PI2-DMPs

使用强化学习的方法PI2更新DMPs以实现轨迹模仿。代码部分主要参考材料[2]，内含PI2的matlab代码

DMPs部分

DMPs(Dynamic Movement Primitives)被设计为可以拟合任意的运动轨迹，并且可以较为方便的让电机执行。

$$\ddot{y}_t = \alpha_y(\beta_y(g - y_t) - \dot{y}_t) + f$$
$$f = \mathbf{g}(t)^T \boldsymbol{\theta}$$
$$[\mathbf{g}(t)]_j = \frac{\psi_j(x_t)}{\sum_{k=1}^{n_{bfs}} \psi_k(x_t)} x_t(g - y_0)$$
$$\psi_j(x_t) = exp(-0.5h_j(x_t - c_j)^2)$$
$$\dot{x}_t = -\alpha_x x_t$$

式一

可以表示为状态空间方程：

$$\frac{d}{dt}\begin{bmatrix} y_t \\ \dot{y}_t \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\alpha_y\beta_y & -\alpha_y \end{bmatrix}\begin{bmatrix} y_t \\ \dot{y}_t \end{bmatrix} + \begin{bmatrix} 0 \\ \alpha_y\beta_y g + f \end{bmatrix}$$

注意到整体而言，DMPs仍然是一个二阶系统，可以通过对状态矩阵的特征值配置，影响系统本身的稳定性，响应状态。代码中默认的配置是$\alpha_y = 25, \beta_y = 6$，这对应状态矩阵特征根为-15和-10，保证了稳定性与大约0.5s时系统对阶跃响应能达到90%（f=0情况下）。

另一方面，f非线性项的存在使中间过程可以是多变的，f项实际上是一个基于权重的高斯函数和。

式五

x_t是一个一次衰减系统（在这里又被称为canonical dynamical system），初值为1，按指数衰减到0。

式四

每一个高斯函数（也称作基函数，base function）都设置在x_t上，为了适应这点，c_j是时间等距的x_t，如$c = [e^{-0.1}, e^{-0.2}, e^{-0.3}, \cdots]$，在时间上等距，在$x$上不等距。代码中将$c_j$记为`mean_canonical[j]`，将$\psi$记作`psi`。

基函数在x_t上分布导致了一个不太好的效应，x_t在时间上的变化是不均匀的，所以如果设置相同的h_j，基函数在时域上的方差（粗细程度）将不同，所以需要h_j引入方差的补偿，让每个基函数的粗细差不多一致（才能令对于每一个t，都有基函数可以使这范围内的形态改变）。代码中的方法是$h_j = (0.55 * (c_j - c_{j-1}))^2$，这使基函数下降到约0.5时，下一个基函数超过了它，成为了主导。代码中的`D`是这里的h_j。

j是基函数的下标，一共有n_{bfs}个基函数，代码中记作`n_bfs`。

式三

基函数组合形成的系数，按时间t与基函数下标j划分，每一个时刻有n_bfs长度的$\mathbf{g}(t)$项，共有t时刻。代码中记作`g_term`

$(g-y_0)$项是空间比例缩放，g代表目标y，$y0$代表DMPs起始阶段的y，这一项能够使DMPs贴合不同空间规模的轨迹。

x_t项是一个归零项，因为其最终会衰减到0，能保证到DMPs终点时f整体强度趋于0，从而保证整个系统会趋于g。

$\frac{\psi_j(x_t)}{\sum_{k=1}^{n_{bfs}}\psi_k(x_t)}$项是基函数的归一化项，分母按同一时刻所有$\psi_j$求和。每一时间，不太活跃的$\psi_j$贡献的部分都接近0。仍然可能存在所有贡献都不大的情况，因此分母上为所有项增加了1e-10避免除0问题。

式二

f代表非线性动力项，这一部分的存在相当于二阶系统上的额外加速度，才使DMPs系统的自由度很大。代码中非线性系统记录为`f_term`，其值为`f`，可以通过`calc_f`来计算当前x_t下的f值。

$\boldsymbol{\theta}$是n_bfs维的权重向量，这部分直接影响最终形态，也是PI2方法要学习的参数。当已知期望轨迹时，也可以使用拟合轨迹的方法来得到它。对于阶跃响应，它的值可以达到几百几千。代码中表示为`weight`。

PI2部分

$$S_m(\tau_{i->N}) = \phi_m(t_N) + \sum_{t=i}^{N-1} r_m(t) + 0.5\sum_{t=i+1}^{N-1}(\boldsymbol{\theta}+\mathbf{M}_m(t)\boldsymbol{\epsilon}_m(t))^T\mathbf{R}(\boldsymbol{\theta}+\mathbf{M}_m(t)\boldsymbol{\epsilon}_m(t))$$

$$\mathbf{M}_m(t) = \frac{\mathbf{R}^{-1}\mathbf{g}(t)\mathbf{g}^T(t)}{\mathbf{g}^T(t)\mathbf{R}^{-1}\mathbf{g}(t)}$$

$$P_m(\tau_{i->N}) = \frac{e^{-\frac{1}{\lambda}S_m(\tau_{i->N})}}{\sum_{l=1}^{capacity}[e^{-\frac{1}{\lambda}S_l(\tau_{i->N})}]}$$

$$\delta\boldsymbol{\theta}(t) = \sum_{m=1}^{capacity}[P_m(\tau_{i->N})\mathbf{M}_m(t)\boldsymbol{\epsilon}_m(t)]$$

$$[\delta\boldsymbol{\theta}]_j = \frac{\sum_{t=0}^{N-1}(N-t)\psi_j(x_t)[\delta\boldsymbol{\theta}(t)]_j}{\sum_{t=0}^{N-1}(N-t)\psi_j(x_t)}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \delta\boldsymbol{\theta}$$

PI2需要一个存放历史轨迹经验池，容量为capacity，m通常指经验池取出的下标。每次运行rollout满经验池，然后更新一次。

式二

对第m条轨迹轨迹，在每一个时刻t，用$\mathbf{g}_t$和$\mathbf{R}$计算。为了降低计算难度，$\mathbf{R}$通常固定为1。另外，考虑到$\mathbf{M}$出现的场景是$\mathbf{M}_m(t)*\boldsymbol{\epsilon}_m(t)$，为了降低矩阵乘法复杂度，通常先计算$\mathbf{g}^T(t)*\boldsymbol{\epsilon}_m(t)$，得到一个标量，再乘其他部分。

式一

$\tau_{i->N}$代表从时刻i开始到结束的轨迹。

$S_m(\tau_{i->N})$分为三个部分，$\phi_m(t_N)$代表第m条轨迹的终值cost；$\sum_{t=i}^{N-1} r_m(t)$代表从时刻i开始到结束的时间步cost和；最后一项是权重正则化方面的cost，下标虽然从$i+1$开始，但matlab代码上仍然按从i开始，可能影响不大。代码中使用`cumsum`的方式实现对每个时刻i计算$S_m(\tau_{i->N})$，权重正则化项合并在了计算奖励`calc_cost`的过程中。

式三

对每个$S_m(\tau_{i->N})$按softmax在同一时刻所有轨迹上计算，代码中默认取$\lambda=0.1$。使低cost的轨迹获得高概率。直接计算会出现丢失精度的问题，如指数达到几千的级别，因此采用了缩放，变为了：

$$exp(-\frac{S_m(\tau_{i->N})-\min_m S_m(\tau_{i->N})}{\lambda(\max_m S_m(\tau_{i->N})-\min_m S_m(\tau_{i->N}))})$$

貌似改变了原先的值，但matlab中也这么使用，或许仍然是有效的。

式四五六

$\delta\boldsymbol{\theta}$是一个二维的矩阵，按时刻和基函数下标分。式四中对每个时刻的$\mathbf{M}_m(t)*\boldsymbol{\epsilon}_m(t)$取capacity条轨迹期望，每一时刻得到的都是$n_{bfs}$维度的向量。

式五中就按时间求和，$\delta\boldsymbol{\theta}$仅剩余$n_{bfs}$大小的一维矢量，式六进行更新。

代码中用`dtheta`代表$\delta\boldsymbol{\theta}$，用weight代表$\boldsymbol{\theta}$

trick

1. n_reuse：每次更新后并不完全清空经验池，会将cost最小的n_reuse个轨迹保留，清除其他。这样每次rollout的次数会减少，能够加快进度。示例中暂定n_reuse=0，因为发现有可能因为随机问题n_reuse个cost非常小，连rollout都不能再次得到更小的cost，从而n_reuse个始终在更新中占优且保持不变，导致$\delta\boldsymbol{\theta}$保持常数不变，参数走向无限更新。
2. eps冻结：每一次rollout，对权重施加的随机数eps，在同一基函数活跃的阶段内（比其他基函数此刻的值都大）保持不变，同时非激活的基函数eps都为0。能够加速学习。

参考

[1] Stulp F, Buchli J, Ellmer A, et al. Model-free reinforcement learning of impedance control in stochastic environments[J]. IEEE Transactions on Autonomous Mental Development, 2012, 4(4): 330-341.

[2] [Computational Learning and Motor Control Lab | Resources / Software browse (usc.edu)](), DMPs & PI2