# Documented Examples

This document contains a number of worked examples demonstrating the operation of the Matlab nonlinear beam shapes implemetation.

**Table of Contents**

# Simple Cantilever Beam

The first example will consider the modelling of a simple cantilever beam. The nonlinear deformation will be predicted both under the beam's own weight, and following the application of tip loads.

The parameters of the beam to be treated are:

Length: 4 m

Height: 0.01 m

Width: 0.1 m

Material Density: 4430 kg/m^3

Young's Modulus: 127e9 N/m^2

Shear Modulus: 4.670e10 N/m^2

```
Length = 4; Height = 0.01; Width = 0.1; Density = 4430; E = 127e9; G = 4.670e10;
```

## Create NBS_Master Object

To begin, we create the master level object that will contain the full problem definition.

```
Mcb_obj = NBS_Master; % <- NBS Master Object
```

The global parameters that effect the entire system are written to this object. For a full aircraft this will include the flight condition information. For now, lets define the gravitational acceleration acting on the beam.

```
Mcb_obj.grav_acc = 9.807;
Mcb_obj.gravVec_G = [0;0;-1];
```

Note that gravVec_G takes the default value [0;0;-1] so does not technically require specifying in this example. The '_G' at the end of the variable name dictates that this quantity is defined in the global coordinate system.

Note: Further information on the available NBS_Master parameters can be found in the class definition file NBS_Master.m .

## Create a flexible nonlinear part to represent the beam

The beam will be modelled as a NBS_flexPart_nonlinear object. This flexible part will have NBS_Master as its parent object. This is defined with:

```
Pcb_obj = NBS_flexPart_nonlinear(Mcb_obj,'Cantilever_Beam','Parent',Mcb_obj); % <- NBS Part Obj
```

The first arguement here is always the NBS master object for the problem. The second argument is the user specified name of the part. The 3rd and 4th keyword/argument pair defines the part's parent.

Now we write the necessary properties to the part object 'Pcb_obj' to specify the beam of this example.

**Stiffness matrix:**

For an Euler Bernoulli beam the stiffness matrix relates the curvature vector to the restoring moment generated over a section of the beam. Since this beam has a constant cross section and material properties, this stiffness matrix is identical everywhere along the beams length. This 3x3 stiffness matrix is given by:

```
Ixx = 25/3*1e-9; Izz = 25/3*1e-7; J = 3.123e-8; %<- from known geometry of the beam cross-sect

Pcb_obj.StiffnessMatrix = diag([ E*Ixx ; G*J ; E*Izz ]);
```

**Damping matrix:**

A stiffness proportional damping law is chosen in this exampe of 5%. Hence,

```
Pcb_obj.DampingMatrix = 0.05*Pcb_obj.StiffnessMatrix;
```

**Mass Per Span:**

For this continuous beam the mass is given as a mass-per-span (kg/m) quantity.

```
Pcb_obj.mps = Density*Width*Height;
```

**Inertia Per Span:**

Similar to the mass, the inertia of the beam cross section is defined per unit span.

```
Pcb_obj.I_varTheta_ps_I = Density*diag([ Ixx ; Ixx+Izz ; Izz ]);
```

**SpanWise Evaluation Points:**

The various integral terms in the formulation are evaluated at a number of sample locations along the beam. These sample locations are specified by the 's' vector written to the part object. For this version of the code, there should be an odd number of these points and they should be equally spaced.

```
Pcb_obj.s = linspace(0,Length,201);
```

**Beam Width and Height**

These properties are used by the plotting routines when drawing the part.

```
Pcb_obj.w = Width; Pcb_obj.h = Height;
```

**Beam Axis Location.**

The fractional location across the beam width where 1D reference line is located.

```
Pcb_obj.beam_cntr = 0.5;
```

**Is Aerodynamic Part**

The user must indicate whether or not each part is to produce an aerodynamic load.

```
Pcb_obj.isAero = false;
```

**Plotting Bounds**

The default axis limits to use when plotting the system. Argument of the form:

[xmin xmax ; ymin ymax ; zmin zmax]

```
Mcb_obj.plotBounds = [-2.5 2.5 ; -1 4 ; -4 1];
```

## Set the shape functions to be used in solving the problem

Now that the main parameter of the problem have been defined, the user must specify the sets of shape functions to be used in treating the kinematic parameters of the formulation.

There are a number of in-built shape-sets available to choose from. Namely:

{ Chebyshev 1st , Chebyshev 2nd , Polynomial }

To proceed with this example we assign one of these shape sets to the bending (theta and psi) and twist (phi) euler angle parameters.

```
Pcb_obj.shape_class_bend = 'chebyshev_1st';
Pcb_obj.shape_class_twist = 'chebyshev_1st';
```

Each of these distributions can be assigned a boundary condition at the root and tip. For a cantilever beam the euler parameters are zero at the root and non-zero at the tip. This is defined with the lines:

```
Pcb_obj.shape_BCs_bend  = [0 1;1 1;1 1];
Pcb_obj.shape_BCs_twist = [0 1;1 1;1 1];
%matrix inputs of the form: [ rootValue tipValue ; root1stDerivative tip1stDerivative ; root2nd
%0 and 1 indicate a zero or non-zero value respectively
```

Finally we indicate the number of shapes to apply to the $\theta$, $\psi$ and $\phi$ Euler shape sets. Lets choose 8 shapes for each parameter.

```
Pcb_obj.qth.n = 8;
Pcb_obj.qsi.n = 8;
Pcb_obj.qph.n = 8;
```

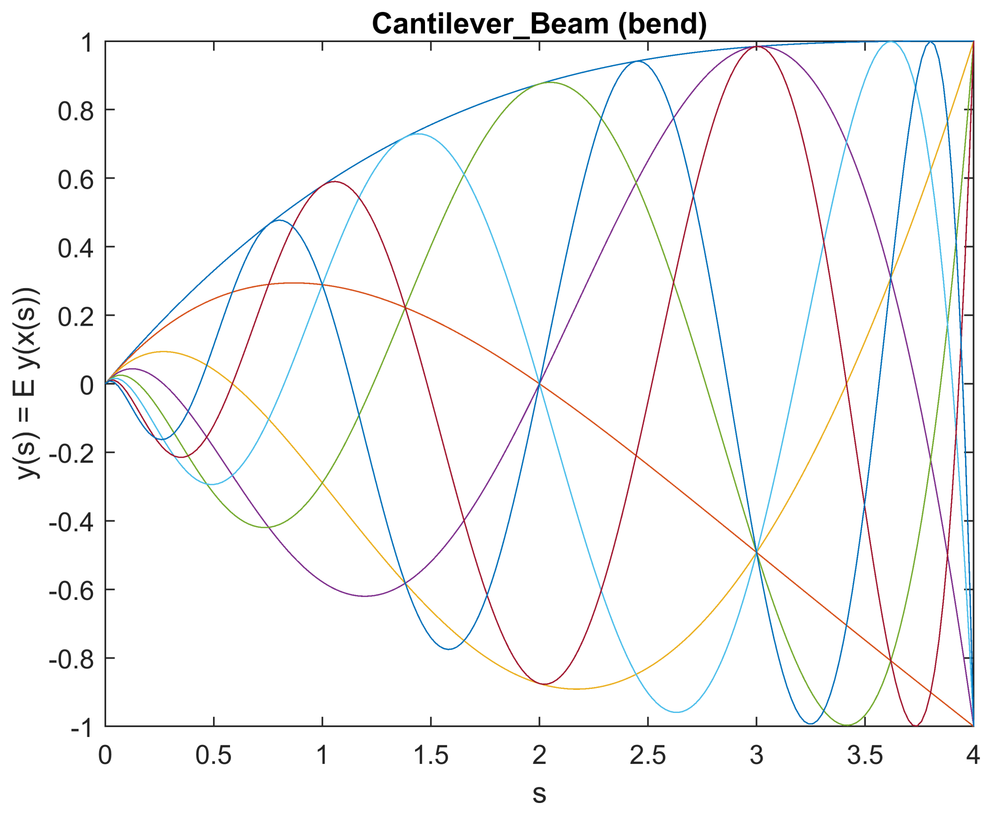
Once the shape set parameters are written to the part object the baseline shape sets can be created using the command:

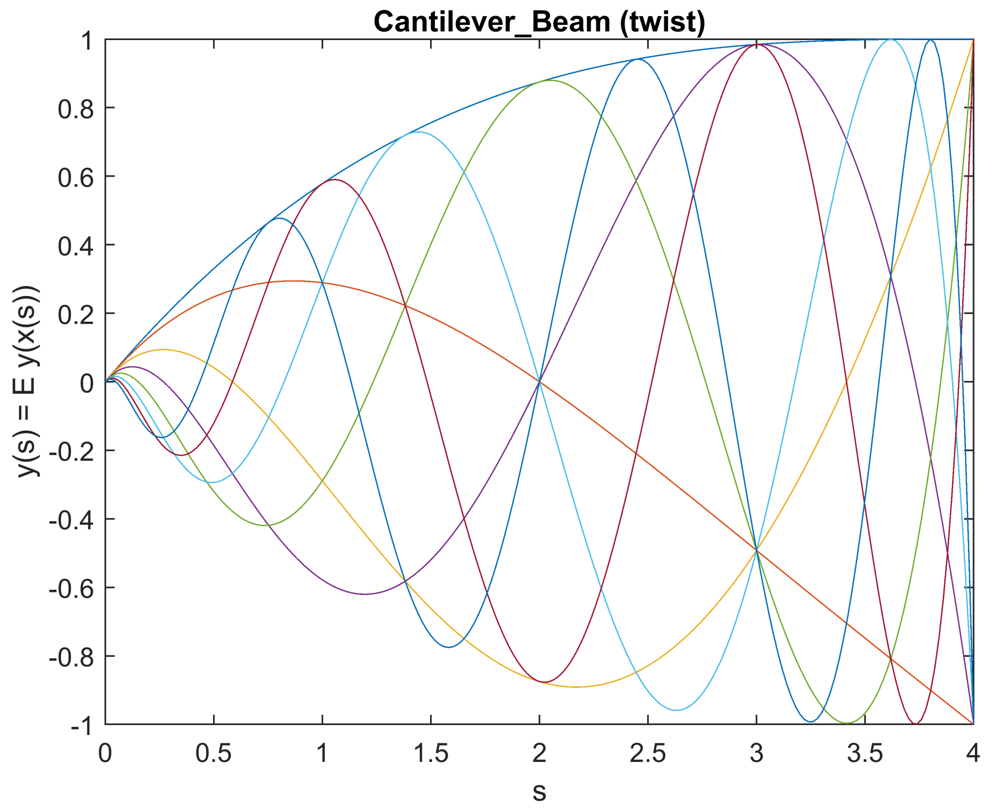
```
%set_dependent_properties(P_obj);
Pcb_obj.populate_shape_set('setName','Cantilever Shape Sets');
```

To view these shape sets one may call

```
Mcb_obj.plotShapes();
```

```
Warning: MATLAB has disabled some advanced graphics rendering features by switching to software OpenGL. For
more information, click here.
```


Cantilever_Beam (bend)
y(s) = E y(x(s))
s
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.5
1
1.5
2
2.5
3
3.5
4


Cantilever_Beam (twist)
y(s) = E y(x(s))
s
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.5
1
1.5
2
2.5
3
3.5
4

## Finalise the object

We are now at the stage where we have a master object Mcb_obj containing all of the problem information including a child object Pcb_obj referring specifically to the flexible beam. To finalise this object we input the command

```
Mcb_obj.set_dependent_properties();
```

This manually populates the remaining dependent properties of the master and part objects.

[Note that true dependent properties are largely avoided in the formulation to avoid the unnecessary recalculation of parameters when accessed during each integration step. The set_dependent_properties method may be removed in subsequent versions of the code].

## Simulate the cantilever beam system

We will now simulate this cantilever beam system. It is important to note here that the NBS_master object and child objects contained within it are all handle classes. Thus to avoid issues when running multiple simulation cases on this system we will create copies of 'Mcb_obj' using its own copy method.

```
Mcb_obj_static = Mcb_obj.copy();
```

(Note: The created copy Mcb_obj_static is entirely unlinked from the original Mcb_obj and thus changes to one will not affect the other. However, Mcb_obj_static has retained exactly the same handle architecture and properties as the original Mcb_obj)

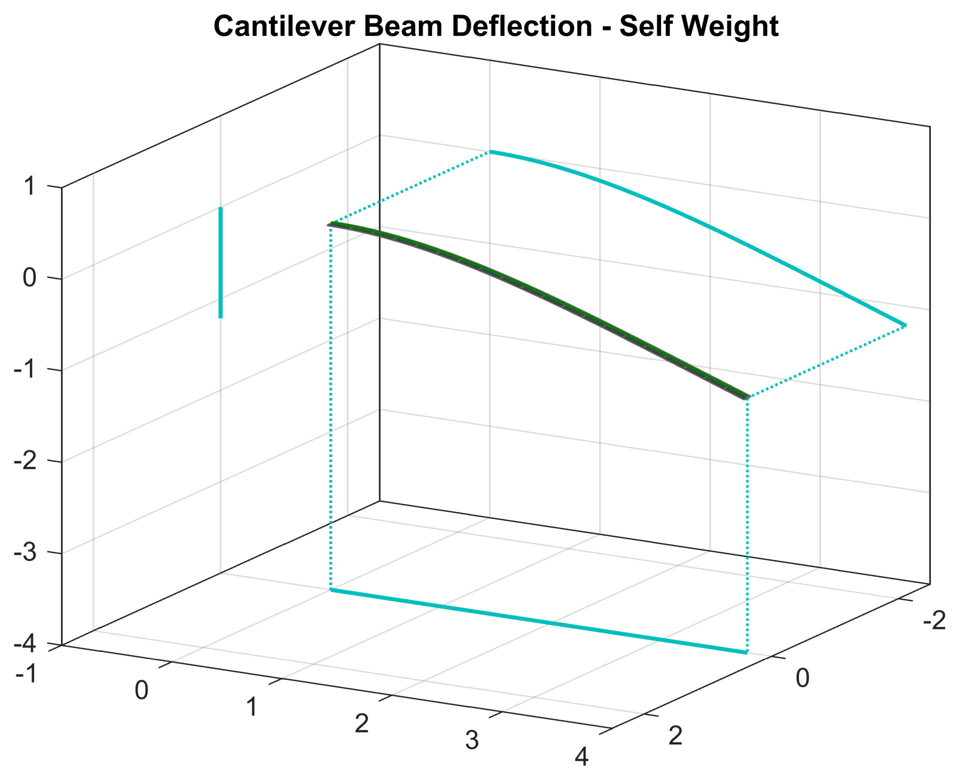
Simulation of the system is executed by the 'runSim' function.

```
Mcb_obj_static = runSim(0,1,'analysisType','static','fromObject',Mcb_obj_static);
```

```
                                         Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0          25          60157.8                     3.85e+05               1
     1          50          226.305       0.437923      3.22e+04               1
     2          75       0.00177315      0.0227796          93.2            1.09
     3         100      1.12938e-13    6.40234e-05       0.00075            1.09
     4         125      1.26681e-25     5.1139e-10      4.56e-09            1.09

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

runTime: 3.1387
```

Use the draw method to display deformed solution.

```
Mcb_obj_static.draw();
title('Cantilever Beam Deflection - Self Weight');
```



Various quantities of interest can be extracted from the simulated object. We will not go into this in detail for this example. For now, the tip displacement is extracted from the static solution using the command:

```
TipPosition = Mcb_obj_static.get_qoiValue('Cantilever_Beam','Gamma_G','Tidx','nt','Sidx','ns')
```

```
TipPosition = 3×1
                   0
   3.782617413135147
  -1.215085207173666
```

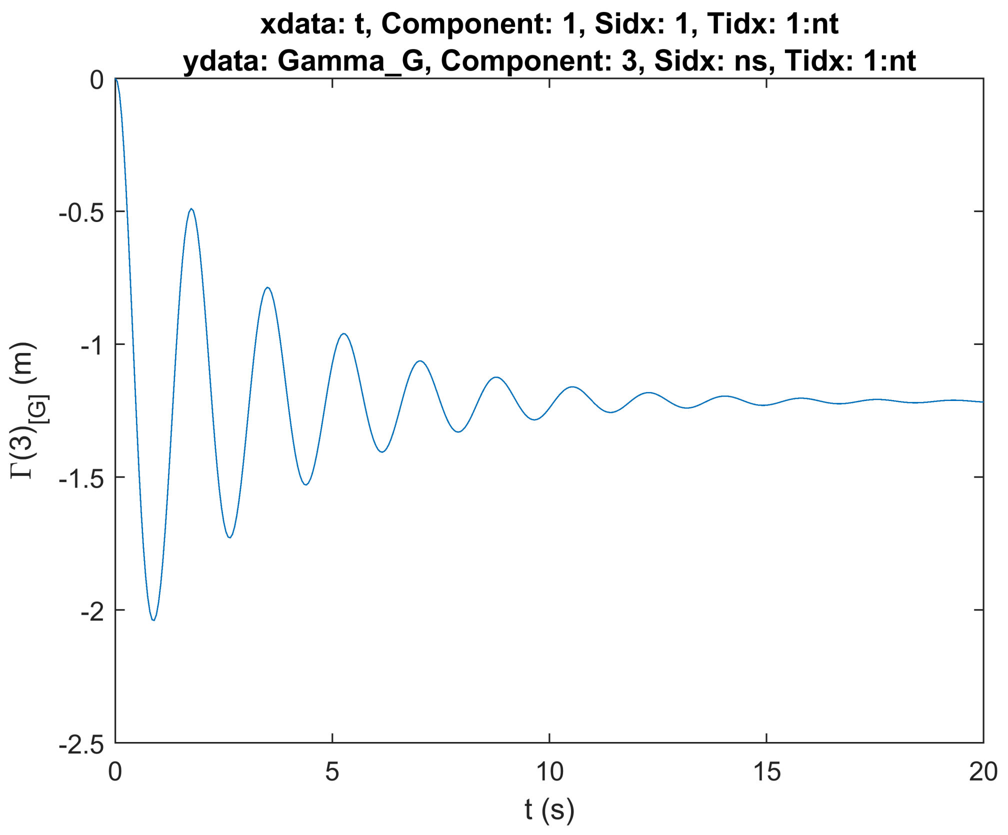
A dynamic simulation may also be performed on the cantilever system. The code below executes this over a simulated period of 20 seconds.

```
Mcb_obj_dynamic = Mcb_obj.copy;
t_start = 0;
t_end = 20;
Mcb_obj_dynamic = runSim(t_start,t_end,'fromObject',Mcb_obj_dynamic);
```

```
runTime: 19.5089
```

Again, we access the tip displacement of the beam, this time over the course of the dynamic simulation.

```
Mcb_obj_dynamic.generate_2dplot('Cantilever_Beam',{'t',1,'1','1:nt'},{'Gamma_G',3,'ns','1:nt'}
```



## Adding a Tip Load

For the final part of this example, two tip loads will be added in turn to the cantilever beam.

The first will be a follower force, this always has the same orientation relative to the beam at its application point (thus the force vector in the intrinsic coordinate system is constant). The force will initially act in the vertical downwards direction with a magnitude of -500 N.

First, lets retrieve the part object from the NBS_Master object.

```
Pcb_obj_static = Mcb_obj_static.get_partObj('Cantilever_Beam');
```

Now add the follower force.

```
Pcb_obj_static.tip_force_local = [0;0;-500];
Pcb_obj_static.tip_force_global = [0;0;0];
```

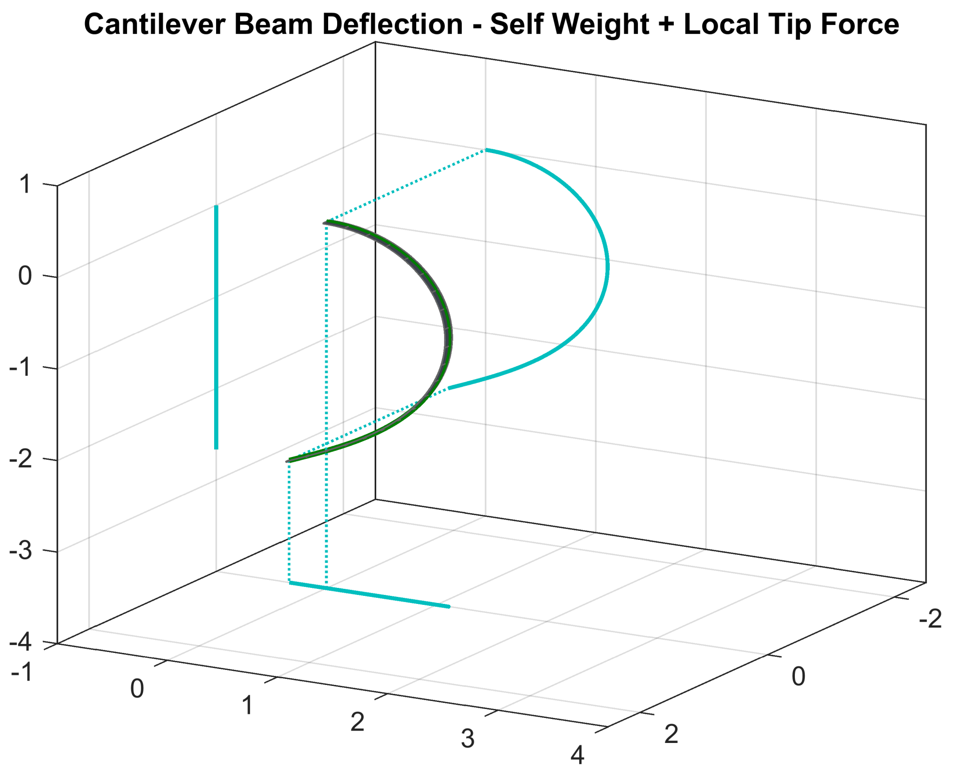
Calculate the static deflection.

```
Mcb_obj_static = runSim(0,1,'analysisType','static','fromObject',Mcb_obj_static);
```

```
                                         Norm of      First-order    Trust-region
 Iteration  Func-count     f(x)          step         optimality     radius
     0          25      3.58834e+06                   5.35e+06               1
     1          50      1.50406e+06             1     2.51e+06               1
     2          75          83706.8        1.4455     8.24e+05             2.5
     3        100         17.2913        0.286548        2.08e+04            3.61
     4        125     7.22057e-07      0.00427983            4.84            3.61
     5        150     2.13342e-21     8.83457e-07        2.36e-06            3.61
     6        175     1.10481e-24     3.53523e-14        2.08e-08            3.61

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

runTime: 2.7903
```

Plot the deformed shape using the draw method.

```
Mcb_obj_static.draw();
title('Cantilever Beam Deflection - Self Weight + Local Tip Force');
```



Next, this tip load is changed to a global force (constant in the global coordinate system).

```
Pcb_obj_static.tip_force_local = [0;0;0];
Pcb_obj_static.tip_force_global = [400;0;-100];
```

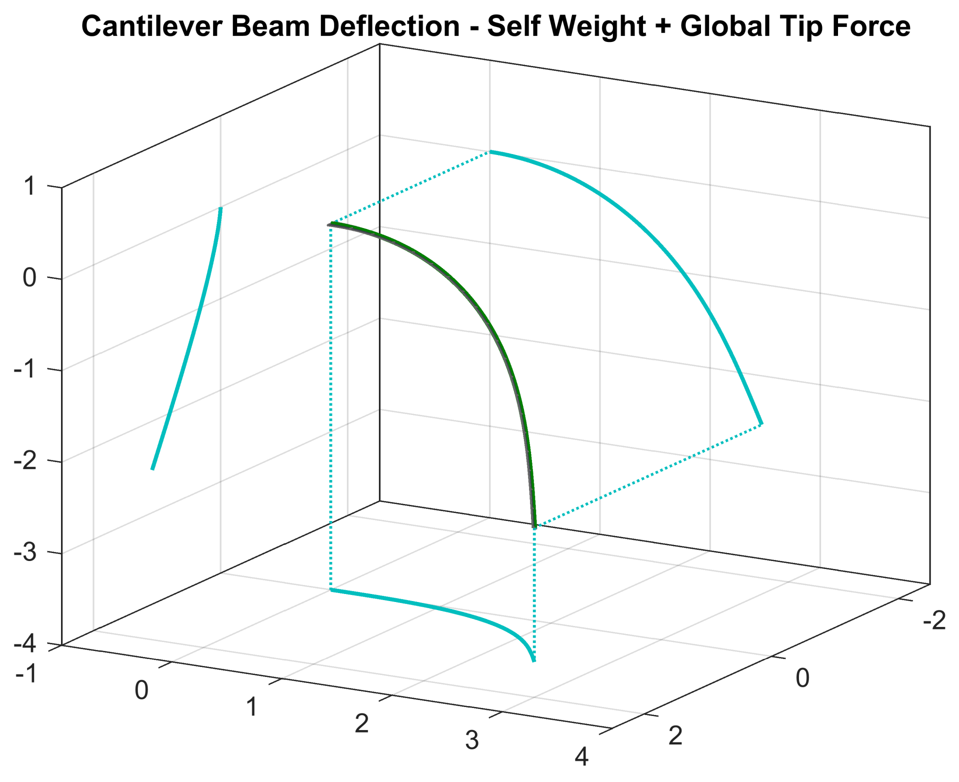
Again, simulate the static system.

```
Mcb_obj_static = runSim(0,1,'analysisType','static','fromObject',Mcb_obj_static);
```

```
                                         Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0          25      2.0958e+06                      4.24e+08               1
     1          50          449889             1        6.24e+07               1
     2          51          449889             1        6.24e+07               1
     3          52          449889          0.25        6.24e+07            0.25
     4          77          390879        0.0625         1.2e+08          0.0625
     5         102          328223       0.15625        3.43e+08           0.156
     6         103          328223       0.15625        3.43e+08           0.156
     7         128          265714     0.0390625        9.95e+07          0.0391
     8         153          214731     0.0976562        1.33e+08          0.0977
     9         154          214731      0.244141        1.33e+08           0.244
    10         179          181489     0.0610352        9.87e+07           0.061
    11         204          143470      0.152588        2.07e+08           0.153
    12         205          143470      0.152588        2.07e+08           0.153
    13         230          117983      0.038147        7.08e+07          0.0381
    14         255         90706.7     0.0953674        1.08e+08          0.0954
    15         280         71536.5     0.0953674        1.17e+08          0.0954
    16         305         60986.8     0.0953674        1.56e+08          0.0954
    17         330         53803.2     0.0953674        1.36e+08          0.0954
    18         355         49816.2     0.0953674        2.03e+08          0.0954
    19         380         39335.9     0.0953674        1.95e+08          0.0954
    20         405         11014.9      0.077996        1.65e+08          0.0954
    21         430         172.323     0.0203358        5.44e+07          0.0954
    22         455        0.145017    0.00359427           7e+05          0.0954
    23         480     4.32618e-08   5.84972e-05        1.06e+03          0.0954
    24         505     3.72511e-21   3.45833e-08        0.000198          0.0954
    25         530     1.15578e-22    1.6622e-14        4.42e-05          0.0954

Equation solved, fsolve stalled.

fsolve stopped because the relative size of the current step is less than the
default value of the step size tolerance squared and the vector of function values
is near zero as measured by the default value of the function tolerance.

<stopping criteria details>

runTime: 8.0543
```

Plot the static deformation of the beam.

```
Mcb_obj_static.draw();
title('Cantilever Beam Deflection - Self Weight + Global Tip Force');
```



## Clamped Patil Hodges Wing

The following test case is based upon the Patil/Hodges wing treated in [ref]; modified to give a spanwise variation in stiffness and chord.

Rather than input all of the parameter values in from the command line as before, this example illustrates the use of input files to generate the initial object. Specifically the files that contian the wing definition are

"+parameter_sets \ testCase_ClampedPatilHodgesWing.m" - to define the NBS_Master object

and

"+parameter_sets \ +FlexPart_Library \ flexPart_Patil_Hodges_16m_halfWing.m" - to define the flexible part representing the wing

These files are now briefly examined.

### NBS_Master Object

```
'.\+parameter_sets\testCase_ExampleClampedWing.m';
```

```
function O = testCase_ExampleClampedWing()

%==========================================================================
%>>Master Level Object<<%
O = NBS_Master;
%--------------------------------------------------------------------------
```

```
%//////Flight Condition
O.V = 30; %m/s                                                             free stream airspeed
O.rho = 0.0881; %kg/m^3                                                    air density
O.aerodynamics = 'strip_unsteady'; %                                         aerodynamic model
O.uVec_freeStream_G = [1;0;0]; %                                           free stream unit velocity vector
%--------------------------------------------------------------------------
O.grav_acc = 9.807;
O.gravVec_G = [0;0;-1];

O.plotBounds = [-0.5 0.5;0 1;-0.5 0.5]*16;

%==========================================================================
%>>Add Flexible Patil/Hodges Half-Wing

O_halfWing = parameter_sets.FlexPart_Library.flexPart_ExampleWing(O,O,'halfWing');
O_halfWing.alpha_root = 4; % the root angle of attack of the wing in degrees
O_halfWing.wingRoot_offset_A = [0;0;0];

%//////Applied Loads
O_halfWing.tip_force_global  = [0;0;0];
O_halfWing.tip_force_local   = [0;0;0];
O_halfWing.tip_moment_global = [0;0;0];
O_halfWing.tip_moment_local  = [0;0;0];
%//////Shape Functions
O_halfWing.shape_class_bend = 'chebyshev_1st';
O_halfWing.shape_class_twist = 'chebyshev_1st';
O_halfWing.shape_BCs_bend  = [0 1;1 1;1 1];
O_halfWing.shape_BCs_twist = [0 1;1 1;1 1];

O_halfWing.qth.n = 8;
O_halfWing.qsi.n = 6;
O_halfWing.qph.n = 6;

O_halfWing.populate_shape_set('PLOT',false);

%==========================================================================
set_dependent_properties(O);
%==========================================================================

end
```

Compared to the previous cantilever beam example, the main difference in the above code is the definition of the flight condition relavent to this aeroelastic test case. Specifically, the free stream velocity 'V', air density 'rho' and free-stream unit flow vector 'uvec_freeStream_G' are assigned values.

Additionally, an unsteady strip theory representation of the aerodynamics is requested.

## Flexible Part Object

```
'.\+parameter_sets\+FlexPart_Library\flexPart_ExampleWing.m';
```

```
function O = flexPart_ExampleWing(master_object,parent_object,name)

O = NBS_flexPart_nonlinear(master_object,name,'Parent',parent_object);
%--------------------------------------------------------------------------
    L = 16;
    ns = 201;
O.s = permute(linspace(0,L,ns),[1 3 2]);
```

```
    nAnodes = 17;
    Anode_Skew_Factor = 1.0; %dictates the degree to which the aero nodes are bunched towards the tip (set t
    Anode_distr = linspace(1,0,nAnodes).^Anode_Skew_Factor;
    s_aero_ = (1 - Anode_distr)*L;
O.s_aero = permute(s_aero_,[1 3 2]);
O.isAero = true;

O.h = ones(1,1,ns)*0.2;
%------------------------------------------------------------------------
    TaperScaling = reshape(linspace(2,0.5,ns),1,1,[]);
    EIxx = 2e4; EIzz = 4e6; GJ  = 1e4;
O.StiffnessMatrix = [
    EIxx 0    0;
    0    GJ   0;
    0    0    EIzz].*TaperScaling;
%O.StiffnessMatrix(4:6,4:6,:) = eye(3);
%------------------------------------------------------------------------
    damping_factor = 0.04;
O.DampingMatrix = damping_factor*O.StiffnessMatrix;
%------------------------------------------------------------------------
    I_tau = 0.1;
O.I_varTheta_ps_I = [
    I_tau 0 0;
    0 I_tau 0;
    0 0 I_tau];
O.I_varTheta_discrete_I = 0;
%------------------------------------------------------------------------
O.mps = 0.75;
O.msDiscrete = 0;
%------------------------------------------------------------------------
O.R_W_WE = eye(3);
O.KAPPA_0_I = zeros(3,1,ns);
%------------------------------------------------------------------------
O.c = ones(1,1,ns).*TaperScaling;
O.beam_cntr = ones(1,1,ns)*0.5;
%------------------------------------------------------------------------
O.AICs = sqrt(1-(0.5*(O.s_aero(2:end)+O.s_aero(1:end-1))).^2./L^2);
O.CrossSectionProfiles = 'NACA0012';
%------------------------------------------------------------------------

end
```

The flexible part object specifying the wing parameters is given above. Once again, in addition to the parameters of the previous example, extra properties pertaining to the aerodynamics are given here. Specifically:

- the provision of 's_aero' defines the aero panel edge locations, i.e. nAnodes = 17 -> 17 s_aero points -> 16 aerodynamic panels
- 'isAero' dictates that this flexible part can generate aerodynamic loads
- 'c' is the aerodynamic chord of the wing
- 'beam_cntr' is the fractional location - from 0-leading edge to 1 trailing edge - along the aerodynamic panels at which the beam axis lies
- 'AICs' these are the span varying influence coefficients that shape the aerodynamic distribution (elliptical in this example). These AIC coefficients directly multiply the loads calculated by the strip theory aerodynamics
- 'CrossSectionProfiles', used in plotting the wing. Here the cross section of the wing is given a NACA0012 profile.

## Simulate the system

To call the formulation using these files we use the 'testCase' keyword followed by an identifier string. The referenced file must have the address and file name of the form '.\+parameter_sets\testCase_[identifier string].m'.

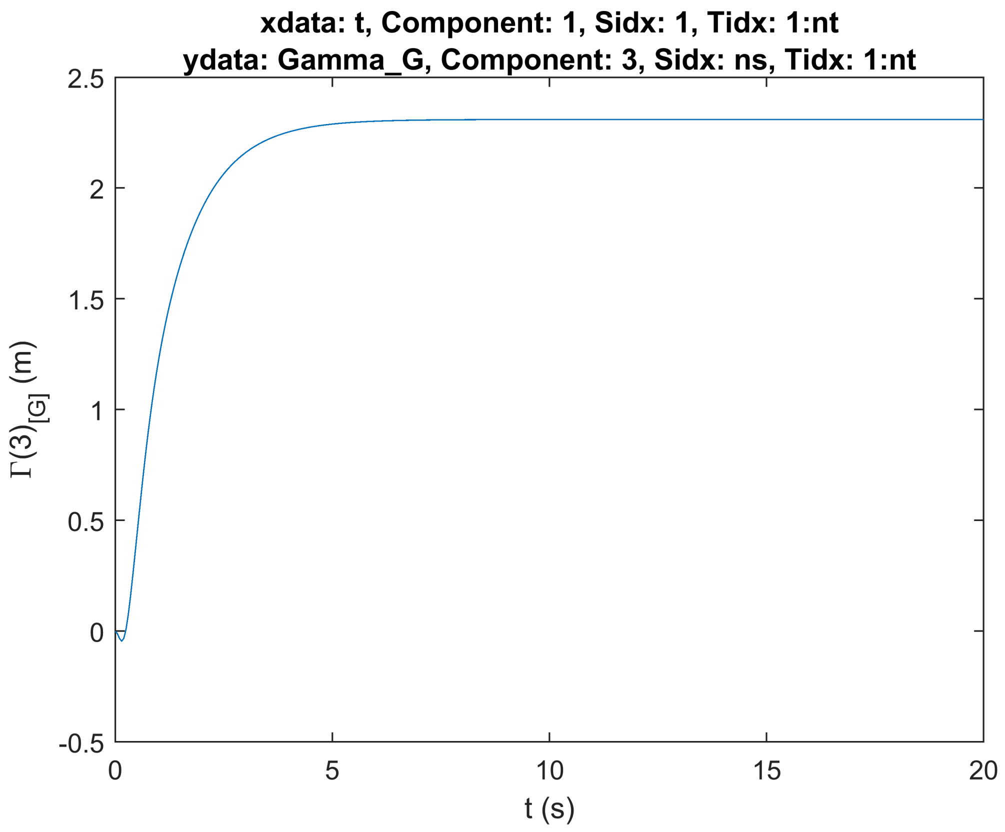
i.e. for this example:

```
T0 = 0; T1 = 20;
Mcph_obj_dynamic = runSim(T0,T1,'analysisType','dynamic','testCase','ExampleClampedWing');
```

```
runTime: 13.9742
```

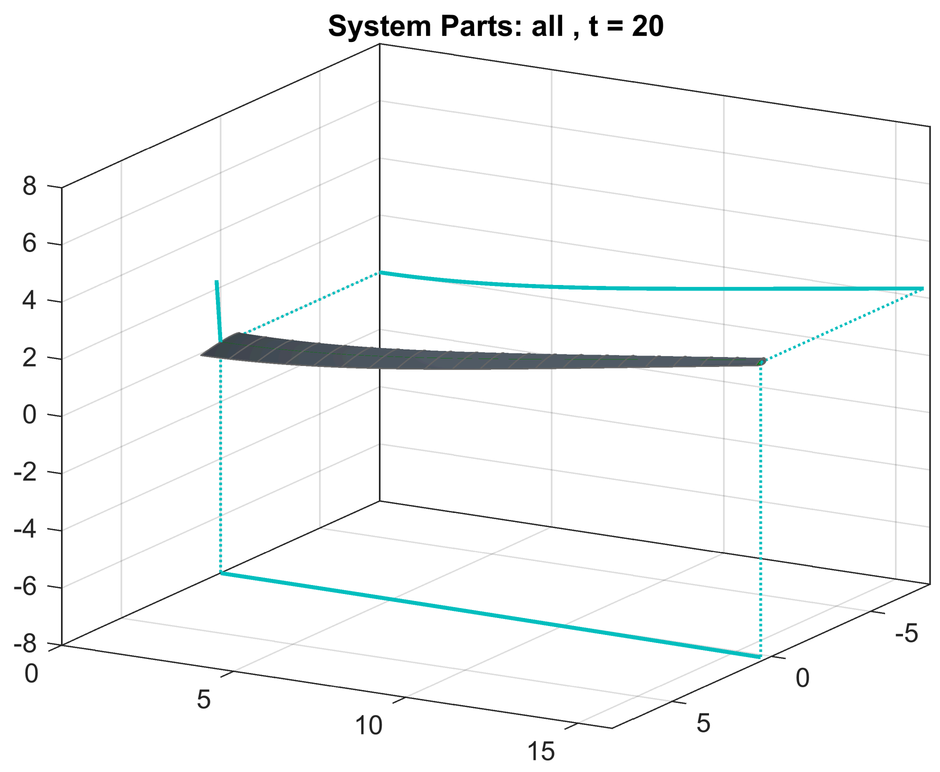
Generate a 2d plot of the tip displacement over time.

```
Mcph_obj_dynamic.generate_2dplot('halfWing',{'t',1,'1','1:nt'},{'Gamma_G',3,'ns','1:nt'});
```



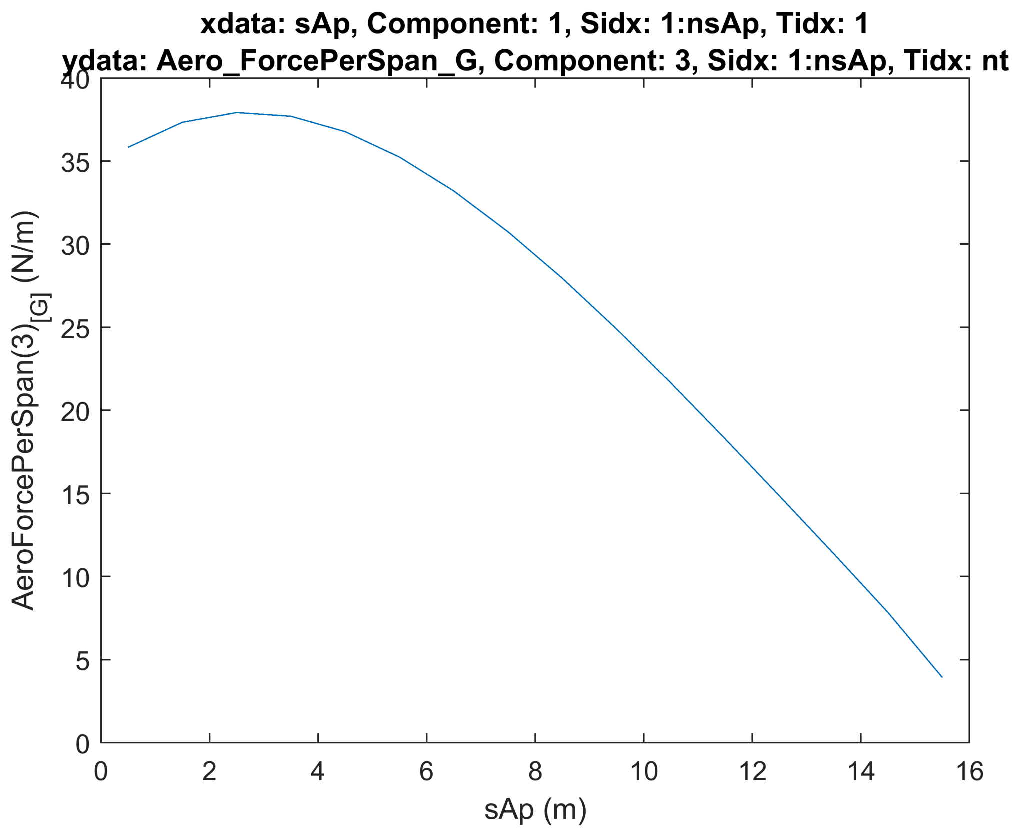
Plot the final shape of the wing at t = 20 seconds.

```
Mcph_obj_dynamic.draw('t',20);
```



Plot the aerodynamic lift distribution over the wing.

```
Mcph_obj_dynamic.generate_2dplot('halfWing',{'sAp',1,'1:nsAp','1'},{'Aero_ForcePerSpan_G',3,'1
```



We can choose to trim the wing using the method staticTrim. As an example, we consider here the the root angle of attack required to produce a net lift over the entire wing of 120 N. The method call takes the form detailed below with the user specifying the trim variables, trim quantities and the objects containing them.

```
property_to_vary = 'alpha_root';
object_containing_property = 'halfWing';
quantity_to_trim = 'Net_Lift';
object_containing_quantity_to_trim = 'aircraft';
trim_quantity_value = 120;

partName_partProperty = {object_containing_property , property_to_vary};
partName_QOIName_QOIValue = {object_containing_quantity_to_trim , quantity_to_trim , trim_quant

Mcph_obj_dynamic.staticTrim(partName_partProperty,partName_QOIName_QOIValue);
```

```
runTime: 4.1783
Property Value: 4
Output Value: 143.195
runTime: 4.5461
Property Value: 4
Output Value: 143.195
runTime: 4.521
Property Value: 3
Output Value: 130.1554
runTime: 4.5482
Property Value: 3
Output Value: 130.1554
runTime: 4.6094
Property Value: 2.2269
```

```
Output Value: 120.0231
runTime: 4.5836
Property Value: 2.2269
Output Value: 120.0231
runTime: 4.5832
Property Value: 2.2252
Output Value: 120.0009
runTime: 4.5762
Property Value: 2.2252
Output Value: 120.0009
runTime: 4.5135
Property Value: 2.2251
Output Value: 120.0001
runTime: 4.5798
Property Value: 2.2251
Output Value: 120.0001
runTime: 4.5626
Property Value: 2.2251
Output Value: 120
runTime: 4.5873
Property Value: 2.2251
Output Value: 120
runTime: 4.529
Property Value: 2.2251
Output Value: 120
runTime: 4.5583
Property Value: 2.2251
Output Value: 120

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>
```

## Full Aircraft Example

The following example will detail the setting up of a full aircraft model. The testcase file that will be loaded for this example is

```
'.\+parameter_sets\testCase_ExampleFullAC.m';
```

The contents of this test case file are shown below:

```
function O = testCase_ExampleFullAC()

%===========================================================================
%>>Master Level Object<<%
O = NBS_Master;
%---------------------------------------------------------------------------
%//////Flight Condition
O.V = 30; %m/s                                                              free stream airspeed
O.rho = 0.0881; %kg/m^3                                                     air density
O.aerodynamics = 'strip_steady'; %                                          aerodynamic model
```

```
O.uVec_freeStream_G = [1;0;0]; %                                free stream unit velocity vector
%------------------------------------------------------------------------------
O.grav_acc = 9.807*1;
O.gravVec_G = [0;0;-1];
%------------------------------------------------------------------------------
O.qRigidT = [0;0;0];
O.qRigidR = [0;0*pi/180;0];
%------------------------------------------------------------------------------
O.plotBounds = [[-1 1];[-1 1];[-1 1]]*40;

%==============================================================================
%>>Add Flexible Patil/Hodges Starboard Wing

O_SBWing = parameter_sets.FlexPart_Library.flexPart_ExampleWing(O,O,'SBWing');
O_SBWing.alpha_root = 4; % the root angle of attack of the wing in degrees
O_SBWing.sweep_root = 5; % the root sweep angle of the wing in degrees
O_SBWing.wingRoot_offset_A = [0;0.5;0];

%//////Shape Functions
O_SBWing.shape_class_bend = 'chebyshev_1st';
O_SBWing.shape_class_twist = 'chebyshev_1st';
O_SBWing.shape_BCs_bend  = [0 1;1 1;1 1];
O_SBWing.shape_BCs_twist = [0 1;1 1;1 1];

O_SBWing.qth.n = 8;
O_SBWing.qsi.n = 6;
O_SBWing.qph.n = 6;

O_SBWing.populate_shape_set('PLOT',false,'setName','Starboard Wing');

%==============================================================================
%>>Add Flexible Patil/Hodges Port Wing

O_PTWing = parameter_sets.FlexPart_Library.flexPart_ExampleWing(O,O,'PTWing');
O_PTWing.alpha_root = 4; O_PTWing.sweep_root = 5;
O_PTWing.wingRoot_offset_A = [0;-0.5;0];

%//////Shape Functions
O_PTWing.shape_class_bend = 'chebyshev_1st';
O_PTWing.shape_class_twist = 'chebyshev_1st';
O_PTWing.shape_BCs_bend  = [0 1;1 1;1 1];
O_PTWing.shape_BCs_twist = [0 1;1 1;1 1];

O_PTWing.qth.n = 8;
O_PTWing.qsi.n = 6;
O_PTWing.qph.n = 6;

O_PTWing.update_R_A_W('reflect',true);
O_PTWing.populate_shape_set('PLOT',false,'setName','Port Wing');

%==============================================================================
%>>Add Fuselage

O_Fuselage = parameter_sets.RigidPart_Library.rigidPart_ExampleFuselage(O,O,'Fuselage');
O_Fuselage.connection_idx_ParentObj = 1;
O_Fuselage.connectionOffset_C = [0;0;0];
O_Fuselage.R_C_W_0 = r_matrix([0;0;-1],pi/2);
O_Fuselage.set_dependent_properties();

%==============================================================================
%>>Add HTP

O_HTP = parameter_sets.RigidPart_Library.rigidPart_ExampleHTP(O,O,'HTP');
O_HTP.connection_idx_ParentObj = 1;
O_HTP.connectionOffset_C = [1;0;0]*O_Fuselage.s(end);
```

```
        O_HTP.R_C_W_0 = O_SBWing.R_A_W.'*r_matrix([0;1;0],0);
        O_HTP.R_C_W_0 = r_matrix([0;1;0],0);
        O_HTP.set_dependent_properties();

        %===========================================================================
        set_dependent_properties(O);
        %===========================================================================

        end
```

In the code above, one can see that this aircraft is constructed by first creating the NBS_Master object. Then adding a starboard wing, port wing, fuselage and horizontal tail plane (HTP) to the object. The individual files describing each part can be found in the folder locations

```
'.\+parameter_sets\+FlexPart_Library'; %for the flexible wings
'.\+parameter_sets\+RigidPart_Library'; %for the rigid fuselage and HTP parts
```

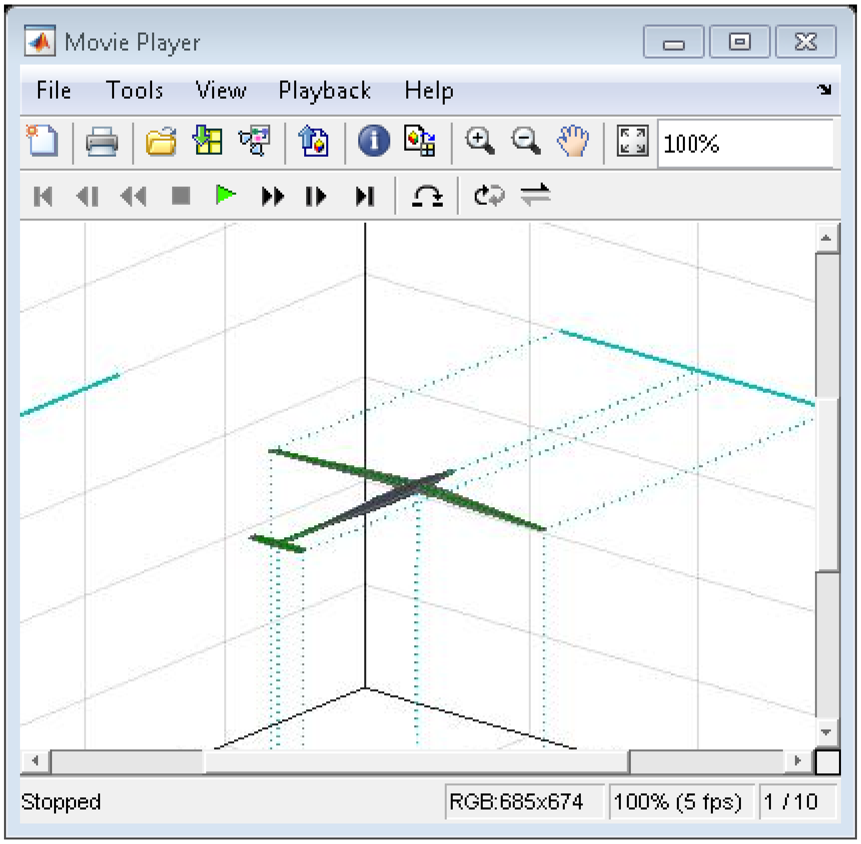
We simulate this system in the same way as the previous examples, using the runSim function.

```
T0 = 0; T1 = 2;
Mcph_obj_dynamic = runSim(T0,T1,'analysisType','dynamic','testCase','ExampleFullAC');
```

```
runTime: 31.9112
```

As well as the previous plot generating functions, a video can also be generated of the dynamic response of the system. This is achieved with the generate_video method; for example.

```
Mcph_obj_dynamic.generate_video('plotCoM',false,'playSpeed',1,'framerate',5,'az',130);
implay('video.avi');
```

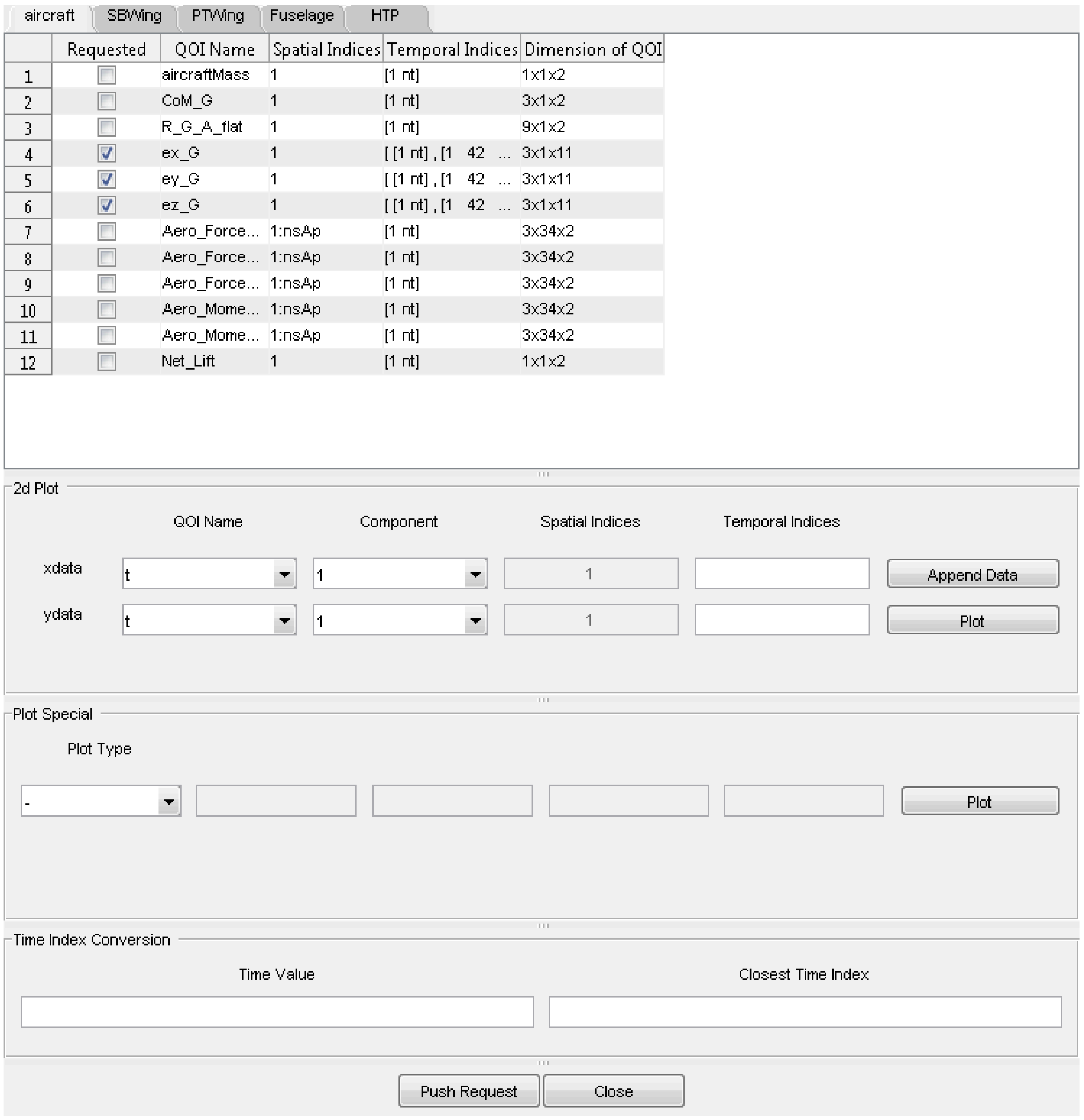


### A note on output generation

Note that the nonlinear beam shapes coded formulation contains a graphical user interface (GUI) to assist with the organisation of output quantities and the generation of plots. The functionality of this will be covered in detail in a separate guide. To use this feature the user must install the GUI layout toolbox in matlab, available from here.

Once the toolbox is installed, the GUI is accessible via the command:

```
Mcph_obj_dynamic.populate_QOIs();
```



| | Requested | QOI Name | Spatial Indices | Temporal Indices | Dimension of QOI |
|---|---|---|---|---|---|
| 1 | ☐ | aircraftMass | 1 | [1 nt] | 1x1x2 |
| 2 | ☐ | CoM_G | 1 | [1 nt] | 3x1x2 |
| 3 | ☐ | R_G_A_flat | 1 | [1 nt] | 9x1x2 |
| 4 | ✓ | ex_G | 1 | [ [1 nt] , [1 42 ... | 3x1x11 |
| 5 | ✓ | ey_G | 1 | [ [1 nt] , [1 42 ... | 3x1x11 |
| 6 | ✓ | ez_G | 1 | [ [1 nt] , [1 42 ... | 3x1x11 |
| 7 | ☐ | Aero_Force... | 1:nsAp | [1 nt] | 3x34x2 |
| 8 | ☐ | Aero_Force... | 1:nsAp | [1 nt] | 3x34x2 |
| 9 | ☐ | Aero_Force... | 1:nsAp | [1 nt] | 3x34x2 |
| 10 | ☐ | Aero_Mome... | 1:nsAp | [1 nt] | 3x34x2 |
| 11 | ☐ | Aero_Mome... | 1:nsAp | [1 nt] | 3x34x2 |
| 12 | ☐ | Net_Lift | 1 | [1 nt] | 1x1x2 |

2d Plot

| | QOI Name | Component | Spatial Indices | Temporal Indices |
|---|---|---|---|---|
| xdata | t | 1 | 1 | |
| ydata | t | 1 | 1 | |