```
classdef NRUERxFD < matlab.System
    % 5G NR UR receiver class implemented in frequency domain
    properties
        % Configuration
        carrierConfig;   % Carrier configuration
        pdschConfig;     % Default PDSCH config
        waveformConfig;  % Waveform config

        % OFDM grid
        rxGrid;

        % Transport block data for last transmission
        targetCodeRate = 490/1024;  % Target code rate
        trBlkSizes;                 % Transport block size

        % Received data in last slots
        pdschEq;       % Equalized PDSCH symbols
        rxBits;        % RX bits

        % DLSCH decoder
        decDLSCH;

        % HARQ Process
        nharq = 8;      % number of HARQ processes

        % RV sequence.  This is the sequence that the TX will cycle
        % through in the RVs
        rvSeq = [0,3,2,1]';

        % TX parameters per HARQ process
        TBId;   % ID of the last TX packet
        rvInd;  % Index of the RV for the current transmission
        newDataAvail;   % If HARQ process can take new data
        txBits;         % Cell array of TX bits


    end
    methods
        function obj = NRUERxFD(carrierConfig, pdschConfig, ...
                varargin)
            % Constructor

            % Save the carrier and PDSCH configuration
            obj.carrierConfig = carrierConfig;
            obj.pdschConfig = pdschConfig;

            % Create the waveform configuration from the carrier
            % configuration
            obj.waveformConfig = nrOFDMInfo(obj.carrierConfig);

            % Set parameters from constructor arguments
            if nargin >= 1
                obj.set(varargin{:});
            end

            % Create DLSCH decoder
            obj.decDLSCH = nrDLSCHDecoder('MultipleHARQProcesses', true, ...
                'TargetCodeRate', obj.targetCodeRate, ...
                'LDPCDecodingAlgorithm', 'Layered belief propagation');
```

```
    end
end
methods (Access = protected)


    function stepImpl(obj, rxGrid, chanGrid, noiseVar, ...
            iharq, rv, newDat)
        % Demodulates and decodes one slot of data
        %
        % Parameters
        % ----------
        % iharq:  HARQ process ID
        % rv:  Rendundancy version
        % newData:  If data is new
        %
        % Note that the last three parameters would normally be
        % sent on the PDCCH


        % Get PDSCH received symbols and channel estimates
        % from received grid
        [pdschInd,pdschInfo] = nrPDSCHIndices(...
            obj.carrierConfig, obj.pdschConfig);
        [pdschRx, pdschHest] = nrExtractResources(pdschInd, rxGrid,...
            chanGrid);

        % Perform the MMSE equalization using the
        % nrEqualizeMMSE() function
        [obj.pdschEq,csi] = nrEqualizeMMSE(pdschRx,pdschHest,noiseVar);


        % TODO:  Get the LLRs with the nrPDSCHDecode() function.
        % Use carrier and PDSCH configuration, the equalized symbols,
        % and the noise variance, noiseVar.
        [dlschLLRs,rxSym] = nrPDSCHDecode(obj.carrierConfig,obj.pdschConfig,obj.pdschEq, noiseVar);

        % Scale LLRs by EbN0.
        % The csi value computed in the nrEqualizeMMSE()
        % function is csi = |pdschHest|^2 + noiseVar.
        % Also, the Eb/N0 = snrEq/Qm where Qm is the number of bits
        % per symbol and snrEq is the SNR after equalization,
        %
        %   snrEq = (|pdschHest|^2 + noiseVar)/noiseVar = csi/noiseVar
        %
        % Hence, Eb/N0 = csi/(noiseVar*Qm).
        % Since the LLRs from the nrPDSCHDecode function are
        % already scaled by 1/noiseVar, we multiply them by  csi/Qm.
        csi = nrLayerDemap(csi); % CSI layer demapping
        numCW = length(csi);
        for cwIdx = 1:numCW
            Qm = length(dlschLLRs{cwIdx})/length(rxSym{cwIdx}); % bits per symbol
            csi{cwIdx} = repmat(csi{cwIdx}.',Qm,1);   % expand by each bit per symbol
            dlschLLRs{cwIdx} = dlschLLRs{cwIdx} .* csi{cwIdx}(:);   % scale
        end

        % Compute the extra overhead from the PT-RS
        Xoh_PDSCH = 6*obj.pdschConfig.EnablePTRS;

        % Calculate the transport block size based on the PDSCH
        % allocation and target code rate
```

```
            obj.trBlkSizes = nrTBS(obj.pdschConfig.Modulation,obj.pdschConfig.NumLayers,...
                numel(obj.pdschConfig.PRBSet),pdschInfo.NREPerPRB,...
                obj.targetCodeRate,Xoh_PDSCH);
            obj.decDLSCH.TransportBlockLength = obj.trBlkSizes;

            % Reset the soft buffer for all codewords
            if newDat
                for cwIdx = 1:numCW
                    obj.decDLSCH.resetSoftBuffer(cwIdx, iharq-1);
                end
            end

            % TODO:  Decode the bits with the obj.decDLSCH() method.
            % Use the scaled LLRs from above.
            obj.rxBits = obj.decDLSCH(dlschLLRs,obj.pdschConfig.Modulation,...
                                      obj.pdschConfig.NumLayers,rv,iharq-1);
        end

    end
end
```

```
Not enough input arguments.

Error in NRUERxFD (line 44)
            obj.carrierConfig = carrierConfig;
```

Published with MATLAB® R2020b