```
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;

        % Channel estimation parameters
        sigFreq = 7;  % Channel smoothing in freq
        sigTime = 3;  % Channel smoothing in time
        lenFreq = 21;  % Filter length in freq
        Wtime;

        % Test bit parameters
        bitsPerSym = 2;

        % Channel and noise estimate
        chanEstGrid;
        chanEstDmr;
        noiseEst;

        % RX symbols and estimated channel on the PDSCH
        pdschChan;
        pdschSym;


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


        end

        function chanEst(obj, rxGrid)
            % Computes the channel estimate
            % TODO:  Get the TX DM-RS symbols and indices
            dmrsSymTx = nrPDSCHDMRS(obj.carrierConfig, obj.pdschConfig);
            dmrsInd = nrPDSCHDMRSIndices(obj.carrierConfig, obj.pdschConfig);

            rxGrid = rxGrid(:);
            % TODO:  Get RX symbols on the DM-RS
            dmrsSymRx = rxGrid(dmrsInd);

            % TODO:  Get the raw channel estimate
            chanEstRaw = dmrsSymRx./dmrsSymTx;

            % Get the symbol numbers and sub-carrier indices of the
            % DM-RS symbols from the DM-RS
            % dmrsSymNum(i) = symbol number for the i-th DM-RS symbol
            nsc = obj.carrierConfig.NSizeGrid*12;
            tot_dmrs_sym = length(dmrsSymRx);
            dmrsSymNum = zeros(tot_dmrs_sym,1);
            dmrsSymNum(1:tot_dmrs_sym/2) = 3;
            dmrsSymNum(tot_dmrs_sym/2+1:end) = 12;

            % dmrsScInd(i) = sub-carrier index for the i-th DM-RS symbol
            sub_indices = obj.pdschConfig.DMRS.DMRSSubcarrierLocations+1;
            tot_sub_idx = [];
            for i=1:length(sub_indices)
                sub_i = sub_indices(i);
                idx_sub = sub_i:12:nsc;
                tot_sub_idx = [tot_sub_idx idx_sub];
            end
%             idx_sub_1 = sub_indices(1):12:nsc;
%             idx_sub_2 = sub_indices(2):12:nsc;
%             idx_sub_3 = sub_indices(3):12:nsc;
%             idx_sub_4 = sub_indices(4):12:nsc;
            tot_sub_idx = sort(tot_sub_idx);

            dmrsScInd = zeros(tot_dmrs_sym,1);
            dmrsScInd(1:tot_dmrs_sym/2) = tot_sub_idx;
            dmrsScInd(tot_dmrs_sym/2+1:end) = tot_sub_idx;

            % TODO:  Get the list of all symbol numbers on which DM-RS was
            % transmitted.  You can use the unique command
            dmrsSymNums = unique(dmrsSymNum);
            ndrmsSym = length(dmrsSymNums);

            % We first compute the channel and noise
            % estimate on each of the symbols on which the DM-RS was
            % transmitted.  We will store these in two arrays
            %   chanEstDmrs(k,i) = chan est on sub-carrier k in DM-RS
            %       symbol i
            %   noiseEstDmrs(i) = noise est for DM-RS symbol i
            chanEstDmrs = zeros(nsc, ndrmsSym);
            noiseEstDmrs  = zeros(ndrmsSym, 1);

            % Loop over the DM-RS symbols
            for i = 1:ndrmsSym

                % TODO:  Find the indices, k, in which the DM-RS
                % dmrsSymNum(k)= dmrsSymNum(i).
                I = dmrsSymNum == dmrsSymNums(i);

                % TODO:  Get the sub-carrier indices and raw channel
                % channel estimate for these RS on the symbol
                ind = dmrsScInd(I);
                raw = chanEstRaw(I);

                % TODO:  Use kernelReg to compute the channel estimate
                % on that DM-RS symbol.  Use the lenFreq and sigFreq
                % for the kernel length and sigma.
                chanEstDmrs(:,i) = kernelReg(ind, raw, nsc, obj.lenFreq, obj.sigFreq);

                % TODO:  Compute the noise estimate on the symbol
                % using the residual method
                noiseEstDmrs(i) = mean(abs(dmrsSymRx(I) - chanEstDmrs(ind,i).*dmrsSymTx(I)).^2);

            end
            obj.chanEstDmr = chanEstDmrs;
            % TODO:  Find the noise estimate over the PDSCH by
            % averaging noiseEstDmrs
            obj.noiseEst = mean(noiseEstDmrs);

            % TODO:  Finally, we interpolate over time.
            % We will use an estimate of the form
            %    obj.chaneEstGrid = chanEstDrms*W
            % so that
            %    chanEstGrid(k,j) = \sum_i chanEstDmrs(k,i)*W(i,j)
            %
            % We use a kernel estimator
            %
            %     W(i,j) = W0(i,j) / \sum_k W0(k,j)
            %     W0(k,j) = exp(-D(k,j)^2/(2*obj.sigTime^2))
            %     D(k,j) = dmrsSymNum(k) - j
            %
            j = (1:14);
            D = dmrsSymNums - j;
            W0 = exp(-(D.^2/(2*obj.sigTime^2)));
            W = W0 ./ sum(W0,1);

            % Save the time interpolation matrix
            obj.Wtime = W;

            % Create the channel estimate grid
            obj.chanEstGrid = chanEstDmrs*W;

        end
    end
    methods (Access = protected)


        function rxBits = stepImpl(obj, rxGrid, chanGrid, noiseVar)
            % Performs channel estimation, equalization and
            % symbol demodulation for one slot of data.
            %
            % Input
            % -----
            % rxGrid:  Received symbols in one slot
            % chanGrid:  Optional true channel estimate.
            % noiseVar:  Optional true noise variance
            %
            % If (chanGrid, noiseVar) are supplied the function skips
            % the channel estimate.  This is useful for testing a true
            % channel estimate without channel estimation error.

            if nargin >= 3
```

```
                % Set the estimated channel and noise to the supplied
                % values if provided.
                obj.chanEstGrid = chanGrid;
                obj.noiseEst = noiseVar;
            else

                % Compute the channel and noise estimate
                obj.chanEst(rxGrid);
            end

            % Get indices on where the PDSCH is allocated
            pdschInd = nrPDSCHIndices(obj.carrierConfig, obj.pdschConfig);

            % TODO:  Get the PDSCH symbols and channel on the indicies
            obj.pdschSym = rxGrid(pdschInd);
            obj.pdschChan = obj.chanEstGrid(pdschInd);

            % TODO:  Perform the MMSE equalization
            obj.pdschSymEq = conj(obj.pdschChan).*obj.pdschSym./(abs(obj.pdschChan).^2 + obj.noiseEst);

            % Demodulate the symbols
            M = 2^obj.bitsPerSym;
            rxBits = qamdemod(obj.pdschSymEq, M, 'OutputType', 'bit',...
                'UnitAveragePower', true);
        end

    end
end
```

```
Not enough input arguments.

Error in NRUERxFD (line 42)
            obj.carrierConfig = carrierConfig;
```

Published with MATLAB® R2020b