Bottom-up Geometry Configuration Sensitivities with EGADS 1.17

Marshall C. Galbraith galbramc@mit.edu
Robert Haimes haimes@mit.edu
Massachusetts Institute of Technology

January 2020

Preamble

This document assumes the reader is familiar with the terminology and process for generating bottom-up constructed geometry using `EGADS`. Furthermore, the reader is assumed to be familiar with the concept of parametric sensitivities.

Introduction

Core geometry routines in `EGADS` are differentiated in order to facilitate analytic configuration sensitivity calculations for bottom up constructed geometric primitive Bodies. The sensitivity calculations are independent of whether the BRep Body is a *Wire*, *Face*, *Sheet* or *Solid*. Thus, this enables analytic sensitivities for configurations such as *sketches* that depend exclusively on the design parameters as well as *blend* and *rule* geometry where the sensitivities are propagated through sections.

Construction dependencies between components of the Body are tracked by populating the geometric entities of the Body with sensitivity information consistent with the construction process. This disambiguates, for example, whether an Edge is a function of a Face, if a Face is a function of the Edge, or if the Face and Edge are independent.

As an example, consider a *sketch* that is a single Edge consisting of a B-spline curve fit from a set of user specified points in the xy-plane. If the B-spline curve is closed, then it can be used to create a Face by making a closed Loop and adding the xy planar surface. The Face can be promoted to either a *FaceBody* or *SheetBody*. If the planar surface is independent of the user specified points, then Edge parameter sensitivities in the Body are independent of the Face. The differentiated routines in `EGADS` can track the sensitivity of the B-spline curve fit to the set of user specified points as well as the track that the sensitivities of the Face w.r.t. the points.

A Body can also be constructed by first fitting a B-spline surface to a set of user specified points and extracting the four bounding curves from the surface to construct the four Edges and the Face. In this case, the sensitivities of the Edges are identical to the the sensitivities of the Face. Again, routines in `EGADS` can be used to propagate the B-spline surface sensitivities w.r.t. the the user specified points used to fit the curves.

Note that `EGADS` does not track parametric dependencies between different Bodies. A parametric CAD-like (e.g., `OpenCSM`) system is required to construct a build tree of Bodies and propagate the design sensitivities through the tree. Similar to the construction process, where the leafs of the build tree involve calling `EGADS` to create a new Body, the sensitivity information is propagated though the tree by "dot" functions. The complete set of "dot" functions are listed in the `EGADS` header file "egads_dot.h".

The high-level (top down) functions `EG_solidBoolean`, `EG_fuseSheets`, `EG_intersection`, `EG_imprintBody`, `EG_filletBody`, `EG_chamferBody`, `EG_hollowBody`, `EG_sweep`, `EG_extrude`, and `EG_rotate`, in `EGADS` are not currently differentiated as they involve complex interactions in OpenCASCADE. Operations involving these functions in a build tree will need to propagate sensitivity information by other means.

Populating Geometric Sensitivities

Geometry constructed with calls to `EG_makeGeometry` are populated with sensitivity information using the function `EG_setGeometry_dot` shown in Listing 1. The *geom* and *data* arguments to `EG_setGeometry_dot` must the same as those used with `EG_makeGeometry`. In general, the *data* returned from `EG_getGeometry` cannot be used as the argument for `EG_setGeometry_dot` as `EG_makeGeometry` may alter the input *data*. The *data_dot* argument is the sensitvity of the *data*. The function `EG_hasGeometry_dot` allows querying if all geometry entities in an `ego` object are populated with sensitivities.

```
int EG_setGeometry_dot( ego geom, const double *data,
                                  const double *data_dot );
int EG_hasGeometry_dot( const ego geom );
```

Listing 1: Setting geometry sensitivity

The sensitvity of a point on the geometry is accessible via the `EG_evaluate_dot` function shown in Listing 2. This is similar to the `EG_evaluate` function, but returns both the point value and it's sensitvity. Furthermore, the sensitvity of the parametric derivatives are also returned. The argument *param_dot* is generally set to *NULL*, but is required if the parametric value includes sensitivities. This occurs, for example, when an arc-length spline knot sequence with sensitivities is used to evaluate points as is done internally when constructing *blend* and *rule* geometry.

```
int EG_evaluate_dot( const ego geom,
                     const double *param, const double *param_dot,
                     double *results, double *results_dot );
```

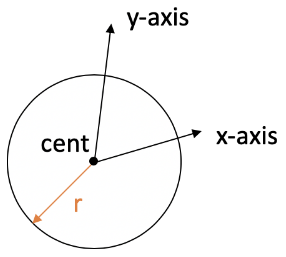
Listing 2: Geometry sensitivity evaluation

Circle Example

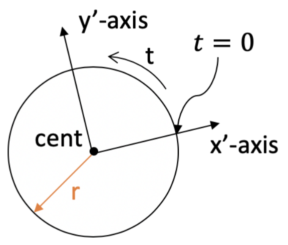
The example *src/circlc1_dot.c*, shown in Listings 3 through 5, illustrates constructing a simple *WireBody* circle and populating the body with sensitivity information using a two step process. The analytic sensitivities are then verified using finite differences via a "ping" test.

The function `makeCircleBody` for creating the circle *WireBody* is shown in Listing 3. The input data for the circle is a centroid, two axes that define the plane of the circle, and the radius of the circle. The axes in the input *data* for `EG_makeGeometry` is orthonormalized to create the geometry as illustrated in Fig. 1. By definition, the parametric t-coordinate is in the range $[0, 2\pi]$ with the branch cut occurring where the orthonormalized x$'$-axis intersects the circle. For simplicity, a single Node is placed at the branch cut to define a *OneNode* Edge. Because the axes are now orthonormal, the unit x$'$-axis needs to be retrieved from the circle with `EG_getGeometry`. The Node on the circle is created using the unit x$'$-axis and circle radius. Finally, the *WireBody* is created using the Edge and Loop.

The functions for setting the parametric sensitivity of the Circle *WireBody* are shown in Listing 4. This function does not generate any new geometry; rather, the sensitivity of the Node and Circle geometry are set by extracted them from *WireBody* during the traversal of it's hierarchy. Thus, in order to interpret the hierarchy, this function implicitly assumes that the circle *WireBody* was created with the function `makeCircleBody`.



(a) Input *data*



(b) Orthonormal axis *data* and t-coordinate

Figure 1: Input *data* and how it's orthornomialized to create a circle with `EG_makeGeometry`

```
int
makeCircleBody( ego context,          /* (in)  EGADS context    */
                const double *xcent,  /* (in)  Center           */
                const double *xaxis,  /* (in)  x-axis           */
                const double *yaxis,  /* (in)  y-axis           */
                const double r,       /* (in)  radius           */
                ego *ebody )          /* (out) Circle wire body */
{
  int     status = EGADS_SUCCESS;
  int     senses[1] = {SFORWARD}, oclass, mtype, *ivec=NULL;
  double  data[10], tdata[2], dx[3], *rvec=NULL;
  ego     ecircle, eedge, enode, eloop, eref;

  /* create the Circle */
  data[0] = xcent[0]; /* center */
  data[1] = xcent[1];
  data[2] = xcent[2];
  data[3] = xaxis[0]; /* x-axis */
  data[4] = xaxis[1];
  data[5] = xaxis[2];
  data[6] = yaxis[0]; /* y-axis */
  data[7] = yaxis[1];
  data[8] = yaxis[2];
  data[9] = r;        /* radius */
  status = EG_makeGeometry(context, CURVE, CIRCLE, NULL, NULL,
                           data, &ecircle);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_getGeometry(ecircle, &oclass, &mtype, &eref, &ivec, &rvec);
  if (status != EGADS_SUCCESS) goto cleanup;

  /* get the orthonormal x-axis from the circle */
  dx[0] = rvec[3];
  dx[1] = rvec[4];
  dx[2] = rvec[5];

  /* create the Node for the Edge */
  data[0] = xcent[0] + dx[0]*r;
  data[1] = xcent[1] + dx[1]*r;
  data[2] = xcent[2] + dx[2]*r;
  status = EG_makeTopology(context, NULL, NODE, 0,
                             data, 0, NULL, NULL, &enode);
  if (status != EGADS_SUCCESS) goto cleanup;

  /* make the Edge on the Circle */
  tdata[0] = 0;
  tdata[1] = TWOPI;

  status = EG_makeTopology(context, ecircle, EDGE, ONENODE,
                           tdata, 1, &enode, NULL, &eedge);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_makeTopology(context, NULL, LOOP, CLOSED,
                           NULL, 1, &eedge, senses, &eloop);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_makeTopology(context, NULL, BODY, WIREBODY,
                           NULL, 1, &eloop, NULL, ebody);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EGADS_SUCCESS;

cleanup:
  if (status != EGADS_SUCCESS) {
    printf(" Failure %d in makeCircleBody\n", status);
  }
  EG_free(ivec); ivec = NULL;
  EG_free(rvec); rvec = NULL;

  return status;
}
```

Listing 3: Function for constructing an Circle *WireBody*

```
int
setCircleBody_dot( ego ebody,        /* (in/out) body with sensitivities */
                   const double *xcent,     /* (in)  Center             */
                   const double *xcent_dot, /* (in)  Center sensitivity */
                   const double *xaxis,     /* (in)  x-axis             */
                   const double *xaxis_dot, /* (in)  x-axis sensitivity */
                   const double *yaxis,     /* (in)  y-axis             */
                   const double *yaxis_dot, /* (in)  y-axis sensitivity */
                   const double r,          /* (in)  radius             */
                   const double r_dot)      /* (in)  radius sensitivity */
{
  int    status = EGADS_SUCCESS;
  int    nnode, nedge, nloop, oclass, mtype, *senses;
  double data[10], data_dot[10], dx[3], dx_dot[3], *rvec=NULL, *rvec_dot=NULL;
  ego    ecircle, *enodes, *eloops, *eedges, eref;

  /* get the Loop from the Body */
  status = EG_getTopology(ebody, &eref, &oclass, &mtype,
                          data, &nloop, &eloops, &senses);
  if (status != EGADS_SUCCESS) goto cleanup;

  /* get the Edge from the Loop */
```

```
status = EG_getTopology(eloops[0], &eref, &oclass, &mtype,
                        data, &nedge, &eedges, &senses);
if (status != EGADS_SUCCESS) goto cleanup;

/* get the Node and the Circle from the Edge */
status = EG_getTopology(eedges[0], &ecircle, &oclass, &mtype,
                        data, &nnode, &enodes, &senses);
if (status != EGADS_SUCCESS) goto cleanup;

/* set the Circle data and sensitivity */
data[0] = xcent[0]; /* center */
data[1] = xcent[1];
data[2] = xcent[2];
data[3] = xaxis[0]; /* x-axis */
data[4] = xaxis[1];
data[5] = xaxis[2];
data[6] = yaxis[0]; /* y-axis */
data[7] = yaxis[1];
data[8] = yaxis[2];
data[9] = r;        /* radius */

data_dot[0] = xcent_dot[0]; /* center */
data_dot[1] = xcent_dot[1];
data_dot[2] = xcent_dot[2];
data_dot[3] = xaxis_dot[0]; /* x-axis */
data_dot[4] = xaxis_dot[1];
data_dot[5] = xaxis_dot[2];
data_dot[6] = yaxis_dot[0]; /* y-axis */
data_dot[7] = yaxis_dot[1];
data_dot[8] = yaxis_dot[2];
data_dot[9] = r_dot;        /* radius */

status = EG_setGeometry_dot(ecircle, CURVE, CIRCLE, NULL, data, data_dot);
if (status != EGADS_SUCCESS) goto cleanup;

status = EG_getGeometry_dot(ecircle, &rvec, &rvec_dot);
if (status != EGADS_SUCCESS) goto cleanup;

/* get the orthonormal x-axis from the circle */
dx[0] = rvec[3];
dx[1] = rvec[4];
dx[2] = rvec[5];
dx_dot[0] = rvec_dot[3];
dx_dot[1] = rvec_dot[4];
dx_dot[2] = rvec_dot[5];

/* set the sensitivity of the Node */
data[0] = xcent[0] + dx[0]*r;
data[1] = xcent[1] + dx[1]*r;
data[2] = xcent[2] + dx[2]*r;
data_dot[0] = xcent_dot[0] + dx_dot[0]*r + dx[0]*r_dot;
data_dot[1] = xcent_dot[1] + dx_dot[1]*r + dx[1]*r_dot;
data_dot[2] = xcent_dot[2] + dx_dot[2]*r + dx[2]*r_dot;
status = EG_setGeometry_dot(enodes[0], NODE, 0, NULL , data, data_dot);
if (status != EGADS_SUCCESS) goto cleanup;
```

```
  status = EGADS_SUCCESS;

cleanup:
  if (status != EGADS_SUCCESS) {
    printf(" Failure %d in setCircleBody_dot\n", status);
  }
  EG_free(rvec); rvec = NULL;
  EG_free(rvec_dot); rvec_dot = NULL;

  return status;
}
```

Listing 4: Function setting the sensitivity of a Circle *WireBody* constructed with `makeCircleBody`

After extracting the geometric entities, the sensitivity of the Circle is set via a call to `EG_setGeometry_dot` using the *data* array and *data_dot* arrays. Note that the *data* array must contain the exact same data used to construct the circle geometry in `makeCircleBody`. This is required to ensure the sensitivities are propagated through the orthonormalization of the axes.

Since the Node was created using the orthonormal x′-axis from `EG_getGeometry`, the sensitivity of the x′-axis must also retrieved with `EG_getGeometry_dot`. The sensitivity of the Node is then set using the sensitivities of the orthonormal x′-axis and the radius. Note that the *data_dot* is a linearization of the *data* array using the chain-rule.

The *main* routine in this example constructs a Circle *WireBody* and verifies the analytic sensitivities by comparing them with sensitivities computed via finite differences. The loop shown in Listing 5 iterates over all the parameters used to create the circle. For each parameter, the sensitivity of that parameter is set to one and the Circle *WireBody* object `ebody1` is populated with analytic sensitivities. The call to `EG_hasGeometry_dot` verifies that all geometric entities in `ebody1` are populated with sensitivity information. Next, a second perturbed Circle *WireBody*, `ebody2`, is created and the tessellation of `ebody1` is mapped to `ebody2` in order for the tessellation to be used to compute finite difference sensitivities. The function `pingBodies` computes the finite difference sensitivity of all surfaces, curves, and Nodes, and compares them with the analytic sensitivities.

```
  for (iparam = 0; iparam < 10; iparam++) {

    /* set the analytic sensitivity of the body */
    x_dot[iparam] = 1.0;
    status = setCircleBody_dot(ebody1,
                               xcent  , xcent_dot,
                               xax    , xax_dot,
                               yax    , yax_dot,
                               x[9]   , x_dot[9]);
    if (status != EGADS_SUCCESS) goto cleanup;
    x_dot[iparam] = 0.0;

    status = EG_hasGeometry_dot(ebody1);
    if (status != EGADS_SUCCESS) goto cleanup;

    /* make a perturbed Circle for finite difference */
    x[iparam] += dtime;
    status = makeCircleBody(context, xcent, xax, yax, x[9], &ebody2);
    if (status != EGADS_SUCCESS) goto cleanup;
    x[iparam] -= dtime;

    /* map the tessellation */
    status = EG_mapTessBody(tess1, ebody2, &tess2);
    if (status != EGADS_SUCCESS) goto cleanup;

    /* ping the bodies */
    status = pingBodies(tess1, tess2, dtime, iparam,
                        "Circle", 1e-7, 1e-7, 1e-7);
    if (status != EGADS_SUCCESS) goto cleanup;

    EG_deleteObject(tess2);
    EG_deleteObject(ebody2);
  }
```

Listing 5: Analytic and finite difference sensitivities for each circle parameter

The verification test of curve sensitivities (via checking all Edges) in `pingBodies` is shown in Listing 6. The analytic sensitivity of the curve is obtained via a call to `EG_evaluate_dot`, which also returns the coordinate of the point, `p1`. The finite difference sensitivity is computed by evaluating point `p2` on the perturbed geometry and taking the difference with `p1`. The finite difference does include a removal of any changes in the parameter in the perturbed Body. Finally, the analytic sensitivity, `p1_dot`, is verified to be within a thin tolerance of the finite difference sensitivity, `fd_dot`.

```
  for (iedge = 0; iedge < nedge; iedge++) {

    status = EG_getInfo(eedges1[iedge], &oclass, &mtype, &top, &prev, &next);
    if (status != EGADS_SUCCESS) goto cleanup;
    if (mtype == DEGENERATE) continue;

    /* extract the tessellation from the original edge */
    status = EG_getTessEdge(tess1, iedge+1, &np1, &x1, &t1);
    if (status != EGADS_SUCCESS) goto cleanup;

    /* get the tessellation from the perturbed edge */
    status = EG_getTessEdge(tess2, iedge+1, &np2, &x2, &t2);
    if (status != EGADS_SUCCESS) goto cleanup;

    for (n = 0; n < np1; n++) {

      /* evaluate original edge and velocities*/
      status = EG_evaluate_dot(eedges1[iedge], &t1[n], NULL, p1, p1_dot);
      if (status != EGADS_SUCCESS) goto cleanup;

      /* evaluate perturbed edge */
      status = EG_evaluate(eedges2[iedge], &t2[n], p2);
      if (status != EGADS_SUCCESS) goto cleanup;

      /* compute the configuration velocity based on finite difference */
      fd_dot[0] = (p2[0] - p1[0])/dtime - p2[3]*(t2[n] - t1[n])/dtime;
      fd_dot[1] = (p2[1] - p1[1])/dtime - p2[4]*(t2[n] - t1[n])/dtime;
      fd_dot[2] = (p2[2] - p1[2])/dtime - p2[5]*(t2[n] - t1[n])/dtime;

      for (d = 0; d < 3; d++) {
        if (fabs(p1_dot[d] - fd_dot[d]) > etol) {
          printf("%s Edge %d iparam=%d, diff fabs(%+le - %+le) = %+le > %e\n",
                 shape, iedge+1, iparam, p1_dot[d], fd_dot[d], fabs(p1_dot[d] -
     fd_dot[d]), etol);
          nerr++;
        }
      }

      //printf("p1_dot = (%+f, %+f, %+f)\n", p1_dot[0], p1_dot[1], p1_dot[2]);
      //printf("fd_dot = (%+f, %+f, %+f)\n", fd_dot[0], fd_dot[1], fd_dot[2]);
      //printf("\n");
    }
  }
```
Listing 6: Edge analytic and finite difference sensitivity equivalence test

Alternative Circle Example

The previous circle example assumed that the body created with `makeCircleBody` was simple enough that the Node and Circle geometry could be readily extracted from the body hierarchy. However, extracting all the geometry from a more complex Body, such as one generated from a sophisticated sketching routine, may not be feasible.

An alternative approach that does not require extracting the geometry from the Body to set sensitivity information is given in *src/circle2_dot.c*. Rather than extracting the geometry from the Body, the function `setCircleBody_dot`, shown in Listing 7, builds a circle *WireBody* and populates the geometric entities with sensitivity information during the construction process. The last call to `EG_makeTopology` to create the *WireBody* will preserve the sensitivity information if all geometry in the children are populated with sensitivity information.

After constructing the second Body with sensitivity information, the sensitivities are copied over to the original body with a call to `EG_copyGeometry_dot`.

```
int
setCircleBody_dot( ego ebody,        /* (in/out) body with sensitivities */
                   const double *xcent,     /* (in)  Center             */
                   const double *xcent_dot, /* (in)  Center sensitivity */
                   const double *xaxis,     /* (in)  x-axis             */
                   const double *xaxis_dot, /* (in)  x-axis sensitivity */
                   const double *yaxis,     /* (in)  y-axis             */
                   const double *yaxis_dot, /* (in)  y-axis sensitivity */
                   const double r,          /* (in)  radius             */
                   const double r_dot)      /* (in)  radius sensitivity */
{
  int    status = EGADS_SUCCESS;
  int    oclass, mtype, senses[1] = {SFORWARD}, *ivec=NULL;
  double data[10], data_dot[10], dx[3], dx_dot[3];
  double tdata[2], *rvec=NULL, *rvec_dot=NULL;
  ego    context, ecircle, enode, eloop, eedge, ebody2, eref;

  status = EG_getContext(ebody, &context);
  if (status != EGADS_SUCCESS) goto cleanup;

  /* the Circle data and sensitivity */
  data[0] = xcent[0]; /* center */
  data[1] = xcent[1];
  data[2] = xcent[2];
  data[3] = xaxis[0]; /* x-axis */
  data[4] = xaxis[1];
  data[5] = xaxis[2];
  data[6] = yaxis[0]; /* y-axis */
  data[7] = yaxis[1];
  data[8] = yaxis[2];
  data[9] = r;        /* radius */

  data_dot[0] = xcent_dot[0]; /* center */
  data_dot[1] = xcent_dot[1];
  data_dot[2] = xcent_dot[2];
  data_dot[3] = xaxis_dot[0]; /* x-axis */
  data_dot[4] = xaxis_dot[1];
  data_dot[5] = xaxis_dot[2];
```

```
data_dot[6] = yaxis_dot[0]; /* y-axis */
data_dot[7] = yaxis_dot[1];
data_dot[8] = yaxis_dot[2];
data_dot[9] = r_dot;        /* radius */

/* create the Circle */
status = EG_makeGeometry(context, CURVE, CIRCLE, NULL, NULL,
                         data, &ecircle);
if (status != EGADS_SUCCESS) goto cleanup;

/* set the Circle sensitivity */
status = EG_setGeometry_dot(ecircle, CURVE, CIRCLE, NULL, data, data_dot);
if (status != EGADS_SUCCESS) goto cleanup;

status = EG_getGeometry(ecircle, &oclass, &mtype, &eref, &ivec, &rvec);
if (status != EGADS_SUCCESS) goto cleanup;
EG_free(rvec);

status = EG_getGeometry_dot(ecircle, &rvec, &rvec_dot);
if (status != EGADS_SUCCESS) goto cleanup;

/* get the orthonormal x-axis from the circle */
dx[0] = rvec[3];
dx[1] = rvec[4];
dx[2] = rvec[5];
dx_dot[0] = rvec_dot[3];
dx_dot[1] = rvec_dot[4];
dx_dot[2] = rvec_dot[5];

/* the Node and it's sensitvities */
data[0] = xcent[0] + dx[0]*r;
data[1] = xcent[1] + dx[1]*r;
data[2] = xcent[2] + dx[2]*r;
data_dot[0] = xcent_dot[0] + dx_dot[0]*r + dx[0]*r_dot;
data_dot[1] = xcent_dot[1] + dx_dot[1]*r + dx[1]*r_dot;
data_dot[2] = xcent_dot[2] + dx_dot[2]*r + dx[2]*r_dot;

/* create the Node for the Edge */
status = EG_makeTopology(context, NULL, NODE, 0,
                         data, 0, NULL, NULL, &enode);
if (status != EGADS_SUCCESS) goto cleanup;

/* set the sensitvity of the Node */
status = EG_setGeometry_dot(enode, NODE, 0, NULL , data, data_dot);
if (status != EGADS_SUCCESS) goto cleanup;

/* make the Edge on the Circle */
tdata[0] = 0;
tdata[1] = TWOPI;

status = EG_makeTopology(context, ecircle, EDGE, ONENODE,
                         tdata, 1, &enode, NULL, &eedge);
if (status != EGADS_SUCCESS) goto cleanup;

status = EG_makeTopology(context, NULL, LOOP, CLOSED,
                             NULL, 1, &eedge, senses, &eloop);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_makeTopology(context, NULL, BODY, WIREBODY,
                             NULL, 1, &eloop, NULL, &ebody2);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_copyGeometry_dot(ebody2, NULL, NULL, ebody);
  if (status != EGADS_SUCCESS) goto cleanup;

  EG_deleteObject(ebody2);

  status = EGADS_SUCCESS;

cleanup:
  if (status != EGADS_SUCCESS) {
    printf(" Failure %d in makeCircleBody_dot\n", status);
  }
  EG_free(ivec); ivec = NULL;
  EG_free(rvec); rvec = NULL;
  EG_free(rvec_dot); rvec_dot = NULL;

  return status;
}
```
Listing 7: Setting sensitivities while building a Body

NACA Airfoil Spline Example

The NACA 4-series airfoil is defined by an equation for the half thickness, y_t,

$$y_t(x;t) = 5t\left[0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4\right] \qquad x \in [0,1]$$

and an equation for camber, y_c,

$$y_c(x;m,p) = \begin{cases} \frac{m}{p^2}\left(2px - x^2\right) & 0 \le x < p \\ \frac{m}{(1-p)^2}\left((1-2p) + 2px - x^2\right) & p \le x \le 1 \end{cases} \qquad x \in [0,1]$$

The parameters, t, m, and p are the maximum thickness, camber, and the location of the maximum camber respectively. The upper and lower coordinates are obtained by adding the thickness perpendicular to the camber via the equations

$$\begin{aligned} x_U &= x - y_t \sin\theta, & y_U &= y_c + y_t\cos\theta \\ x_L &= x + y_t \sin\theta, & y_L &= y_c - y_t\cos\theta \end{aligned}$$

where

$$\theta = \arctan\left(\frac{dy_c}{dx}\right)$$
$$\frac{dy_c}{dx} = \begin{cases} \frac{2m}{p^2}\left(p - x\right) & 0 \le x < p \\ \frac{2m}{(1-p)^2}\left(p - x\right) & p \le x \le 1 \end{cases}$$

The thickness equation does not close at $x = 1$ and hence produces a blunt trailing edge. Modifying the last coefficient (i.e. changing -0.1015 to -0.1036) produces a sharp trailing edge with minimal other modifications to the thickness.

The example given in *src/naca_dot.c* produces a spline approximation of either a blunt or sharp trailing edge NACA airfoil. The function `makeNacaBody` produces a *FaceBody* by creating a set of points from the NACA equation and fitting a spline to those points and then defining a *plane* for the Face.

The portion of the `makeNacaBody` that evaluates the NACA equation at a cosine distribution of points and then creates a spline fit via a call to `EG_approximate` is shown in Listing 8.

```
int makeNacaBody( ego context,        /* (in) EGADS context     */
                  const int sharpte, /* (in) sharp or blunt TE */
                  const double m,    /* (in) camber            */
                  const double p,    /* (in) maxloc            */
                  const double t,    /* (in) thickness         */
                  ego *ebody )      /* (out) result pointer    */
{
  int     status = EGADS_SUCCESS;
  int     ipnt, *header=NULL, sizes[2], sense[3], nedge, oclass, mtype;
  double  *pnts = NULL, *rvec = NULL;
  double  data[18], tdata[2], tle;
  double  x, y, zeta, s, yt, yc, theta, ycm, dycm;
  ego     eref, enodes[4], eedges[3], ecurve, eline, eloop, eplane, eface;
```

```
/* mallocs required by Windows compiler */
pnts = (double*)EG_alloc((3*NUMPNTS)*sizeof(double));
if (pnts == NULL) {
  status = EGADS_MALLOC;
  goto cleanup;
}

/* points around airfoil (upper and then lower) */
for (ipnt = 0; ipnt < NUMPNTS; ipnt++) {
  zeta = TWOPI * ipnt / (NUMPNTS-1);
  s    = (1 + cos(zeta)) / 2;

  if (sharpte == 0) {
    yt = 5.*t * (A * sqrt(s) + s * (B + s * (C + s * (D + s * Eb))));
  } else {
    yt = 5.*t * (A * sqrt(s) + s * (B + s * (C + s * (D + s * Es))));
  }

  if (s < p) {
    ycm  = (s * (2*p -   s)) / (p*p);
    dycm = (    (2*p - 2*s)) / (p*p);
  } else {
    ycm  = ((1-2*p) + s * (2*p -   s)) / pow(1-p,2);
    dycm = (              (2*p - 2*s)) / pow(1-p,2);
  }
  yc    = m * ycm;
  theta = atan(m * dycm);

  if (ipnt < NUMPNTS/2) {
    x = s  - yt * sin(theta);
    y = yc + yt * cos(theta);
  } else if (ipnt == NUMPNTS/2) {
    x = 0.;
    y = 0.;
  } else {
    x = s  + yt * sin(theta);
    y = yc - yt * cos(theta);
  }

  pnts[3*ipnt  ] = x;
  pnts[3*ipnt+1] = y;
  pnts[3*ipnt+2] = 0.;
}

/* create spline curve from upper TE, to LE, to lower TE
 *
 * finite difference must use knots equally spaced (sizes[1] == -1)
 * arc-length based knots (sizes[1] == 0) causes the t-space to change.
 */
sizes[0] = NUMPNTS;
sizes[1] = KNOTS;
status = EG_approximate(context, 0, DXYTOL, sizes, pnts, &ecurve);
if (status != EGADS_SUCCESS) goto cleanup;
```

Listing 8: NACA airfoil spline creating in makeNacaBody

The next portion of makeNacaBody, shown in Listing 9, creates Nodes at the trailing edge and leading edge of the airfoil spline. Because the spline is an approximation, the spline may not always exactly go through the $(0, 0, 0)$ coordinate. Thus, the leading edge Node is created by evaluating the spline at the parametric t-value that corresponds to the middle point used in the spline fit. This results in a Node that is coincident with the spline and located close to $(0, 0, 0)$. A *line* is created to close the Loop for a blunt trailing edge using the first and last spline fit points.

```
/* create Node at upper trailing edge */
ipnt = 0;
data[0] = pnts[3*ipnt  ];
data[1] = pnts[3*ipnt+1];
data[2] = pnts[3*ipnt+2];
status = EG_makeTopology(context, NULL, NODE, 0,
                         data, 0, NULL, NULL, &enodes[0]);
if (status != EGADS_SUCCESS) goto cleanup;

/* node at leading edge as a function of the spline */
status = EG_getGeometry(ecurve, &oclass, &mtype, &eref, &header, &rvec);
if (status != EGADS_SUCCESS) goto cleanup;

ipnt = (NUMPNTS - 1) / 2 + 3; /* index, with knot offset of 3 (cubic)*/
tle = rvec[ipnt];             /* t-value (should be very close to (0,0,0) */

status = EG_evaluate(ecurve, &tle, data);
if (status != EGADS_SUCCESS) goto cleanup;

status = EG_makeTopology(context, NULL, NODE, 0,
                         data, 0, NULL, NULL, &enodes[1]);
if (status != EGADS_SUCCESS) goto cleanup;

if (sharpte == 0) {
  /* create Node at lower trailing edge */
  ipnt = NUMPNTS - 1;
  data[0] = pnts[3*ipnt  ];
  data[1] = pnts[3*ipnt+1];
  data[2] = pnts[3*ipnt+2];
  status = EG_makeTopology(context, NULL, NODE, 0,
                           data, 0, NULL, NULL, &enodes[2]);
  if (status != EGADS_SUCCESS) goto cleanup;

  enodes[3] = enodes[0];
} else {
  enodes[2] = enodes[0];
}

/* make Edge for upper surface */
tdata[0] = 0;   /* t-value at lower TE */
tdata[1] = tle;

/* construct the upper Edge */
status = EG_makeTopology(context, ecurve, EDGE, TWONODE,
                         tdata, 2, &enodes[0], NULL, &eedges[0]);
if (status != EGADS_SUCCESS) goto cleanup;
```

```
/* make Edge for lower surface */
tdata[0] = tdata[1]; /* t-value at leading edge */
tdata[1] = 1;        /* t value at upper TE */

/* construct the lower Edge */
status = EG_makeTopology(context, ecurve, EDGE, TWONODE,
                         tdata, 2, &enodes[1], NULL, &eedges[1]);
if (status != EGADS_SUCCESS) goto cleanup;

if (sharpte == 0) {
  nedge = 3;

  /* create line segment at trailing edge */
  ipnt = NUMPNTS - 1;
  data[0] = pnts[3*ipnt  ];
  data[1] = pnts[3*ipnt+1];
  data[2] = pnts[3*ipnt+2];
  data[3] = pnts[0] - data[0];
  data[4] = pnts[1] - data[1];
  data[5] = pnts[2] - data[2];

  status = EG_makeGeometry(context, CURVE, LINE, NULL, NULL, data, &eline);
  if (status != EGADS_SUCCESS) goto cleanup;

  /* make Edge for this line */
  tdata[0] = 0;
  tdata[1] = sqrt(data[3]*data[3] + data[4]*data[4] + data[5]*data[5]);

  status = EG_makeTopology(context, eline, EDGE, TWONODE,
                           tdata, 2, &enodes[2], NULL, &eedges[2]);
  if (status != EGADS_SUCCESS) goto cleanup;
} else {
  nedge = 2;
}
```

Listing 9: Node and Edge creation in makeNacaBody

Finally, a Loop is created from the Edges along with a planar surface to create a *FaceBody* as shown in Listing 10.

```
/* create loop of the Edges */
sense[0] = SFORWARD;
sense[1] = SFORWARD;
sense[2] = SFORWARD;

status = EG_makeTopology(context, NULL, LOOP, CLOSED,
                         NULL, nedge, eedges, sense, &eloop);
if (status != EGADS_SUCCESS) goto cleanup;

/* create a plane for the loop */
data[0] = 0.;
data[1] = 0.;
data[2] = 0.;
data[3] = 1.; data[4] = 0.; data[5] = 0.;
data[6] = 0.; data[7] = 1.; data[8] = 0.;
```

```
status = EG_makeGeometry(context, SURFACE, PLANE, NULL, NULL, data, &eplane);
if (status != EGADS_SUCCESS) goto cleanup;

/* create the Face from the plane and the Loop */
status = EG_makeTopology(context, eplane, FACE, SFORWARD,
                         NULL, 1, &eloop, sense, &eface);
if (status != EGADS_SUCCESS) goto cleanup;

/* create the FaceBody */
status = EG_makeTopology(context, eplane, BODY, FACEBODY,
                         NULL, 1, &eface, sense, ebody);
if (status != EGADS_SUCCESS) goto cleanup;
```

Listing 10: Surface and *FaceBody* creation in `makeNacaBody`

The function `setNacaBody_dot` populates a body created with `makeNacaBody` with sensitivity information w.r.t. the NACA parameters t, m, and p. After extracting the surface, curve, and Nodes from the Body, the spline sensitivity is computed by again generating the NACA points for the spline along with the point sensitivity from differentiating the NACA equations as shown in Listing 11. Using the NACA points and point sensitivities, the spline curve sensitivity is computed with a call to `EG_approximate_dot`.

```
/* points around airfoil (upper and then lower) */
for (ipnt = 0; ipnt < NUMPNTS; ipnt++) {
  zeta = TWOPI * ipnt / (NUMPNTS-1);
  s    = (1 + cos(zeta)) / 2;

  if (sharpte == 0) {
    yt     = 5.*t     * (A * sqrt(s) + s * (B + s * (C + s * (D + s * Eb))));
    yt_dot = 5.*t_dot * (A * sqrt(s) + s * (B + s * (C + s * (D + s * Eb))));
  } else {
    yt     = 5.*t     * (A * sqrt(s) + s * (B + s * (C + s * (D + s * Es))));
    yt_dot = 5.*t_dot * (A * sqrt(s) + s * (B + s * (C + s * (D + s * Es))));
  }

  if (s < p) {
    ycm      =         ( s * (2*p -   s)) / (p*p);
    ycm_dot  = p_dot * (-2 * s * (p - s)) / (p*p*p);
    dycm     =         (     (2*p - 2*s)) / (p*p);
    dycm_dot = p_dot * (-2 * (p   - 2*s)) / (p*p*p);
  } else {
    ycm      =         ( (1-2*p) + s * (2*p -   s)) / pow(1-p,2);
    ycm_dot  = p_dot * (        2*(s - p)*(s - 1)) / pow(p-1,3);
    dycm     =         (                (2*p - 2*s)) / pow(1-p,2);
    dycm_dot = p_dot * (          -2*(1 + p - 2*s)) / pow(p-1,3);
  }
  yc         = m * ycm;
  yc_dot     = m_dot * ycm + m * ycm_dot;
  theta      = atan(m * dycm);
  theta_dot = (m_dot * dycm + m * dycm_dot) / (1 + m*m*dycm*dycm);

  if (ipnt < NUMPNTS/2) {
    x     = s  - yt * sin(theta);
    y     = yc + yt * cos(theta);
```

```
      x_dot =         - yt_dot * sin(theta) - theta_dot * yt * cos(theta);
      y_dot = yc_dot + yt_dot * cos(theta) - theta_dot * yt * sin(theta);
    } else if (ipnt == NUMPNTS/2) {
      x     = 0;
      y     = 0;
      x_dot = 0;
      y_dot = 0;
    } else {
      x     = s  + yt * sin(theta);
      y     = yc - yt * cos(theta);
      x_dot =        + yt_dot * sin(theta) + theta_dot * yt * cos(theta);
      y_dot = yc_dot - yt_dot * cos(theta) + theta_dot * yt * sin(theta);
    }

    pnts[3*ipnt  ] = x;
    pnts[3*ipnt+1] = y;
    pnts[3*ipnt+2] = 0.;

    pnts_dot[3*ipnt  ] = x_dot;
    pnts_dot[3*ipnt+1] = y_dot;
    pnts_dot[3*ipnt+2] = 0.;
  }

  /* populate spline curve sensitivities */
  sizes[0] = NUMPNTS;
  sizes[1] = KNOTS;
  status = EG_approximate_dot(ecurve, 0, DXYTOL, sizes, pnts, pnts_dot);
  if (status != EGADS_SUCCESS) goto cleanup;
```

Listing 11: NACA airfoil spline sensitivity in `setNacaBody_dot`

Setting the Node and blunt trailing edge line sensitivities is shown in Listing 12. For the leading edge Node, the mid t-value and the sensitivity of the t-value is extracted from the spline data. If arc-length spaced knot sequence are used in the spline fit (the default) then the t-values also have sensitivities w.r.t. the spline points. Thus, both the t-value and it's sensitivity must be used in the call to `EG_evaluate_dot` to get the sensitivity for the leading edge Node.

```
  /* set the sensitivity of the Node at trailing edge */
  ipnt = 0;
  status = EG_setGeometry_dot(enodes[0], NODE, 0, NULL,
                              &pnts[3*ipnt], &pnts_dot[3*ipnt]);
  if (status != EGADS_SUCCESS) goto cleanup;

  /* set the sensitivity of the Node at leading edge */
  status = EG_getGeometry_dot(ecurve, &rvec, &rvec_dot);
  if (status != EGADS_SUCCESS) goto cleanup;

  ipnt = (NUMPNTS - 1) / 2 + 3; /* index, with knot offset of 3 (cubic)*/
  tle     = rvec[ipnt];         /* t-value (should be very close to (0,0,0) */
  tle_dot = rvec_dot[ipnt];     /* t-value sensitivity */

  status = EG_evaluate_dot(ecurve, &tle, &tle_dot, data, data_dot);
  if (status != EGADS_SUCCESS) goto cleanup;
  status = EG_setGeometry_dot(enodes[1], NODE, 0, NULL, data, data_dot);
  if (status != EGADS_SUCCESS) goto cleanup;
```

```
if (sharpte == 0) {
  /* trailing edge line and lower trailing edge node from the 3rd edge */
  status = EG_getTopology(eedges[2], &eline, &oclass, &mtype, data,
                          &nchild, &echildren, &senses);
  if (status != EGADS_SUCCESS) goto cleanup;
  enodes[2] = echildren[0]; /* lower trailing edge */

  /* set the sensitivity of the Node at lower trailing edge */
  ipnt = NUMPNTS - 1;
  status = EG_setGeometry_dot(enodes[2], NODE, 0, NULL,
                              &pnts[3*ipnt], &pnts_dot[3*ipnt]);
  if (status != EGADS_SUCCESS) goto cleanup;

  /* set the sensitivity of the line segment at trailing edge */
  ipnt = NUMPNTS - 1;
  data[0] = pnts[3*ipnt  ];
  data[1] = pnts[3*ipnt+1];
  data[2] = pnts[3*ipnt+2];
  data[3] = pnts[0] - data[0];
  data[4] = pnts[1] - data[1];
  data[5] = pnts[2] - data[2];

  data_dot[0] = pnts_dot[3*ipnt  ];
  data_dot[1] = pnts_dot[3*ipnt+1];
  data_dot[2] = pnts_dot[3*ipnt+2];
  data_dot[3] = pnts_dot[0] - data_dot[0];
  data_dot[4] = pnts_dot[1] - data_dot[1];
  data_dot[5] = pnts_dot[2] - data_dot[2];

  status = EG_setGeometry_dot(eline, CURVE, LINE, NULL, data, data_dot);
  if (status != EGADS_SUCCESS) goto cleanup;
}
```

Listing 12: NACA airfoil Node sensitivities in `setNacaBody_dot`

Finally the plane is not a function of any of the NACA airfoil parameters, and hence all of the sensitivities for the surface are simply set to zero as shown in Listing 13.

```
/* plane data */
data[0] = 0.;
data[1] = 0.;
data[2] = 0.;
data[3] = 1.; data[4] = 0.; data[5] = 0.;
data[6] = 0.; data[7] = 1.; data[8] = 0.;

/* set the sensitivity of the plane */
data_dot[0] = 0.;
data_dot[1] = 0.;
data_dot[2] = 0.;
data_dot[3] = 0.; data_dot[4] = 0.; data_dot[5] = 0.;
data_dot[6] = 0.; data_dot[7] = 0.; data_dot[8] = 0.;

status = EG_setGeometry_dot(eplane, SURFACE, PLANE, NULL, data, data_dot);
if (status != EGADS_SUCCESS) goto cleanup;
```

Listing 13: NACA airfoil surface sensitivity in `setNacaBody_dot`

Ruled NACA Wing Example

The example *src/ruled_naca_dot.c* builds upon the previous NACA airfoil example to generate a simple Ruled wing. This example uses the exact same `makeNacaBody` and `setNacaBody_dot` functions from the previous example, and instead modifies the *main* function to build a Ruled wing using two airfoil sections.

In the Listing 14 a NACA airfoil *FaceBody* is first created and the Face is extracted from the body as the first section. The second section is created by copying the first section while applying a z-translation transformation. The two sections are then used to create the Ruled wing with the call to `EG_ruled`.

```
    /* make a NACA body for the 1st section */
    status = makeNacaBody( context, sharpte, x[im], x[ip], x[it], &enaca1 );
    if (status != EGADS_SUCCESS) goto cleanup;

    /* get the Face from the FaceBody */
    status = EG_getTopology(enaca1, &eref, &oclass, &mtype,
                            range, &nface, &efaces, &senses);
    if (status != EGADS_SUCCESS) goto cleanup;
    esecs1[0] = efaces[0];

    /* make a 2nd section via a transformation */
    mat[ 0] = 1.; mat[ 1] = 0.; mat[ 2] = 0.; mat[ 3] = 0.;
    mat[ 4] = 0.; mat[ 5] = 1.; mat[ 6] = 0.; mat[ 7] = 0.;
    mat[ 8] = 0.; mat[ 9] = 0.; mat[10] = 1.; mat[11] = x[is];

    status = EG_makeTransform(context, mat, &exform);
    if (status != EGADS_SUCCESS) goto cleanup;
    status = EG_copyObject(esecs1[0], exform, &esecs1[1]);
    if (status != EGADS_SUCCESS) goto cleanup;
    status = EG_deleteObject(exform);
    if (status != EGADS_SUCCESS) goto cleanup;

    /* create the ruled body */
    status = EG_ruled(nsec, esecs1, &ebody1);
    if (status != EGADS_SUCCESS) goto cleanup;
```

Listing 14: Creating a Ruled wing

In Listing 15, the airfoil *FaceBody* is first populated with sensitivity information. Note that the object `esecs1[0]` is populated with this function call since it is the Face object contained within `enaca1`. Next, the sensitivities are transferred to the second section via the call to `EG_copyGeometry_dot`. This call includes the transformation matrix along with the sensitivity of the transformation matrix. Once the two sections are populated with sensitivity information, the Ruled wing is populated with sensitivity information with a call to `EG_ruled_dot`. Similar to calls to `EG_copyGeometry_dot` where the *data* array must be consistent with the data used to construct the initial geometric object, the sections in the call to `EG_ruled_dot` must be equivalent to the sections used to create the body.

```
 /* set the analytic sensitivity of the airfoil */
 x_dot[iparam] = 1.0;
 status = setNacaBody_dot(enaca1,
                          sharpte,
                             x[im], x_dot[im],
                             x[ip], x_dot[ip],
                             x[it], x_dot[it]);
if (status != EGADS_SUCCESS) goto cleanup;

/* esecs1[0] is automatically populated with sensitivities as
 * esecs1[0] == efaces[0] which is part of enaca1 */

/* copy sensitivities via the transformation */
mat[ 0] = 1.; mat[ 1] = 0.; mat[ 2] = 0.; mat[ 3] = 0.;
mat[ 4] = 0.; mat[ 5] = 1.; mat[ 6] = 0.; mat[ 7] = 0.;
mat[ 8] = 0.; mat[ 9] = 0.; mat[10] = 1.; mat[11] = x[is];

mat_dot[ 0] = 0.; mat_dot[ 1] = 0.; mat_dot[ 2] = 0.; mat_dot[ 3] = 0.;
mat_dot[ 4] = 0.; mat_dot[ 5] = 0.; mat_dot[ 6] = 0.; mat_dot[ 7] = 0.;
mat_dot[ 8] = 0.; mat_dot[ 9] = 0.; mat_dot[10] = 0.; mat_dot[11] = x_dot[is];

status = EG_copyGeometry_dot(esecs1[0], mat, mat_dot, esecs1[1]);
if (status != EGADS_SUCCESS) goto cleanup;

/* set the ruled body sensitivities */
status = EG_ruled_dot(ebody1, nsec, esecs1);
if (status != EGADS_SUCCESS) goto cleanup;
x_dot[iparam] = 0.0;
```
Listing 15: Setting Ruled wing sensitivities