

## **FastShip Technical Note**

### **Using the Macro to Match an Input Sectional Area Curve**

1 February, 1999  
Larry Leibman  
Bruce Hays

#### **1.0 Introduction**

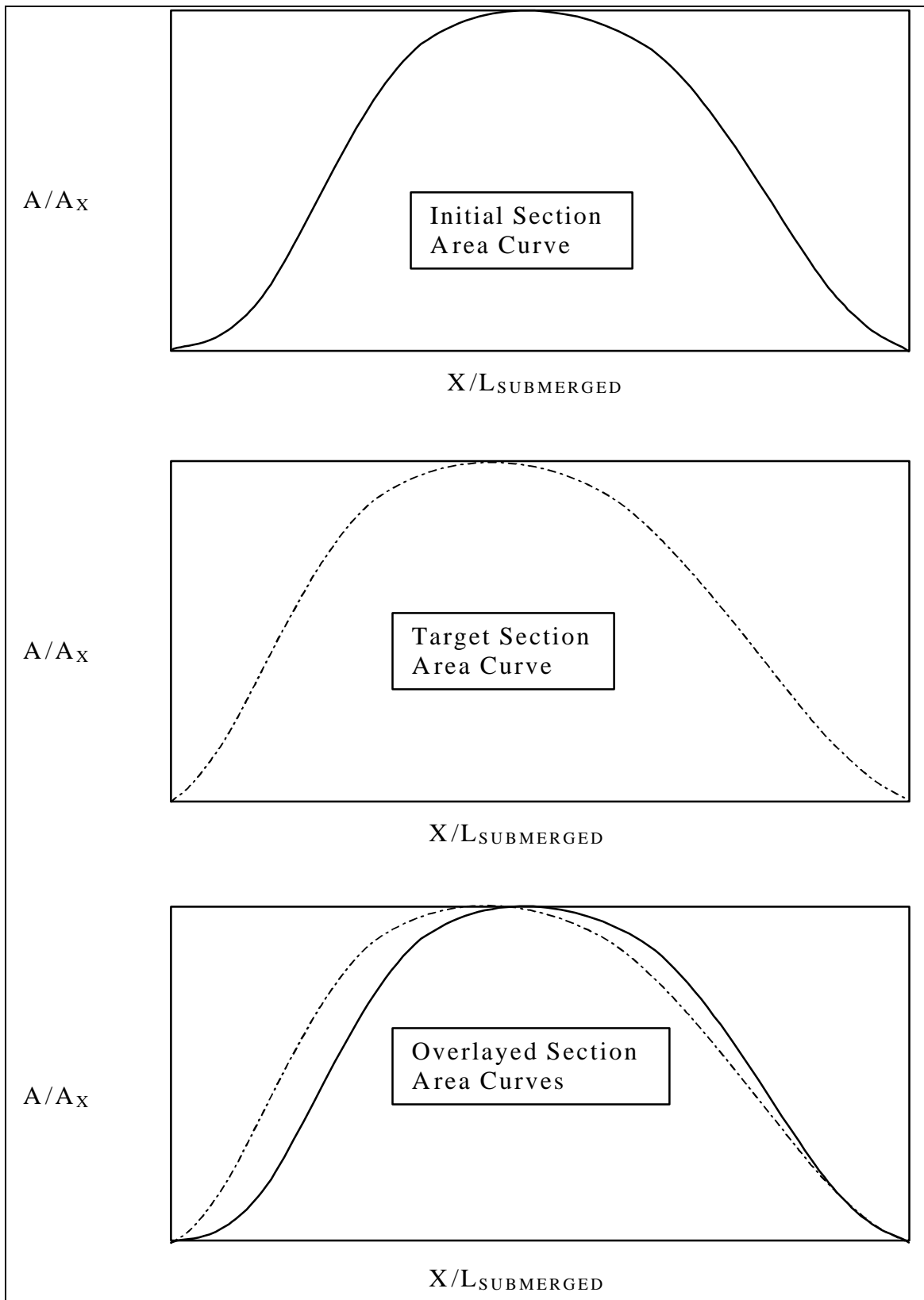
A *FastShip* macro has been written to automatically modify a ship surface model so that the sectional area curve matches a user-defined input sectional area curve. The macro is a combination of *FastShip* commands and PERL, and is a good example of a more complex automation of *FastShip* commands that can increase your productivity.

#### **2.0 Description**

The macro, *matchSAC.mac*, allows a user to match a user-specified sectional area curve. In the following discussion we make reference to the “*initial*” and “*target*” sectional area curves. The initial sectional area curve refers to that curve associated with the *FastShip* hull form model prior to starting the macro. The target sectional area curve refers to that curve associated with the hull form that the user is attempting to develop (i.e. the modified hull form). The basic approach to converting the initial hull form to the target hull form involves shifting control net vertices longitudinally in order to cause an associated shift in immersed area. Since moving the vertices affects the hull in a larger region than just the station at which it is located, this is by necessity an iterative process. One of the main parts of this process is to determine how much to shift the vertices. The shift is determined by the difference between the initial section area curve and the target section area curve. The procedure for determining the shift involves two steps.

1. The first step is to overlay the target section area curve over the initial section area curve. In doing so, we make the basic and important assumption that the overall submerged length of the hull (i.e. the dimensional length of the sectional area curve) is unchanged between initial and target hullforms. Therefore, in overlaying the two curves, we non-dimensionalize both curves as  $x/L_{\text{SUBMERGED}}$  which varies from 0 to 1. We assume that the minimum abscissa in the target curve corresponds to 0 and the maximum corresponds to 1. Thus, the actual  $L_{\text{pp}}$  value is not important in this case.
2. The second step is to split the initial and target curves at their respective maximums. This is done in order to avoid having a multi-valued relationship between section area and longitudinal position, which would make it difficult to solve for  $x$  give a section area value.

This methodology is depicted in Figure 1 and Figure 2.



**Figure 1 Overlaying Section Area Curves**

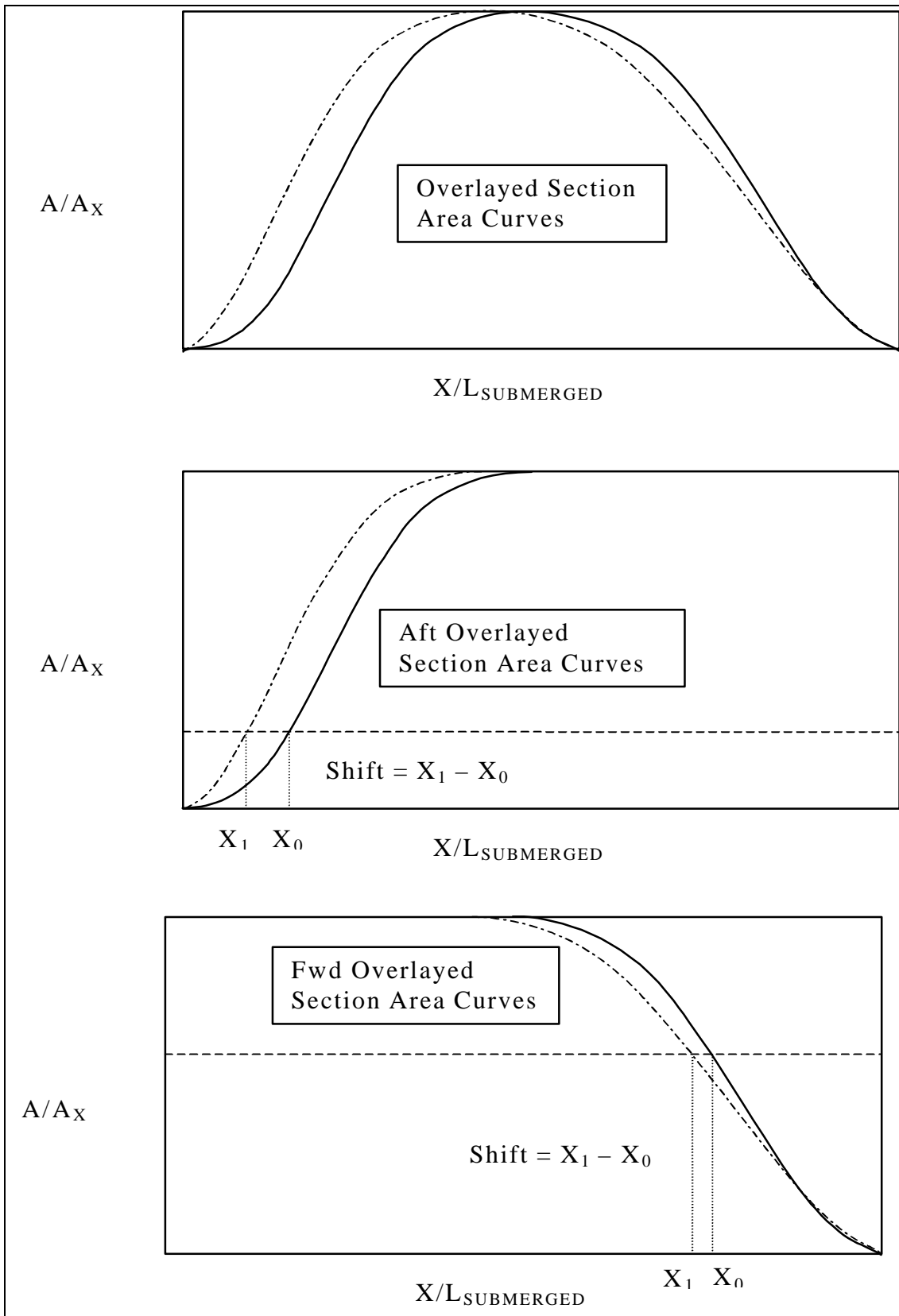


Figure 2 Splitting the Section Area Curves

### 3.0 Technique

Prior to starting the macro it is important that the user define a number of stations along the hull to adequately define the initial sectional area curve. More stations should be placed in areas with more rapid change in underwater shape and in areas of abrupt discontinuities. The macro will make use of whatever stations are defined when the macro is started. Furthermore, the user may wish to refine the sections to some tolerance before starting the macro, as again the macro will make use of whatever section refinement level has been defined.

With the hull on the screen, read the macro by selecting **Read Macro...** from the **Utils** menu, and selecting *matchSAC.mac* (this assumes that you have already put the *matchSAC.mac* file on your hard disk in a suitable location, such as the *config* subdirectory).

The macro will prompt you as follows:

- **Enter the filename containing the target section area curve:**  
The user should enter the name of the file containing target sectional area curve at the command line. It is not necessary to use double quotes around the filename, even if the name has spaces in it. If no file path is given, it is assumed that the specified file exists in the FastShip data path. The format of this file is described later.
- **Enter the part specification for the hull parts to be modified:**  
This part specification should include all surfaces for which hydrostatic properties should be included in the sectional area curve calculation and to which modifications should be applied to achieve the target section area curve. If, for example, the model includes a superstructure which is not to be modified, exclude this in the part specification. The macro will automatically supply a “...” to the end if this specification if the user does not do so. It is not necessary to use double quotes around the specification, even if the name has spaces in it.
- **Does the hull need to be mirrored?:**  
Enter y or n at the command line to specify whether or not only half of a symmetric hull has been modeled.
- **Enter the flotation plane constant, i.e. z value:**  
Enter a numerical value representing the flotation plane at which the sectional area curve calculations are to be performed. Remember to prefix the correct sign to the number to account for positive z direction.

When all of the arguments are entered, the macro begins its calculations. The user will notice hydrostatics calculations being performed followed by a series of activity associated with commands being executed in the macro. These are the *modify-net* commands necessary to adjust the hull form shape. Depending on the speed of your computer and the size of your surface model, the calculation may take anywhere from a few seconds to a few minutes to complete. At the conclusion of the macro, the modified hull form is displayed together with three sectional area curves overlaid on top of one another. The cyan curve represents the target sectional area curve. The green curve represents the initial sectional area curve. Finally, the red curve represents the sectional area curve for the final modified hull form. Ideally, this red curve would be identical to the target sectional area curve. However, since the procedure employed behaves as though transforming the control net vertices transforms the surface by an equal amount (clearly a rather gross approximation), the macro may require additional iterations. Other reasons for discrepancies between the final and target sectional area curves are presented in Section 4.0.

An example demonstrating the use of this macro is given now. First the user prepares a target sectional area curve. The format of this file is shown below in Figure 3. It consists of any introductory information describing the file followed by two columns of numbers defining the longitudinal location and the sectional area value, respectively. It is assumed that the numbers are separated by a comma. There can be as much white space as desired. The target sectional area curve must be sorted on longitudinal location. Furthermore, it is assumed that the minimum value of longitudinal location defines the bow of the vessel. The user is free to use any dimensionalization of both columns because the macro redimensionalizes the abscissa and ordinate from 0 to 1. This gives the user the flexibility of using dimensional data from a similar vessel or non-dimensionalized data. It is assumed, however, that the target sectional area curve has only a single maximum at the station of maximum area (although this maximum may persist for some finite distance as in the case of parallel middle body). No local maxima are permitted. In this example we have created an input file called *matchSample.sac* and placed the file in the FastShip data path.

Once the target sectional area curve has been defined, the user opens the hull form model to be modified and defines and refines an adequate number of stations as described above (if appropriate stations are already defined for the model it is not necessary to define new ones). The sample hull form model for this example is shown below in Figure 4. The user then starts the macro by choosing **Read Macro** from the **Utils** menu of FastShip and opens the macro called *matchSAC.mac*. The program prompts first for the name of the file containing the target sectional area curve. Here we enter *matchSample.sac* at the command line (no explicit path is necessary since the data path will be assumed). Next we are prompted to enter the part specification. We enter the part specification as */top* since we wish to include the hull and the skeg (the entire model) in the calculations. The next prompt asks whether the hull needs to be mirrored, and we enter *y* since only half of the hull has been explicitly modeled. Finally we are prompted for the flotation plane. We enter *0* since the  $z=0$  planes corresponds to our design waterline in this example.

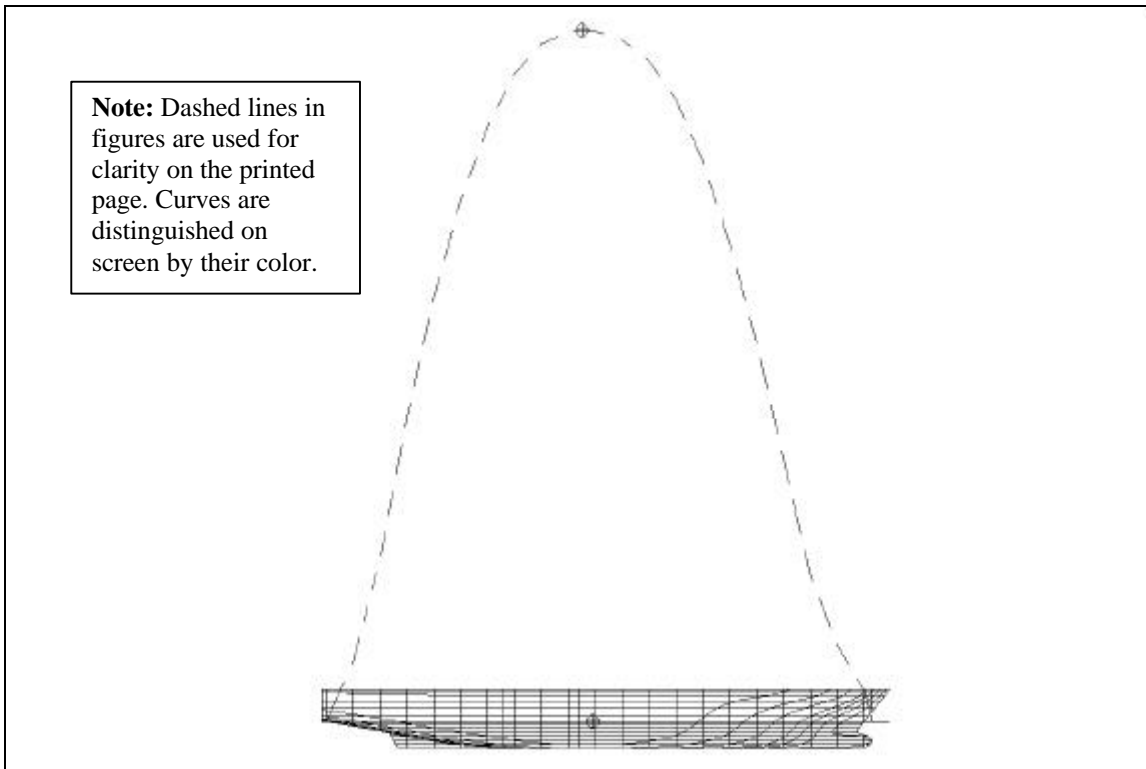
The macro executes and at the conclusion shows a comparison of sectional area curves as shown below in Figure 5. In this figure, the curve with long dashes represents the initial sectional area curve, the curve with short dashes represents the target sectional area curve, and the solid curve represents the sectional area curve after the macro has run to completion. It is evident from this figure that a single iteration of the macro went a long way toward matching the target curve. However, we wish to see if we can get even closer, so we refine sections again (the macro edits the hull which causes sections to become unrefined) and rerun the macro with the same inputs. The results this time are shown below in Figure 6 (the curves follow the same symbology described in the previous figure). After two iterations we have come sufficiently close to matching the target sectional area curve.

```
# Test section area curve for sectional area curve matching
# Proteus Engineering
# 2/1/99
# Lpp = 550.
```

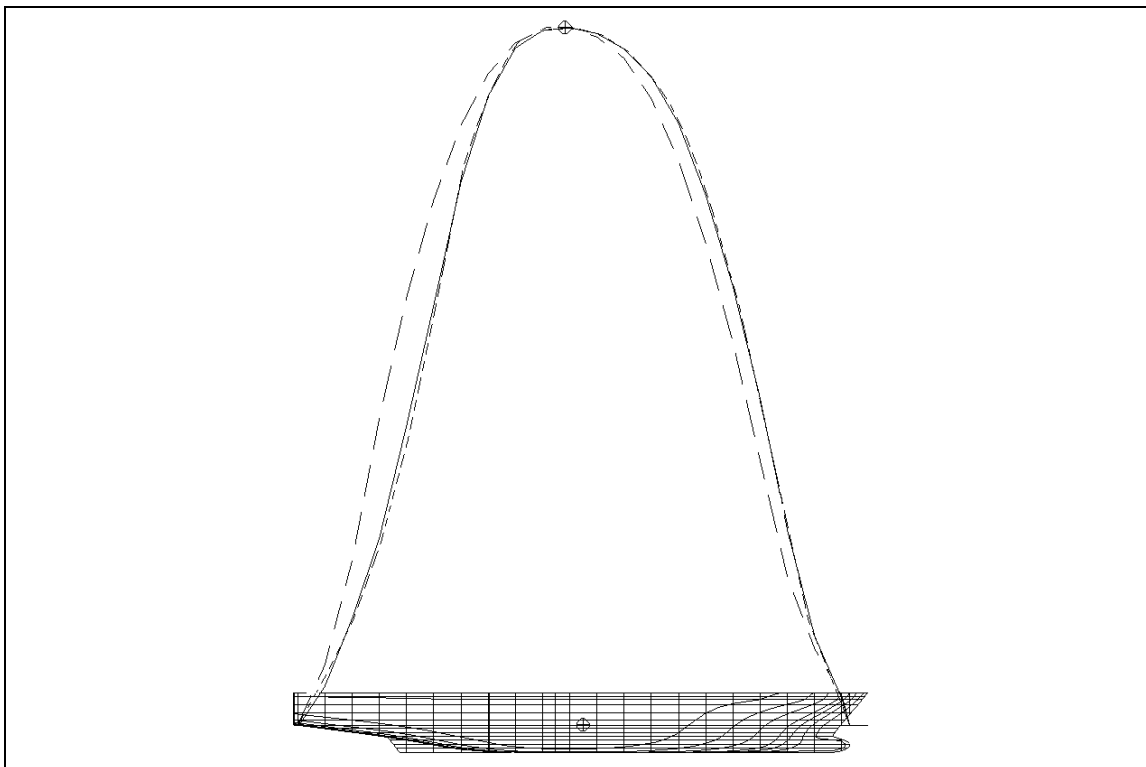
```
0.,0.
0.5,0.05
1.,0.1
1.5,0.15
2.,0.24
3.,0.42
4.,0.6
5.,0.74
5.5,0.80
6.,0.85
6.5,0.89
7.,0.92
8.,0.965
9.,0.99
10.,1.
11.,1.
12.,0.98
12.5,0.95
13.,0.91
13.5,0.86
14.,0.8
15.,0.6
16.,0.41
```

17.,0.26  
18.,0.15  
19.,0.07  
20.,0.0

**Figure 3: Sample input file containing target sectional area curve**



**Figure 4: Initial sectional area curve for test hull form**



**Figure 5: Sectional area curves after one iteration for test hull form**

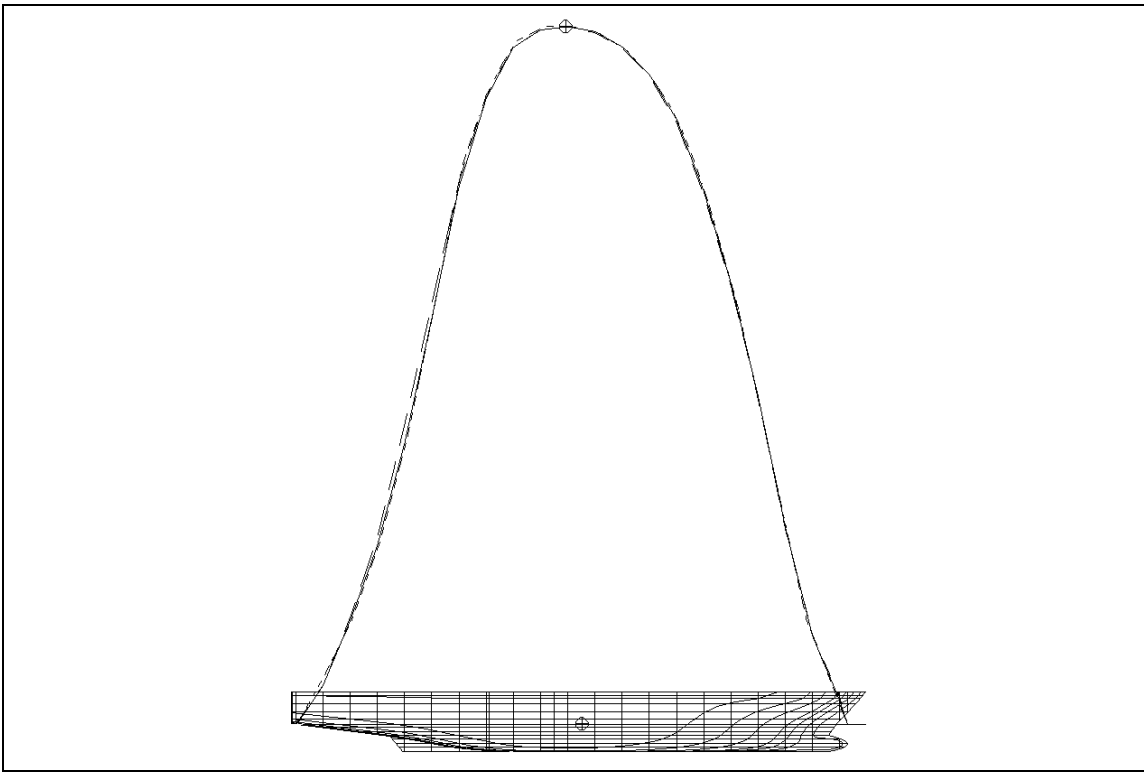


Figure 6: Sectional area curves after two iterations for test hull form

## 4.0 Potential Problems

There are a number of restrictions and assumptions regarding the use of this macro, any of which when violated could cause unexpected results in the final hull form. These are as follows:

- This macro assumes that the initial surface model has an adequate section definition. If no sections are defined the macro will not work properly. If inadequate sections are defined, the resulting linear interpolations between initial and target section area curves will be inaccurate in general.
- The bow of the hull is assumed to correspond to the minimum abscissa of the target sectional area curve, and the FastShip hull model is assumed to have the bow pointing to the right. These are two conditions that the macro cannot determine on its own. If either of these two conditions is violated, the algorithm will not work properly.
- The topology and nature of the target hull form must be the same as that of the initial hull form. The macro cannot introduce surface features into the hull where none existed before, nor can it remove surface features that already exist in the model. Therefore, it is important that the target sectional area curve reflect only those features already existing in the model. For example, if a bulbous bow is in the model, the target sectional area curve should reflect this. The target sectional area curve should not reflect a submerged transom stern if the model does not have one.
- The matching problem is non-dimensionalized on the overall submerged length of the model since this defines the ends of the sectional area curve. Therefore, by definition there is zero shift at the ends of the sectional area curve. Two important side effects of this are that the submerged length remains constant and the area at the ends of the submerged length remain constant (e.g. you cannot change immersed transom area via this macro, this must be done manually with the usual FastShip editing tools).
- There is one and only one maximum in both the initial and target sectional area curves corresponding to the maximum sectional area (although these maxima can extend over some finite distance as in the case of parallel middle body). There must not be any local maxima, such as at a bulbous bow which necks down where it meets the hull fairbody, as this would cause ambiguity in the interpolation scheme described earlier.

- The macro can only adjust the hull shape by moving control net vertices longitudinally. If it appears after repeated iterations of the macro that there is an area of the hull where the macro just can't seem to match the target curve, it probably indicates a lack of control in that region. The user can increase the amount of control locally by inserting a control net column via the Insert Net item under the NURBS menu. The user should be aware that this option usually causes a modest change in hull shape in the area the net column is inserted. A more global increase in control can be achieved without changing the hull shape by selecting the Double item under the NURBS menu.

## 5.0 Questions & Comments

Any questions or comments regarding this macro or this technical note may be addressed to:

FastShip Technical Support  
c/o Proteus Engineering  
345 Pier One Road  
Stevensville, MD 21666  
Telephone: +1 (410) 643-7496  
Fax: +1 (410) 643-7535  
Email: [fastship@proteusengineering.com](mailto:fastship@proteusengineering.com)  
Web: <http://www.proteusengineering.com>

## MACRO LISTING

```
#####  
# Name: MatchSAC  
#  
# Description: FXPerl script to match a target sectional area curve.  
#  
# Programmer: Larry Leibman, Proteus Engineering  
#  
# Date: January 31, 1999  
#  
# Operating System: MS Windows 95/98/NT  
#  
# Revisions:  
#  
# Comments:  
# The macro assumes that the current model has sections adequately defined  
# and uses the current section refinement level.  
# The maximum ordinate of the input SAC must be 1 by definition. If it  
# isn't, we will make it so internally thru nondimensionalization. The  
# range of the abscissa of the input SAC must be from 0 to 1. If it  
# isn't, we will make it so internally thru nondimensionalization.  
# The bow is assumed to be at the end of the input section area curve with  
# the smallest values. Conversely the stern is assumed to be at the end  
# with the higher values. The FastShip hull model is assumed to have the  
# bow pointing to the right.  
# The matching problem is nondimensionalized on the overall submerged  
# length of the model since this defines the ends of the section area  
# curve. Therefore, by definition there is zero shift at the ends of  
# section area curve and the submerged length remains constant.  
# The procedure employed behaves as though transforming the control net  
# vertices transforms the surface by an equal amount. Since this is not  
# strictly true, the process becomes iterative requiring repetitive  
# macro execution. This must be done manually by the user.  
# The procedure cannot introduce surface features where none existed before  
# nor cannot it remove surface features that already exist in the model.  
# Therefore, it is important that the target section area curve reflect  
# only those features already existing in the model. For example if a  
# bulbous bow is in the model, the target SAC should reflect this. The  
# target SAC should not reflect a submerged transom stern if the model  
# does not have one, and the submerged transom area cannot be changed.  
#####  
  
# Perform initialization logic  
  
local ($BIGNUMBER, $SMALLNUMBER) = 1.e100,-1.e100;  
&get_sys ();  
local ($sac_file) = "$sys{'data_path'}matchSAC.tmp";  
$idf_file = "$sys{'data_path'}matchSAC_IDF.tmp";  
print("\nStarting matchSAC.mac.....");  
  
# Query the user for the name of the input (target) SAC, part spec, flotation  
# plane, mirror flag  
  
local ($target_file) = &input(  
"Enter the filename containing the target section area curve: ");  
local ($part_spec) = &input(  
"Enter the part specification for the hull parts to be modified: ");  
local ($mirrorAns) = &input("Does the hull need to be mirrored?: ");  
local ($float_pln) = &input(  
"Enter the flotation plane const., ie. z value: ");  
  
# Parse the target sectional area curve data; if the target filename does not  
# have any path information, prepend the data path; strip off any quotes which  
# the user might have put in to protect spaces;  
  
local (@lines);  
if (!(($target_file =~ /\[\\\/\]))  
{  
$target_file = $sys{'data_path'}. $target_file;  
}  
$target_file =~ s/^"/g;
```

```

open (TARGET_FILE, $target_file) ||
  die("\nCould not open $target_file. Does file exist?");
@lines = <TARGET_FILE>;
close(TARGET_FILE);

local ($targetXMin) = $BIGNUMBER;
local ($targetXMax) = $SMALLNUMBER;
local ($targetAMin) = $BIGNUMBER;
local ($targetAMax) = $SMALLNUMBER;
local ($) = 0;
local (@xTargetSAC, @yTargetSAC);

for (@lines)
{
  chop $_;
  if (/(\-?d*\.\?d*e*\-?d*)\s*,\s*(\-?d*\.\?d*e*\-?d*)/)
  {
    $xTargetSAC[$i] = $1;
    $yTargetSAC[$i] = $2;
  }
  $targetXMin = &min ($targetXMin, $xTargetSAC[$i]);
  $targetXMax = &max ($targetXMax, $xTargetSAC[$i]);
  $targetAMin = &min ($targetAMin, $yTargetSAC[$i]);
  $targetAMax = &max ($targetAMax, $yTargetSAC[$i]);
  $i++;
}
local ($numTargetSAC) = $i;
local ($targetOffset) = $targetXMin;
local ($targetLenSubmerged) = $targetXMax - $targetXOffset;

# Non-dimensionalize target SAC to submerged length and max area

foreach $i (0..$numTargetSAC-1)
{
  $xTargetSAC[$i] = ($xTargetSAC[$i] - $targetOffset) / $targetLenSubmerged;
  $yTargetSAC[$i] /= $targetAMax;
}

# Sort on x from decreasing to increasing (the dreaded bubble sort)

for ($i=0; $i<$numTargetSAC-1; $i++)
{
  for ($j=0; $j<$numTargetSAC-1-$i; $j++)
  {
    if ($xTargetSAC[$j+1] < $xTargetSAC[$j])
    {
      $tmpX = $xTargetSAC[$j];
      $tmpY = $yTargetSAC[$j];
      $xTargetSAC[$j] = $xTargetSAC[$j+1];
      $yTargetSAC[$j] = $yTargetSAC[$j+1];
      $xTargetSAC[$j+1] = $tmpX;
      $yTargetSAC[$j+1] = $tmpY;
    }
  }
}

# Calculate the initial hydrostatics; use user-defined sections

local ($part_specroot) = $part_spec;
$part_specroot =~ s/\.{3}$//;
$part_spec = $part_specroot."...";
print "\npart_spec = $part_spec, part_specroot = $part_specroot";

set-sections on;
unlink ($sac_file);
set-io-user-units off;
if ($mirrorAns =~ /[yY]/)
{
  measure-volume mirror $part_specroot {$float_pln};
  write-model {area:mirror:$float_pln} $part_spec $sac_file;
}
else
{
  measure-volume $part_specroot {$float_pln};
}

```

```

    write-model {area:no mirror:$float_pln} $part_spec $sac_file;
}

&get_hydro;
local (@wet_max) = split(/,/,$hydro{'wet_max'});
local (@wet_min) = split(/,/,$hydro{'wet_min'});
local ($initialOffset) = $sys{'zup'} == 1 ? $wet_min[0] : $wet_max[0];
local ($initialLenSubmerged) = $wet_max[0] - $wet_min[0];
local ($initialAMax) = $hydro{'ax'};
local ($zfac) = $sys{'zup'} == 1 ? 1 : -1;

# Parse IDF file containing SAC data for initial hull

local (@xInitialSAC, @yInitialSAC, $numInitialSAC);
open (INITIAL_FILE, $sac_file) ||
    die("\nCould not open $sac_file. Is hardware lock present?");
while (<INITIAL_FILE>)
{
    chop $_;
    if (A$PART/)
    {
        $_ = <INITIAL_FILE>;
        $numInitialSAC = <INITIAL_FILE>;
        chop $numInitialSAC;
        for ($i=0; $i<$numInitialSAC; $i++)
        {
            $_ = <INITIAL_FILE>;
            chop $_;
            ($xInitialSAC[$i],$yInitialSAC[$i]) = split(/,/,$_);
            $xInitialSAC[$i] = $zfac * ($xInitialSAC[$i] - $initialOffset) /
                $initialLenSubmerged;
            $yInitialSAC[$i] /= $initialAMax;
        }
    }
}
close(INITIAL_FILE);

# Sort on x from decreasing to increasing (the dreaded bubble sort)

for ($i=0; $i<$numInitialSAC-1; $i++)
{
    for ($j=0; $j<$numInitialSAC-1-$i; $j++)
    {
        if ($xInitialSAC[$j+1] < $xInitialSAC[$j])
        {
            $tmpX = $xInitialSAC[$j];
            $tmpY = $yInitialSAC[$j];
            $xInitialSAC[$j] = $xInitialSAC[$j+1];
            $yInitialSAC[$j] = $yInitialSAC[$j+1];
            $xInitialSAC[$j+1] = $tmpX;
            $yInitialSAC[$j+1] = $tmpY;
        }
    }
}

# Compute the shift curve; make sure the shift is zero at the ends

local (@y_shift) = &Compute_Shift_Curve ($numInitialSAC, $numTargetSAC,
    @xInitialSAC, @yInitialSAC,
    @xTargetSAC, @yTargetSAC);
local (@x_shift) = @xInitialSAC;

if ($x_shift[0] <= 0.)
{
    $x_shift[0] = 0.;
    $y_shift[0] = 0.;
}
else
{
    unshift (@x_shift, 0.);
    unshift (@y_shift, 0.);
}

if ($x_shift[$#x_shift] >= 1.)

```

```

{
  $x_shift[$#x_shift] = 1.;
  $y_shift[$#x_shift] = 1.;
}
else
{
  push (@x_shift, 1.);
  push (@y_shift, 1.);
}

# Apply the shift curve to the target parts; start by retrieving the nurbs
# data defining the current model

local (%nurbs) = &read_nurbs ($part_spec);
local (@parts);
local (%max_col, %max_row);
local ($key, $value, @subkeys, $key_found);

while (($key,$value) = each %nurbs)
{
  @subkeys = split (/;/,$key);
  $key_found = 0;
  for (@parts)
  {
    if ($_ eq $subkeys[1])
    {
      $key_found = 1;
      last;
    }
  }
  if (!$key_found)
  {
    push (@parts, $subkeys[1]);
  }
  if ($subkeys[0] =~ /vertex/)
  {
    $max_col{$subkeys[1]} = &max ($max_col{$subkeys[1]}, $subkeys[2]);
    $max_row{$subkeys[1]} = &max ($max_row{$subkeys[1]}, $subkeys[3]);
  }
}
for (@parts)
{
  $quoted_part = sprintf ("%s\\", $_);
  modify-net open $quoted_part;
  foreach $col (0..$max_col{$_})
  {
    foreach $row (0..$max_row{$_})
    {
      if(&was_esc_hit()) {return;}
      $xpos = $nurbs{'vertex',$_,$col,$row,'x'};
      $ypos = $nurbs{'vertex',$_,$col,$row,'y'};
      $zpos = $nurbs{'vertex',$_,$col,$row,'z'};
      $h = $nurbs{'vertex',$_,$col,$row,'h'};
      $xfrac = $zfac * ($xpos - $initialOffset) / $initialLenSubmerged;
      if ($xfrac >= 0 && $xfrac <= 1)
      {
        $xfrac = &Interpolate ($xfrac, $#x_shift+1, @x_shift, @y_shift);
      }
      $xpos = $xfrac * $initialLenSubmerged/$zfac + $initialOffset;
      modify-net {$col,$row}{$xpos, $ypos, $zpos, $h};
    }
  }
  modify-net close;
}

```

# Print out the results

```

delete-part "/top/target section area curve" confirm;
delete-part "/top/initial section area curve" confirm;
set-sections on;
unlink ($sac_file);
set-io-user-units off;
if ($mirrorAns =~ /[Y]/)
{

```

```

    measure-volume mirror $part_specroot {$float_pln};
}
else
{
    measure-volume $part_specroot {$float_pln};
}

# Open the .idf file for writing

unlink ($idf_file);
open (IDF_FILE, ">".$idf_file) || die ("Couldn't create the $idf_file!");

foreach $i (0..$numTargetSAC-1)
{
    print "\n$xTargetSAC[$i], $yTargetSAC[$i]";
}

&write_idf_header ("target section area curve");
printf (IDF_FILE "$numTargetSAC\n");
foreach $i (0..$numTargetSAC-1)
{
    $ypos = $yTargetSAC[$i] * $initialAMax * ($sys{'zup'} == -1 ? -1 : 1);
    printf (IDF_FILE "%f, 0.,%f, unknown\n",
        $initialOffset + $zfac * $xTargetSAC[$i]*$initialLenSubmerged, $ypos);
}
printf (IDF_FILE "%s\n", 'END ENTITY');

&write_idf_header ("initial section area curve");
printf (IDF_FILE "$numInitialSAC\n");
foreach $i (0..$numInitialSAC-1)
{
    $ypos = $yInitialSAC[$i] * $initialAMax * ($sys{'zup'} == -1 ? -1 : 1);
    printf (IDF_FILE "%f, 0.,%f, unknown\n",
        $initialOffset + $zfac * $xInitialSAC[$i]*$initialLenSubmerged, $ypos);
}
printf (IDF_FILE "%s\n", 'END ENTITY');

close (IDF_FILE);

set-current-part "/top";
read-model {sections} $idf_file;
set-offset-color "/top/initial section area curve" green;
set-offset-color "/top/target section area curve" cyan;

# End of Main routine

sub Compute_Shift_Curve
#-----
# This routine computes a shift curve given two input curves. The input consists
# of a source curve defined by abscissa and ordinate arrays (x_source, y_source)
# and a target curve also defined by abscissa and ordinate arrays (x_target,
# y_target). It is assumed that the two curves are sorted on x and have only a
# single maximum, although this maximum may be more than one point (such as a
# sectional area curve with parallel midbody). It is also assumed that the two
# input curves are sorted in the same direction, from smaller to larger x or
# vice-versa.
#-----
{
    local ($num_source, $num_target, @vals) = @_;
    local (@x_source, @y_source, @x_target, @y_target);
    local (@x_lo_vals, @y_lo_vals, @x_hi_vals, @y_hi_vals);
    local (@x_shift, @y_shift);

    for ($i=0; $i<$num_source; $i++)
    {
        $x_source[$i] = $vals[$i];
        $y_source[$i] = $vals[$i+$num_source];
    }

    for ($i=0; $i<$num_target; $i++)
    {
        $x_target[$i] = $vals[2*$num_source+$i];
        $y_target[$i] = $vals[2*$num_source+$num_target+$i];
    }
}

```

```
# Break input curves into regions
```

```
local ($i_source_lo, $i_source_hi) = 0,0;
foreach $i (0..$#y_source)
{
  if ($y_source[$i] > $y_source[$i_source_lo])
  {
    $i_source_lo = $i;
    $i_source_hi = $i;
  }
  elseif ($y_source[$i] == $y_source[$i_source_lo])
  {
    $i_source_hi = $i;
  }
}

```

```
local ($i_target_lo, $i_target_hi) = 0,0;
local ($i_lo, $i_hi) = 0,0;
$x_lo_vals[0] = $x_target[0];
$y_lo_vals[0] = $y_target[0];
foreach $i (0..$#y_target)
{
  if ($y_target[$i] > $y_target[$i_target_lo])
  {
    $i_target_lo = $i;
    $i_target_hi = $i;
    $x_lo_vals[++$i_lo] = $x_target[$i];
    $y_lo_vals[$i_lo] = $y_target[$i];
  }
  elseif ($y_target[$i] == $y_target[$i_target_lo])
  {
    $i_target_hi = $i;
  }
  else
  {
    $x_hi_vals[++$i_hi] = $x_target[$i];
    $y_hi_vals[$i_hi] = $y_target[$i];
  }
}
$x_hi_vals[0] = $x_target[$i_target_hi];
$y_hi_vals[0] = $y_target[$i_target_hi];

```

```
# For each abscissa in source curve, determine a shift value in target
```

```
foreach $i (0..$#x_source)
{
  $x_shift[$i] = $x_source[$i];
  if ($i <= $i_source_lo)
  {
    $y_shift[$i] = &Interpolate ($y_source[$i], $#x_lo_vals+1,
      @y_lo_vals, @x_lo_vals);
  }
  elseif ($i >= $i_source_hi)
  {
    $y_shift[$i] = &Interpolate ($y_source[$i], $#x_hi_vals+1,
      @y_hi_vals, @x_hi_vals);
  }
  else
  {
    $y_shift[$i] = ($x_shift[$i] - $x_source[$i_source_lo]) /
      ($x_source[$i_source_hi] - $x_source[$i_source_lo]) *
      ($x_target[$i_target_hi] - $x_target[$i_target_lo]) +
      $x_target[$i_target_lo];
  }
}
return (@y_shift);
}

```

```
sub Interpolate
```

```
#-----
# This routine performs linear interpolation given 2 input arrays (xVals and
# yVals) defining the abscissae and ordinates, respectively, of a single-valued
```

```
# function (in x) and an x interpolant (xInterp). It is assumed that the input
# arrays are sorted on x (increasing or decreasing). If the input x interpolant
# falls outside of the bounds of the input arrays, the returned y value is that
# at the appropriate end of the array of ordinates.
```

```
#-----
{
  local ($xInterp, $num_vals, @vals) = @_;
  local (@xVals, @yVals);
  local ($yInterp) = 0.;

  for ($i=0; $i<$num_vals; $i++)
  {
    $xVals[$i] = $vals[$i];
    $yVals[$i] = $vals[$i+$num_vals];
  }

  # Check first to see if the input x is within the bounds of the abscissal array

  if ($xInterp < $xVals[0] && $xInterp < $xVals[$#xVals])
  {
    if ($xVals[0] < $xVals[$#xVals])
    {
      $yInterp = $yVals[0];
    }
    else
    {
      $yInterp = $yVals[$#yVals];
    }
  }
  elseif ($xInterp > $xVals[0] && $xInterp > $xVals[$#xVals])
  {
    if ($xVals[0] > $xVals[$#xVals])
    {
      $yInterp = $yVals[0];
    }
    else
    {
      $yInterp = $yVals[$#yVals];
    }
  }
  else
  {
    foreach $i (0..$#xVals-1)
    {
      if (($xVals[$i] <= $xInterp && $xVals[$i+1] >= $xInterp) ||
          ($xVals[$i] >= $xInterp && $xVals[$i+1] <= $xInterp))
      {
        $yInterp = ($xInterp - $xVals[$i]) / ($xVals[$i+1] - $xVals[$i]) *
          ($yVals[$i+1] - $yVals[$i]) + $yVals[$i];
        last;
      }
    }
  }
  return $yInterp;
}
```

```
sub read_nurbs
```

```
#-----
# This routine obtains the current nurbs information for an input part
# specification and returns it in an associative array. The user must have a
# non-demo mode version of FastShip since we call the show-nurbs function.
#-----
```

```
{
  local ($part_spec) = @_;
  local ($count) = 0;
  local (%nurbs);
  local (@lines);
  &get_sys ();
  local ($nurbs_file) = "$sys{'prog_path'}info.tmp";
```

```
#..Write NURBS file
```

```
show-nurbs $part_spec;
```

```
#..Open FastShip NURBS file
```

```
open (NURBS_FILE, $nurbs_file) || die "Can't open FastGen NURBS file!\n";
```

```
FastShip Sectional Area Curve Matching Macro
```

```

@lines = <NURBS_FILE>;

#..Parse NURBS file and read input data
for (@lines)
{
#.....Read knot vectors and control net vertices
  \*\*\* NURBS Info: (\D*\b)/ && ($part = '/top/'. $1);
  if (/Order \[u,v\]: \[s*(d*),s*(d*)\]/)
  {
      $nurbs{'u_order',$part} = $1;
      $nurbs{'v_order',$part} = $2;
  }
  if (/Nvert \[u,v\]: \[s*(d*),s*(d*)\]/)
  {
      $nurbs{'u_vert',$part} = $1;
      $nurbs{'v_vert',$part} = $2;
      $nurbs = 1;
  }
  /Knot vector \[(\D)\]/ && ($parameter = $1);
  if (/^\s+(\d+),*\s*(\d*),*\s*(\d*),*\s*(\d*),*\s*(\d*),*/)
  {
      $nurbs{'knot',$part,$parameter,$count} = $1;
      $nurbs{'knot',$part,$parameter,$count+1} = $2;
      $nurbs{'knot',$part,$parameter,$count+2} = $3;
      $nurbs{'knot',$part,$parameter,$count+3} = $4;
      $nurbs{'knot',$part,$parameter,$count+4} = $5;
      $count = $count + 5;
  }
  if (/^\[s*(\d*),s*(\d*)\]:\s*(\d*\.\d*e*\d*),\s*(\d*\.\d*e*\d*),\s*(\d*\.\d*e*\d*),\s*(\d*\.\d*e*\d*)/)
  {
      $nurbs{'vertex',$part,$1,$2,'x'} = $3;
      $nurbs{'vertex',$part,$1,$2,'y'} = $4;
      $nurbs{'vertex',$part,$1,$2,'z'} = $5;
      $nurbs{'vertex',$part,$1,$2,'h'} = $6;
  }
}
undef @lines;
close NURBS_FILE;
return %nurbs;
}

sub write_idf_header
#-----
# Subroutine to create the required IDF header for exporting the target section
# area curve.
#-----
{
# Get the current date and time

($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
$today = sprintf ("%2d/%2d/%2d", $mday,$mon, $year);
$now = sprintf ("%2d:%02d:%02d", $hour, $min, $sec);
&get_sys;

# Write the header information to the globally defined IDF_FILE handle

printf (IDF_FILE "%s\n", '$IDF');
printf (IDF_FILE "4.00\n");
printf (IDF_FILE "%s\n",'$ENTITY');
printf (IDF_FILE "SECTIONS\n");
printf (IDF_FILE "%s\n", '$VESSEL NAME');
printf (IDF_FILE "Test Case\n");
printf (IDF_FILE "%s\n", '$DATA SOURCE');
printf (IDF_FILE "matchSAC.mac\n");
printf (IDF_FILE "%s\n", '$DATE');
printf (IDF_FILE "%s\n", $today);
printf (IDF_FILE "%s\n", '$TIME');
printf (IDF_FILE "%s\n", $now);
printf (IDF_FILE "%s\n", '$UNITS');
printf (IDF_FILE "SI\n");
printf (IDF_FILE "%s\n", '$COORDINATE SYSTEM');
printf (IDF_FILE "%d, 1, %d\n",
    $sys{'zup'} == -1 ? 1 : -1, $sys{'zup'} == -1 ? 1 : -1);
}

```

```
printf (IDF_FILE "%s\n", $COMMENTS);
printf (IDF_FILE "Target sectional area curve data.\n");
printf (IDF_FILE "%s\n", $GEOMETRY);
printf (IDF_FILE "1\n");
printf (IDF_FILE "%s\n", $PART);
printf (IDF_FILE "@_0\n");
printf (IDF_FILE "1\n");
printf (IDF_FILE "%s\n", $CURVE);
printf (IDF_FILE "Section 1\n");
printf (IDF_FILE "three-d\n");
}
1;
```