784  11/84 0]';

by = [5179/57600 0 7571/16695 393/640 -92097/339200 187/2100 1/40]';
bz = [  35/384   0  500/1113  125/192  -2187/6784    11/84    0]';

c = [0 1/5 3/10 4/5 8/9 1 1]';
```

Avec le code…

```
A = [...]';
c = [...]';
by = [...]';
bz = [...]';

// ...

n_K = 7;
K = zeros( 1, n_K );

for m = 1:n_K
    K(m) = f( t_out(k) + h*c(m), ...
              y_out(k) + h*c(m)*K*A(:,m) );
end

y_tmp = y_out(k) + h*K*by;
z_tmp = y_out(k) + h*K*bz;

% Determine s and modify h as appropriate
```

Usage

Quelle valeur pour *h*?

- Dans les méthodes type RK4, on définit le pas d'intégration, l'intervalle et le nombre de points

Pour Dormand-Prince, on donne une valeur initiale à *h*

- ode45 adapte cette première valeur au problème *h*

On ne sait pas combien de points seront nécessaires pour faire le calcul sur l'intervalle

```
% Initialize t_out and y_out
      % Initialize our location to k = 1
      %
      % while t_out(k) < tf
      %     Use Dormand Prince to find two approximations
      %     y_tmp and z_tmp to approximate y(t) at
      %     t = t_out(k) + h for the current value of h
      %
      %     Calculate the scaling factor 's'
      %
      %     if s >= 2,
      %         We use z_tmp to approximate y_out(k + 1)
      %         t_out(k + 1) is the previous t-value plus h
      %         Increment k and double the value of h for the
      %         next iteration.
```

```
        %     else if s >= 1,
%          We use z_tmp to approximate y_out(k + 1)
%          t_out(k + 1) is the previous t-value plus h
%          In this case, h is neither too large or too
%          small, so only increment k
%     else s < 1
%          Divide h by two and try again with the smaller
%          value of h (just go through the loop again
%          without updating t_out, y_out, or k)
%     end
%
%     We must make one final check before we end the loop:
%        if t_out(k) + h > tf, we must reduce the
%        size of h so that t_out(k) + h == tf
% end
```

Un essai !

```
[TOUT,YOUT] = ode45(@(t,Q) FD_planar_arm_modifie(t,Q,specs,Fext,tt,Tau,g),[0 tf],qfd_ini,options);
```

```
function dQ=FD_planar_arm_modifie(t,Q,specs,Fext,tt,Tau,g)
[…]

t1=interp1(tt,Tau(1,:),t);
t2=interp1(tt,Tau(2,:),t);

dQ=[…];

end
```

Comme le pas est variable, il est nécessaire d'interpoler la valeur du couple à chaque évaluation de la fonction « FD_planar_arm » !

Synthèse

+ Méthodes de description génériques des systèmes de solides rigides polyarticulés (et graphes associés)

+ Méthodes de résolution des équations de la dynamique (globales ou récursives)

+ Méthodes d'intégration d'équations différentielles

+ Méthodes adaptatives

Simulation des systèmes de solides rigides polyarticulés