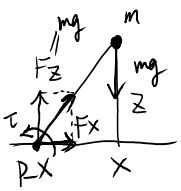
HW3

2015/04/01 20:27

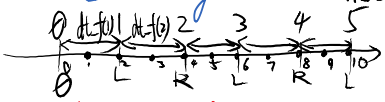


$$\ddot{x} = (x - p\_x^{(t)} + u\_x^{(t)}) \cdot g\_z$$

$$\ddot{y} = (y - p\_y^{(t)} + u\_y^{(t)}) \cdot g\_z$$

Okay. fminsearch does the job ... Perhaps CMA-ES could be faster (soft constraint only).

Variable timing: m=5, n=2 Vars: each INTERVAL



Limits: (0.3 ~ 2.0) only

Problem: total time now different. Can we achieve some "orthogonality"?

Put this straight: can we maintain u_x, u_y function when changing timing?

Not if these are sampled uniformly. Some coupling has to be there.

So "u" will be uniform across... Or no? Can it be uniform within a step?

Yes, but score func needs change. Basically: $\int f^2(t)\,dt \Rightarrow \frac{T}{n}\sum f^2(\frac{i}{n}T)$.

Var = [dt_step; u_x; u_y] (m; m·n; m·n)

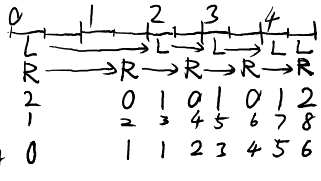
However, should NOT indirectly penalize total time ⇒ weighted average of sum sqr,

So integrate: (x.^2).'*dts

Random thoughts: choice of meta-params in the score function could be Voodoo.

Swing Penalty: Each swing incurs $\left(\frac{x}{t^2}\right)^2$.

Need to pass to score fn: "swing_from":



R: 2→0, 0→0, 1→2

L: 2→1, 1→1, 0→2

Fuck, a bug in PI found. Correcting it stops fminsearch from converging.

FUCK THIS $#.#!

At least see if CMA-ES can handle.

Also, using grid size 0.05 (10 div/step) and cubic spline trajectory interpolation.

I should first see if previous result can apply to current sim/opt.

Running CMA-ES till death doesn't sound reasonable.

There has to be a way... Wait a sec.

$f(t, x, y, \dot{x}, \dot{y})$ rely on $f(t-\delta t, x', y', \dot{x}', \dot{y}')$ only... DP!

Okay. 刘文 gave you a good crash course on DDP.

OrigDyn: $\dot{x} = f(t, x, u)$

OrigObj: $\lambda_1 \int (x-c)^2 dt + \lambda_2 \int \dot{x}^2 dt + \lambda_3 \int u^2 dt + \cdots$

Step Dyn : $x_{i+1} = F(x_i, u_i)$

Step Cost : $L(x_i, u_i) = (x_i - c_i)^T (x_i - c_i) + u_i^T u_i + \cdots$ Here we have G quadratic fn of both X & u. Can specify as $a + bx + \frac{1}{2}x^T C x \cdots$

Value Fcn : $V_i(x) = \min_{u_i} \left[L(x_i, u_i) + V_{i+1}(F(x_i, u_i)) \right]$ Boundary : $V_N(x) = 0$

Before iter : x_i, u_i cur. val. available.

$Q^i(x_i, u_i) = L^i(x_i, u_i) + V^{i+1}(F^i(x_i, u_i))$

$Q^i_u = L^i_u + V^{i+1}_x F^i_u$

$Q^i_{xx} = L^i_{xx} + V^{i+1}_x F^i_{xx}$

Okay. Now explicitly plug everything in... Hold on a sec again.

$\vec{x} = (x, y, \dot{x}, \dot{y})^T$ $\vec{u} = (u_x, u_y)$

Isn't the problem kinda easy, even if we group all timesteps within one stance interval? This is what you'll have to do for Part 2&3 anyway... (maybe).

~~DDP step $\Longleftrightarrow$ foot step~~ ($1..m$). ~~Time subdivision : N (just as p2_opt).~~

~~State : $x_i : [x; y; \dot{x}; \dot{y}]$~~ ... No. Dim of state = 4N!

But we still can include the timing... Yes. Makes sense to me now...

Because it's just DP. Who said you can't have multiple transition rules!

$V(i = 1..M+1, j = 0..N, x)$

i: which stance ($M+1 \Rightarrow$ end); j: which sub-div ($t = t_0(i) + \frac{j}{N}(t_0(i+1) - t_0(i))$); x: where you start from.

Hold on yet again. Turns out DDP also avoids the grid flood.

Reason: Know all u_i $\Rightarrow$ sim to get x_i. And DDP only change u_i locally so x_i are also changed locally... Or really? (can always set learning rate to avoid large jumps). So the "derivative-ful" form is what you should use. However, the "grid flood" version & the "conceptual bridge" btwn this and plain DP — should always keep in mind.

So in the end we keep "current iter" values of both x_i & u_i, and do local optimization completely around them. Now read the paper carefully and try associate terms to the "plain DP" form (so that you can adapt it to your own form).

Updating section above. Main problem: Tensors. Fuck this... Gotta learn!

$L, V : (0,0)$ $F : (1,0)$ $X \in \mathbb{R}^c$ $U \in \mathbb{R}^d$

$1 \times c$ $Q^t_x = L^t_x + V^{t+1}_x F^t_x$

$1 \times d$ $Q^t_u = L^t_u + V^{t+1}_x F^t_u$

Impl w/ plain mat.

Rank ≤ 2 : normal rep'n

1 grad : $F_u(i,j) = \frac{\partial F_i}{\partial u_j}$

$1\times C$ $Q_x^t = L_x^t + V_x^{t+1} F_x^t$

$1\times d$ $Q_u^t = L_u^t + V_x^{t+1} F_u^t$

$c\times c$ $Q_{xx}^t = L_{xx}^t + V_x^{t+1} F_{xx}^t + (F_x^t)^T V_{xx}^{t+1} (F_x^t)$

$d\times C$ $Q_{ux}^t = L_{ux}^t + V_x^{t+1} F_{ux}^t + (F_u^t)^T V_{xx}^{t+1} (F_x^t)$

$d\times d$ $Q_{uu}^t = L_{uu}^t + V_x^{t+1} F_{uu}^t + (F_u^t)^T V_{xx}^{t+1} (F_u^t)$

$d\times C$ $K^t = (Q_{uu}^t)^{-1} Q_{ux}^t$

$d\times 1$ $\delta u^t = (Q_{uu}^t)^{-1} (Q_u^t)^T$

$1\times C$ $V_x^t = Q_x^t - Q_u^t K^t$

$c\times c$ $V_{xx}^t = Q_{xx}^t - Q_{xu}^t K^t$ $= (Q_{ux}^t)^T$

Impl w plain mat.

Rank ≤ 2 : normal rep'n

| grad : $Fu(i,j) = \frac{\partial F_i}{\partial u_j}$

| hess : $Qux(i,j) = \frac{\partial^2 Q}{\partial u_i \partial x_j}$

Rank $=3$: $Vx\,F$??

| $Fux(i,j,k) = \frac{\partial^2 F_i}{\partial u_j \partial x_k}$

$1\times C \cdot C\times ?\times ?$

It'd be great if we could modularize this. At least try P1 first...

Actually if we can add a terminal cost fn $V^N(x)$ as initial... P2'd be solved.

Okay. So the DP topology can still remain linear... Ah. I see the problem.

Doing that requires grid search on X again...

Conclusion: Forget about P2 and multi-choice for now.

Need to provide: $\{F^t, L^t\}$, up to hessian w.r.t. X, u.

$L^t(x,u) = \frac{1}{2}(x-c)^T(x-c) + \frac{\lambda}{2}u^T u = \frac{1}{2}x^T x - c^T x + \frac{1}{2}c^T c + \frac{\lambda}{2}u^T u$

$c = (p_x \; p_y \; 0 \; 0)^T$

$L_x^t = (x-c)^T \quad L_{xx}^t = I \quad L_{ux}^t = 0$

$L_u^t = \lambda u^T \quad L_{uu}^t = \lambda I$

$F^t(\vec{x}, u)$: Try also simplify into quadratic $\Rightarrow$ actually linear (euler first order).

$\vec{x} = (x, \dot{x})$ need to know x^{t+1}, $\dot{x}^{t+1}$ (actually $t+\Delta t$)

$x^{t+1} = x^t + \Delta t \cdot \dot{x}$

$\dot{x}^{t+1} = \dot{x}^t + (\frac{c_t}{m} p_x + u^t)\underbrace{\frac{c_t}{m}\Delta t}_{dv}$

$$\vec{x}^{t+1} = \begin{pmatrix} 1 & \Delta t \\ dv & 1 \end{pmatrix}_A \vec{x}^t + \begin{pmatrix} 0 \\ dv \end{pmatrix}_B u^t + \begin{pmatrix} 0 \\ -dv \cdot p_x \end{pmatrix}_C$$

(y component is identical, no cross terms).

$F_x^t = A$, $F_u^t = B$, $F_{xx}^t = F_{ux}^t = F_{uu}^t = 0$

Actually dynamics is decoupled as well $\Rightarrow$ completely separate! ✓ Done.

Back to P2. Since DDP does not require grid search, can we use "whole step"?

$\vec{X} = (x_0^k, \dot{x}_0^k)$ (because intermediate $x/\dot{x}$ does not influence decision)

$\vec{u} = (u_0^k, \dots, u_{N-1}^k)$ F/L (next/cost) pair: must use ODE integrator. Or can we cheat?

$n_x = 2$, $n_u = N$, $x_1 = x_0 + \dot{x}_0\alpha \quad x_2 = x_1 + \dot{x}_1\alpha = (x_0 + \dot{x}_0\alpha) + (\dot{x}_0 + x_0\beta)\alpha$

$\dot{x}_1 = \dot{x}_0 + x_0\beta + \dots$ All boils down to: Is it linear?

Need to specify what we want.

Cost: quad. of $(x_0 \cdots x_N) = \tilde{X}$

if: $\tilde{X} = \tilde{A}x + \tilde{B}u + \tilde{C}$ then quad of x and u.

We need to keep in mind that this is a special case. Also we

Proof: $\tilde{X}^T Q \tilde{X} = (Ax+Bu+c)^T Q (Ax+Bu+c)$ — now have cross terms.
$= x^T A^T Q A x + x^T A^T Q B u$

Idea: Generate $\tilde{A}, \tilde{B}, \tilde{C}$. Have to live with interlace.

n=3 $x' = Ax+Bu+c$

	x_0	$\dot{x}_0$	u_0	u_1	u_2	
x_0	I		0	0	0	0
$\dot{x}_0$						
x_1	A		B	0	0	c
$\dot{x}_1$						
x_2	A^2		AB	B	0	CtAC
$\dot{x}_2$						
x_3	A^3		A^2B	AB	B	CtACtA²C
$\dot{x}_3$						

$\tilde{A}_{(2n+2)\times 2}$ $\tilde{B}_{(2n+2)\times n}$ $\tilde{C}_{(2n+2)\times 1}$

Then: $L(i,x,u) = \frac{1}{2}(\tilde{x}-\tilde{p})^T(\tilde{x}-\tilde{p}) + \frac{\lambda}{2}u^T u$

$\tilde{p} = (p,0,p,0,\dots)^T$

1×2 $L_x = (\tilde{x}-\tilde{p})^T \tilde{A}$

$1\times n$ $L_u = \lambda u^T$

2×2 $L_{xx} = \frac{\partial}{\partial x}\left((\tilde{A}x+\tilde{B}u+\tilde{C}-\tilde{p})^T\right)\tilde{A} = \tilde{A}^T\tilde{A}$

$n\times 2$ $L_{ux} = 0$

$n\times n$ $L_{uu} = \lambda I$

Idea: encode timing as u_n... NO. This breaks.

$A = \begin{bmatrix} 1 & \Delta t \\ \Delta v & 1 \end{bmatrix}$ $B = \begin{bmatrix} 0 \\ \Delta v \end{bmatrix}$ $C = \begin{bmatrix} 0 \\ -\Delta v p \end{bmatrix}$

But: $\Delta t = \frac{u_n}{n}$, $\Delta v = \frac{G}{2}\Delta t$ ...

So... Why shouldn't we simply say: $u := (u, \Delta t)$? Good. We're not dealing with... Okay. Then there's "density". You can try penalize. And other cost params can... Well. Now you need to plan ahead what "one step" cost is... Say, the "swing" is not easy. $\left(\frac{\Delta x}{t^2}\right)^2$... But anyway, with the $\frac{1}{t^4}$, we no longer have quad cost... Wait. What if we use velocity? No...

$A \cdot A = \begin{bmatrix} 1 & \Delta v \\ \Delta t & 1 \end{bmatrix}\begin{bmatrix} 1 & \Delta v \\ \Delta t & 1 \end{bmatrix} = 1 + \Delta v \Delta t$ Remember: if $u=0$, x is exponential to time!

Use the "i" param. $\vec{X} := (x, \dot{x}, \Delta t)$, $u := \begin{cases} u & i\%(n+1) = 0..n-1 \\ \Delta t & i\%(n+1) = n \end{cases}$

x: 0, 1, 2, ... $n-1$, n (u_0, u_{n-1}, u_n)

x_{n+1}: 0, 1, 2 (u_{n+1}: 0, 1)

Now x is quadratic to self, but this is okay...

$x = x + \dot{x}\cdot\Delta t$

$\dot{x} = x\cdot k\Delta t + \dot{x} + u\cdot k\Delta t - p\cdot k\Delta t$

$x = \frac{1}{2}\vec{x}^T \begin{pmatrix} & & 1 \\ & & \\ 1 & & \end{pmatrix}\vec{x} + (1,0,0)\vec{x}$ $\quad \frac{\partial x}{\partial \vec{x}} = \vec{x}^T\begin{pmatrix} & & 1 \\ & & \\ 1 & & \end{pmatrix} + (1,0,0)$

$\dot{x} = \frac{1}{2}\vec{x}^T\begin{pmatrix} k & & 0 \\ & & \\ 0 & & k \end{pmatrix}\vec{x} + (0\ 1\ -pk)\vec{x} + u^T(0\ 0\ k)\vec{x}$ $\quad \Delta t = (0\ 0\ 1)\vec{x}$

Sounds like it could work.

$\frac{\partial F}{\partial x} = \begin{bmatrix} x^T\begin{pmatrix} & & 1 \\ & & \\ 1 & & \end{pmatrix} + (1\ 0\ 0) \\ x^T\begin{pmatrix} k & & 0 \\ & & \\ 0 & & k \end{pmatrix} + (0\ 1\ -pk) \\ (0\ 0\ 1) \end{bmatrix}$

$\frac{\partial F}{\partial u} = \begin{bmatrix} 0 \\ (0\ 0\ k)\vec{x} \\ 0 \end{bmatrix}$

$\frac{\partial^2 F_1}{\partial x^2} = \begin{pmatrix} & & 1 \\ & & \\ 1 & & \end{pmatrix}$ $\frac{\partial^2 F_2}{\partial x^2} = \begin{pmatrix} k & & 0 \\ & & \\ 0 & & k \end{pmatrix}$ $\frac{\partial^2 F_3}{\partial x^2} = 0$

$\frac{\partial^2 F_1}{\partial u \partial x} = 0$ $\frac{\partial^2 F_2}{\partial u \partial x} = (0\ 0\ k)$ $\frac{\partial^2 F_3}{\partial u \partial x} = 0$

$\frac{\partial^2 F_1}{\partial u^2} = 0$ $\frac{\partial^2 F_2}{\partial u^2} = 0$ $\frac{\partial^2 F_3}{\partial u^2} = 0$

"Step transition": $x = x$ $\dot{x} = \dot{x}$ $\Delta t = u$

$\frac{\partial F}{\partial x} = \begin{bmatrix} 1 & & 0 \end{bmatrix}$ 2nd order all 0.

$\frac{\partial F}{\partial u} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Associate "swing" cost here (yes, nonlinear, but should still work). ⇒ Need to put this more explicitly.

Otherwise: normal step:

$x_1 = (0\ 0\ 0)^T$ $u_1 = \Delta t_1$

$x_2 = (0\ 0\ \Delta t_1)^T$ u_2 } n time-steps

$\vdots$ $\vdots$

x_{n+1} u_{n+1}

x_{n+2} $u_{n+2} = \Delta t_2$

$\vdots$ $\vdots$

Otherwise: normal step:

$$F: \begin{cases} x = x + \dot{x}\cdot\Delta t \\ \dot{x} = k\cdot x\cdot\Delta t + k\cdot u\cdot\Delta t + \dot{x} - pk\cdot\Delta t \\ \Delta t = \Delta t \end{cases} \quad \text{(quadratic)}$$

$$L: \frac{1}{2}(x - P_i)^T(x - P_i) + \frac{k_u}{2}u^T u \quad \text{(quad.)}$$

$i\,\%\,(n+1) = 1$: "Δt step":

$$F = \begin{pmatrix} 1 & \\ & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \quad \text{(linear)}$$

$$L = k_d \frac{d_j}{u^2} \quad \text{(nonlinear)}$$

$$\frac{\partial L}{\partial x} = 0, \quad \frac{\partial L}{\partial u} = -2k_d d_j\left(\frac{1}{u^3}\right) \quad \frac{\partial^2 L}{\partial u^2} = 6k_d d_j\left(\frac{1}{u^4}\right)$$

Actually x becomes $x, y \ldots$

x, y now coupled!

(well, not directly.)

$$L = \frac{1}{2}\left((x - p_x)^2 + (y - p_y)^2 + \dot{x}^2 + \dot{y}^2 + k_u u_x^2 + k_u u_y^2\right)\Delta t$$

Relabel: $m = 3, h = 3;\ nn = 4, q = 12$