```matlab
classdef ArrayPlatform < matlab.System
    % ArrayWithAxes.  Class containing an antenna array and axes.
    properties

        fc = 28e9;  % Carrier frequency

        % Element within each array.
        elem = [];

        % Gridded interpolant object for the element gain in linear scale
        elemGainInterp = [];

        % Antenna array.
        arr = [];

        % Steering vector object
        svObj = [];

        % Azimuth and elevation angle of the element peak directivity
        axesAz = 0;
        axesEl = 0;

        % Axes of the element local coordinate frame of reference
        axesLoc = eye(3);

        % Velocity vector in 3D in m/s
        vel = zeros(1,3);

        % Position in m
        pos = zeros(1,3);

        % Normalization matrix
        Qinv = [];
    end

    methods
        function obj = ArrayPlatform(varargin)
            % Constructor

            % Set key-value pair arguments
            if nargin >= 1
                obj.set(varargin{:});
            end
        end



        function computeNormMatrix(obj)
            % The method performs two key tasks:
            % * Measures the element pattern and creates a gridded
            %   interpolant object to interpolate the values at other
            %   angles
            % * Compute the normalization matrix to account for mutual
            %   coupling

            % TODO:  Get the pattern for the element using the elem.pattern
            % method
            [elemGain,az,el] = obj.elem.pattern(obj.fc,'Type','directivity');
```

```matlab
        % TODO:  Create the gridded interpolant object
        % for element gain
        obj.elemGainInterp = griddedInterpolant({el,az},elemGain);

        % Get a vector of values with the elements az(i) and el(j).
        [azMat, elMat] = meshgrid(az, el);
        azVal = azMat(:);
        elVal = elMat(:);
        elemGainVal = elemGain(:);

        % TODO:  Create a steering vector object with the array
        obj.svObj = phased.SteeringVector('SensorArray', obj.arr);

        % TODO:  Get the steering vectors
        Sv0 = obj.svObj(obj.fc, [azVal elVal]');

        % TODO:  Compute un-normalized spatial signature
        % by multiplying the steering vectors with the element gain
        elemGainLin = db2mag(elemGainVal);
        SvNoNorm = Sv0.*elemGainLin';

        % TODO:  Compute the normalization matrix by integrating the
        % un-normalized steering vectors
        cosel = cos(deg2rad(elVal));
        Q = (1/length(elVal))*(SvNoNorm.*cosel')*(SvNoNorm') / mean(cosel);

        % Ensure matrix is Hermitian
        Q = (Q+Q')/2;

        % TODO:  Save the inverse matrix square root of Q
        obj.Qinv = sqrtm(Q);

    end

    function alignAxes(obj,az,el)
        % Aligns the axes to given az and el angles

        % Set the axesAz and axesEl to az and el
        obj.axesAz = az;
        obj.axesEl = el;

        % Creates axes aligned with az and el
        obj.axesLoc = azelaxes(az,el);
    end

    function dop = doppler(obj,az,el)
        % Computes the Doppler shift of a set of paths
        % The angles of the paths are given as (az,el) pairs
        % in the global frame of reference.

        % Finds unit vectors in the direction of each path
        npath = length(el);
        [u1,u2,u3] = sph2cart(deg2rad(az),deg2rad(el),ones(1,npath));
        u = [u1; u2; u3];

        % Compute the Doppler shift of each path via an inner product
        % of the path direction and velocity vector.
        vcos = obj.vel*u;
        vc = physconst('lightspeed');
        dop = vcos*obj.fc/vc;
```

```
        end

        function releaseSV(obj)
            % Creates the steering vector object if it has not yet been
            % created.  Otherwise release it.  This is needed since the
            % sv object requires that it takes the same number of
            % inputs each time.
            if isempty(obj.svObj)
                obj.svObj = phased.SteeringVector('SensorArray',obj.arr);
            else
                obj.svObj.release();
            end

        end

        function n = getNumElements(obj)
            % Gets the number of elements
            n = obj.arr.getNumElements();

        end

        function elemPosGlob = getElementPos(obj)
            % Gets the array elements in the global reference frame

            % Get the element position in the local reference frame
            elemPosLoc = obj.arr.getElementPosition();

            % Convert to the global reference frame
            elemPosGlob = local2globalcoord(elemPosLoc, 'rr', ...
                zeros(3,1), obj.axesLoc) + reshape(obj.pos,3,1);
        end

    end

    methods (Access = protected)

        function setupImpl(obj)
            % setup:  This is called before the first step.
            obj.computeNormMatrix();


        end

        function releaseImpl(obj)
            % release:  Called to release the object
            obj.svObj.release();
        end

       function [Sv, elemGain] = stepImpl(obj, az, el, relSV)
            % Gets normalized steering vectors and element gains for a set of angles
            % The angles az and el should be columns vectors along which
            % the outputs are to be computed.
            % If the relSV == true, then the steering vector object is
            % released.  This is needed in case the dimensions of the past
            % call are the different from the past one

            % Release the SV
            if nargin < 4
                relSV = true;
            end
            if relSV
```

```
            obj.releaseSV();
        end

        % TODO:  Convert the global angles (az, el) to local
        % angles (azLoc, elLoc).  Use the
        % global2localcoord() method with the 'ss' option.
        locCoord = global2localcoord([az; el; ones(size(az));],'ss',[0;0;0],obj.axesLoc);
        azLoc = locCoord(1,:);
        elLoc = locCoord(2,:);

        % TODO: Get the SV in the local coordinates
        Sv0 = obj.svObj(obj.fc, [azLoc(:) elLoc(:)]');

        % TODO:  Get the directivity gain of the element from the
        % local angles.
        elemGain = obj.elemGainInterp(elLoc, azLoc);
        elemGainLin = db2mag(elemGain(:));
        SvNoNorm = Sv0.*elemGainLin';

        % TODO:  Compute the normalized steering vectors

        Sv = obj.Qinv \ SvNoNorm;

    end

end


end
```

```
ans =

  ArrayPlatform with properties:

               fc: 2.8000e+10
             elem: []
    elemGainInterp: []
              arr: []
            svObj: []
           axesAz: 0
           axesEl: 0
          axesLoc: [3×3 double]
              vel: [0 0 0]
              pos: [0 0 0]
             Qinv: []
```

Published with MATLAB® R2020b