知能システム論1 (13)

2012.7.4

情報システム学研究科
情報メディアシステム学専攻
知能システム学講座
末廣尚士

- 多変数関数の一次近似とニュートン法

- テーラー展開

$$f(x)=f(x_0)+\left(\frac{\partial f}{\partial x}\right)(x-x_0)\ldots$$

0次　　1次

$J=\left(\frac{\partial f}{\partial x}\right)$ を使って簡単化して表現すると $f(x)\approx f(x_0)+J(x-x_0)$

- ニュートン法

$f(x)=0$ となる x を求めたい．x_0 は解に近く，$f(x_0)$ は小さいとする．

近似的に $0=f(x_0)+J(x-x_0)$ とし，

これを $x=x_0-J^{-1}f(x_0)$ と解いて，あらたな近似値を求め繰り返す．

-（擬似）逆行列と特異値分解

- 特異値分解

$A:(m\times n)$ は、 $A=UWV^T$ の形で表現することが出来る

$U:(m\times r)$

$V:(n\times r)$

U, Vは列直交行列, rは行列のrank

$W=diag(\sigma_1,\sigma_2,\ldots,\sigma_r)\quad(\sigma_1\geq\sigma_2\geq\ldots\geq\sigma_r\geq 0)$

列直交行列

$U=(\boldsymbol{e}_1,\boldsymbol{e}_2,,,,\boldsymbol{e}_r)\qquad \boldsymbol{e}_i\cdot\boldsymbol{e}_j=\begin{cases}1(i=j)\\0(i\neq j)\end{cases}$

つまり

$U^TU=I\qquad I$ は $r\times r$

- 逆行列と特異値分解

m = n=r の場合（Aが正則行列の場合）: 逆行列

$$A^{-1} = V W^{-1} U^T \qquad W^{-1} = diag(1/\sigma_1, 1/\sigma_2, \ldots, 1/\sigma_n)$$

$$A^{-1} A = V W^{-1} U^T U W V^T = V W W^{-1} V^T = V V^T = I$$

そうでない場合： 擬似逆行列

$$A^{+} = V W^{-1} U^T \quad W^{-1} = diag(1/\sigma_1, 1/\sigma_2, \ldots, 1/\sigma_r)$$

$$A A^{+} A = U W V^T V W^{-1} U^T U W V^T = U W V^T = A$$

- 他の逆行列を求める方法との比較

よくある方法

m > n の場合

$$A^{+}=\left(A^{T}A\right)^{-1}A^{T}$$

$$AA^{+}A=A\left(A^{T}A\right)^{-1}A^{T}A=A$$

問題点:

- $A^{T}A$ が正則とは限らない
- 正則であったとしても、危ない場合がある

特異値分解を行うとこれらの問題点が明確になるし、対応を工夫することが出来る
ただし、計算コストはかかる。

- 特異値分解とヤコビアン

Ｊ：(6×6)　は、$J=U\,W\,V^T$　の形で表現することが出来る

$U:(6\times6)$　(直交行列)

$V^T:(6\times6)$　(直交行列)

$W=diag(\sigma_1,\sigma_2,\ldots,\sigma_6)$　$(\sigma_1\geq\sigma_2\geq\ldots\geq\sigma_6\geq0)$

$$\Delta\boldsymbol{p}=J\,\Delta\boldsymbol{q}=U\,W\,V^T\Delta\boldsymbol{q}$$

$$\Delta\tilde{\boldsymbol{p}}=U^T\Delta\,\boldsymbol{p}=W\,V^T\Delta\boldsymbol{q}=W\,\Delta\tilde{\boldsymbol{q}}$$

$\Delta\boldsymbol{p}$ と $\Delta\boldsymbol{q}$ の座標軸(直交基底)をうまく選ぶと、軸方向に
σ_i 倍するだけの変換になる。

逆行列は軸方向に$1/\sigma_i$ 倍すればよいことになる。

- ヤコビアンの逆行列と特異点

$$\Delta \boldsymbol{p} = J \Delta \boldsymbol{q} = U W V^T \Delta \boldsymbol{q}$$

$$J^+ = V W^{-1} U^T \qquad W^{-1} = diag(1/\sigma_1, 1/\sigma_2, \ldots, 1/\sigma_6)$$

$$\Delta \boldsymbol{q} = V W^{-1} U^T \Delta \boldsymbol{p}$$

σ_i が大きい： 動かしやすい方向

σ_i が小さい： 動かしにくい方向

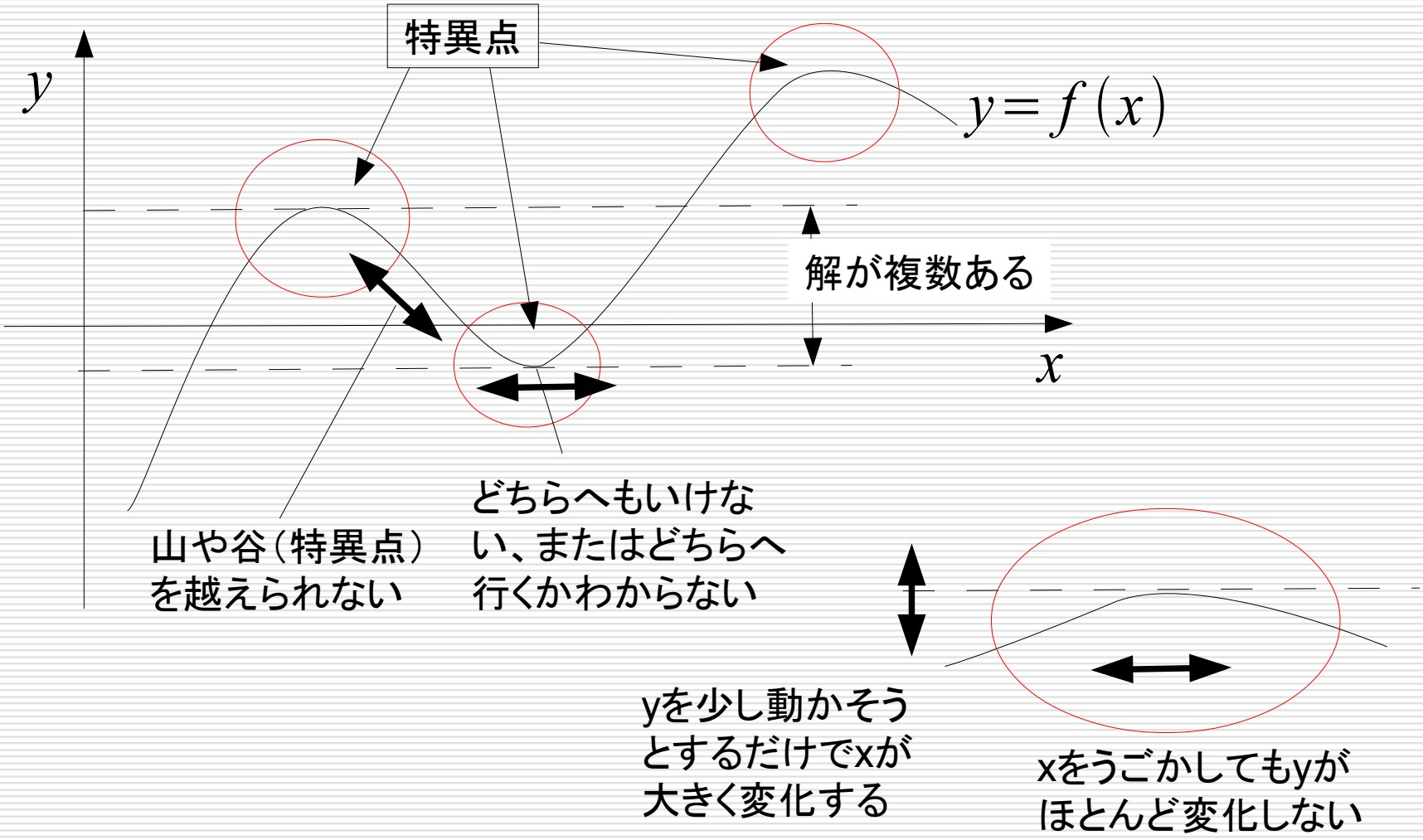
σ_i がゼロ： 動かせない方向がある ⇒ 特異点ということ

σ_i の最大と最小の比：特性値

一般に擬似逆行列では、$\sigma_i = 0$ のところは計算しない。すなわち対応する $1/\sigma_i$ の部分を0としたことと等しい。

さらに、アームの逆運動学解を求める場合などでは、$\sigma_i \approx 0$ の場合も0にすることで、解の発散を防ぐ。

- 特異点


特異点
y
$y=f(x)$
解が複数ある
x
山や谷（特異点）を越えられない
どちらへもいけない、またはどちらへ行くかわからない
yを少し動かそうとするだけでxが大きく変化する
xをうごかしてもyがほとんど変化しない

- 注意点

- 収束性
  - 下手をすると大きく変化しすぎて違うところに飛び出してしまう。
- 停留
  - local minimum に落ち込む
- 特異点
  - 行列が正則でなくなる
- 複数解の存在
  - 複数解のどれになるかの保障がない

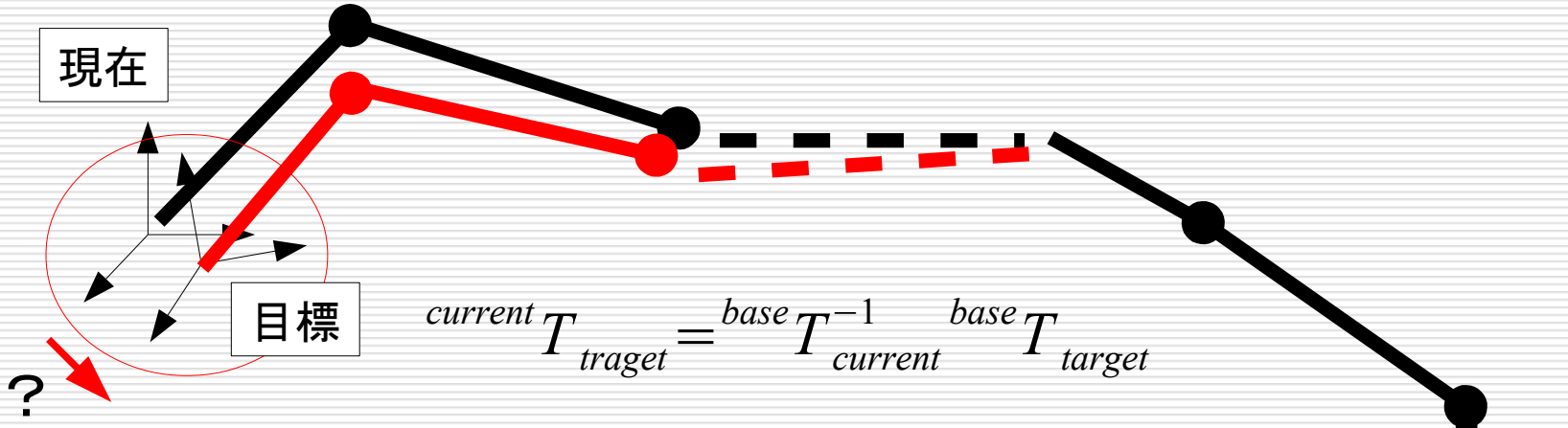
- arm6dof.py(1)

- 下準備
  - 特異値分解で使う領域の確保
  - 現在の関節角、目標位置、姿勢

```
def arm_sol2(self,trans,step=0.01,lim=0.0001,th_lim=0.05,rlim=0.001) :
    w_size=min(6,self.dof)
    cc_diff=empty((6),dtype=float64)
    th_diff=empty((self.dof),dtype=float64)
    w_inv=empty((w_size),dtype=float64)
    th=self.get_joints()
    p_target = trans.vec
    o_target= trans.mat
```

- ヤコビアンによる逆運動学解

手先座標の現状と目標の差の表現



$$ {}^{current}T_{traget} = {}^{base}T_{current}^{-1}\ {}^{base}T_{target} $$

目標と現在の差の座標系表現

現在座標系から見た並進、回転ベクトル表現

ベース座標系から見た並進、回転ベクトル表現

$$ {}^{current}T_{traget} \Rightarrow {}^{current}\boldsymbol{diff},\ {}^{current}\boldsymbol{rot} \Rightarrow {}^{base}\boldsymbol{diff} = {}^{base}T_{cuurent}\ {}^{current}\boldsymbol{diff},\quad {}^{base}\boldsymbol{rot} = {}^{base}T_{current}\ {}^{current}\boldsymbol{rot} $$

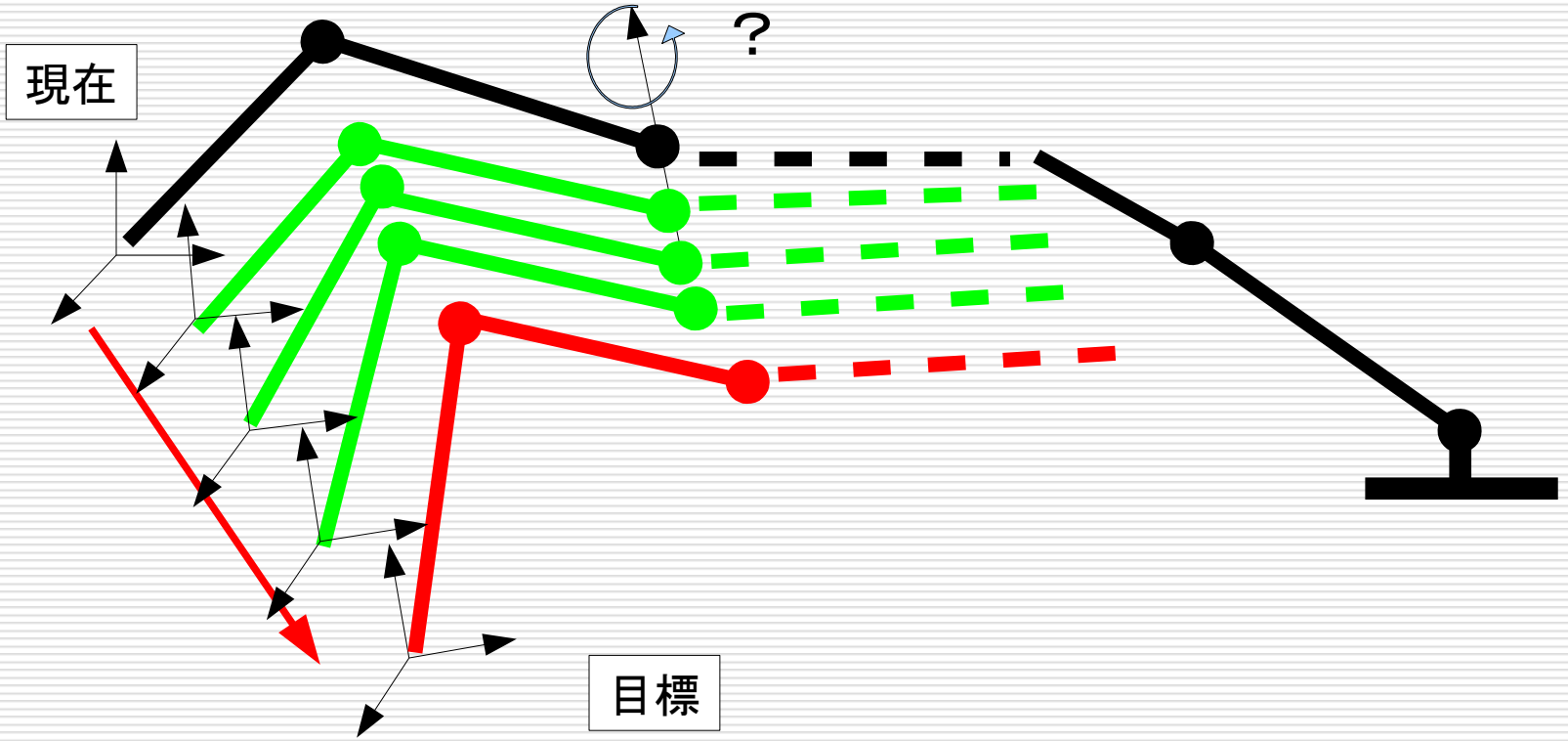
- arm6dof.py(2)

- 手先座標の現状と目標の差の計算

```
    while 1 :
#         c_frame=self.wrist.where(self.base)
        c_frame=self.hand.where(self.base)
        c_pos = c_frame.vec
        p_diff=p_target-c_pos
        c_ori = c_frame.mat
        o_diff = ((- c_ori) * o_target).rot_axis()
```

- ヤコビアンによる逆運動学解

差が大きいときは、軌道を分割する



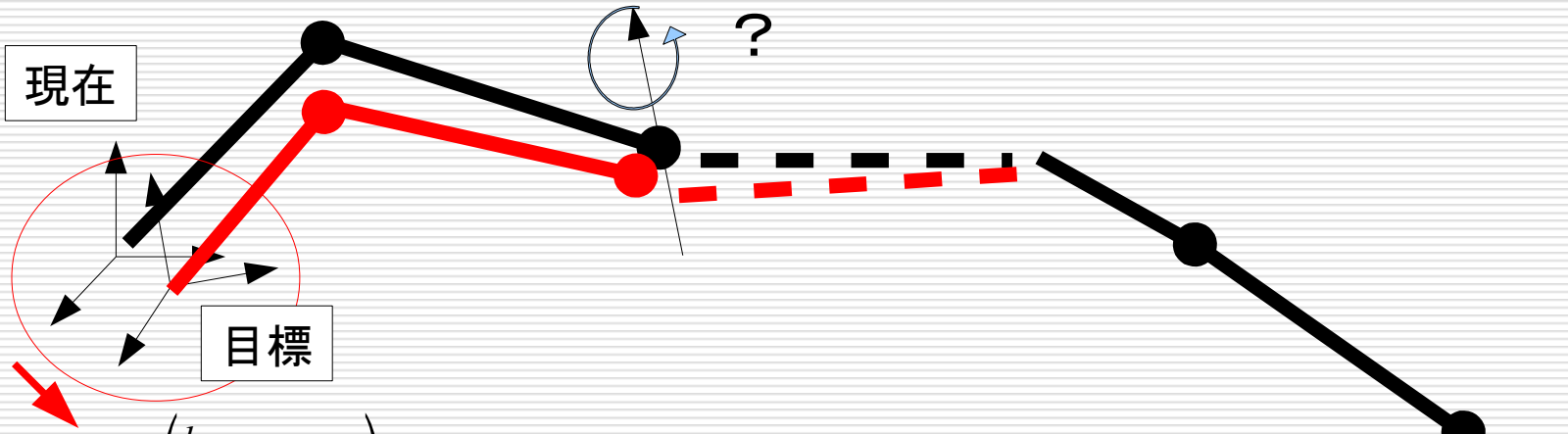
- arm6dof.py(3)

- 目標との差のチェック
  - 十分小さければ終了
  - あまりに大きければ、ステップを刻む
  - それなりの大きさならそのまま
- 回転軸ベクトルの座標変換および大きさ調整

```
dist=sqrt(p_diff.dot(p_diff)+o_diff[0]**2)
if dist < lim :
    break
elif dist > step :
    p_diff=(step/dist)*p_diff
    o_diff[0]=(step/dist)*o_diff[0]
o_diff[1]=o_diff[0]*(c_ori*o_diff[1])
```

- ヤコビアンによる逆運動学解

関節パラメタをどう変化させたらよいか



$$\Delta \boldsymbol{p} = \begin{pmatrix} {}^{base}\boldsymbol{diff} \\ {}^{base}\boldsymbol{rot} \end{pmatrix}$$ これが十分小さいとすると、、、

$$\Delta \boldsymbol{p} = \begin{pmatrix} {}^{base}\boldsymbol{diff} \\ {}^{base}\boldsymbol{rot} \end{pmatrix} = \begin{pmatrix} \Delta \boldsymbol{p} \\ \Delta \boldsymbol{\phi} \end{pmatrix} = J \Delta \boldsymbol{q} \Rightarrow \Delta \boldsymbol{q} = J^{-1} \Delta \boldsymbol{p}$$

または $\Delta \boldsymbol{q} = J^{+} \Delta \boldsymbol{p}$（正則でない場合）

- arm6dof.py(4)

- ヤコビアンの計算と特異値分解
- 特異値が小さいところの切り落とし

```
self.calc_jacob()
u_mat,w_vec,v_trn=svd(self.j_mat,compute_uv=1,full_matrices=0)
w_max=max(w_vec)
w_min=w_max*rlim
for j in range(6) :
    if w_vec[j]<w_min :
        w_inv[j]=0.0
    else :
        w_inv[j]=1.0/w_vec[j]
```

- arm6dof.py(5)

- 手先の微小移動に対応する関節角度変化分の計算
- 関節角の変化が大きくなりすぎないように制限する

```
        for j in range(3) :
            cc_diff[j]=p_diff[j]
            cc_diff[j+3]=o_diff[1][j]
        u_trn=u_mat.transpose()
        tmp=dot(u_trn,cc_diff)
#         print tmp
        tmp=tmp*w_inv
        v_mat=v_trn.transpose()
        th_diff=dot(v_mat,tmp)
        th_max=max(th_diff)
        if th_max > th_lim :
            for j in range(self.dof) :
                th_diff[j] *= th_lim/th_max
```

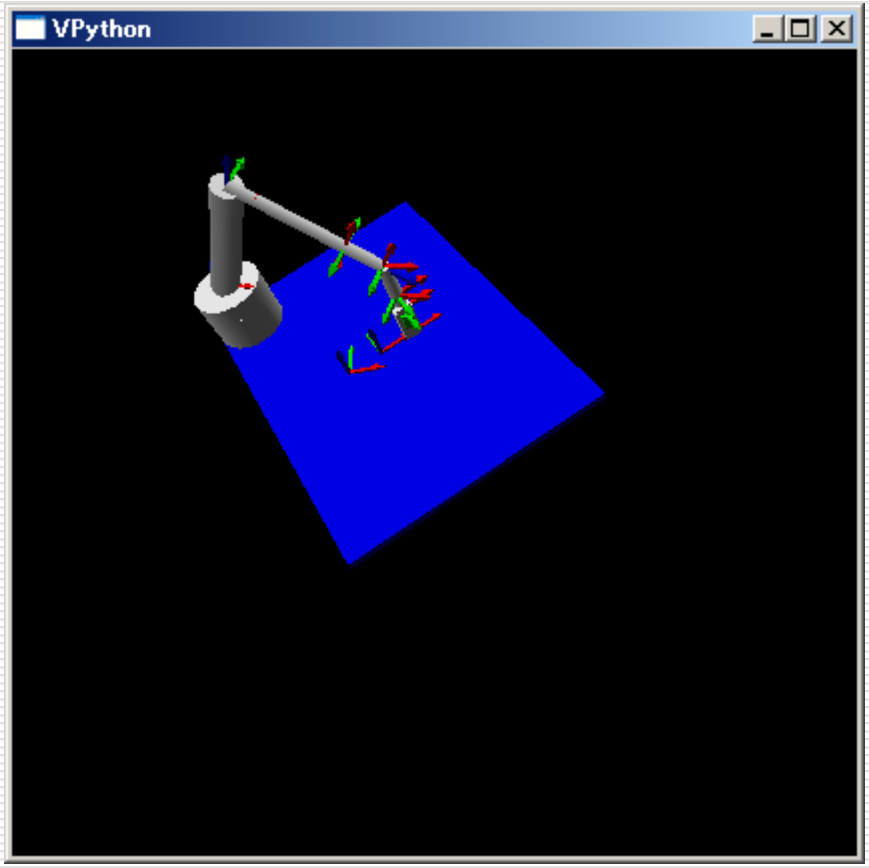
- arm6dof.py(6)

- 関節角変化分の適用：繰り返しへ
- 終了処理

```
        for j in range(self.dof):
            th[j] += th_diff[j]
        self.set_joints(th)
#
    self.solution2=th
    return th
```

- 実行例(env_arm6dof.py)

```
>>> create_env()
>>> arm.solve2(box.where(arm.base))
[-0.8519709146401257 5,
0.5716525262084165 6,
1.662819893182902 1,
3.141606464604186 6,
-0.9071121316229018 3,
-0.8519807720817074 2]
>>> arm.hand.where(box).xyzabc()
[-1.9155179310503812e-006,
2.4515453263471154e-006,
-3.4869339414211886e-006,
1.0661780502463189e-006,
-1.3523143813456838e-005,
1.3495879990349744e-006]
```



- 比較

```
>>> arm.solve(box.where(arm.base)*arm.wrist.where(arm.hand))
.
.
.
>>> arm.solutions[1]
[-0.8519663271732721 4, 0.5716571227757810 7, 1.6628010732898744,
-3.1415926535897931, -0.9071344575241379, -0.8519663271732722 5]
>>> arm.set_joints(arm.solutions[1])
>>> arm.hand.where(box).xyzabc()
[-2.775557561562891 4e-017, 6.2450045135165055e-017,
1.1102230246251565e-016, -2.2754674832649873e-018,
-6.9388939039072284e-017, 0.0]
```

- なぜ数値解法？

- 幾何的、解析的に解けない場合が多数ある。
  - 多変数、非線形問題の基本的な解法となっている。
- 冗長自由度の問題を扱える
  - 冗長自由度をそのまま（無駄に動かさずに）解ける擬似逆行列の最小推移性
  - 冗長自由度を有効に使うパラメタを自然に、新たに導入できる
- 過剰拘束であっても何とかなる
  - 解の誤差が最小2乗となる基本的な解法
- 分解運動速度制御
  - arm_sol2でアームが動いたのをそのまま利用

- moveの完成(arm6dof.py)

```
def arm_move(self, smthng, smwhr, wrt=None) :
        if smthng.__class__ == str :
           smthng = self.tools[smthng]
        if smthng.ancestor(self.wrist) :
           if isinstance(smwhr,CoordinateObject) :
              target=smwhr.where(self.base)
           elif smwhr.__class__ == FRAME :
              if wrt :
                 target = wrt.where(self.base) * smwhr
              else :
                 target = (- self.base.where()) * smwhr
           else :
              print "error: wrong place expression"
              return False
           self.solve2(target*(- smthng.where(self.hand)))
#            time.sleep(1)
        else :
          print "error: cannot move unfixed object"
          return False
```

- armへの命令追加

- ready, park

```
def ready(self):
    if self.ready_angle :
        self.set_joints(self.ready_angle)
    else :
        print "no ready angle"
def park(self) :
    if self.park_angle :
        self.set_joints(self.park_angle)
    else :
        print "no park angle"
```

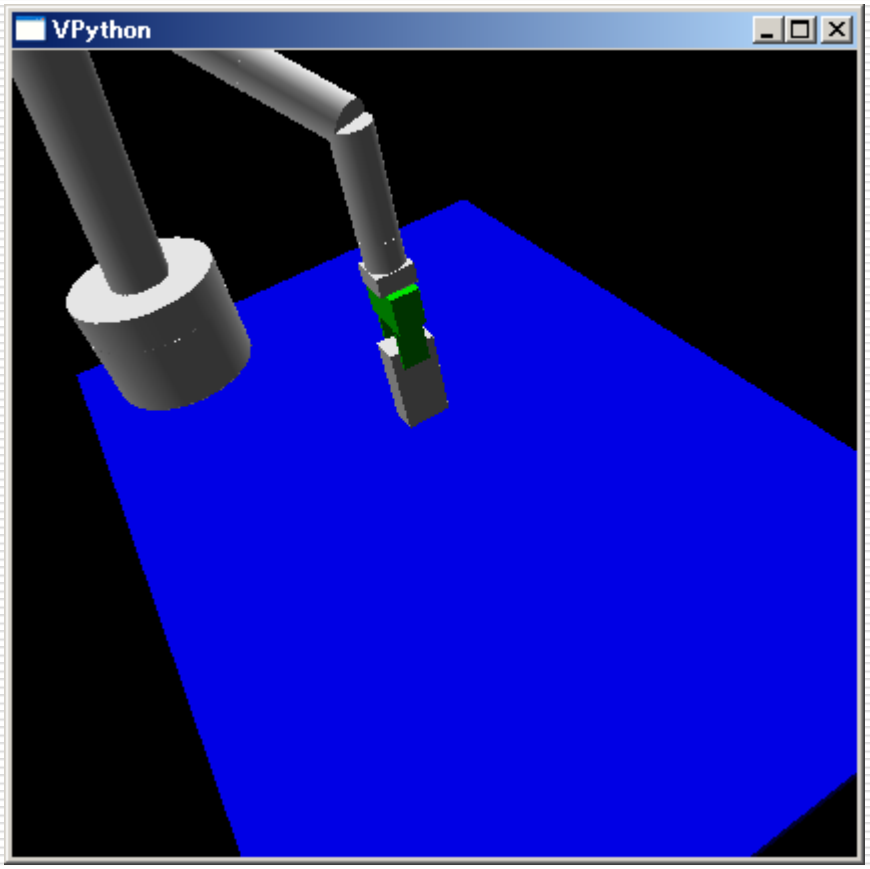
- armへの命令追加

- create_arm()

```
arm.ready_angle=[0,pi/6,pi/2,0,pi/3,0]
arm.park_angle=[0,0,0,0,0,0]
```

- 作業例

- boxを動かす。
  - test_model_v_vbody2.py と robo_sys.py の比較
  - do_task_old() と do_task() の比較



- 次回予告

- 分解運動速度制御
- 自分の好きな構造のロボットを作る

  larm_w_hand_sol.pyを使う.

  arm7dof.pyやarm7dof2.pyを参考に，これまで自分で作ったarmをベースにモディファイする.
- 環境はenv.py