```
classdef Burner < handle
    %BURNER Burner assembly

    properties (SetAccess = private)
        segments = {}; % Cell array of burner segments
        maxStep = 1e-3; % [m] Maximum solution step size
        states; % State array from solutios
        startFlow; % Starting flow of the burner
        injectionFunc; % Function handle of injection function

        width;
        startHeight;
        lengths;
        angles = []; % [deg] Array of angles for each element

        h; % Fuel heating value

        cea;
    end

    methods

        function obj = Burner()
            obj.cea = nasa.CEARunner();
        end

        function setHeatingValue(obj, h)
            obj.h = h;
        end

        function setMaxStep(obj, step)
            obj.maxStep = step;
        end

        function setInjectionFunc(obj, fh)
            assert(isa(fh, 'function_handle'), 'Injection function
 must be a handle!');
            obj.injectionFunc = fh;
        end

        function setStartFlow(obj, flow)

 assert(isa(flow, 'aeroBox.flowFields.FlowElement'), 'Starting flow
 must be burner flow type!');
            obj.startFlow = flow;
        end

        function setGeometry(obj, w, h, l, a)
            assert(numel(l) == numel(a), 'Must have consistant
 dimensions!');
            obj.angles = a;
            obj.width = w;
```

```
            obj.startHeight = h;
            obj.lengths = l;
        end

        function setup(obj)
            % Sets up the burner for solving

            % Make assertions to verify all componenets are ready to
setup
            assert(~isempty(obj.injectionFunc), 'Missing injection
function!');
            assert(~isempty(obj.startFlow), 'Missing starting flow!');
            assert(~isempty(obj.angles), 'Missing segment angles!');
            assert(~isempty(obj.h), 'Missing fuel heating value!');

            for i = 1:numel(obj.angles)
                % Create the burner segment
                obj.segments{i} = aae550.final.BurnerSegment('cea',
obj.cea);

                % Set the geometry
                obj.segments{i}.geometry.setWidth(obj.width);
                if i == 1

obj.segments{i}.geometry.setHeight(obj.startHeight);
                else
                    obj.segments{i}.geometry.setHeight(obj.segments{i
- 1}.geometry.getHeight(obj.lengths(i - 1)));
                end
                obj.segments{i}.geometry.setLength(obj.lengths(i));
                obj.segments{i}.geometry.setAngle(obj.angles(i));

                % Set the injection function
                obj.segments{i}.setInjectionFunc(obj.injectionFunc);

                % Set the heating value of the fuel
                obj.segments{i}.setHeatingValue(obj.h);

                % Set to use global injection
                obj.segments{i}.setGlobalInjection();

            end
        end

        function solve(obj)
            % Solves throught the combustor
            obj.setup(); % Sets up the combustor
            x = 0;
            tempFlow = obj.startFlow;
            obj.states = [];

            totalLength = sum(obj.lengths);
            waitString = @(x) sprintf('%0.2f m of %0.2f m', x,
totalLength);
```

```
            wh = waitbar(0, waitString(x));
            updateFH = @(x) waitbar(x / totalLength, wh,
 waitString(x));
            for i = 1:numel(obj.segments)
                % Solve the current burner segment
                obj.segments{i}.setWaitFH(updateFH, max(totalLength /
 1000, 0.01));
                obj.segments{i}.setFlowElement(tempFlow);
                [tempFlow, newStates] =
 obj.segments{i}.solve(ceil(obj.lengths(i) / obj.maxStep), x);
                % Get the length of the combustor for the next
 iteration
                x = x + obj.segments{i}.geometry.length;
                obj.states = [obj.states newStates];
            end
            close(wh);
        end

        function plotGeometry(obj)
            % Plots the burner segments
            fh = figure();
            x = [];
            y = [];
            xx = 0;
            for i = 1:numel(obj.segments)
                [xnew, ynew] = obj.segments{i}.getPlotArrays();
                x = [x xnew + xx];
                y = [y ynew];
                xx = xx + obj.lengths(i);
            end
            plot(x, y);
        end

    end

end
```

Published with MATLAB® R2015b