loss_dp.m

Table of Contents

Summary: Double-Pendulum loss function; the loss is $1 - \exp(-0.5 * d^2 * a)$, where $a > 0$ and d^2 is the squared difference between the actual and desired position of the tip of the outer pendulum. The mean and the variance of the loss are computed by averaging over the Gaussian distribution of the state $p(x) = \mathcal{N}(m, s)$ with mean m and covariance matrix s. Derivatives of these quantities are computed when desired.

```
function [L, dLdm, dLds, S2] = loss_dp(cost, m, s)
```

Input arguments:

```
cost            cost structure
  .width        array of widths of the cost (summed together)
  .expl         (optional) exploration parameter
  .angle        (optional) array of angle indices
  .target       target state                                    [D x  1 ]
m               mean of state distribution                      [D x  1 ]
s               covariance matrix for the state distribution    [D x  D ]
```

Output arguments:

```
L     expected cost                                             [1 x  1 ]
dLdm  derivative of expected cost wrt. state mean vector        [1 x  D ]
dLds  derivative of expected cost wrt. state covariance matrix  [1 x D^2]
S2    variance of cost                                          [1 x  1 ]
```

High-Level Steps

1. Precomputations
2. Define static penalty as distance from target setpoint
3. Calculate loss

```
function [L, dLdm, dLds, S2] = loss_hc(cost, m, s)
```

Code

```
if isfield(cost,'width'); cw = cost.width; else cw = 1; end
if ~isfield(cost,'expl') || isempty(cost.expl); b = 0; else b =  cost.expl; end


% 1. Some precomputations
D0 = size(s,2); D = D0;                                 % state dimension
D1 = D0 + 2*length(cost.angle);           % state dimension (with sin/cos)
```

```
M = zeros(D1,1); M(1:D0) = m; S = zeros(D1); S(1:D0,1:D0) = s;
Mdm = [eye(D0); zeros(D1-D0,D0)]; Sdm = zeros(D1*D1,D0);
Mds = zeros(D1,D0*D0); Sds = kron(Mdm,Mdm);

% 2. Define static penalty as distance from target setpoint
Q = diag(ones(numel(m), 1));

target = [cost.target(:); gTrig(cost.target(:), 0*s, cost.angle)];
% 3. Calculate loss
L = 0; dLdm = zeros(1,D0); dLds = zeros(1,D0*D0); S2 = 0;
for i = 1:length(cw)                    % scale mixture of immediate costs
    cost.z = target; cost.W = Q/cw(i)^2;
  [r, rdM, rdS, s2, s2dM, s2dS] = lossSat(cost, M, S);

  L = L + r; S2 = S2 + s2;
  dLdm = dLdm + rdM(:)'*Mdm + rdS(:)'*Sdm;
  dLds = dLds + rdM(:)'*Mds + rdS(:)'*Sds;

  if (b~=0 || ~isempty(b)) && abs(s2)>1e-12
    L = L + b*sqrt(s2);
    dLdm = dLdm + b/sqrt(s2) * ( s2dM(:)'*Mdm + s2dS(:)'*Sdm )/2;
    dLds = dLds + b/sqrt(s2) * ( s2dM(:)'*Mds + s2dS(:)'*Sds )/2;
  end
end

% normalize
n = length(cw); L = L/n; dLdm = dLdm/n; dLds = dLds/n; S2 = S2/n;
```

Published with MATLAB® R2014a