1 Object Oriented Programming (Draft)

Repository of Codes

All codes of this section are available open source @ https://github.com/ahmed-rashed/Cross-SectionProperties_remote

`struct`

```
>> A.var1=9
A =
  struct with fields:
    var1: 9

>> A.var2=10
A =
  struct with fields:
    var1: 9
    var2: 10

>> A.Var2=10
A =
  struct with fields:
    var1: 9
    var2: 10
    Var2: 10
```

Why using a Class? Why not just use a `struct`?[*]

- `struct` only have field-names
  - There is no control over assignment (or access) to struct field-names
    - For example, if a user incorrectly spells the field-name, MATLAB will simply add a new field to the struct
  - It is possible to assign wrong data type to a struct field-name
- On the other hand, classes
  - return an error if the user misspelled a field name
  - can validate individual field value/type when assigned
  - can restrict access to particular fields
    - For example, allow reading, disallow changing
  - are easy to identify with the " whos " and " class " functions and the Workspace browser
  - are easy to reuse
    - You can easily inherit a class and create new subclasses with additional properties

Object Oriented Programming

- Refer to appendix ??.

[*] https://alaakhreis.com/tech/oop-matlab/

Tutorials and Documentation

- A concise tutorial is available at https://alaakhreis.com/tech/oop-matlab/
- www.mathworks.com/help/matlab/matlab_oop/developing-classes-typical-workflow.html
- www.mathworks.com/help/matlab/sample-classes.html
- [?]

Classes in Matlab

- According to Matlab syntax,
  - class member variables are called ***properties***
  - class member functions are called ***methods***

Key Terms

Class definition Description of what is common to every instance of a class.

Properties Data storage for class instances

Methods Special functions that implement operations that are usually performed only on instances of the class

Events Messages defined by classes and broadcast by class instances when some specific action occurs

Attributes Values that modify the behavior of properties, methods, events, and classes

Listeners Objects that respond to a specific event by executing a callback function when the event notice is broadcast

Objects Instances of classes, which contain actual data values stored in the objects' properties

Subclasses Classes that are derived from other classes and that inherit the methods, properties, and events from those classes (subclasses facilitate the reuse of code defined in the superclass from which they are derived).

Superclasses Classes that are used as a basis for the creation of more specifically defined classes (that is, subclasses).

Packages Folders that define a scope for class and function naming

Sample Class Definition

Listing 1: cArea_simple.m

```
classdef cArea_simple   %The file-name must be the same as the class-name
    %This class takes the propeties of an area (A,Iy,Iz,Iyz), and calculates the properties of this area.

    % The attributes used to calcute the area properties
    properties
        A
        Iy=0
        Iz=0
        Iyz=0
    end

    methods
        % Constructor
        function oThisArea=cArea_simple(A,Iy,Iz,Iyz)
            if nargin>=1
                oThisArea.A=A;
            end

            if nargin>=2
                oThisArea.Iy=Iy;
            end

            if nargin>=3
                oThisArea.Iz=Iz;
            end

            if nargin==4
                oThisArea.Iyz=Iyz;
            end
        end

        function Ip=Ip(oThisArea)
            Ip=oThisArea.Iy+oThisArea.Iz;
        end

        function alpha1=alpha1(oThisArea)
            alpha1=atan2(-oThisArea.Iyz,(oThisArea.Iy-oThisArea.Iz)/2);
        end

        function I_34=I_34(oThisArea)
            I_34=sqrt(((oThisArea.Iy-oThisArea.Iz)/2)^2+(-oThisArea.Iyz)^2);
        end

        function I_3=I_3(oThisArea)
            I_3=(oThisArea.Iy+oThisArea.Iz)/2;
        end

        function I_4=I_4(oThisArea)
            I_4=oThisArea.I_3;
        end

        function I1=I1(oThisArea)
```

```
            I1=oThisArea.I_3+oThisArea.I_34;
        end

        function I2=I2(oThisArea)
            I2=oThisArea.I_3-oThisArea.I_34;
        end

        function alpha_3=alpha_3(oThisArea)
            alpha_3=pi/4-oThisArea.alpha1;
        end

        function rho_y=rho_y(oThisArea)
            rho_y=sqrt(oThisArea.Iy/oThisArea.A);
        end

        function rho_z=rho_z(oThisArea)
            rho_z=sqrt(oThisArea.Iz/oThisArea.A);
        end

        function Sy=Sy(oThisArea,z_max)
            Sy=oThisArea.Iy/abs(z_max);
        end

        function Sz=Sz(oThisArea,y_max)
            Sz=oThisArea.Iz/abs(y_max);
        end

        function oNewArea=rotatedArea(oThisArea,alpha_rad)
            temp_col=TransMatrix(-alpha_rad)*[oThisArea.Iy;oThisArea.Iz
               ;-oThisArea.Iyz];
            oNewArea=cArea_simple(oThisArea.A,temp_col(1),temp_col(2),
               temp_col(3));
        end
    end
end
```

Using the Class

```
>> oArea1=cArea_simple
oArea1 =
  cArea_simple with properties:
      A: []
     Iy: 0
     Iz: 0
    Iyz: 0

>> oArea1=cArea_simple(5)
oArea1 =
  cArea_simple with properties:
      A: 5
     Iy: 0
     Iz: 0
    Iyz: 0

>> oArea1=cArea_simple(5,6)
oArea1 =
```

```
  cArea_simple with properties:
      A: 5
     Iy: 6
     Iz: 0
    Iyz: 0

>> oArea1=cArea_simple(5,6,7)
oArea1 =
  cArea_simple with properties:
      A: 5
     Iy: 6
     Iz: 7
    Iyz: 0

>> oArea1=cArea_simple(5,6,7,8)
oArea1 =
  cArea_simple with properties:
      A: 5
```

```
        Iy: 6
        Iz: 7
       Iyz: 8

>> I_p(oArea1)
ans =
       13

>> oArea1.I_p()
ans =
       13

>> oArea1.I_p
ans =
       13

>> oArea1.Sy(5)
ans =
```

```
    1.2000

>> oArea1.rotatedArea(pi/2)
ans =
  cArea_simple with properties:
        A: 5
       Iy: 7.0000
       Iz: 6.0000
      Iyz: 8

>> oArea2=oArea1.rotatedArea(pi/2)
oArea2 =
  cArea_simple with properties:
        A: 5
       Iy: 7.0000
       Iz: 6.0000
      Iyz: 8
```

1.1 Defining a Class

Defining a Class

```
classdef (Attribute1 = value1, Attribute2 = value2,...) ClassName
    ...
end
```

Class Attributes

- The complete list of attributes is available at www.mathworks.com/help/matlab/matlab_oop/class-attributes.html

Abstract

- An abstract class cannot be instantiated itself, but serves as a way to define a unified interface for use by its subclasses
- When a property or method is attributed as abstract, MATLAB automatically attributes its class as abstract
- Details are available in www.mathworks.com/help/matlab/matlab_oop/abstract-classes-html

AllowedSubclasses List classes that can subclass this class. Check details at www.mathworks.com/help/matlab/matlab_oop/control-allowed-subclasses.html

Hidden If true, this class does not appear in the output of the superclasses or help functions.

Sealed If true, this class cannot be subclassed.

1.2 Properties

- A default property value can be assigned to the properties in the properties definition block, as was implemented in code 1

Property Attributes[*]
Most popular property attributes are:

- **Abstract** Logical value. Default: false
  - If true, the property has no implementation, but a concrete subclass must redefine this property without Abstract being set to true.
  - When a property[†] is attributed as abstract, MATLAB automatically attributes its class as abstract
- **SetAccess** Enumeration value. Default: public
  - Restricts the access from where an attribute % can be set. Possible values:
    - public — unrestricted access
    - protected — access from class or derived classes
    - private — access by class members only
- **GetAccess** Same as SetAcces, but for reading property values.
- **Hidden** Determines whether the property should be shown in a property list
- **Dependent** A dependent property is calculated by a class method whenever the dependent property is requested
- **Constant**
  - A constant property have the same value in all instances of the class
    - That is, a constant property is static
  - A constant property can only be initialized in its defining property block, its value cannot be changed later on
    - However, there is a workaround in `www.mathworks.com/help/matlab/matlab_oop/static-data.html#buvyy2x`

Property Access Methods

- `www.mathworks.com/help/matlab/matlab_oop/property-access-methods.html`

[*] [`www.mathworks.com/help/matlab/matlab_oop/property-attributes.html`]

[†] or method

1.3 Methods

There are specialized kinds of methods[*]:

Constructor method create instances of the class.

- A constructor method must have the same name as the class
- Typically, constructor methods accept input arguments to assign the properties and return an initialized object
- The constructor must return the object it creates as an output argument
  - This output argument is initialized before executing the first line of the constructor
  - You can call other class methods from the constructor because the object is already initialized
- To be able to create object arrays, the constructor must support calling with no input argument[†]

Ordinary methods [‡]define functions that operate on objects of the class.

- Ordinary methods can:
  - perform computations
  - overload MATLAB built-in functions
  - call other methods and functions
  - return modified objects
- One of the input arguments must be the parent object
- Same as ordinary Matlab functions, ordinary methods cannot modify input arguments.
- Either of the following statements can be used to call an ordinary method (where "`obj`" is an object of the parent class that defined the "`method1`" method:
  - obj.method1(arg)
  - method1(obj,arg)

Destructor methods are called automatically when the object is destroyed, for example if you call delete(object) or there are no longer any references to the object. See `www.mathworks.com/help/matlab/matlab_oop/handle-class-destructors.html`

Property access methods enable a class to define code to execute whenever a property value is queried or set. See `www.mathworks.com/help/matlab/matlab_oop/property-access-methods.html`

Conversion methods are overloaded constructor methods from other classes that enable your class to convert its own objects to the class of the overloaded constructor. For example, if your class implements a double method, then this method is called instead of the double class constructor to convert your class object to a MATLAB double object. See `www.mathworks.com/help/matlab/matlab_oop/converting-objects-to-another-class.html`

[*] [`www.mathworks.com/help/matlab/matlab_oop/how-to-use-methods.html`]

[†] C.f. sec. 1.5.

[‡] [`www.mathworks.com/help/matlab/matlab_oop/ordinary-methods.html`]

Method Attributes[§]

Some important attributes are:

Abstract

- When a method[*] is attributed as abstract, MATLAB automatically attributes its class as abstract
- An abstract method has no implementation, only a syntax line (e.g., `[a,b] = myMethod(x,y)`)
    - Subclasses are not required to define the same number of input and output arguments

Access Determines what code can call this method:

> **public** Unrestricted access
>
> **protected** Access from methods in class or subclasses
>
> **private** Access by class methods only (not from subclasses)

- Check more details at www.mathworks.com/help/matlab/matlab_oop/selective-access- html

Hidden When false, the method name shows in the list of methods displayed using the `methods` or `methodsview` commands. If set to true, the method name is not included in these listings and `ismethod` does not return true for this method name.

Sealed If true, the method cannot be redefined in a subclass.

Static

Static methods are associated with a class, but not with specific instances of this class. These methods do not require an object of the class as an input argument. Hence static methods:

- can be called without creating an object of the class
- cannot modify/use properties/methods of the class
    - but they can still modify/use constant properties of the class

1.4 Documenting Classes

The doc command[†]

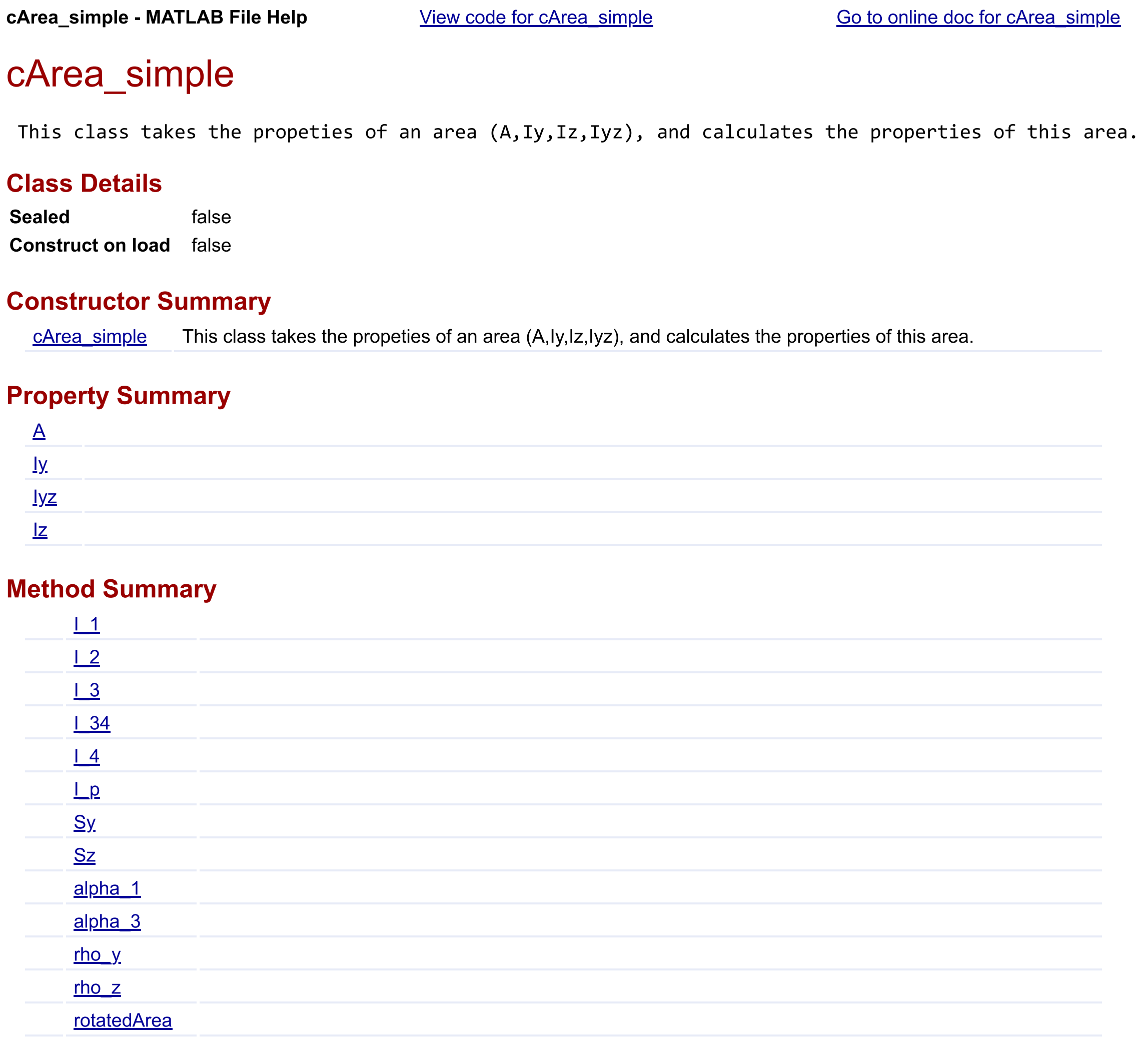
The doc command can be used to display the information, that MATLAB derives from the class definition, in HTML format.

- For example, `>> doc cArea_simple` yields

[§] [www.mathworks.com/help/matlab/matlab_oop/method-attributes.html]

[*] or property

[†] [www.mathworks.com/help/matlab/matlab_prog/create-help-for-classes.html]



cArea_simple - MATLAB File Help View code for cArea_simple Go to online doc for cArea_simple

cArea_simple

```
This class takes the propeties of an area (A,Iy,Iz,Iyz), and calculates the properties of this area.
```

Class Details

Sealed	false
Construct on load	false

Constructor Summary

cArea_simple	This class takes the propeties of an area (A,Iy,Iz,Iyz), and calculates the properties of this area.

Property Summary

- A
- Iy
- Iyz
- Iz

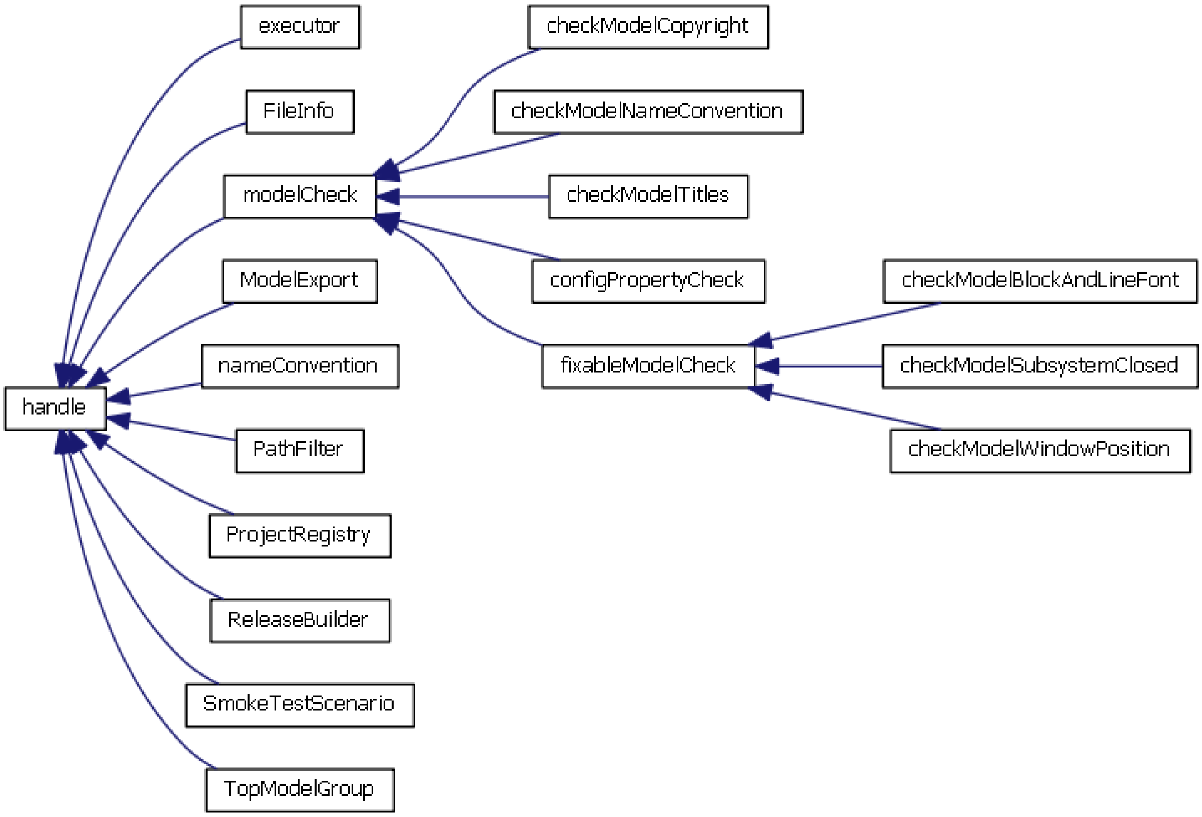
Method Summary

- I_1
- I_2
- I_3
- I_34
- I_4
- I_p
- Sy
- Sz
- alpha_1
- alpha_3
- rho_y
- rho_z
- rotatedArea

Doxygen[*]

If you need more powerful documentation features such as dependency graphs, you may check `www.mathworks.com/matlabcentral/fileexchange/25925-using-doxygen-with-matlab`

[*] Check appendix ??.



1.5 Object Arrays

[?, ch. 10]

Preallocating an Object Array
To Preallocate an object array, assign the last element of the array.

- MATLAB calls the class constructor with no arguments to initialize the previous array elements
- Therefore, the constructor must support being called with no input argument
  - A simple solution is to test `nargin` and let the case when `nargin==0` execute no code, but not error
- Then, you can assign the array elements one by one

```
>> oAreaArray(2,2)=cArea_simple(3.2)
oAreaArray =
  2x2 cArea_simple array with properties:
    A
    Iy
    Iz
    Iyz

>> oAreaArray(2,1)=cArea_simple(1.4)
oAreaArray =
  2x2 cArea_simple array with properties:
    A
    Iy
    Iz
    Iyz
```

Properties of an Object Array

Referencing a property of an object array using dot notation returns a comma-separated list* of the property values.

```
>> oAreaArray.A
ans =
     []
ans =
    1.4000
ans =
     []
ans =
    3.2000

>> oAreaArray(1,1).A,oAreaArray(2,1).
   A,oAreaArray(1,2).A,oAreaArray
   (2,2).A
ans =
     []
ans =
    1.4000
ans =
     []
ans =
    3.2000
```

```
>> oAreaArray.Iy
ans =
     0
ans =
     0
ans =
     0
ans =
     0

>> oAreaArray(1,1).Iy,oAreaArray(2,1)
   .Iy,oAreaArray(1,2).Iy,oAreaArray
   (2,2).Iy
ans =
     0
ans =
     0
ans =
     0
ans =
     0
```

- To convert the elements of a comma-separated list to a row vector, enclose it in `[]`

```
>> [oAreaArray.A]
ans =
    1.4000    3.2000
```

```
>> [oAreaArray.Iy]
ans =
     0     0     0     0
```

- To create a cell array from the comma-separated list, enclose it in `{}`

```
>> {oAreaArray.A}
ans =
  1x4 cell array
    {0x0 double}    {[1.4000]}    {0x0 double}    {[3.2000]}

>> {oAreaArray.Iy}
ans =
  1x4 cell array
    {[0]}    {[0]}    {[0]}    {[0]}
```

*C.f. sec. ??.

Designing the Constructor and Methods for Initializing/Handling Object Arrays

Listing 2: cArea.m

```
classdef cArea  %The file-name must be the same as the class-name
    %This class takes the propeties of an area (A,Iy,Iz,Iyz), and
       calculates the properties of this area.

    % The attributes used to calcute the area properties
    properties
        A(1,1)
        Iy(1,1)=0
        Iz(1,1)=0
        Iyz(1,1)=0
    end

    methods
        % Constructor
        function oThisArea_arr=cArea(A_arr,Iy_arr,Iz_arr,Iyz_arr)
            if nargin>=1
                N=numel(A_arr);
                A_arr_size=size(A_arr);
                dims_c=num2cell(A_arr_size);
                oThisArea_arr(dims_c{:})=oThisArea_arr;
                for n=1:N
                    oThisArea_arr(n).A=A_arr(n);
                end
            end

            if nargin>=2
                if any(size(Iy_arr)~=A_arr_size),error('Iy_arr␣and␣A_arr
                   ␣must␣have␣identical␣size.'),end
                for n=1:N
                    oThisArea_arr(n).Iy=Iy_arr(n); %#ok<AGROW>
                end
            end

            if nargin>=3
                if any(size(Iz_arr)~=A_arr_size),error('Iz_arr␣and␣A_arr
                   ␣must␣have␣identical␣size.'),end
                for n=1:N
                    oThisArea_arr(n).Iz=Iz_arr(n); %#ok<AGROW>
                end
            end

            if nargin==4
                if any(size(Iyz_arr)~=A_arr_size),error('Iyz_arr␣and␣
                   A_arr␣must␣have␣identical␣size.'),end
                for n=1:N
                    oThisArea_arr(n).Iyz=Iyz_arr(n); %#ok<AGROW>
                end
            end
        end

        function Ip_arr=Ip(oThisArea_arr)
            Ip_arr=reshape([oThisArea_arr.Iy]+[oThisArea_arr.Iz],size(
               oThisArea_arr));
        end

        function alpha1_arr=alpha1(oThisArea_arr)
            alpha1_arr=reshape(atan2(-[oThisArea_arr.Iyz],([
               oThisArea_arr.Iy]-[oThisArea_arr.Iz])/2),size(
               oThisArea_arr));
        end

        function I_34_arr=I_34(oThisArea_arr)
            I_34_arr=reshape(sqrt((([oThisArea_arr.Iy]-[oThisArea_arr.Iz
               ])/2).^2+(-[oThisArea_arr.Iyz]).^2),size(oThisArea_arr));
        end

        function I_3_arr=I_3(oThisArea_arr)
            I_3_arr=reshape(([oThisArea_arr.Iy]+[oThisArea_arr.Iz])/2,
               size(oThisArea_arr));
        end

        function I_4_arr=I_4(oThisArea_arr)
            I_4_arr=oThisArea_arr.I_3;
        end

        function I1_arr=I1(oThisArea_arr)
            I1_arr=oThisArea_arr.I_3+oThisArea_arr.I_34;
        end

        function I2_arr=I2(oThisArea_arr)
            I2_arr=oThisArea_arr.I_3-oThisArea_arr.I_34;
        end

        function alpha_3_arr=alpha_3(oThisArea_arr)
            alpha_3_arr=pi/4-oThisArea_arr.alpha1;
        end

        function rho_y_arr=rho_y(oThisArea_arr)
            rho_y_arr=reshape(sqrt([oThisArea_arr.Iy]./[oThisArea_arr.A
               ]),size(oThisArea_arr));
        end

        function rho_z_arr=rho_z(oThisArea_arr)
            rho_z_arr=reshape(sqrt([oThisArea_arr.Iz]./[oThisArea_arr.A
               ]),size(oThisArea_arr));
        end

        function Sy_arr=Sy(oThisArea_arr,z_max_arr)
            if ~isscalar(z_max_arr) && any(size(oThisArea_arr)~=size(
               z_max_arr))
                error('z_max_arr␣must␣have␣the␣same␣size␣as␣
                   oThisArea_arr,␣or␣be␣a␣scalar')
            end
            Sy_arr=reshape([oThisArea_arr.Iy]./abs(z_max_arr),size(
               oThisArea_arr));
        end

        function Sz_arr=Sz(oThisArea_arr,y_max_arr)
            if ~isscalar(y_max_arr) && any(size(oThisArea_arr)~=size(
               y_max_arr))
                error('y_max_arr␣must␣have␣the␣same␣size␣as␣
                    oThisArea_arr,␣or␣be␣a␣scalar')
            end
            Sz_arr=reshape([oThisArea_arr.Iz]./abs(y_max_arr),size(
                oThisArea_arr));
        end

        function oNewArea_arr=rotatedArea(oThisArea_arr,alpha_rad_arr)
            if ~isscalar(alpha_rad_arr) && any(size(oThisArea_arr)~=size
                (alpha_rad_arr))
                error('alpha_rad_arr␣must␣have␣the␣same␣size␣as␣
                    oThisArea_arr,␣or␣be␣a␣scalar')
            end

            oThisArea_arr_size=size(oThisArea_arr);
            dims_c=num2cell(oThisArea_arr_size);
            if isscalar(alpha_rad_arr)
                temp_cols=TransMatrix(-alpha_rad_arr)*[[oThisArea_arr.Iy
                    ];[oThisArea_arr.Iz];-[oThisArea_arr.Iyz]];
                oNewArea_arr=reshape(cArea([oThisArea_arr.A],temp_cols
                    (1,:),temp_cols(2,:),temp_cols(3,:)),
                    oThisArea_arr_size);
            else
                oNewArea_arr(dims_c{:})=cArea;
                for n=1:numel(oNewArea_arr)
                    temp_col=TransMatrix(-alpha_rad_arr(n))*[
                        oThisArea_arr(n).Iy;oThisArea_arr(n).Iz;-
                        oThisArea_arr(n).Iyz];
                    oNewArea_arr(n)=cArea(oThisArea_arr(n).A,temp_col(1)
                        ,temp_col(2),temp_col(3));
                end
            end
        end
    end
end
```

Object Array Initialization

```
>> A_arr=[1,2,3;4,5,6]
A_arr =
     1     2     3
     4     5     6

>> Iy_arr=[7,8,9;10,11,12]
Iy_arr =
     7     8     9
    10    11    12

>> Iz_arr=[13,14,15;16,17,18]
Iz_arr =
    13    14    15
    16    17    18

>> oArea_arr=cArea(A_arr,Iy_arr,
    Iz_arr)
oArea_arr =
  2x3 cArea array with properties:
```

```
    A
    Iy
    Iz
    Iyz

>> [oArea_arr.A]
ans =
     1     4     2     5     3     6

>> reshape([oArea_arr.A],size(
    oArea_arr))
ans =
     1     2     3
     4     5     6

>> reshape([oArea_arr.Iy],size(
    oArea_arr))
ans =
     7     8     9
```

```
      10        11        12

>> reshape([oArea_arr.Iz],size(
    oArea_arr))
ans =
      13        14        15
      16        17        18
```

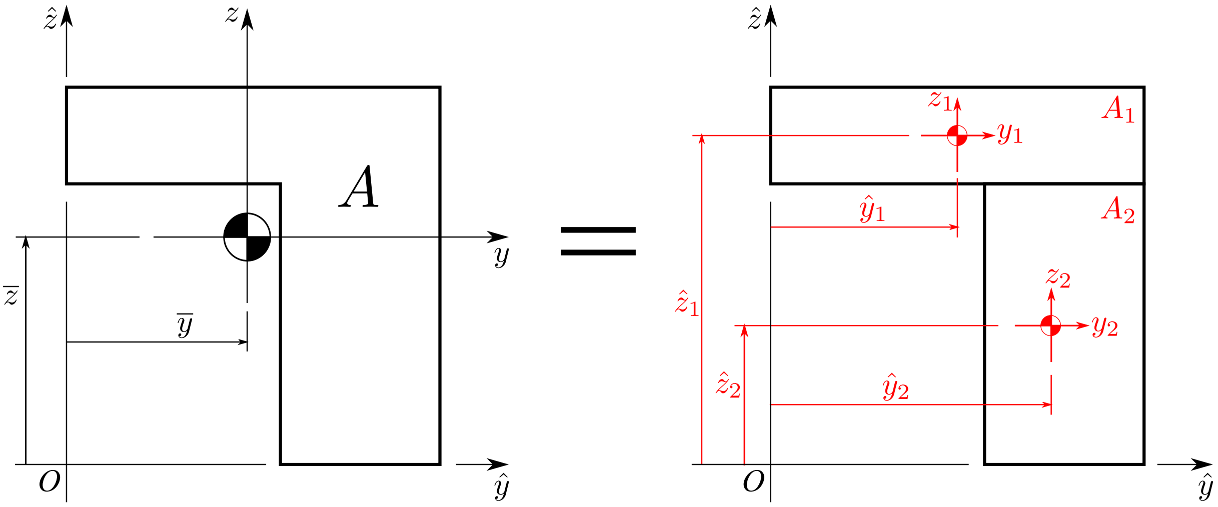
```
>> reshape([oArea_arr.Iyz],size(
    oArea_arr))
ans =
      0         0         0
      0         0         0
```

Methods of an Object Array

```
>> oArea_arr.I_p
ans =
      20        22        24
      26        28        30

>> oArea_arr.rotatedArea(pi/2)
ans =
  2x3 cArea array with properties:
     A
     Iy
     Iz
```

```
     Iyz

>> oArea_arr.rotatedArea(pi/2*ones
    (2,3))
ans =
  2x3 cArea array with properties:
     A
     Iy
     Iz
     Iyz
```

1.5.1 Application 1; Properties of a Composite Cross-sections



$$A = \sum_i A_i \tag{1}$$

$$Q_{\hat{y}} = \sum_i Q_{\hat{y},i} \qquad \& \qquad Q_{\hat{z}} = \sum_i Q_{\hat{z},i} \tag{2}$$

$$= \sum_i A_i\, \hat{z}_i \qquad \& \qquad = \sum_i A_i\, \hat{y}_i \tag{3}$$

$$\overline{z} = \frac{Q_{\hat{y}}}{A} \qquad \& \qquad \overline{y} = \frac{Q_{\hat{z}}}{A} \tag{4}$$

$$z_i = \hat{z}_i - \overline{z} \qquad\qquad y_i = \hat{y}_i - \overline{y} \tag{5}$$

$$I_{\hat{y}} = \sum_i I_{\hat{y},i} \qquad \& \qquad I_{\hat{z}} = \sum_i I_{\hat{z},i} \qquad \& \qquad I_{\hat{y}\hat{z}} = \sum_i I_{\hat{y}\hat{z},i} \tag{6}$$

$$= \sum_i \left(\bar{I}_{y,i} + A_i\,\hat{z}_i^2\right) \quad \& \quad = \sum_i \left(\bar{I}_{z,i} + A_i\,\hat{y}_i^2\right) \quad \& \quad = \sum_i \left(\bar{I}_{yz,i} + A_i\,\hat{y}_i\,\hat{z}_i\right) \tag{7}$$

$$I_y = \sum_i I_{y,i} \qquad I_z = \sum_i I_{z,i} \qquad I_{yz} = \sum_i I_{yz,i} \tag{8}$$

$$= \sum_i \left(\bar{I}_{y,i} + A_i\,z_i^2\right) \qquad = \sum_i \left(\bar{I}_{z,i} + A_i\,y_i^2\right) \qquad = \sum_i \left(\bar{I}_{yz,i} + A_i\,y_i z_i\right) \tag{9}$$

$$= I_{\hat{y}} - A\,\overline{z}^2 \qquad = I_{\hat{z}} - A\,\overline{y}^2 \qquad = I_{\hat{y}\hat{z}} - A\,\overline{y}\,\overline{z} \tag{10}$$

where:

- $(\hat{y}_i, \hat{z}_i)$ are the coordinates of the centroid of A_i,
- $Q_{\hat{y},i}$, $Q_{\hat{z},i}$, $I_{\hat{y},i}$, $I_{\hat{z},i}$ & $I_{\hat{y}\hat{z},i}$ are the 1st and 2nd moments of the area A_i around the $\hat{y} - \hat{z}$ axes
- $\bar{I}_{y,i}$, $\bar{I}_{z,i}$ and $\bar{I}_{yz,i}$ are the 2nd moments of area A_i around its centroidal axes

The cCompositeArea_simple Class

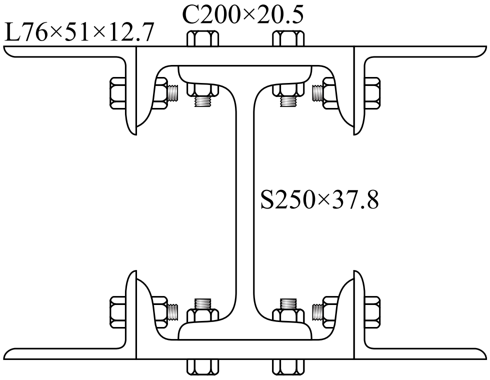
Listing 3: File cCompositeArea_simple.m

```
classdef cCompositeArea_simple  %The file-name must be the same as the class-name


    properties (Access=protected)
        oArea_vec(1,:) cArea   %row vector
        y_hat_vec(1,:)         %row vector
        z_hat_vec(1,:)         %row vector
    end

    methods
        % Constructor
        function oThisCompositeArea=cCompositeArea_simple(oArea_vec,y_hat_vec,z_hat_vec)
            if nargin == 3
                if length(oArea_vec)~=length(y_hat_vec),error('oArea_vec␣and␣y_hat_vec␣must␣have␣the␣same␣lengths'),end
                if length(oArea_vec)~=length(z_hat_vec),error('C_Area_vec␣and␣z_hat_vec␣must␣have␣the␣same␣lengths'),end

                %Assign class properties
                oThisCompositeArea.oArea_vec=oArea_vec;
                oThisCompositeArea.y_hat_vec=y_hat_vec;
                oThisCompositeArea.z_hat_vec=z_hat_vec;
            elseif nargin ~= 0
                error('This␣class␣can␣be␣constructed␣using␣zero␣or␣3␣inputs.');
            end
        end

        function oArea_vec=get_oArea_vec(oThisCompositeArea)
            oArea_vec=oThisCompositeArea.oArea_vec;
        end

        function y_hat_vec=get_y_hat_vec(oThisCompositeArea)
            y_hat_vec=oThisCompositeArea.y_hat_vec;
        end

        function z_hat_vec=get_z_hat_vec(oThisCompositeArea)
            z_hat_vec=oThisCompositeArea.z_hat_vec;
        end

        function A=A(oThisCompositeArea)
            A=sum([oThisCompositeArea.oArea_vec.A]);
        end

        function Qy=Qy(oThisCompositeArea)
            Qy=sum([oThisCompositeArea.oArea_vec.A].*oThisCompositeArea.
               z_hat_vec);
        end

        function Qz=Qz(oThisCompositeArea)
            Qz=sum([oThisCompositeArea.oArea_vec.A].*oThisCompositeArea.
               y_hat_vec);
        end

        function y_bar=y_bar(oThisCompositeArea)
            y_bar=oThisCompositeArea.Qz()/oThisCompositeArea.A();
        end

        function z_bar=z_bar(oThisCompositeArea)
            z_bar=oThisCompositeArea.Qy()/oThisCompositeArea.A();
        end

        function Iy_hat=Iy_hat(oThisCompositeArea)
            Iy_hat=sum([oThisCompositeArea.oArea_vec.Iy])+sum([
               oThisCompositeArea.oArea_vec.A].*oThisCompositeArea.
               z_hat_vec.^2);
        end

        function Iz_hat=Iz_hat(oThisCompositeArea)
            Iz_hat=sum([oThisCompositeArea.oArea_vec.Iz])+sum([
               oThisCompositeArea.oArea_vec.A].*oThisCompositeArea.
               y_hat_vec.^2);
        end

        function Iyz_hat=Iyz_hat(oThisCompositeArea)
            Iyz_hat=sum([oThisCompositeArea.oArea_vec.Iyz])+sum([
               oThisCompositeArea.oArea_vec.A].*oThisCompositeArea.
               y_hat_vec.*oThisCompositeArea.z_hat_vec);
        end

        function Ip_hat=Ip_hat(oThisCompositeArea)
            Ip_hat=oThisCompositeArea.Iy_hat+oThisCompositeArea.Iz_hat;
        end
        function Iy=Iy(oThisCompositeArea)
            Iy=oThisCompositeArea.Iy_hat-oThisCompositeArea.A*
               oThisCompositeArea.z_bar.^2;
        end

        function Iz=Iz(oThisCompositeArea)
            Iz=oThisCompositeArea.Iz_hat-oThisCompositeArea.A*
               oThisCompositeArea.y_bar.^2;
        end

        function Iyz=Iyz(oThisCompositeArea)
            Iyz=oThisCompositeArea.Iyz_hat-oThisCompositeArea.A*
               oThisCompositeArea.y_bar*oThisCompositeArea.z_bar;
        end

        function Ip=Ip(oThisCompositeArea)
            Ip=oThisCompositeArea.Iy+oThisCompositeArea.Iz;
        end
    end
end
```

Example 1 (Composite Cross-Section).

For the shown cross-section, calculate:

1. A, $\overline{y}$, $\overline{z}$, I_y, I_z, I_{yz}, I_P, ρ_y, ρ_z, S_y & S_z
2. I'_y, I'_z, I'_{yz} at $\alpha = 5°$
3. α_1, I_1 & I_2

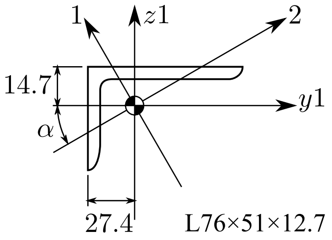


Solution

From standard tables of rolled-Steel shapes [?, app. C], properties of cross-section component are identified as:

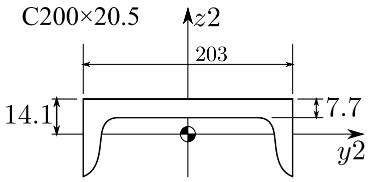
Dims. in mm.

$A_1 = 1450$
$\bar{I}_{y,1} = .278$
$\bar{I}_{z,1} = .799$
$\tan(\alpha) = 0.413$



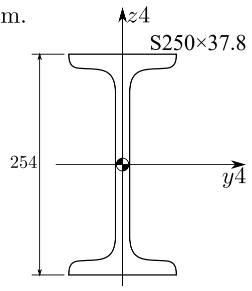
Dims. in mm.

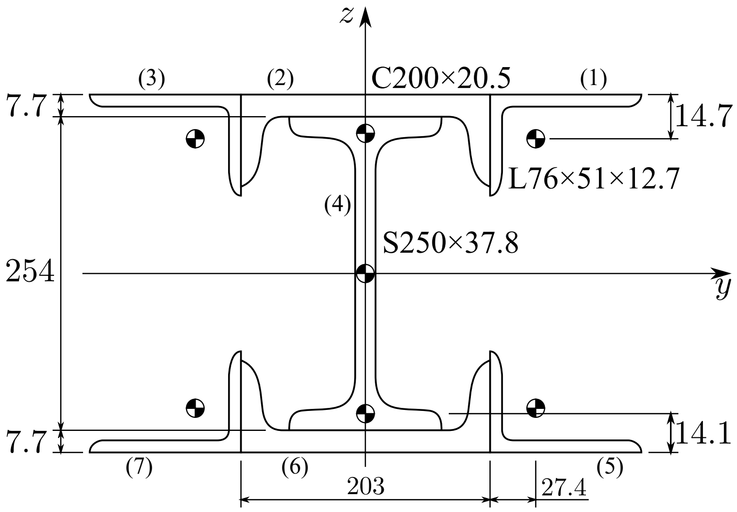
$A_2 = 2610$
$\bar{I}_{y,2} = 0.633$
$\bar{I}_{z,2} = 15$



Dims. in mm.

$A_4 = 4810$
$\bar{I}_{y,4} = 51.2$
$\bar{I}_{z,4} = 2.8$





Listing 4: File test_cCompositeArea_simple.m

```
clc
clear all %#ok<*CLALL>

% Problem 1.2-e
A_vec=[1450;2610;1450;4810;1450;2610;1450]*1e-6;
Iy_vec=[0.278;0.633;0.278;51.2;0.278;0.633;0.278]*1e-6;
Iz_vec=[0.799;15;0.799;2.8;0.799;15;0.799]*1e-6;
Iyz_hat_1=-tan(2*(atan(.413)+pi/2))*(Iy_vec(1)-Iz_vec(1))/2;
Iyz_vec=[Iyz_hat_1;0;-Iyz_hat_1;0;-Iyz_hat_1;0;Iyz_hat_1];
oA_vec=cArea(A_vec,Iy_vec,Iz_vec,Iyz_vec);

y1=203e-3/2+27.4e-3;
z1=254e-3/2+7.7e-3-14.7e-3;
z2=254e-3/2+7.7e-3-14.1e-3;
y_hat_vec=[y1;0;-y1;0;y1;0;-y1];
```

```
z_hat_vec=[z1;z2;z1;0;-z1;-z2;-z1];

oSec=cCompositeArea_simple(oA_vec,y_hat_vec,z_hat_vec) %#ok<*NOPTS>

A=oSec.A
y_bar=oSec.y_bar
z_bar=oSec.z_bar
Iy=oSec.Iy
Iz=oSec.Iz
Iyz=oSec.Iyz
Ip=oSec.Ip

oArea=cArea(oSec.A,oSec.Iy,oSec.Iz,oSec.Iyz)
rho_y=oArea.rho_y
rho_z=oArea.rho_z
Sy=oArea.Sy((254/2+7.7)*1e-3)
Sz=oArea.Sz((203/2+76)*1e-3)
oArea_dash=oArea.rotatedArea(deg2rad(5))
alpha1=oArea.alpha1
I1=oArea.I1
I2=oArea.I2
```

```
>> test_cCompositeArea_simple

oSec =
  cCompositeArea_simple with no
      properties.

A =
    0.0158

y_bar =
     0

z_bar =
   3.4245e-18

Iy =
   2.1302e-04

Iz =
   1.3236e-04

Iyz =
     0

Ip =
   3.4538e-04

oArea =
  cArea with properties:
      A: 0.0158
     Iy: 2.1302e-04
     Iz: 1.3236e-04
    Iyz: 0

rho_y =
    0.1160

rho_z =
    0.0914

Sy =
    0.0016

Sz =
   7.4571e-04

oArea_dash =
  cArea with properties:
      A: 0.0158
     Iy: 2.1241e-04
     Iz: 1.3298e-04
    Iyz: 7.0028e-06

alpha1 =
     0

I1 =
   2.1302e-04

I2 =
   1.3236e-04
```

1.6 Inheritance

Inheritance

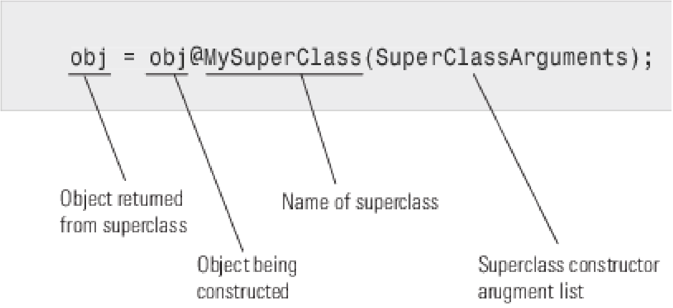
A subclass object inherits all superclass' properties and methods.

- Subclasses can override superclass methods
- www.mathworks.com/help/matlab/matlab_oop/subclass-constructors.html
- www.mathworks.com/help/matlab/matlab_oop/modifying-superclass-methods-and-proper
  html
- www.mathworks.com/help/matlab/matlab_oop/modify-superclass-properties.html

The Subclass Constructor

Before constructing a subclass, the superclass must be initialized.

- This is done by calling the superclass constructor
- By default, MATLAB calls the superclass constructor without arguments
- If you want the superclass constructor called with specific arguments, explicitly call the superclass constructor from the subclass constructor



```
obj = obj@MySuperClass(SuperClassArguments);
```

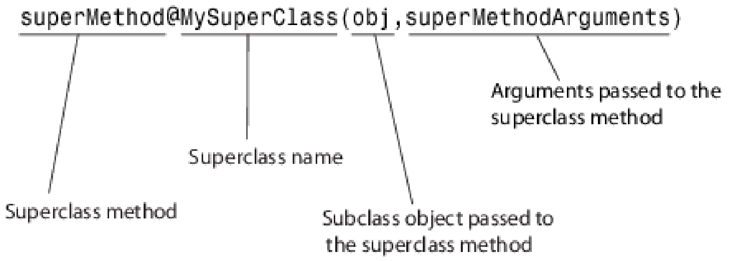
Object returned from superclass
Object being constructed
Name of superclass
Superclass constructor arugment list

  - The call to the superclass constructor must come before any other references to the subclass object
  - Calls to superclass constructor cannot be conditional
    - You cannot place superclass construction calls in loops, conditions, switches, try/catch, or nested functions
    - To call a superclass constructor with different arguments that depend on some condition, create a comma-separated list [sec. ??] and pass it as the input to the superclass constructor

- When a subclass does not define a constructor, the default constructor passes its inputs to the direct superclass constructor
  - This behavior is useful when there is no need for a subclass to define a constructor

Calling a Superclass Method from within a Subclass Method

A superclass method can be called only from a subclass method with the same name.


superMethod@MySuperClass(obj,superMethodArguments)
Arguments passed to the superclass method
Superclass name
Superclass method
Subclass object passed to the superclass method

Identifying Class Type

`isa` determines if a class is of (or derived from) a specified class.

`class` determine the class of an object

1.6.1 Application 2; Composite-Cross-Sections; Revisited

The `cCompositeArea` Class

Listing 5: File cCompositeArea.m

```
classdef cCompositeArea < cArea

    properties
        y_bar(1,1)
        z_bar(1,1)
    end

    properties (Access=protected)
        oArea_vec(1,:) cArea   %row vector
        y_hat_vec(1,:)         %row vector
        z_hat_vec(1,:)         %row vector
    end

    methods
        % Constructor
        function oThisCompositeArea=cCompositeArea(oArea_vec,y_hat_vec,
           z_hat_vec)
            if nargin==0
                superClassArgs={};
            elseif nargin == 3
                if length(oArea_vec)~=length(y_hat_vec),error('oArea_vec
                   ␣and␣y_hat_vec␣must␣have␣the␣same␣lengths'),end
                if length(oArea_vec)~=length(z_hat_vec),error('
                   C_Area_vec␣and␣z_hat_vec␣must␣have␣the␣same␣lengths')
                   ,end

                A=sum([oArea_vec.A]);
                Qy_hat=dot([oArea_vec.A],z_hat_vec);
                Qz_hat=dot([oArea_vec.A],y_hat_vec);
                y_bar=Qz_hat/A;
                z_bar=Qy_hat/A;
                Iy_hat=sum([oArea_vec.Iy])+dot([oArea_vec.A],z_hat_vec
                   .^2);
                Iz_hat=sum([oArea_vec.Iz])+dot([oArea_vec.A],y_hat_vec
                   .^2);
                Iyz_hat=sum([oArea_vec.Iyz])+dot([oArea_vec.A],y_hat_vec
                   .*z_hat_vec);
                Iy=Iy_hat-A*z_bar.^2;
                Iz=Iz_hat-A*y_bar.^2;
                Iyz=Iyz_hat-A*y_bar*z_bar;
                superClassArgs={A,Iy,Iz,Iyz};
            else
                error('This␣class␣can␣be␣constructed␣using␣zero␣or␣3␣
                   inputs.');
            end

            %Construct the super class
            oThisCompositeArea@cArea(superClassArgs{:});

            %Construct the sub class
            oThisCompositeArea.oArea_vec=oArea_vec;
            oThisCompositeArea.y_hat_vec=y_hat_vec;
            oThisCompositeArea.z_hat_vec=z_hat_vec;
            oThisCompositeArea.y_bar=y_bar;
            oThisCompositeArea.z_bar=z_bar;
        end

        function oArea_vec=get_oArea_vec(oThisCompositeArea)
            oArea_vec=oThisCompositeArea.oArea_vec;
        end

        function y_hat_vec=get_y_hat_vec(oThisCompositeArea)
            y_hat_vec=oThisCompositeArea.y_hat_vec;
        end

        function z_hat_vec=get_z_hat_vec(oThisCompositeArea)
            z_hat_vec=oThisCompositeArea.z_hat_vec;
        end

        function y_bar=get_y_bar(oThisCompositeArea)
            y_bar=oThisCompositeArea.y_bar;
        end

        function z_bar=get_z_bar(oThisCompositeArea)
            z_bar=oThisCompositeArea.z_bar;
        end

        function Qy_hat=Qy_hat(oThisCompositeArea)
            Qy_hat=oThisCompositeArea.A*oThisCompositeArea.z_bar;
        end

        function Qz_hat=Qz_hat(oThisCompositeArea)
            Qz_hat=oThisCompositeArea.A*oThisCompositeArea.y_bar;
        end

        function Iy_hat=Iy_hat(oThisCompositeArea)
            Iy_hat=oThisCompositeArea.Iy+oThisCompositeArea.A*
               oThisCompositeArea.z_bar.^2;
        end

        function Iz_hat=Iz_hat(oThisCompositeArea)
```

```
            Iz_hat=oThisCompositeArea.Iz+oThisCompositeArea.A*
                oThisCompositeArea.y_bar.^2;
        end

        function Iyz_hat=Iyz_hat(oThisCompositeArea)
            Iyz_hat=oThisCompositeArea.Iyz+oThisCompositeArea.A*
                oThisCompositeArea.y_bar*oThisCompositeArea.z_bar;
        end

        function Ip_hat=Ip_hat(oThisCompositeArea)
            Ip_hat=oThisCompositeArea.Iy_hat+oThisCompositeArea.Iz_hat;
        end
    end
end
```

Example 2 (Example 1; Revisited).
Resolve example 1 using the `cCompositeArea` class.

Solution

Listing 6: test_cCompositeArea.m

```
clc
clear all %#ok<*CLALL>

% Problem 1.2-e
A_vec=[1450;2610;1450;4810;1450;2610;1450]*1e-6;
Iy_vec=[0.278;0.633;0.278;51.2;0.278;0.633;0.278]*1e-6;
Iz_vec=[0.799;15;0.799;2.8;0.799;15;0.799]*1e-6;
Iyz_hat_1=-tan(2*(atan(.413)+pi/2))*(Iy_vec(1)-Iz_vec(1))/2;
Iyz_vec=[Iyz_hat_1;0;-Iyz_hat_1;0;-Iyz_hat_1;0;Iyz_hat_1];
oA_vec=cArea(A_vec,Iy_vec,Iz_vec,Iyz_vec);

y1=203e-3/2+27.4e-3;
z1=254e-3/2+7.7e-3-14.7e-3;
z2=254e-3/2+7.7e-3-14.1e-3;
y_hat_vec=[y1;0;-y1;0;y1;0;-y1];
z_hat_vec=[z1;z2;z1;0;-z1;-z2;-z1];

oSec1=cCompositeArea(oA_vec,y_hat_vec,z_hat_vec) %#ok<*NOPTS>

A=oSec1.A
y_bar=oSec1.y_bar
z_bar=oSec1.z_bar
Iy=oSec1.Iy
Iz=oSec1.Iz
Iyz=oSec1.Iyz
Ip=oSec1.Ip
rho_y=oSec1.rho_y
rho_z=oSec1.rho_z
Sy=oSec1.Sy((254/2+7.7)*1e-3)
Sz=oSec1.Sz((203/2+76)*1e-3)
oSec_dash=oSec1.rotatedArea(deg2rad(5))
alpha1=oSec1.alpha1
I1=oSec1.I1
I2=oSec1.I2
```

```
>> test_cCompositeArea

oSec =
  cCompositeArea with properties:
    y_bar: 0
    z_bar: 0
        A: 0.0158
       Iy: 2.1302e-04
       Iz: 1.3236e-04
      Iyz: 0

A =
     0.0158

y_bar =
      0

z_bar =
      0

Iy =
   2.1302e-04

Iz =
   1.3236e-04

Iyz =
      0

Ip =
   3.4538e-04

rho_y =
     0.1160

rho_z =
     0.0914

Sy =
     0.0016

Sz =
   7.4571e-04

oSec_dash =
  cArea with properties:
      A: 0.0158
     Iy: 2.1241e-04
     Iz: 1.3298e-04
    Iyz: 7.0028e-06

alpha1 =
      0

I1 =
   2.1302e-04

I2 =
   1.3236e-04
```

1.6.2 Application 3; Symmetric Composite-Cross-Sections

The cCompositeArea_Symm_Y Class

Listing 7: File cCompositeArea_Symm_Y.m

```
classdef cCompositeArea_Symm_Y < cCompositeArea
    %This class takes only the upper or lower half of composite area

    properties (Access=private)
        ind_vec
    end

    methods
        % Subclass constructor
        function oThisCompositeArea_YSymm=cCompositeArea_Symm_Y(
            oArea_vec_half,y_hat_vec_half,z_vec_half)
             if nargin==0
                 superClassArgs={};
             elseif nargin == 3
                 if length(oArea_vec_half)~=length(y_hat_vec_half),error(
                    'oArea_vec_half,␣y_vec_half␣and␣z_vec_half␣must␣have␣
                    the␣same␣lengths'),end
                 if length(oArea_vec_half)~=length(z_vec_half),error('
                    oArea_vec_half,␣y_vec_half␣and␣z_vec_half␣must␣have␣
                    the␣same␣lengths'),end
```

```
                %Determine the index of non-bisected elements
                ind_vec=find(~((z_vec_half(:).'==0) & ([oArea_vec_half.
                   Iyz]==0)));

                %Multiply the properties of non bisected elements by 2
                oArea_vec_temp=oArea_vec_half;
                for ii=ind_vec
                    oArea_vec_temp(ii)=cArea(2*oArea_vec_temp(ii).A,2*
                       oArea_vec_temp(ii).Iy,2*oArea_vec_temp(ii).Iz,2*
                       oArea_vec_temp(ii).Iyz);
                end

                superClassArgs={oArea_vec_temp,y_hat_vec_half,z_vec_half
                   };
            else
                error('This␣class␣can␣be␣constructed␣using␣zero␣or␣3␣
                   inputs.');
            end

            %Construct the superclass
            oThisCompositeArea_YSymm@cCompositeArea(superClassArgs{:});
            oThisCompositeArea_YSymm.Iy=oThisCompositeArea_YSymm.Iy_hat;
            oThisCompositeArea_YSymm.Iyz=0;
            oThisCompositeArea_YSymm.z_bar=0;

            if nargin == 3
                %Construct the sub class
                oThisCompositeArea_YSymm.ind_vec=ind_vec;
            end
        end

        function oArea_vec_half=get_oArea_vec(oThisCompositeArea_YSymm)
            %Divide the properties of superclass non bisected elements
               by 2
            %oArea_vec_half=oThisCompositeArea_YSymm.oArea_vec;
            oArea_vec_half=get_oArea_vec@cCompositeArea(
               oThisCompositeArea_YSymm);
            for ii=oThisCompositeArea_YSymm.ind_vec
                oArea_vec_half(ii)=cArea(oArea_vec_half(ii).A/2,
                   oArea_vec_half(ii).Iy/2,oArea_vec_half(ii).Iz/2,
                   oArea_vec_half(ii).Iyz/2);
            end
        end
    end
end
```

The cCompositeArea_Symm_Z Class

Listing 8: File cCompositeArea_Symm_Z.m

```
classdef cCompositeArea_Symm_Z  < cCompositeArea
    %This class takes only the right or left half of composite area data

    properties (Access=private)
        ind_NonBisected_vec
    end
```

```
methods
    % Subclass constructor
    function oThisCompositeArea_ZSymm=cCompositeArea_Symm_Z(
       oArea_vec_half,y_vec_half,z_hat_vec_half)
        if nargin==0
            superClassArgs={};
        elseif nargin == 3
            if length(oArea_vec_half)~=length(y_vec_half),error('
               oArea_vec_half,␣y_vec_half␣and␣z_vec_half␣must␣have␣
               the␣same␣lengths'),end
            if length(oArea_vec_half)~=length(z_hat_vec_half),error(
               'oArea_vec_half,␣y_vec_half␣and␣z_vec_half␣must␣have␣
               the␣same␣lengths'),end

            %Determine the index of non bisected elements
            ind_NonBisected_vec=find(~((y_vec_half(:).'==0) & ([
               oArea_vec_half.Iyz]==0)));

            %Multiply the properties of non bisected elements by 2
            oArea_vec_temp=oArea_vec_half;
            for ii=ind_NonBisected_vec
                oArea_vec_temp(ii)=cArea(2*oArea_vec_temp(ii).A,2*
                   oArea_vec_temp(ii).Iy,2*oArea_vec_temp(ii).Iz,2*
                   oArea_vec_temp(ii).Iyz);
            end
            superClassArgs={oArea_vec_temp,y_vec_half,z_hat_vec_half
               };
        else
            error('This␣class␣can␣be␣constructed␣using␣zero␣or␣3␣
               inputs.');
        end

        %Construct the super class
        oThisCompositeArea_ZSymm@cCompositeArea(superClassArgs{:});
        oThisCompositeArea_ZSymm.Iz=oThisCompositeArea_ZSymm.Iz_hat;
        oThisCompositeArea_ZSymm.Iyz=0;
        oThisCompositeArea_ZSymm.y_bar=0;

        if nargin == 3
            %Construct the sub class
            oThisCompositeArea_ZSymm.ind_NonBisected_vec=
               ind_NonBisected_vec;
        end
    end

    function oArea_vec_half=get_oArea_vec(oThisCompositeArea_ZSymm)
        %Divide the properties of super class non bisected elements
           by 2
        %oArea_vec_half=oThisCompositeArea_ZSymm.oArea_vec;
        oArea_vec_half=get_oArea_vec@cCompositeArea(
           oThisCompositeArea_ZSymm);
        for ii=oThisCompositeArea_ZSymm.ind_NonBisected_vec
            oArea_vec_half(ii)=cArea(oArea_vec_half(ii).A/2,
               oArea_vec_half(ii).Iy/2,oArea_vec_half(ii).Iz/2,
               oArea_vec_half(ii).Iyz/2);
        end
    end
end
```

```
end
```

The cCompositeArea_Symm_YZ Class

Listing 9: File cCompositeArea_Symm_YZ.m

```
classdef cCompositeArea_Symm_YZ < cCompositeArea
    %This class takes only quarter the composite area

    properties (Access=private)
        ind_y_symm_only_z_symm_only_vec
        ind_non_symm_vec
    end

    methods
        % Subclass constructor
        function oThisCompositeArea_YZSymm=cCompositeArea_Symm_YZ(
            oArea_vec_quarter,y_vec_quarter,z_vec_quarter)
            if nargin==0
                superClassArgs={};
            elseif nargin == 3
                if length(oArea_vec_quarter)~=length(y_vec_quarter),
                    error('oArea_vec_quarter,␣y_vec_quarter␣and␣
                    z_vec_quarter␣must␣have␣the␣same␣lengths'),end
                if length(oArea_vec_quarter)~=length(z_vec_quarter),
                    error('oArea_vec_quarter,␣y_vec_quarter␣and␣
                    z_vec_quarter␣must␣have␣the␣same␣lengths'),end

                %Determine the index of non bisected elements
                ind_y_symm_vec=find((z_vec_quarter(:).'==0) & ([
                    oArea_vec_quarter.Iyz]==0));
                ind_z_symm_vec=find((y_vec_quarter(:).'==0) & ([
                    oArea_vec_quarter.Iyz]==0));

                ind_y_symm_only_z_symm_only_vec=setxor(ind_y_symm_vec,
                    ind_z_symm_vec);
                ind_non_symm_vec=setdiff(setdiff(1:length(
                    oArea_vec_quarter),ind_y_symm_vec),ind_z_symm_vec);

                oArea_vec_temp=oArea_vec_quarter;
                for ii=ind_y_symm_only_z_symm_only_vec
                    oArea_vec_temp(ii)=cArea(2*oArea_vec_temp(ii).A,2*
                        oArea_vec_temp(ii).Iy,2*oArea_vec_temp(ii).Iz,2*
                        oArea_vec_temp(ii).Iyz);
                end
                for ii=ind_non_symm_vec
                    oArea_vec_temp(ii)=cArea(4*oArea_vec_temp(ii).A,4*
                        oArea_vec_temp(ii).Iy,4*oArea_vec_temp(ii).Iz,4*
                        oArea_vec_temp(ii).Iyz);
                end
                superClassArgs={oArea_vec_temp,y_vec_quarter,
                    z_vec_quarter};
            else
                error('This␣class␣can␣be␣constructed␣using␣zero␣or␣3␣
                    inputs.');
            end
            %Construct the super class
            oThisCompositeArea_YZSymm@cCompositeArea(superClassArgs{:});
            oThisCompositeArea_YZSymm.Iy=oThisCompositeArea_YZSymm.
               Iy_hat;
            oThisCompositeArea_YZSymm.Iz=oThisCompositeArea_YZSymm.
               Iz_hat;
            oThisCompositeArea_YZSymm.Iyz=0;
            oThisCompositeArea_YZSymm.y_bar=0;
            oThisCompositeArea_YZSymm.z_bar=0;

            if nargin == 3
                %Construct the sub class
                oThisCompositeArea_YZSymm.
                   ind_y_symm_only_z_symm_only_vec=
                   ind_y_symm_only_z_symm_only_vec;
                oThisCompositeArea_YZSymm.ind_non_symm_vec=
                   ind_non_symm_vec;
            end
        end

        function oArea_vec_quarter=get_oArea_vec(
           oThisCompositeArea_YZSymm)
            %Divide the properties of super class non bisected elements
               by 2
            %oArea_vec_quarter=oThisCompositeArea_YZSymm.oArea_vec;
            oArea_vec_quarter=get_oArea_vec@cCompositeArea(
               oThisCompositeArea_YZSymm);
            for ii=oThisCompositeArea_YZSymm.
               ind_y_symm_only_z_symm_only_vec
                oArea_vec_quarter(ii)=cArea(oArea_vec_quarter(ii).A/2,
                   oArea_vec_quarter(ii).Iy/2,oArea_vec_quarter(ii).Iz
                   /2,oArea_vec_quarter(ii).Iyz/2);
            end

            %Divide the properties of super class non bisected elements
               by 4
            for ii=oThisCompositeArea_YZSymm.ind_non_symm_vec
                oArea_vec_quarter(ii)=cArea(oArea_vec_quarter(ii).A/4,
                   oArea_vec_quarter(ii).Iy/4,oArea_vec_quarter(ii).Iz
                   /4,oArea_vec_quarter(ii).Iyz/4);
            end
        end
    end
end
```

Listing 10: File test_cCompositeArea_Symm.m

```
clc
clear all %#ok<*CLALL>

% Problem 1.2-e
test_cCompositeArea
clc

%Z Symmetry
iindex=[1,2,4,5,6];
oSec2=cCompositeArea_Symm_Z(oA_vec(iindex),y_hat_vec(iindex)-y_bar,
   z_hat_vec(iindex))     %#ok<*NOPTS>
```

```
oSec2.A-oSec1.A
%oSec2.y_bar-oSec1.y_bar
oSec2.z_bar-oSec1.z_bar
% oSec2.Iy_hat-oSec1.Iy_hat
% oSec2.Iz_hat-oSec1.Iz_hat
% oSec2.Iyz_hat-oSec1.Iyz_hat
% oSec2.Ip_hat-oSec1.Ip_hat
oSec2.Iy-oSec1.Iy
oSec2.Iz-oSec1.Iz
oSec2.Iyz-oSec1.Iyz
oSec2.Ip-oSec1.Ip
[oSec2.get_oArea_vec.A]-[oA_vec(iindex).A]
[oSec2.get_oArea_vec.Iy]-[oA_vec(iindex).Iy]
[oSec2.get_oArea_vec.Iz]-[oA_vec(iindex).Iz]
[oSec2.get_oArea_vec.Iyz]-[oA_vec(iindex).Iyz]

%Y Symmetry
iindex=1:4;
oSec3=cCompositeArea_Symm_Y(oA_vec(iindex),y_hat_vec(iindex),z_hat_vec(
   iindex)-z_bar)
oSec3.A-oSec1.A
oSec3.y_bar-oSec1.y_bar
% oSec3.z_bar-oSec1.z_bar
% oSec3.Iy_hat-oSec1.Iy_hat
% oSec3.Iz_hat-oSec1.Iz_hat
% oSec3.Iyz_hat-oSec1.Iyz_hat
% oSec3.Ip_hat-oSec1.Ip_hat
oSec3.Iy-oSec1.Iy
oSec3.Iz-oSec1.Iz
oSec3.Iyz-oSec1.Iyz
oSec3.Ip-oSec1.Ip
[oSec3.get_oArea_vec.A]-[oA_vec(iindex).A]
[oSec3.get_oArea_vec.Iy]-[oA_vec(iindex).Iy]
[oSec3.get_oArea_vec.Iz]-[oA_vec(iindex).Iz]
[oSec3.get_oArea_vec.Iyz]-[oA_vec(iindex).Iyz]

%YZ Symmetry
iindex=[1,2,4];
oSec4=cCompositeArea_Symm_YZ(oA_vec(iindex),y_hat_vec(iindex)-y_bar,
   z_hat_vec(iindex)-z_bar)
oSec4.A-oSec1.A
% oSec4.y_bar-oSec1.y_bar
% oSec4.z_bar-oSec1.z_bar
% oSec4.Iy_hat-oSec1.Iy_hat
% oSec4.Iz_hat-oSec1.Iz_hat
% oSec4.Iyz_hat-oSec1.Iyz_hat
% oSec4.Ip_hat-oSec1.Ip_hat
oSec4.Iy-oSec1.Iy
oSec4.Iz-oSec1.Iz
oSec4.Iyz-oSec1.Iyz
oSec4.Ip-oSec1.Ip
[oSec4.get_oArea_vec.A]-[oA_vec(iindex).A]
[oSec4.get_oArea_vec.Iy]-[oA_vec(iindex).Iy]
[oSec4.get_oArea_vec.Iz]-[oA_vec(iindex).Iz]
[oSec4.get_oArea_vec.Iyz]-[oA_vec(iindex).Iyz]
```

```
>> test_cCompositeArea_Symm
oSec2 =
  cCompositeArea_Symm_Z with
      properties:
    y_bar: 0
    z_bar: 0
        A: 0.0158
       Iy: 2.1302e-04
       Iz: 1.3236e-04
      Iyz: 0

ans =
   3.4694e-18

ans =
     0

ans =
     0

ans =
     0

ans =
     0

ans =
     0

ans =
     0     0     0     0     0

ans =
     0     0     0     0     0

ans =
     0     0     0     0     0

ans =
     0     0     0     0     0

oSec3 =
  cCompositeArea_Symm_Y with
      properties:
    y_bar: 0
    z_bar: 0
        A: 0.0158
       Iy: 2.1302e-04
       Iz: 1.3236e-04
      Iyz: 0

ans =
     0

ans =
     0

ans =
     0
```

```
ans =
     0

ans =
     0

ans =
     0

ans =
     0     0     0     0

ans =
     0     0     0     0

ans =
     0     0     0     0

ans =
     0     0     0     0

oSec4 =
  cCompositeArea_Symm_YZ with
      properties:
    y_bar: 0
    z_bar: 0
        A: 0.0158
       Iy: 2.1302e-04
       Iz: 1.3236e-04
      Iyz: 0

ans =
     0

ans =
     0

ans =
     0

ans =
     0

ans =
     0

ans =
     0     0     0

ans =
     0     0     0

ans =
     0     0     0

ans =
     0     0     0
```

1.6.3 Application 4; Anti-Symmetric Composite-Cross-Sections

The cCompositeArea_AntiSymm_Y Class

Listing 11: File cCompositeArea_AntiSymm_Y.m

```
classdef cCompositeArea_AntiSymm_Y < cCompositeArea
    %This class takes only the right or left half of composite area data

    properties (Access=private)
        ind_vec
    end

    methods
        % Subclass constructor
        function oThisCompositeArea_AntiSymm_YSymm=
           cCompositeArea_AntiSymm_Y(oArea_vec_half,y_vec_half,
           z_vec_half)
            if nargin==0
                superClassArgs={};
            elseif nargin == 3
                if length(oArea_vec_half)~=length(y_vec_half),error('
                   oArea_vec_half␣and␣y_vec_half␣must␣have␣the␣same␣
                   lengths'),end
                if length(oArea_vec_half)~=length(z_vec_half),error('
                   C_Area_vec␣and␣z_vec_half␣must␣have␣the␣same␣lengths'
                   ),end

                %Determine the index of non bisected elements
                ind_vec=find(~(z_vec_half(:).'==0));

                %Multiply the properties of non bisected elements by 2
                oArea_vec_temp=oArea_vec_half;
                for ii=ind_vec
                    oArea_vec_temp(ii)=cArea(2*oArea_vec_temp(ii).A,2*
                       oArea_vec_temp(ii).Iy,2*oArea_vec_temp(ii).Iz,2*
                       oArea_vec_temp(ii).Iyz);
                end
                superClassArgs={oArea_vec_temp,y_vec_half,z_vec_half};
            else
                error('This␣class␣can␣be␣constructed␣using␣zero␣or␣3␣
                   inputs.');
            end

            %Construct the super class
            oThisCompositeArea_AntiSymm_YSymm@cCompositeArea(
               superClassArgs{:});
            oThisCompositeArea_AntiSymm_YSymm.Iy=
               oThisCompositeArea_AntiSymm_YSymm.Iy_hat;
            oThisCompositeArea_AntiSymm_YSymm.Iz=
               oThisCompositeArea_AntiSymm_YSymm.Iz_hat;
            oThisCompositeArea_AntiSymm_YSymm.Iyz=
               oThisCompositeArea_AntiSymm_YSymm.Iyz_hat;
            oThisCompositeArea_AntiSymm_YSymm.y_bar=0;
            oThisCompositeArea_AntiSymm_YSymm.z_bar=0;

            if nargin == 3
                %Construct the sub class
                oThisCompositeArea_AntiSymm_YSymm.ind_vec=ind_vec;
            end
        end

        function oArea_vec_half=get_oArea_vec(
           oThisCompositeArea_AntiSymm_YSymm)
            %Divide the properties of super class non bisected elements
               by 2
            %oArea_vec_half=oThisCompositeArea_AntiSymm_YSymm.oArea_vec;
            oArea_vec_half=get_oArea_vec@cCompositeArea(
               oThisCompositeArea_AntiSymm_YSymm);
            for ii=oThisCompositeArea_AntiSymm_YSymm.ind_vec
                oArea_vec_half(ii)=cArea(oArea_vec_half(ii).A/2,
                   oArea_vec_half(ii).Iy/2,oArea_vec_half(ii).Iz/2,
                   oArea_vec_half(ii).Iyz/2);
            end
        end
    end
end
```

The cCompositeArea_AntiSymm_Z Class

Listing 12: File cCompositeArea_AntiSymm_Z.m

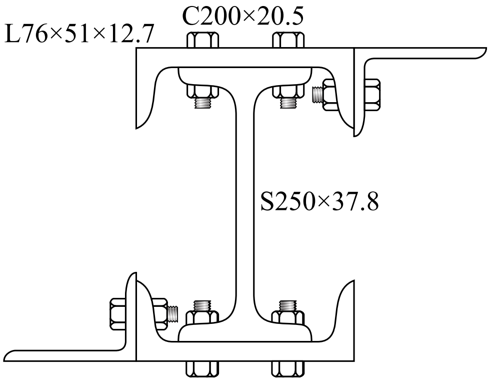
```
classdef cCompositeArea_AntiSymm_Z < cCompositeArea
    %This class takes only the right or left half of composite area data

    properties (Access=private)
        ind_vec
    end

    methods
        % Subclass constructor
        function oThisCompositeArea_AntiSymm_ZSymm=
           cCompositeArea_AntiSymm_Z(oArea_vec_half,y_vec_half,
           z_vec_half)
            if nargin==0
                superClassArgs={};
            elseif nargin == 3
                if length(oArea_vec_half)~=length(y_vec_half),error('
                   oArea_vec_half,␣␣y_vec_half␣and␣z_vec_half␣must␣have␣
                   the␣same␣lengths'),end
                if length(oArea_vec_half)~=length(z_vec_half),error('
                   oArea_vec_half,␣␣y_vec_half␣and␣z_vec_half␣must␣have␣
                   the␣same␣lengths'),end

                %Determine the index of non bisected elements
                ind_vec=find(~(y_vec_half(:).'==0));

                %Multiply the properties of non bisected elements by 2
                oArea_vec_temp=oArea_vec_half;
                for ii=ind_vec
                    oArea_vec_temp(ii)=cArea(2*oArea_vec_temp(ii).A,2*
                       oArea_vec_temp(ii).Iy,2*oArea_vec_temp(ii).Iz,2*
                       oArea_vec_temp(ii).Iyz);
                end
                superClassArgs={oArea_vec_temp,y_vec_half,z_vec_half};
            else
```

```
                error('This␣class␣can␣be␣constructed␣using␣zero␣or␣3␣
                   inputs.');
            end

            %Construct the super class
            oThisCompositeArea_AntiSymm_ZSymm@cCompositeArea(
               superClassArgs{:});
            oThisCompositeArea_AntiSymm_ZSymm.Iy=
               oThisCompositeArea_AntiSymm_ZSymm.Iy_hat;
            oThisCompositeArea_AntiSymm_ZSymm.Iz=
               oThisCompositeArea_AntiSymm_ZSymm.Iz_hat;
            oThisCompositeArea_AntiSymm_ZSymm.Iyz=
               oThisCompositeArea_AntiSymm_ZSymm.Iyz_hat;
            oThisCompositeArea_AntiSymm_ZSymm.y_bar=0;
            oThisCompositeArea_AntiSymm_ZSymm.z_bar=0;

            if nargin == 3
                %Construct the sub class
                oThisCompositeArea_AntiSymm_ZSymm.ind_vec=ind_vec;
            end
        end

        function oArea_vec_half=get_oArea_vec(
           oThisCompositeArea_AntiSymm_ZSymm)
            %Divide the properties of super class non bisected elements
               by 2
            %oArea_vec_half=oThisCompositeArea_AntiSymm_ZSymm.oArea_vec;
            oArea_vec_half=get_oArea_vec@cCompositeArea(
               oThisCompositeArea_AntiSymm_ZSymm);
            for ii=oThisCompositeArea_AntiSymm_ZSymm.ind_vec
                oArea_vec_half(ii)=cArea(oArea_vec_half(ii).A/2,
                   oArea_vec_half(ii).Iy/2,oArea_vec_half(ii).Iz/2,
                   oArea_vec_half(ii).Iyz/2);
            end
        end
    end
end
```

Example 3 (Anti-Symmetric Composite Cross-Section).

For the shown cross-section, calculate:

1. A, $\overline{y}$, $\overline{z}$, I_y, I_z, I_{yz}, I_{P}, ρ_y, ρ_z, S_y & S_z
2. I'_y, I'_z, I'_{yz} at $\alpha = 5°$
3. α_1, I_1 & I_2



Solution

Listing 13: File test_cCompositeArea_AntiSymm.m

```
clc
clear all %#ok<*CLALL>
```

```
% Problem 1.2-f
test_cCompositeArea
clc

%Full section
iindex=[1,2,4,6,7];
oSec1=cCompositeArea(oA_vec(iindex),y_hat_vec(iindex),z_hat_vec(iindex))
       %#ok<*NOPTS>

A=oSec1.A
y_bar=oSec1.y_bar
z_bar=oSec1.z_bar
Iy=oSec1.Iy
Iz=oSec1.Iz
Iyz=oSec1.Iyz
Ip=oSec1.Ip
rho_y=oSec1.rho_y
rho_z=oSec1.rho_z
Sy=oSec1.Sy((254/2+7.7)*1e-3)
Sz=oSec1.Sz((203/2+76)*1e-3)
oSec_dash=oSec1.rotatedArea(deg2rad(5))
alpha1=oSec1.alpha1
I1=oSec1.I1
I2=oSec1.I2

%AntiSymm_YSymm
iiindex=iindex(1:3);
oSec2=cCompositeArea_AntiSymm_Y(oA_vec(iiindex),y_hat_vec(iiindex),
   z_hat_vec(iiindex))
oSec2.A-oSec1.A
% oSec2.get_y_bar-oSec1.get_y_bar
% oSec2.get_z_bar-oSec1.get_z_bar
% oSec2.Iy_hat-oSec1.Iy_hat
% oSec2.Iz_hat-oSec1.Iz_hat
% oSec2.Iyz_hat-oSec1.Iyz_hat
% oSec2.Ip_hat-oSec1.Ip_hat
oSec2.Iy-oSec1.Iy
oSec2.Iz-oSec1.Iz
oSec2.Iyz-oSec1.Iyz
oSec2.Ip-oSec1.Ip
[oSec2.get_oArea_vec.A]-[oA_vec(iiindex).A]
[oSec2.get_oArea_vec.Iy]-[oA_vec(iiindex).Iy]
[oSec2.get_oArea_vec.Iz]-[oA_vec(iiindex).Iz]
[oSec2.get_oArea_vec.Iyz]-[oA_vec(iiindex).Iyz]

%AntiSymm_ZSymm
iiindex=iindex(1:4);
oSec3=cCompositeArea_AntiSymm_Z(oA_vec(iiindex),y_hat_vec(iiindex),
   z_hat_vec(iiindex))
oSec3.A-oSec1.A
% oSec3.get_y_bar-oSec1.get_y_bar
% oSec3.get_z_bar-oSec1.get_z_bar
% oSec3.Iy_hat-oSec1.Iy_hat
% oSec3.Iz_hat-oSec1.Iz_hat
% oSec3.Iyz_hat-oSec1.Iyz_hat
% oSec3.Ip_hat-oSec1.Ip_hat
oSec3.Iy-oSec1.Iy
```

```
oSec3.Iz-oSec1.Iz
oSec3.Iyz-oSec1.Iyz
oSec3.Ip-oSec1.Ip
[oSec3.get_oArea_vec.A]-[oA_vec(iiindex).A]
[oSec3.get_oArea_vec.Iy]-[oA_vec(iiindex).Iy]
[oSec3.get_oArea_vec.Iz]-[oA_vec(iiindex).Iz]
[oSec3.get_oArea_vec.Iyz]-[oA_vec(iiindex).Iyz]
```

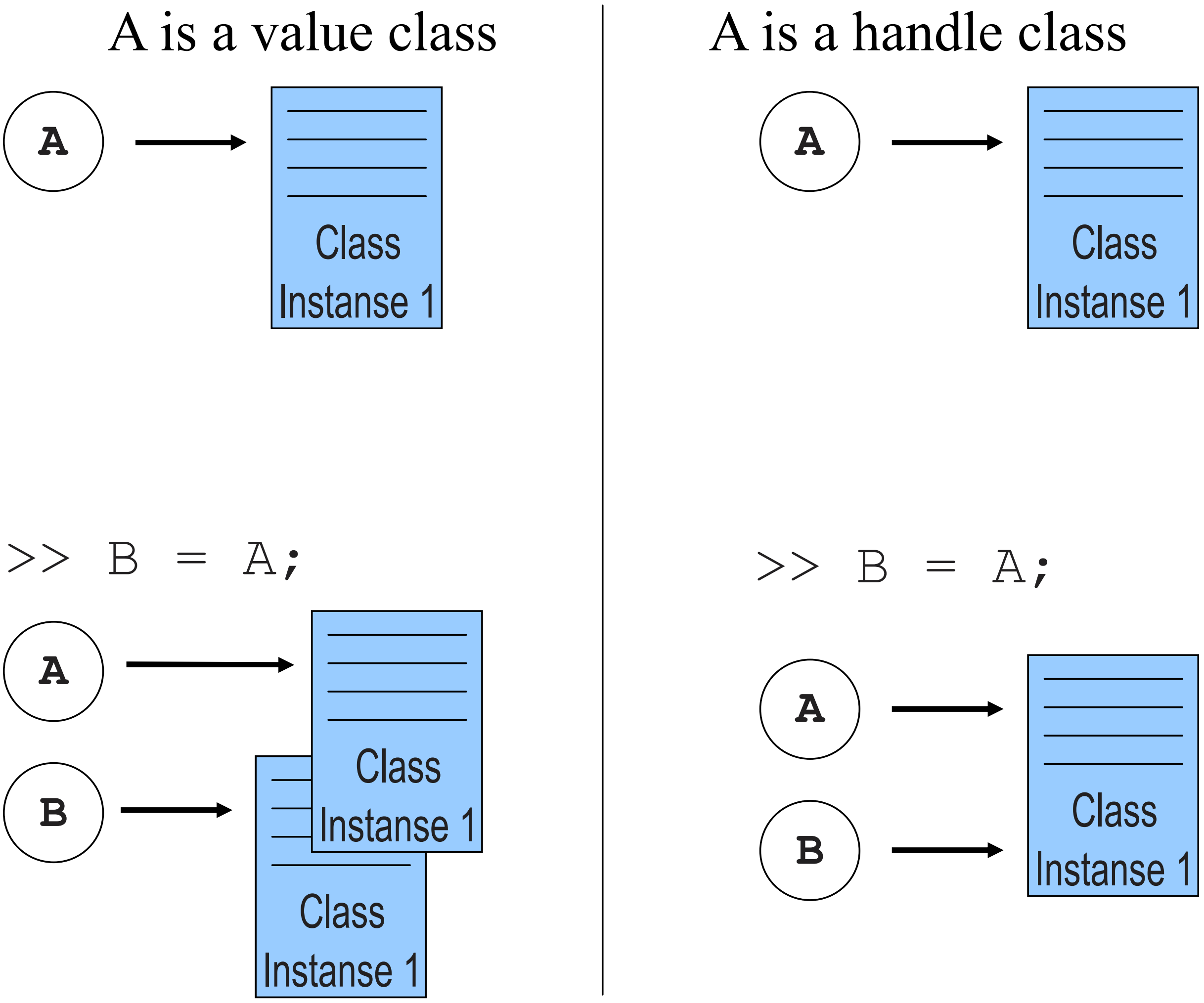
```
>> test_cCompositeArea_AntiSymm
oSec1 =
  cCompositeArea with properties:
    y_bar: 0
    z_bar: 0
        A: 0.0129
       Iy: 1.7070e−04
       Iz: 8.2582e−05
      Iyz: 4.5376e−05

A =
    0.0129

y_bar =
     0

z_bar =
     0

Iy =
   1.7070e−04

Iz =
   8.2582e−05

Iyz =
   4.5376e−05

Ip =
   2.5329e−04

rho_y =
    0.1149

rho_z =
    0.0799

Sy =
    0.0013

Sz =
   4.6525e−04

oSec_dash =
  cArea with properties:
      A 0.0129
     Iy: 1.7791e−04
     Iz: 7.5372e−05
    Iyz: −3.7036e−05
```

```

alpha1 =
   −0.8001

I1 =
   1.8989e−04

I2 =
   6.3395e−05

oSec2 =
  cCompositeArea_AntiSymm_Y with
      properties:
    y_bar: 0
    z_bar: 0
        A: 0.0129
       Iy: 1.7070e−04
       Iz: 8.2582e−05
      Iyz: 4.5376e−05

ans =
     0

ans =
     0

ans =
     0

ans =
     0

ans =
     0

ans =
     0     0     0

ans =
     0     0     0

ans =
     0     0     0

ans =
     0     0     0

oSec3 =
  cCompositeArea_AntiSymm_Z with
```

```
        properties:
      y_bar: 0
      z_bar: 0
          A: 0.0129
         Iy: 1.7070e-04
         Iz: 8.2582e-05
        Iyz: 4.5376e-05

ans =
     0

ans =
     0

ans =
     0
```

```
ans =
     0

ans =
     0

ans =
     0     0     0     0

ans =
     0     0     0     0

ans =
     0     0     0     0

ans =
     0     0     0     0
```

1.7 Value vs. Handle Classes



- Remember that, same as ordinary functions, Matlab passes input arguments to ordinary methods by value
- If you want a method of a class to modify properties of an object of the class
  - and since the class object itself is one of the input arguments to the ordinary method
  - then any modifications to properties of this class object will be lost
  - This is the disadvantage of the so called value classes
- This disadvantage is solved in the so called "handle classes"
  - Handle class is a class that is passed by handle* to the method of the object class
  - Thus, any modifications to properties of this class object will be preserved
- In summary, handle class is a class that is passed by reference

*In Fortran and C++ terminology, this is called "pass by reference". In C terminology, this is achievable through "pass by pointer".

  - In the context of the C programming language, you can think in handle class as a pointer to the class

Sample Value Class

Listing 14: Definition of “cValueClass” Class

```
classdef cValueClass
    properties
        prop1
    end

    methods
        %Constructor
        function oThisClass=cValueClass(value)
            oThisClass.prop1=value;
        end

        %Modify class properties
        %This property will not work as expected because this class is a
            value class
        function setProp1(oThisClass,value)
            oThisClass.prop1=value;
        end
    end
end
```

```
>> oValueClass=cValueClass(5);
>> oValueClass.prop1
ans =
     5
>> oValueClass.setProp1(10);
>> oValueClass.prop1
ans =
     5
>> oValueClass.prop1=10
ans =
     10
```

Sample Handle Class

Listing 15: Definition of “cHandleClass” Class

```
classdef cHandleClass < handle
    % This class encapsulates the following properties
    % Hence, you cannot access them directly
    % Therefore, we have to create the getProp1 and setProp1 methods
    properties (Access = private)
        prop1
    end

    methods
        %Constructor
        function oThisClass=cHandleClass(value)
            oThisClass.prop1=value;
        end
        function prop1=getProp1(oThisClass)
            prop1=oThisClass.prop1;
        end

        %Modify class properties
        function setProp1(oThisClass,value)
            oThisClass.prop1=value;
        end
    end
end
```

```
>> oHandleClass=cHandleClass(5);
>> oHandleClass.getProp1
ans =
     5
>> oHandleClass.setProp1(10);
>> oHandleClass.getProp1
ans =
     10
```

1.8 Advanced Topics

Validate Property Values

- www.mathworks.com/help/matlab/matlab_oop/validate-property-values.html

OOP in Matlab versus other Languages

- www.mathworks.com/help/matlab/matlab_oop/matlab-vs-other-oo-languages.html

Static Data

- www.mathworks.com/help/matlab/matlab_oop/static-data.html