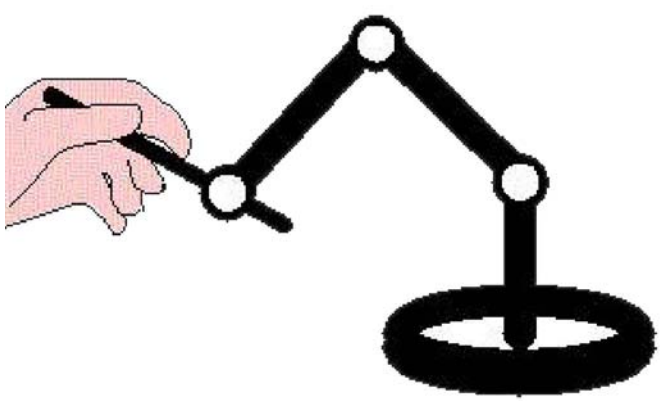
KYUSHU UNIVERSITY 100th 2011
知の新世紀を拓く

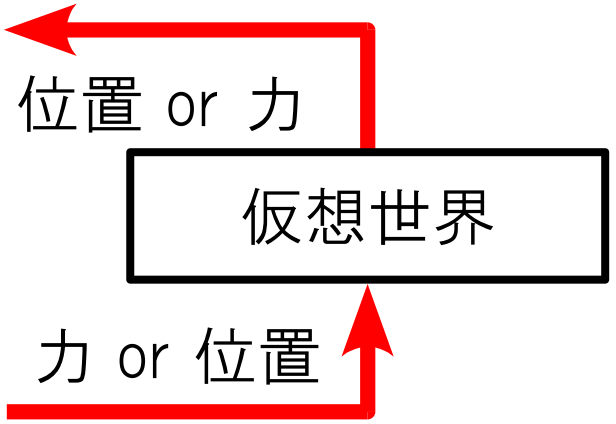
力覚提示における物理モデル，状態方程式，そして演算手順の設計

九州大学大学院
工学研究院機械工学部門
きく　うえ　　りょう
菊　植　亮
http://rk.mech.kyushu-u.ac.jp/

九州大学

力覚提示とは？





- 使用者とデバイスとの境界面において，なんらかの位置と力の関係を実現すること.
- 位置と力の関係は，仮想世界の実時間シミュレーションで決定される.
- 力覚提示は実時間シミュレーションの出入り口.

本日の内容

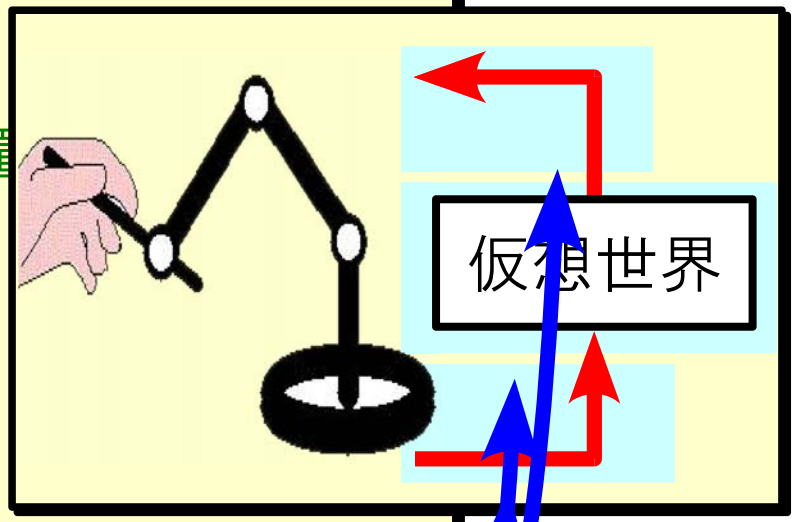
- アルゴリズムと状態方程式
- 2つのアプローチ:『陽』vs『陰』
- 『陽』に関連して: ペナルティベースの方法
- 『陰』に関連して: プロクシを用いた方法

＊ 数式が多いですが，軽い気分で見てください．

アルゴリズム
と
状態方程式

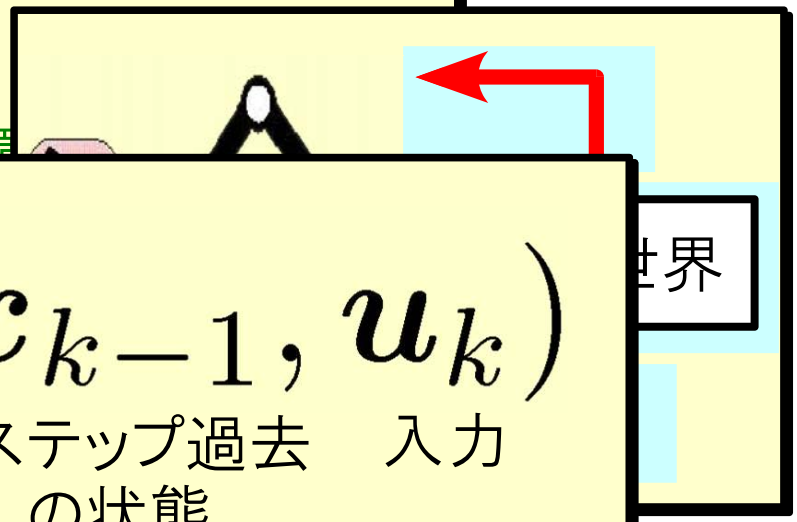
実時間シミュレーションの基本形

```
struct State {…};
struct Input {…};
State vecX; /* 状態ベクトル(位置
Input vecU; /* 入力ベクトル */
  :
  :
for(;;)
{
    waitForTimerTick();
     /* ある時間(1～30ms)の経過を待つ*/
    vecU = readInput();
      /* 入力(センサ値の読込) */
    vecX = update(vecX,vecU);
      /* 計算(仮想世界の「状態」の更新) */
    writeOutput(vecX);
      /* 出力(デバイスへの値の書出) */
}
```



実時間シミュレーションの基本形

```
struct State {…};
struct Input {…};
State vecX; /* 状態ベクトル(位置
Input
  :
for(;
{
    wa
      /*
    ve
      /
    ve
      /
    wr
      /
}
```



$$\boldsymbol{x}_k \leftarrow \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$$

現在の状態　　1ステップ過去の状態　　入力

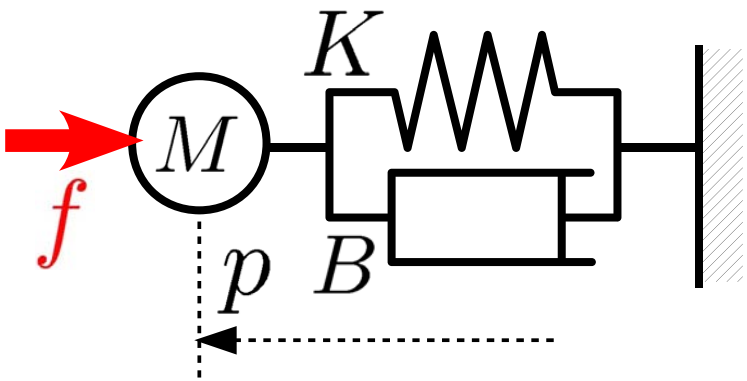
```
State update(State vecX, Input vecU)
{
      : /* 何らかの計算 */
  return newVecX;
}
```

シミュレーション：現実世界の現象の模倣

- 現実の物理法則は連続時間の微分方程式（状態方程式）で表現される.

$$M\ddot{p} + B\dot{p} + Kp = f$$

$$\Rightarrow \quad \dot{\boldsymbol{x}} = \begin{bmatrix} -B/M & -K/M \\ 1 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1/M \\ 0 \end{bmatrix} u$$

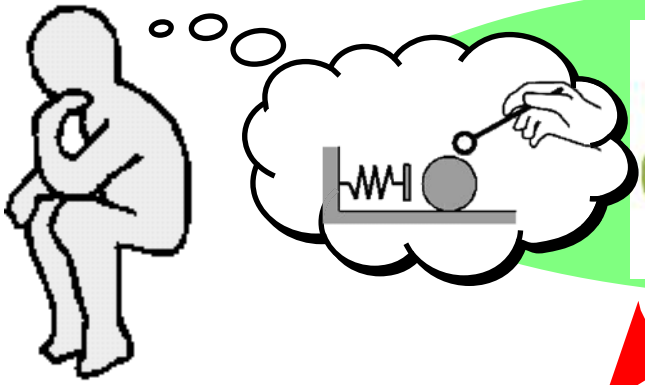


ただし $\boldsymbol{x} = \begin{bmatrix} \dot{p} \\ p \end{bmatrix}$

$$\Rightarrow \quad \dot{\boldsymbol{x}} = \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{u})$$

状態の時間変化率　　現在の状態　　現在の入力

状態方程式とアルゴリズム



仮想世界の
イメージと
物理法則

変換
（陽vs陰）

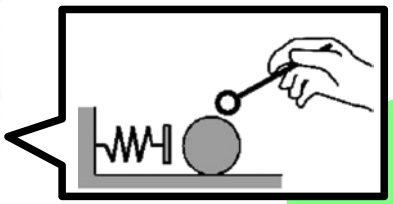
状態方程式
（連続時間表現）

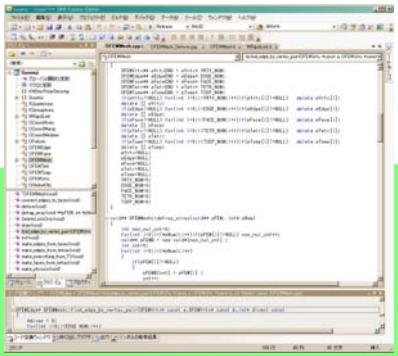
$$\dot{x} = \Phi(x, u)$$

意味づけ

アルゴリズム
（離散時間表現）

$$x_k \leftarrow \tilde{\Phi}(x_{k-1}, u_k)$$





プログラム

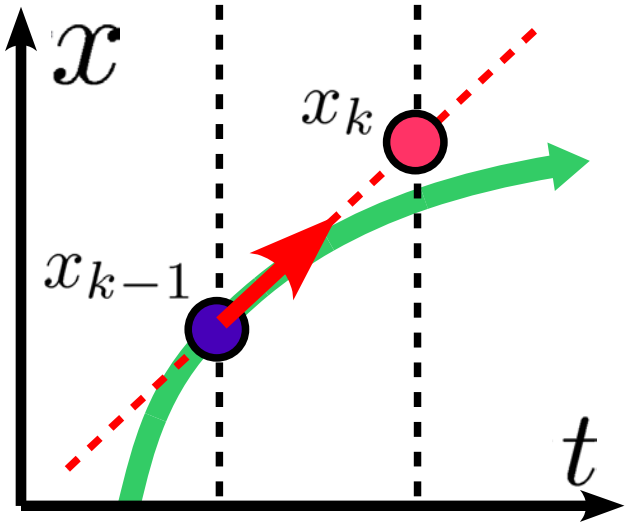
陽解法と陰解法

基本：状態方程式→アルゴリズム

$$\dot{x} = \Phi(x, u)$$

$$\frac{x_k - x_{k-1}}{T} = \Phi(x_{k-1}, u_k)$$

$$x_k \leftarrow x_{k-1} + T\Phi(x_{k-1}, u_k)$$




```
for(;;)
{
    waitForTimerTick();
    vecU = readInput();
    vecX = vecX + T* funcPhi(vecX,vecU);
    writeOutput(vecX);
}
```

連続時間の関数そのまま

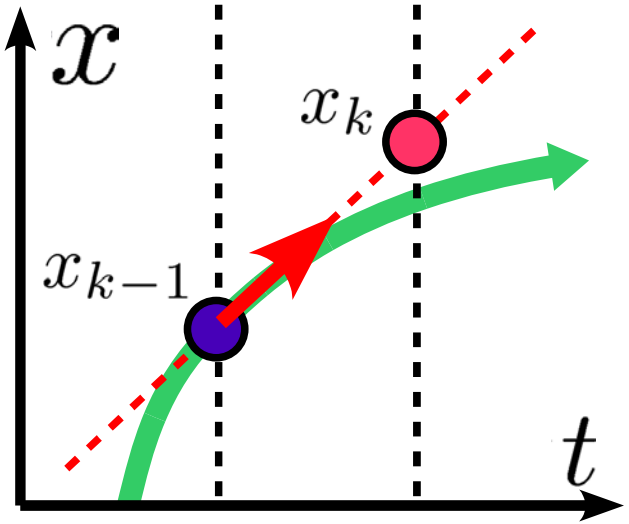
- これは「陽解法」と呼ばれる積分法

陽解法と陰解法

- 陽解法(陽的積分)
  - 例:前進(陽的)オイラー法

$$\frac{\boldsymbol{x}_k - \boldsymbol{x}_{k-1}}{T} = \boldsymbol{\Phi}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$$

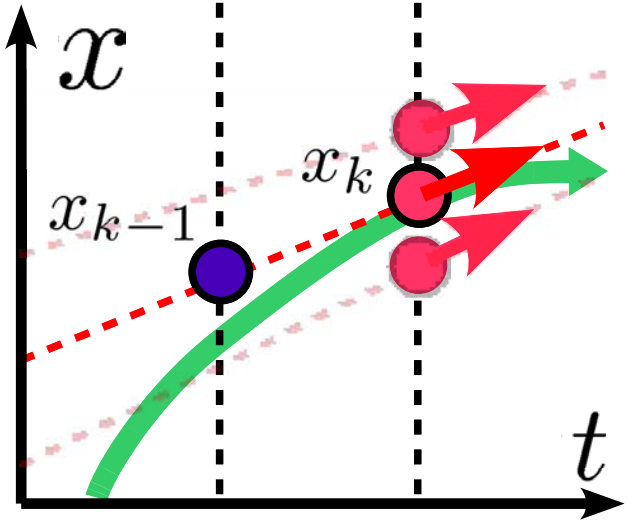
$$\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k-1} + T\boldsymbol{\Phi}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$$



- 陰解法(陰的積分)
  - 例:後退(陰的)オイラー法

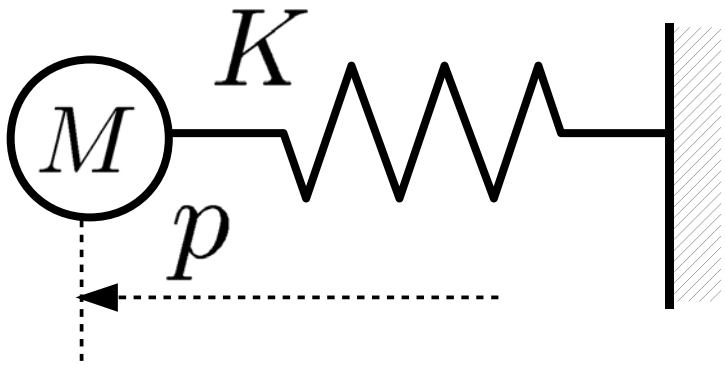
$$\frac{\boldsymbol{x}_k - \boldsymbol{x}_{k-1}}{T} = \boldsymbol{\Phi}(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

$\boldsymbol{x}_k \leftarrow$ 上式の解 (数値解 or 解析解)



比較：陽解法 vs 陰解法

- 例：バネ・質量系



$$M\ddot{p} + Kp = 0$$

$$\frac{d}{dt}\begin{bmatrix} \dot{p} \\ p \end{bmatrix} = \begin{bmatrix} 0 & -K/M \\ 1 & 0 \end{bmatrix}\begin{bmatrix} \dot{p} \\ p \end{bmatrix}$$

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$$

- 陽解法

$$\frac{\boldsymbol{x}_k - \boldsymbol{x}_{k-1}}{T} = \boldsymbol{A}\boldsymbol{x}_{k-1}$$

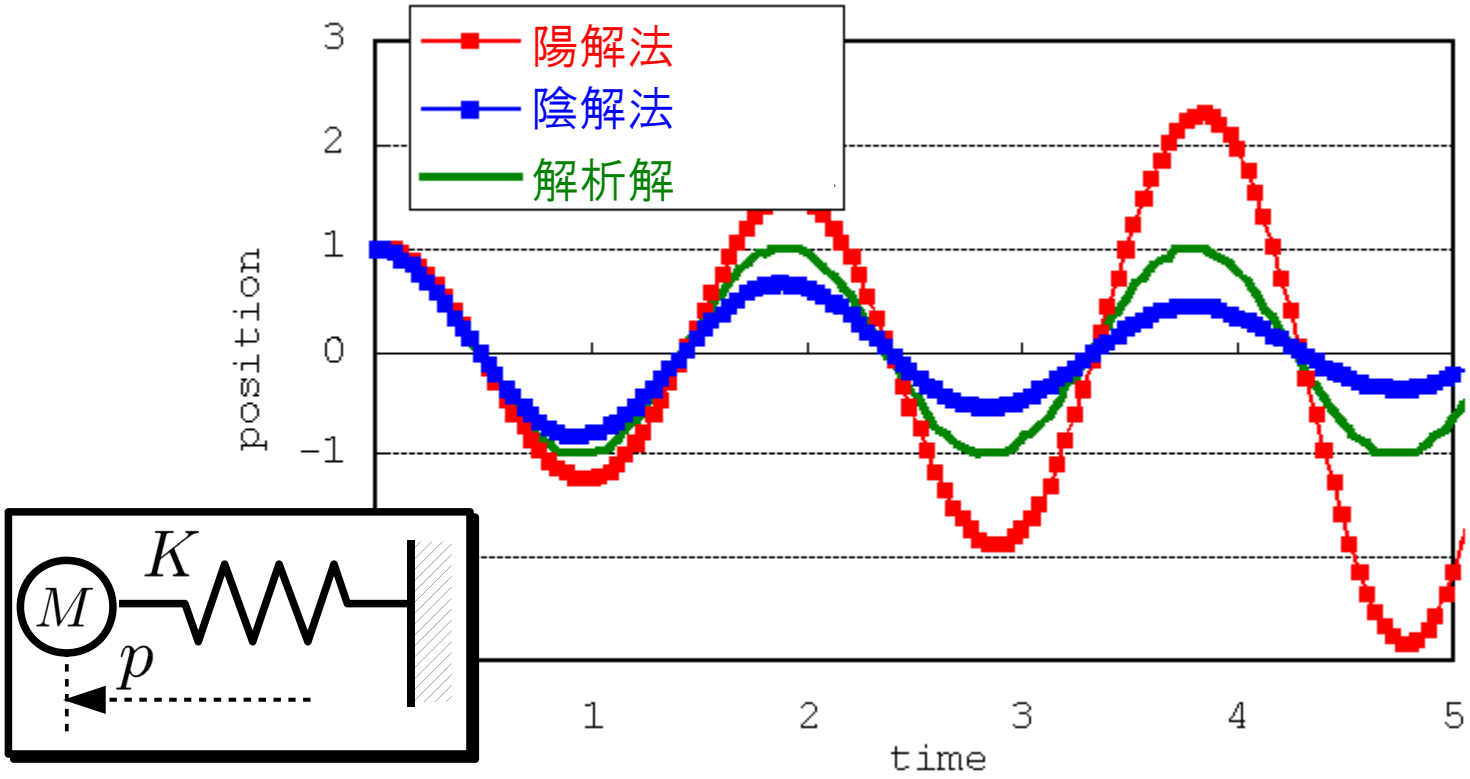
$$\boldsymbol{x}_k \leftarrow (\boldsymbol{I} + T\boldsymbol{A})\boldsymbol{x}_{k-1}$$

- 陰解法

$$\frac{\boldsymbol{x}_k - \boldsymbol{x}_{k-1}}{T} = \boldsymbol{A}\boldsymbol{x}_k$$

$$\boldsymbol{x}_k \leftarrow (\boldsymbol{I} - T\boldsymbol{A})^{-1}\boldsymbol{x}_{k-1}$$

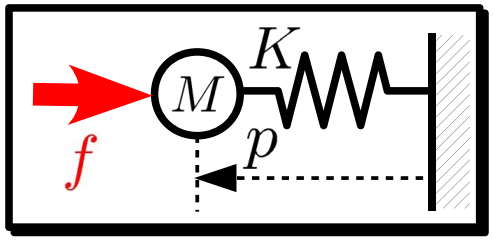
比較：陽解法 vs 陰解法（続き）



- 陽解法→発散，陰解法→収束．どちらも不正確．
- しかし，陰的のほうが安定 → 実用上好都合．
- “Numerical Dissipation”：アリかナシか？

プログラムの分かりやすさ：陽vs陰

$$M\ddot{p} + Kp = f$$



- 陽解法
  - 「力の計算」と「質点の運動方程式」が分離：読みやすい

$$\begin{aligned} g_k &\leftarrow -Kp_{k-1} + f_k && \text{力の算出} \\ v_k &\leftarrow v_{k-1} + Tg_k/M && \text{速度の更新} \\ p_k &\leftarrow p_{k-1} + Tv_{k-1} && \text{位置の更新} \end{aligned}$$

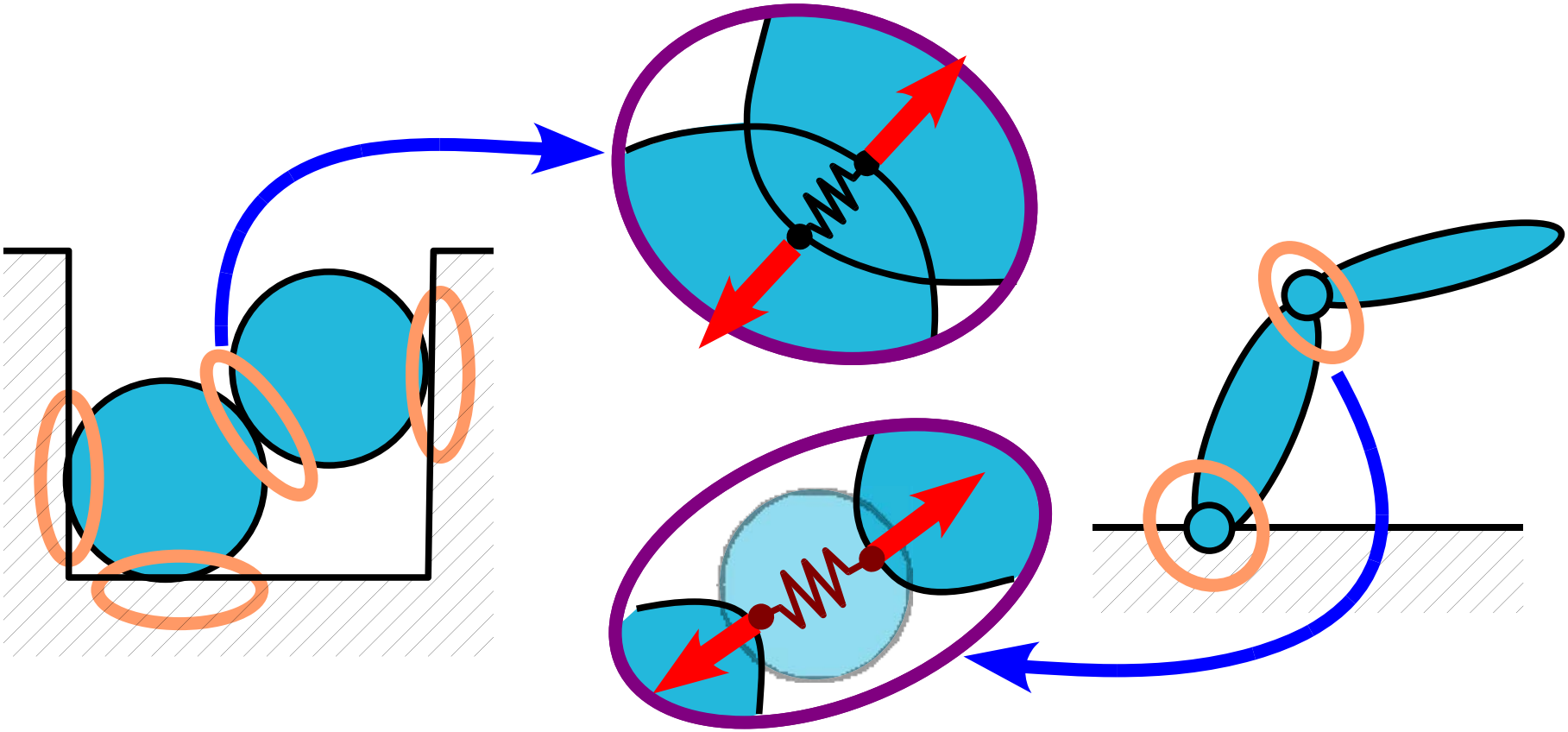
- 陰解法
  - 連立方程式を解析的に解いた後の式：意味不明

$$\begin{aligned} g_k &\leftarrow -Kp_{k-1} + f_k && \text{力の算出} \\ v_k &\leftarrow (Mv_{k-1} + Tg_k)/(M + T^2K) && \text{速度の更新} \\ p_k &\leftarrow (Mp_{k-1} + MTv_{k-1})/(M + T^2K) && \text{位置の更新} \end{aligned}$$

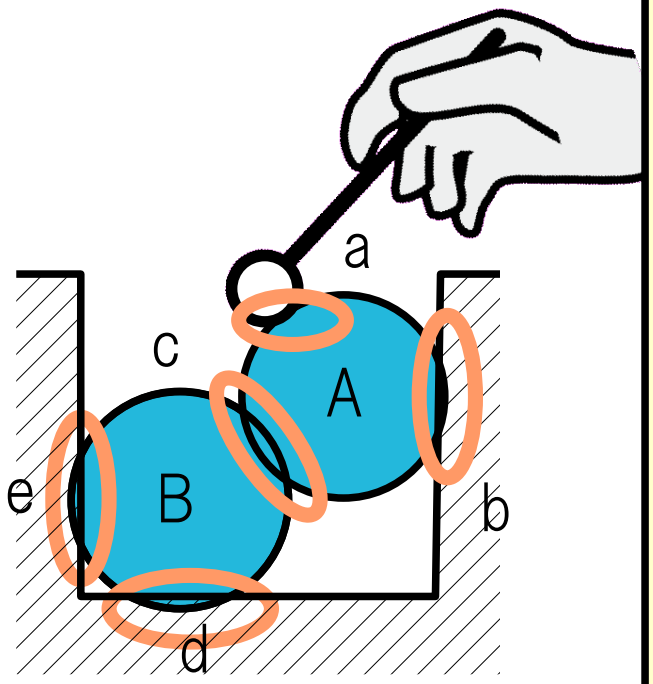
ペナルティベースの方法（＋陽解法）

複雑なシステムのシミュレーション

- 簡易な方法:「ペナルティベース」の方法
  - 仮想的なバネダンパを考える.
  - 禁じられている状態(めり込み等)を力によって「罰する」

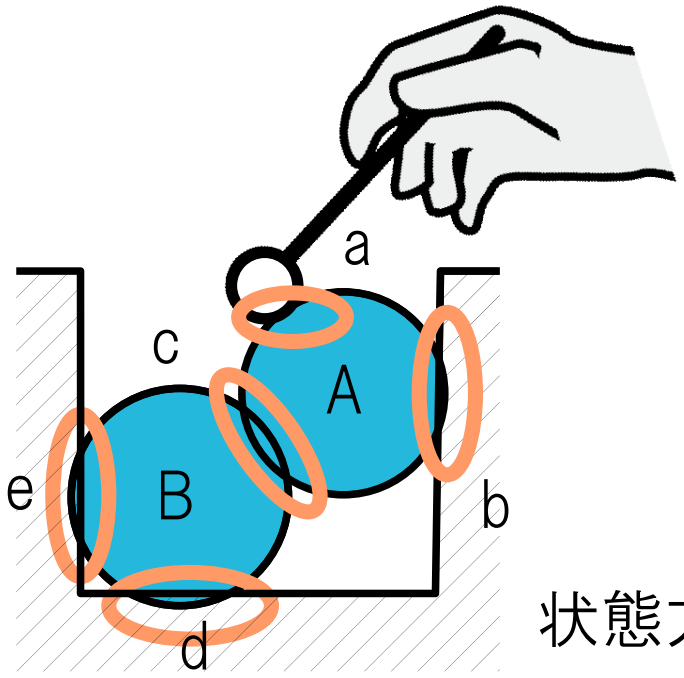


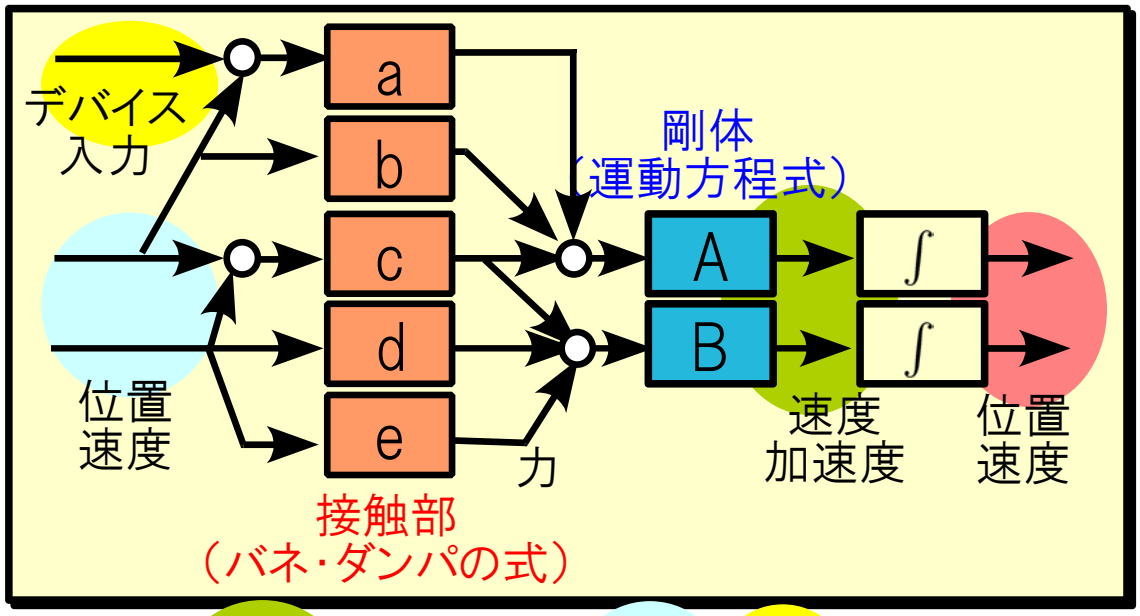
ペナルティベースの方法




```
/*力の算出*/
frc1 = 接触部aの力;
frc2 = 接触部bの力;
frc3 = 接触部cの力;
frc4 = 接触部dの力;
frc5 = 接触部eの力;
/*速度の算出*/
vel1 += T*(frc1+frc2-frc3)/M1;
vel2 += T*(frc4+frc5+frc3)/M2;
/*位置の算出*/
pos1 += T* vel1;
pos2 += T* vel2;
```

ペナルティベースの方法





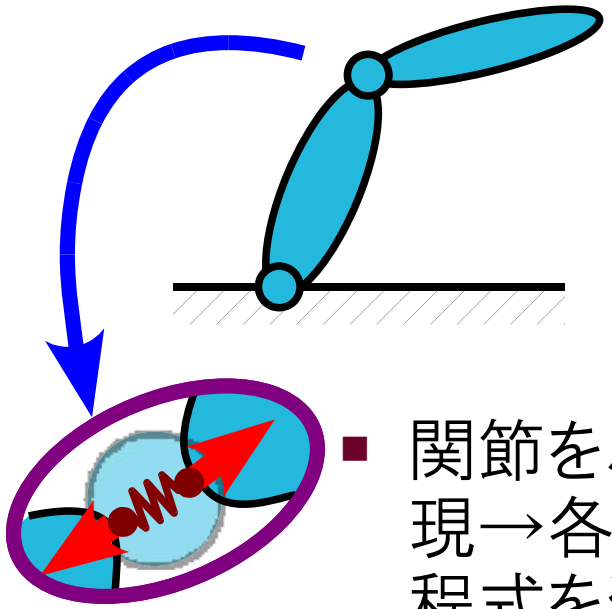
状態方程式：$\dot{x} = \Phi(x, u)$

アルゴリズム：$x_k \leftarrow \tilde{\Phi}(x_{k-1}, u_k)$

- 系全体の状態方程式を意識しなくても，アルゴリズムが書ける. [Kikuuwe&Yamamoto, IROS08]
  - プログラムを「モジュール化」できる.

ペナルティベースの方法は何にでも使える

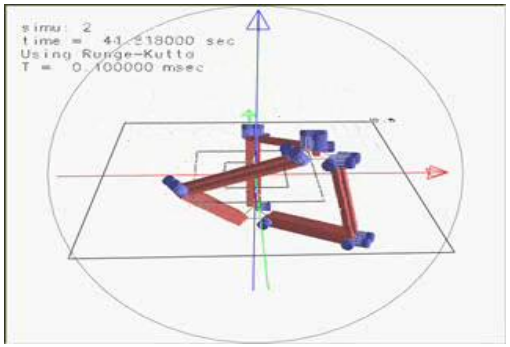
- 例：剛体リンク機構
  - 動力学の式を書き下すととても複雑！



$$\tau_1 = [\phi_{17} + \phi_{21} l_1^2 + \phi_{27} + 2 l_1 (C_2\phi_{22} - S_2\phi_{23})]\ddot{\theta}_1$$
$$+ [\phi_{27} + l_1 (C_2\phi_{22} - S_2\phi_{23})]\ddot{\theta}_2$$
$$- l_1 (S_2\phi_{22} + C_2\phi_{23})(\dot{\theta}_2^2 + 2\dot{\theta}_1\dot{\theta}_2)$$
$$+ \hat{g}(C_1\phi_{12} - S_1\phi_{13} + l_1 C_1\phi_{21} + C_{12}\phi_{22} - S_{12}\phi_{23})$$
$$\tau_2 = [\phi_{27} + l_1 (C_2\phi_{22} - S_2\phi_{23})]\ddot{\theta}_1 + \phi_{27}\ddot{\theta}_2$$
$$+ l_1 (S_2\phi_{22} + C_2\phi_{23})\dot{\theta}_1^2 + \hat{g}(C_{12}\phi_{22} - S_{12}\phi_{23})$$

- 関節をバネ・ダンパで表現→各リンクの運動方程式を独立して計算できる.

10リンクのパラレルリンク系



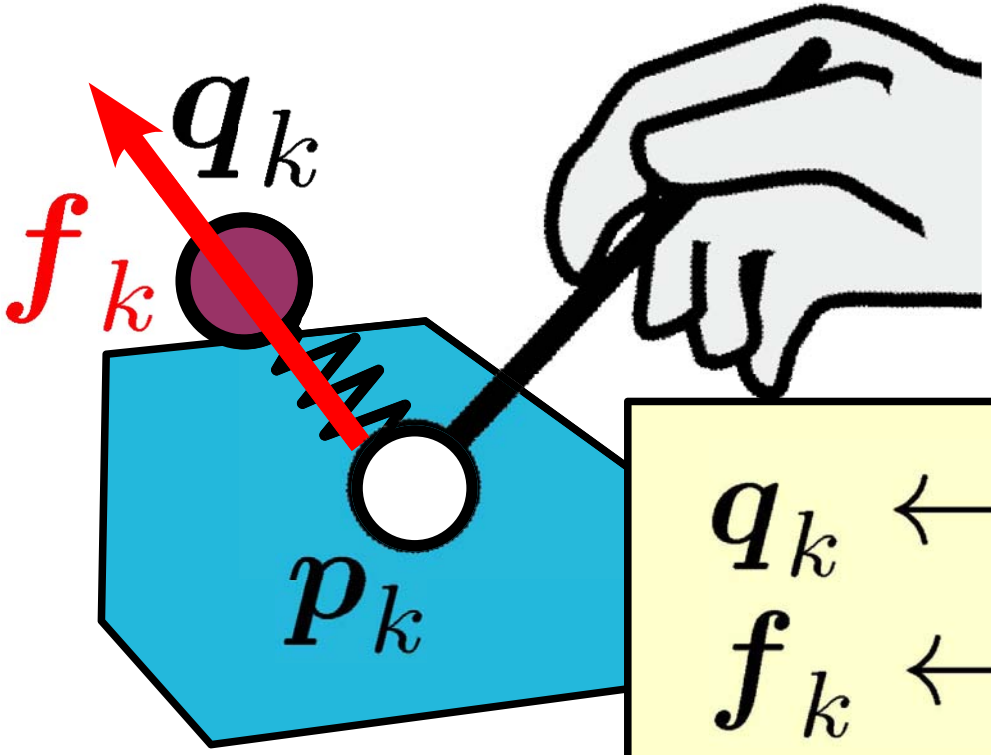
- 欠点：不安定になりやすい
  - 陽解法→ バネを硬くすると発散.

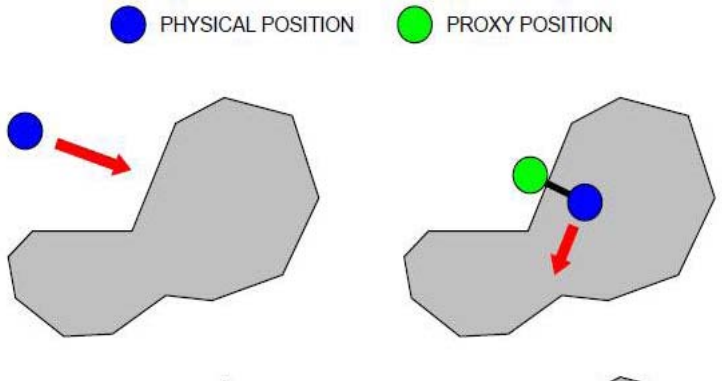
プロクシを使った力覚提示（＋陰解法）

状態方程式をあまり意識しない力覚提示プログラム構築

- 例:「プロクシ」(参照点)を用いた力覚提示
  - プロクシの位置は，基本的に**幾何学**で決める.

[Ruspini et al., SIGGRAPH97]

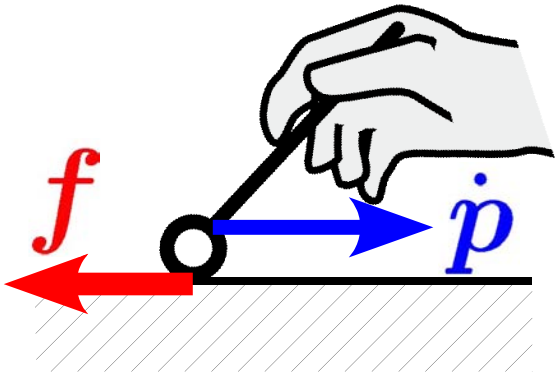


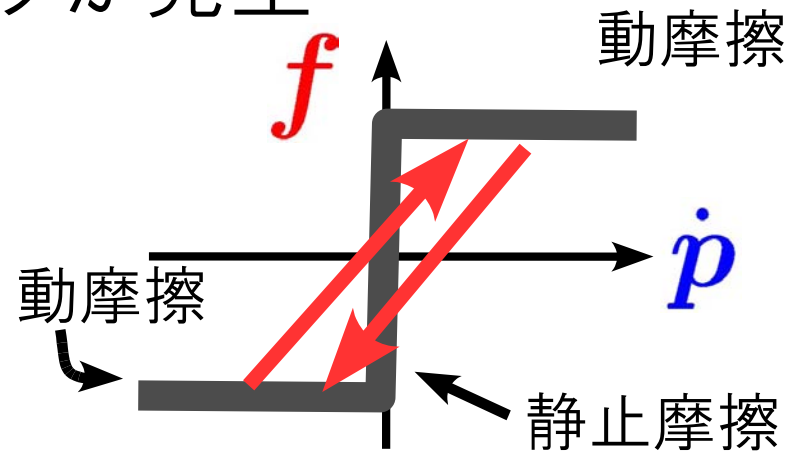


$$q_k \leftarrow \mathcal{G}(q_{k-1}, p_k)$$
$$f_k \leftarrow K(q_k - p_k)$$

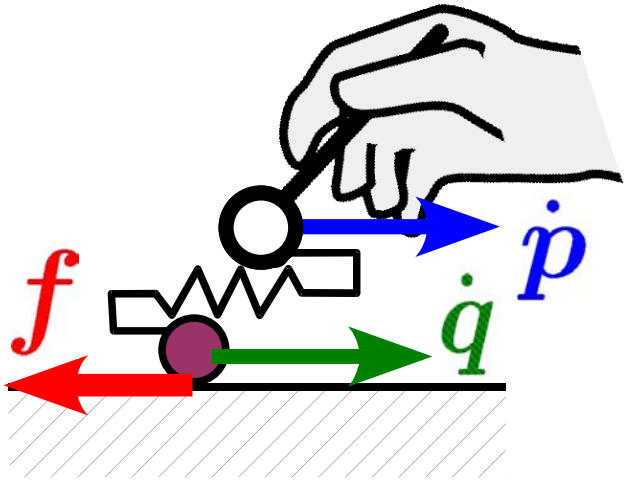
摩擦力も提示できる.

- プロクシなし：チャタリングが発生





- プロクシあり



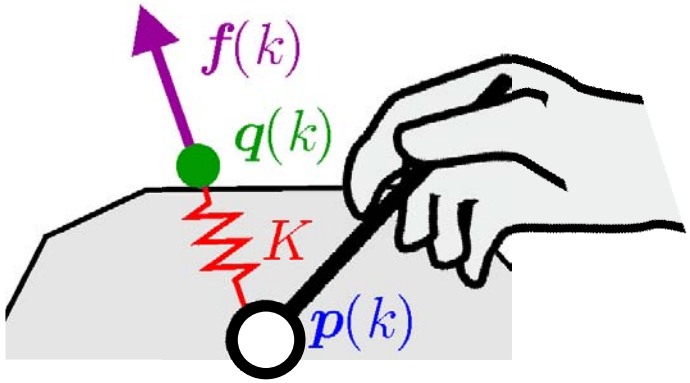
$$\boldsymbol{q}_k \leftarrow \begin{cases} \boldsymbol{q}_{k-1} & \text{if } \|\boldsymbol{p}_k - q_{k-1}\| < \dfrac{F}{K} \\ \boldsymbol{p}_k - \dfrac{F}{K}\dfrac{\boldsymbol{p}_k - \boldsymbol{q}_{k-1}}{\|\boldsymbol{p}_k - \boldsymbol{q}_{k-1}\|} & \text{otherwise} \end{cases}$$

- プロクシが，デバイスに一定距離を保ってついていくようにプロクシ位置を更新.

[Hayward & Armstrong, 2000]
[Hasegawa et al. 2003]

力学的にはどういう意味？

[菊植・藤本, JRSJ2007]



- アルゴリズム

$$q_k \leftarrow \mathcal{G}(q_{k-1}, p_k)$$
$$f_k \leftarrow K(q_k - p_k)$$

代数的に等価

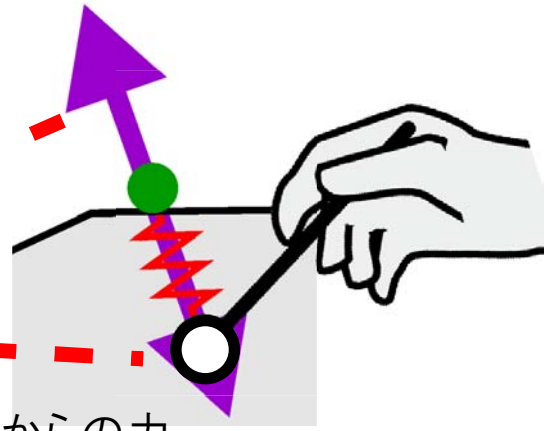
あらたな関数 $\mathcal{M}$ を定義

$$\forall \kappa > 0, \forall \boldsymbol{q} \in \mathbb{R}^n, \forall \boldsymbol{v} \in \mathbb{R}^n, \forall \boldsymbol{f} \in \mathbb{R}^n,$$
$$\boldsymbol{f} = \mathcal{M}(\boldsymbol{q}, \boldsymbol{v}) \iff \boldsymbol{q} = \mathcal{G}\left(\kappa, \boldsymbol{q} - \frac{\boldsymbol{f}}{\kappa}, \boldsymbol{q} - T\boldsymbol{v}\right)$$

- 力の釣り合いの式

$$f_k = \mathcal{M}(q_k, (q_k - q_{k-1})/T)$$
$$f_k = K(q_k - p_k)$$

環境からの力



仮想バネからの力

アルゴリズムから状態方程式へ

陰解法

$$q_k \leftarrow \mathcal{G}(q_{k-1}, p_k)$$
$$f_k \leftarrow K(q_k - p_k)$$

プロクシ位置更新のアルゴリズム

$$f = \mathcal{M}(q, \dot{q})$$
$$f = K(q - p)$$

fを消去

力の釣り合いの式

$$\mathcal{M}(q, \dot{q}) - K(q - p) = 0$$

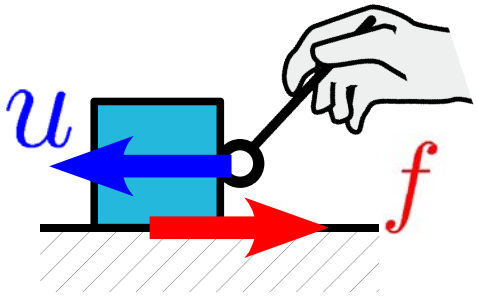
状態　状態の時間変化率　状態　入力

陰関数で表される状態方程式

状態方程式はしばしば不連続になる

状態方程式 $\dot{x} = \Phi(x, u)$ の右辺が

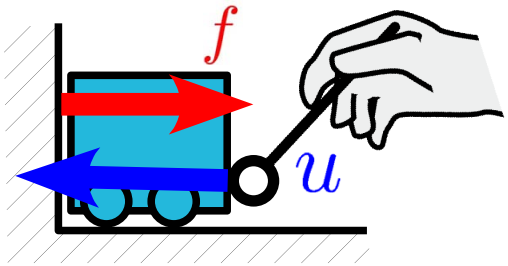
- 不連続の場合
  - クーロン摩擦力



$$\begin{bmatrix} \dot{v} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} (-f+u)/M \\ v \end{bmatrix}$$

$$f = \quad v$$

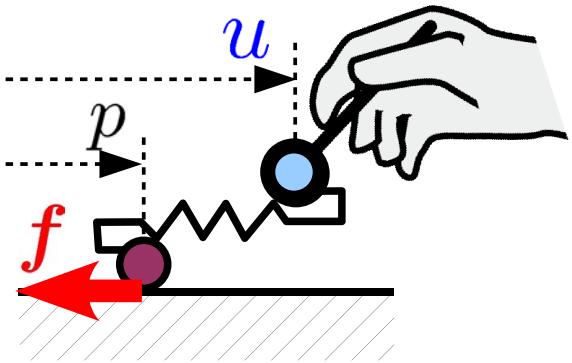
- 全域一価関数でない場合
  - 剛体どうしの接触



$$\begin{bmatrix} \dot{v} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} (-f+u)/M \\ v \end{bmatrix}$$

$$f = \quad p$$

- 陰関数である場合
  - バネ力と摩擦力が釣り合うように速度を決定.

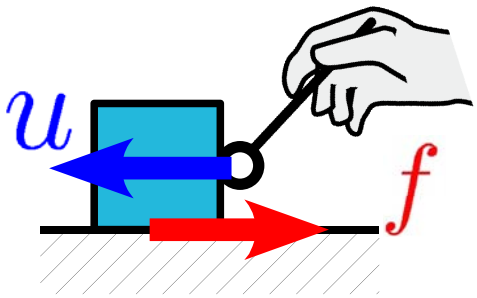


$$\dot{p} = v \text{ s.t. } K(u-p) = F\mathrm{sgn}(v)$$

状態方程式はしばしば不連続になる

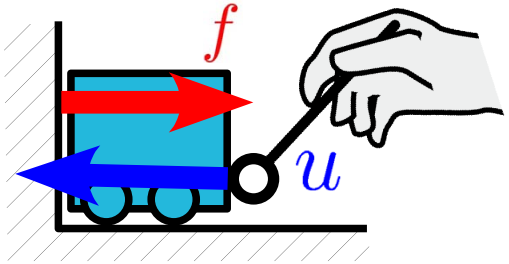
状態方程式 $\dot{x} = \Phi(x, u)$ の右辺が

- 不連続の場合
  - クーロン摩擦力



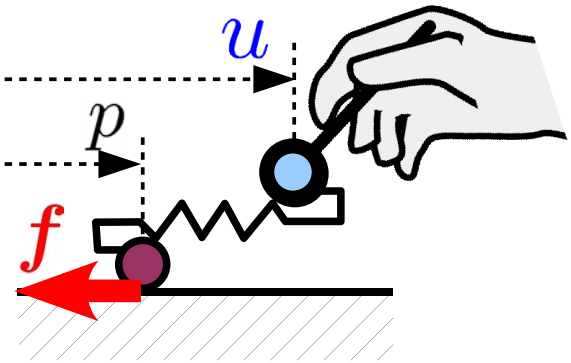
$$\begin{bmatrix} \dot{v} \\ \cdot \end{bmatrix} = \begin{bmatrix} (-f+u)/M \\ \cdot \end{bmatrix}$$

- 全域一価関数でない場合
  - 剛体どうしの接触



$$\begin{bmatrix} \dot{v} \\ \cdot \end{bmatrix} = \begin{bmatrix} (-f+u)/M \\ \cdot \end{bmatrix}$$

- 陰関数である場合
  - バネ力と摩擦力が釣り合うように速度を決定.

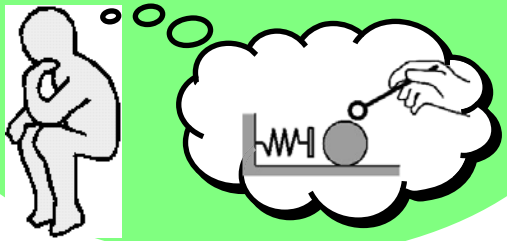


- 陰解法ならどれも積分可能
- たとえ不連続でも物理的に明確な意味がある.
- アルゴリズムは幾何学的に考えた方が早い場合も.

まとめ

状態方程式とアルゴリズム

仮想世界のイメージ



力学

状態方程式

$$\dot{x} = \Phi(x, u)$$

質量・ダンピングの追加など

[菊植・藤本, JRSJ2007]

[Kikuuwe&Fujimoto, WHC07]

力学的意味づけ

(主に陰解法)

力学的改良

変換

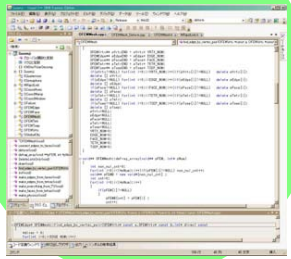
(陽解法 or 陰解法)

主に幾何学

アルゴリズム

$$x_k \leftarrow \tilde{\Phi}(x_{k-1}, u_k)$$

プログラム



まとめ

- 力覚提示アルゴリズムの裏側には状態方程式がある.
- 状態方程式が同じでも，積分方法が異なれば，仮想世界の挙動は大きく異なる.
  - 陽解法：プログラムの見通し良好／計算量少／発散しやすい
  - 陰解法：プログラムが分かりにくい／計算量大／安定.
- 状態方程式は，アルゴリズムに物理的意味付けを与える.
  - アルゴリズム作成中，ワケが分からなくなった場合に何らかの手掛かりが得られるかも