```
classdef ARIC < handle
```

ARIC - Neuro-fuzzy controller class

ARIC properties

```
    properties
        A, b, c, D, e, f, v, s, z, rhat;
    end

    properties
        n, h, rho, rhoh, beta, betah, gamma;
    end

    methods
```

Class construction Input argument 'param' contains the settings the learning process.

```
        function obj = ARIC(param)
        % ARIC Construct an instance of this class
        % n = number of states + 1 (4 in the cart pole example)
        % h = number of rules (13 in the cart pole example)
        obj.n = param.aric.n;
        obj.h = param.aric.h;

        obj.rho = param.aric.rho;
        obj.rhoh = param.aric.rhoh;
        obj.beta = param.aric.beta;
        obj.betah = param.aric.betah;
        obj.gamma = param.aric.gamma;

        % Action-state Evaluation Method weights
        obj.A = rand(obj.n, obj.n); % SQUARE MATRIX
        obj.b = rand(1, obj.n); % ROW VECTOR WITH LENGTH N
        obj.c = rand(1, obj.n); % ROW VECTOR WITH LENGTH N

        % Action Selection Network
        obj.D = rand(obj.h, obj.n); % HxN MATRIX
        obj.e = rand(1, obj.n); % ROW VECTOR WITH LENGTH N
        obj.f = rand(1, obj.h); % ROW VECTOR WITH LENGTH H
        obj.v = rand(); % SCALAR
        obj.s = rand(); % SCALAR
        obj.z = rand(obj.h, 1); % COLUMN VECTOR WITH LENGHT H

        end
```

Not enough input arguments.

Error in ARIC (line 23)
obj.n = param.aric.n;

Master loop This function is called by Simulink in order to compute the control force based on the states

```
    function F = getControllerOutput(obj, x)
```

```
            % Action selection network
            u = obj.fuzzyInference(x); % Compute u based on fuzzy
 rules
            p = obj.confidenceComputation(x); % Compute confidence p
            obj.updateWeights(x); % Apply learning algorithms
            F = obj.stochasticActionModifier(u, p); % Compute
 Stochastic Object Modifier F
        end
```

Action-State Evaluation

```
        function [y, v] = stateEvaluation(obj, x)
            % Implementation of Maxime's AEN1 - neural network of AEN
            y = arrayfun(@sigmoid, obj.A*x);  % Neural network
            v = obj.b*x + obj.c*y;
        end

        function rhat = internalReinforcement(obj, x, vNew)
           % Implementation of Maxime's AEN2 - Computation of the
 internal reinforcement r_hat
           if abs(x(3)) > pi/15   %[rad]
               rhat = -1 - obj.v;
           elseif abs(x(1)) > 2.4  %[m]
               rhat = -1 - obj.v;
           else
               rhat = obj.gamma*vNew - obj.v;
           end

           % Update v
           obj.v = vNew;
        end

        function updateWeights(obj, x)
            % Compute necessary values from AEN
            [y, vNew] = obj.stateEvaluation(x);
            obj.rhat = obj.internalReinforcement(x, vNew);

            % Update AEN weights
            obj.b = obj.b + (obj.beta*obj.rhat*x)'; % b is stored as a
 row vector
            obj.c = obj.c + (obj.beta*obj.rhat*y)'; % c is stored as a
 row vector
            obj.A = obj.A + (obj.betah*obj.rhat).*sign(obj.c)'.*y.*(1
 - y)*x';

            % Update ASN weights
            obj.e = obj.e + (obj.rho*obj.rhat*obj.s*x)';
            obj.f = obj.f + (obj.rho*obj.rhat*obj.s*obj.z)';
            obj.D = obj.D + obj.rhoh*obj.rhat*obj.z.*(1-
obj.z).*sign(obj.f)'*obj.s*x';
        end
```

Action Selection Network methods

```
        function u = fuzzyInference(obj, x)
```

```
            % Based on Bart's FuzzyInference

            w = fuzzifier(x, obj.D);
            m = defuzzifier(w);

            utop=0;
            ubot=0;

            for i = 1:length(w)
                utop = utop + obj.f(i)*m(i)*w(i);
                ubot = ubot + w(i)*obj.f(i);
            end
            u = utop/ubot;
        end

        function p = confidenceComputation(obj, x)
            obj.z = arrayfun(@sigmoid, obj.D*x);
            p = obj.e*x + obj.f*obj.z;
        end

        function u_mod = stochasticActionModifier(obj, u, p)
           % Implementation of functions o and s in the paper

           q = (p + 1)/2;
           if u > 0
              u_mod = q;
           else
               u_mod = 1 - q;
           end

           if sign(u_mod) ~= sign(u)
               obj.s = 1 - p;
           else
               obj.s = -p;
           end
        end

    end

end
```

Published with MATLAB® R2019b