Sandia National Laboratories

Operated for the U.S. Department of Energy by
Sandia Corporation

Albuquerque, New Mexico 87185-0834

date: June 22, 2000

to: Distribution

from: Matthew M. Hopkins, Org. 9114, MS 0834

subject: Template for parameter continuation and operability window estimation using *Perl* scripts and *Goma* for a slot coater. (GT-015.1)

keywords: Continuation, Hunting, Operability Window, slot coater, linear stability, LSA, Perl

input records: Continuation

Introduction

In this tutorial we will supply the details necessary to use scripts in conjunction with *Goma* and the *SEACAS* tools to perform parameter continuation and operability window estimation for a slot coater model problem.

The bulk of this procedure is complicated due to the amount of information that must be organized. Because of this, another memo has been supplied describing the detailed interaction of the *Perl* script(s), *Goma*, and the *SEACAS* tool suite. (*Perl* is a scripting language similar to standard Unix shell scripting environments, but more powerful.) Detailed knowledge of *Perl* is absolutely not required–only a few lines will need to be edited in the script file *arc.pl*. Please see publication GTM-021.0 for these details. It is further assumed that the reader is familiar with performing linear stability analysis (LSA) with *Goma*. See publication GSR-01.1 for these details.

Although this scripting process is general enough to be useful in other contexts (essentially any problem with a deformable mesh), we will concentrate on a particular slot coating model. We take all of our physical parameters from Ian Gates' dissertation ("Slot Coating Flows: Feasibility, Quality", University of Minnesota, 1999). After reading this tutorial the reader should feel comfortable with setting up and performing parameter continuation for other problems.

This tutorial depends on the script files (with a *.pl* extension), and the files residing in the directory `slot_50cms_orig` and the subdirectory `initial_steady_state`. Please get these files before continuing. It is assumed that the *Perl* files are executable, and that their first line directs the shell to the proper location of the *Perl* executable. If this is not the case, please contact Duane Labreche (dalabre@sandia.gov) or the author (mmhopki@sandia.gov). If you do not have a *Perl* executable, they are publicly available at *www.perl.org*.

Background

As detailed in publication GTM-021.0, the scripting process for parameter continuation requires the user to modify the *Goma* input files, geometry files, and the script *arc.pl*. The purpose of these edits is to allow the controlling script (*arc.pl*) to find keywords and replace them with the correct numerical values to perform a local continuation run. A local continuation run is completed in a single execution of *Goma* on a single base mesh. In addition to these keyword substitutions, the script automatically performs the necessary remeshing and solution remapping steps. A sequence of local continuation runs yields the global continuation run.

Parameter continuation is an intricate part of our current method for determining operability windows. To determine an operability window, one starts at a known stable steady-state solution, and continues in the chosen parameter(s) until an unstable solution is encountered. The value of the parameter(s) at which this occurs is a boundary point of the operability window. Repeating this process with different starting values, or different continuation directions, will map out a set of points on the operability window's boundary. Although the entire process has not been automated, the bulk of the work is in the "continue in the parameter(s) until we find an unstable solution" step, which is automated here.

There are two types of continuation runs available in *Goma*: single parameter and multi-parameter. Single parameter continuation is easier to set up and understand, but has a debilitating restriction. Multi-parameter continuation is far more general, but is more complicated. Of course, single parameter continuation can be performed in a multiple parameter continuation context. We will present a single parameter continuation first to get our feet wet, then dive into a multi-parameter continuation.

There have been many improvements to the slot coater model, and solution analysis, since GT-002.1 (a tutorial detailing how to find initial steady-state solutions). Specifically, we have free menisci (all contact lines are free to relocate), more accurate geometry (rounded lands), linear stability analysis, and new parameter continuation techniques. These advances allow for substantially more realistic models and quantitative analysis.

Initial Steady-State Solution

Finding an initial steady-state solution is not a trivial task. `get_steady_state.sh`, a short shell script, resides in the subdirectory `initial_steady_state`. It will find the initial base case for our model problem. See publication GT-002.1 for a discussion of finding the initial steady-state solution for a similar slot coater model. Publication GT-011.0 discusses finding the initial steady-state solution to a multi-layer slide coater. One could use these two examples as a starting point for finding an initial steady-state solution in other deforming mesh and free surface problems.

Single Parameter Continuation in Back Pressure

We will start at a base case with back pressure = -11444 dynes/cm^2 in the vacuum box, and continue up in back pressure. "Up" in this case refers to the magnitude of back pressure. Continuing from -

11444 dynes/cm^2 to -12000 dynes/cm^2 is continuing up in back pressure. All of the other parameters (such as web speed) will remain fixed throughout.

Copy the following files:

From	To
`slot-baseC.geom`	`slot-base.geom`
`slot-cont-baseC.input`	`slot-cont-base.input`
`slot-stab-baseC.input`	`slot-stab-base.input`

The "C" and "H" extensions on the files in this directory indicate single and multi-parameter continuation, respectively. Other *Goma* materials refer to single parameter and multi-parameter continuation runs as continuation runs and hunting runs.

We will use the keyword `VACUUM_VAL` to designate the back pressure in the scripting process.

The controlling script, *arc.pl*, has the following lines at the top of the file:

```
# Parameter settings for a slot coater continuation run.
@params =     ("VACUUM_VAL");
@start_vals = (      -11444);
@end_vals =   (      -30000);
$eval_shift = -400.0;
$eval_base = -100.0;
$max_steps_per_segment = 20;
```

There are a few other single and multi-parameter continuation runs set up and commented out. In *Perl*, the comment character is #. A quick look should convince the reader that no knowledge of *Perl* is required.

What do these settings do? For now, the only important ones are the first three. The `@params` line lists the starting value keyword. Another keyword is derived from this by appending an `_END`, and is replaced by the global ending value. The `@start_vals` and `@end_vals` lines list the global starting and ending parameter value. Note that the starting value must match that used in the initial steady-state solution.

The only change we made in the geometry file `slot-base.geom` was to replace the *aprepro* line in the unedited version:

```
# Vacuum backpressure     [=] cgs            {vacuum = -11.444e+03}
```

with:

```
# Vacuum backpressure     [=] cgs            {vacuum = VACUUM_VAL}
```

This keyword will be replaced by *arc.pl* with the actual numerical starting value for the upcoming local continuation run. In setting up the first local continuation run, this will be replaced with -11444.

When given a choice, the author's experience indicates that one should place the keyword(s) in the geometry file rather than directly in the input file. If this value is used in any further *aprepro* computations, then it must have already been defined. Furthermore, this keeps the input files as clean as possible. This will become more apparent when we look at multi-parameter continuation runs.

The lines in `slot-cont-base.input` required by the script are:

```
{Include(slot.geom)}
FEM file                              = slot.exoII
Output EXODUS II file                 = slot-cont-out.exoII
Initial Guess                         = read_exoII_file slot-cont-in.exoII
Continuation                          = first
Boundary condition data float tag = 1
Initial parameter value               = { vacuum }
Final parameter value                 = VACUUM_VAL_END
Maximum number of path steps          = MAX_STEPS
BC = CAPILLARY SC 23 {surface_tension} {vacuum}  0.0
```

The *aprepro* `Include` command is required to read in the appropriate geometry file. The next three lines are filename requirements. The `Initial parameter value` card references the *aprepro* `vacuum` variable set in the geometry file. The keyword `VACUUM_VAL_END` will be replaced with the global ending value.

If we want to continue in back pressure until we cross from stable to unstable solutions (i.e., finding a boundary point of the operability window), we will choose an ending value that we know to be unreasonably extreme. We expect that no stable solution exists at a back pressure of -30000 dynes/cm^2 By selecting this as the ending value we should cross the boundary between stable and unstable solutions.

The `Boundary condition data float tag` and `BC` cards indicate which boundary value will be our continuation parameter. This is where single parameter continuation loses out to multi-parameter continuation. Only one boundary condition can be a continuation parameter. Even if this exact same back pressure were used in another boundary condition, the fact that it would be a different boundary condition would necessitate the use of multi-parameter continuation.

The keyword `MAX_STEPS` is replaced by the value `$max_steps_per_segment` from *arc.pl*. It is the maximum number of continuation steps *Goma* will take in a single local continuation run. This is how forced remeshing and remapping is implemented. It could have simply been set in the input file, but we want to keep the parameters that change most often in one place (the *arc.pl* script).

Recall from publication GTM-021.0 that the script first computes all of the steady-state solutions (slices), and then performs LSA on each slice. The above information completely describes what needs to be done for the first phase. The second phase (performing LSA) requires us to edit the `slot-stab-base.input` file. The relevant lines in `slot-stab-base.input` required by the script are:

```
{Include(slot.geom)}
FEM file                        = slot.exoII
```

```
Output EXODUS II file          = slot-stab-out.exoII
Initial Guess                  = read_exoII_file slot-stab-in.exoII
Eigen Initial Shifts           = E_SHIFT E_SHIFT E_SHIFT E_SHIFT
```

The only new card here is `Eigen Initial Shifts`. The `E_SHIFT` keyword is replaced with a guess for the real part of the eigenvalue with largest real part. It was found that a good initial guess for the eigenvalue shift is more important than having multiple initial shifts. This is why each of the shifts are all the same. This could be changed but has not been done.

The *arc.pl* variables `$eval_shift` and `$eval_base` control the initial guess from one slice to the other. The `$eval_shift` value is the initial eigenvalue shift. Since the eigenvalue with leading real part should not change drastically from one slice to the other, the script uses the previous converged eigenvalue to construct a good shift for the next slice. The `$eval_base` value is used as a default value when the previous eigenvalue solution had problems (i.e., failed to converge or claimed convergence to an eigenvalue that is incorrect). The methods to declare an eigenvalue solution "bad", and how to generate a good guess for the next eigenvalue problem, are heuristic in nature. The interested user is encouraged to inspect the *Perl* code directly or contact the author for details.

From the form of the file names, it is apparent that we have chosen the base name "slot" to index all of our files (see publication GTM-021.0). To execute the entire process, simply run *arc.pl slot*. When run, the script will report what it is currently doing (i.e., remeshing, running *Goma*, etc.). Here's a sample of the global continuation run output:

```
================================================================================
  Welcome to A.R.C.
================================================================================
Running: cp slot-base.exoII slot.exoII
--------------------------------------------------------------------------------
Running: cp slot-base.geom slot.geom
Running: cp slot-cont-base.input slot-cont.input
Replaced 1 instance(s) of MAX_STEPS in slot-cont-base.input
Replacing VACUUM_VAL_END with -30000, found 0 in slot-base.geom, and 1 in slot-
cont-base.input.
Replacing VACUUM_VAL with -11444, found 1 in slot-base.geom, and 0 in slot-cont
-base.input.
At the end of setup_continuation_run,
@params = VACUUM_VAL
@start_vals = -11444
@end_vals   = -30000
start_val = -11444
Running: goma -a -i slot-cont.input -se slot-cont--11444.err -so slot-cont--114
44.out
Found a continuation run (as opposed to a hunting run).
Running: mv slot-cont-in.exoII slot-cont-in--11444.exoII
Running: cp slot-cont-out.exoII slot-cont-out--11444.exoII
Hit maximum number of steps per segment.  Performing a remesh/remap.
Running: cp slot-cont-out.exoII remesh-in.exoII
Running: cubit -nojournal -batch -nographics remesh.jou > tmp.cubit.out 2>&1
Running: cp slot-cont-out.exoII remap.e
Piping to: | mapvar -a remap > tmp.mapvar.out 2>&1
Running: cp remap.int slot-cont-in.exoII
```

```
Running: cp remap.int slot.exoII
------------------------------------------------------------------------------
Running: cp slot-base.geom slot.geom
Running: cp slot-cont-base.input slot-cont.input
Replaced 1 instance(s) of MAX_STEPS in slot-cont-base.input
Replacing VACUUM_VAL_END with -30000, found 0 in slot-base.geom, and 1 in slot-
cont-base.input.
Replacing VACUUM_VAL with -12204, found 1 in slot-base.geom, and 0 in slot-cont
-base.input.
At the end of setup_continuation_run,
@params     = VACUUM_VAL
@start_vals = -12204
@end_vals   = -30000
Running: goma -a -i slot-cont.input -se slot-cont--12204.err -so slot-cont--122
04.out
Running: mv slot-cont-in.exoII slot-cont-in--12204.exoII
Running: cp slot-cont-out.exoII slot-cont-out--12204.exoII
Hit maximum number of steps per segment.  Performing a remesh/remap.
Running: cp slot-cont-out.exoII remesh-in.exoII
Running: cubit -nojournal -batch -nographics remesh.jou > tmp.cubit.out 2>&1
Running: cp slot-cont-out.exoII remap.e
Piping to: | mapvar -a remap > tmp.mapvar.out 2>&1
Running: cp remap.int slot-cont-in.exoII
Running: cp remap.int slot.exoII
------------------------------------------------------------------------------
Running: cp slot-base.geom slot.geom
Running: cp slot-cont-base.input slot-cont.input
Replaced 1 instance(s) of MAX_STEPS in slot-cont-base.input
Replacing VACUUM_VAL_END with -30000, found 0 in slot-base.geom, and 1 in slot-
cont-base.input.
Replacing VACUUM_VAL with -12964, found 1 in slot-base.geom, and 0 in slot-cont
-base.input.
At the end of setup_continuation_run,
@params     = VACUUM_VAL
@start_vals = -12964
@end_vals   = -30000
Running: goma -a -i slot-cont.input -se slot-cont--12964.err -so slot-cont--129
64.out
Running: mv slot-cont-in.exoII slot-cont-in--12964.exoII
Running: cp slot-cont-out.exoII slot-cont-out--12964.exoII
Hit maximum number of steps per segment.  Performing a remesh/remap.
Running: cp slot-cont-out.exoII remesh-in.exoII
Running: cubit -nojournal -batch -nographics remesh.jou > tmp.cubit.out 2>&1
Running: cp slot-cont-out.exoII remap.e
Piping to: | mapvar -a remap > tmp.mapvar.out 2>&1
Running: cp remap.int slot-cont-in.exoII
Running: cp remap.int slot.exoII
------------------------------------------------------------------------------
.
.
.
```

Here's a sample of the output from the LSA computations:

```
------------------------------------------------------------------------------
```

```
It required 10 mesh(es) to complete the continuation run (as well as I could).
Proceeding with LSA on each slice of each mesh.
Running: cp slot-base.exoII slot.exoII
Found 20 solution slice(s) at: -11444 -11484 -11524 -11564 -11604 -11644 -11684
 -11724 -11764 -11804 -11844 -11884 -11924 -11964 -12004 -12044 -12084 -12124 -
12164 -12204
------------------------------------------------------------------------------
Ratio along hunting/continuation line: 0.00%
Piping to: | algebra slot-cont-out--11444.exoII slot-stab-in.exoII > tmp.algebr
a.out 2>&1
Running: cp slot-base.geom slot.geom
Running: cp slot-stab-base.input slot-stab.input
Replacing VACUUM_VAL_END with -30000, found 0 in slot.geom, and 0 in slot-stab.
input.
Replacing VACUUM_VAL with -11444, found 1 in slot.geom, and 0 in slot-stab.inpu
t.
Using an eigenshift of -100
Running: goma -a -i slot-stab.input -se slot-stab--11444.err -so slot-stab--114
44.out
Running: cp slot-stab-out.exoII slot-stab--11444.exoII
Local step 1, parameter -11444, Goma says the leading evalue is -1.037502e+02
------------------------------------------------------------------------------
Ratio along hunting/continuation line: 0.22%
Piping to: | algebra slot-cont-out--11444.exoII slot-stab-in.exoII > tmp.algebr
a.out 2>&1
Running: cp slot-base.geom slot.geom
Running: cp slot-stab-base.input slot-stab.input
Replacing VACUUM_VAL_END with -30000, found 0 in slot.geom, and 0 in slot-stab.
input.
Replacing VACUUM_VAL with -11484, found 1 in slot.geom, and 0 in slot-stab.inpu
t.
Using an eigenshift of -69.1668
Running: goma -a -i slot-stab.input -se slot-stab--11484.err -so slot-stab--114
84.out
Running: cp slot-stab-out.exoII slot-stab--11484.exoII
Local step 2, parameter -11484, Goma says the leading evalue is -1.036438e+02
------------------------------------------------------------------------------
Ratio along hunting/continuation line: 0.43%
Piping to: | algebra slot-cont-out--11444.exoII slot-stab-in.exoII > tmp.algebr
a.out 2>&1
Running: cp slot-base.geom slot.geom
Running: cp slot-stab-base.input slot-stab.input
Replacing VACUUM_VAL_END with -30000, found 0 in slot.geom, and 0 in slot-stab.
input.
Replacing VACUUM_VAL with -11524, found 1 in slot.geom, and 0 in slot-stab.inpu
t.
Using an eigenshift of -69.0958666666667
Running: goma -a -i slot-stab.input -se slot-stab--11524.err -so slot-stab--115
24.out
Running: cp slot-stab-out.exoII slot-stab--11524.exoII
Local step 3, parameter -11524, Goma says the leading evalue is -1.036925e+02
------------------------------------------------------------------------------
.
.
```

.

The "ratio along hunting/continuation line" reported while computing the LSA slices is somewhat misleading. This is the ratio along the path defined by the user's beginning and ending values. In this case, they are -11444 \dcm and -30000 dynes/cm^2 . For this particular problem, the global continuation run failed at approximately -17804 dynes/cm^2 , or an ending ratio of 34.27% (i.e., not 100%).

The script will also determine how the global continuation run ended. It may have successfully completed (reached the global ending parameter value), but this particular run failed for a different reason. Here's the output:

```
Checking for successful continuation run ... Uh-oh!
Failed on a deformation gradient.  One of two things probably
happened.  We may have failed at the beginning of a fresh
continuation/hunting run.  This means that the initial files you
supplied were not consistent (the -base.exoII and -cont-in.exoII
files, along with the input parameters in -cont-base.input).  The
other thing that may have happened is a deformation gradient,
followed by a remesh/remap step, followed by another deformation
gradient.  This latter problem usually indicates a more serious
failure.  You may have just approached the boundary of stable
solutions in your parameter space.  Our work is done here.
```

There is another script, *findevals.pl*, that will search all of the output files from the LSA runs, and output only the leading real parts. This is useful to find out when the steady state switched from stable to unstable. Another script, *gplot_goma.pl*, will save all of the parameter values and their associated leading real parts in (*x*,*y*) format to a file named `evals.dat`. It then attempts to run *gnuplot* to plot the leading real parts. If the user doesn't have access to *gnuplot* (it is public domain), then simply comment out those lines in the script and use it to generate the (*x*,*y*) data for your own purposes.

Multi-Parameter Continuation in Back Pressure and Web Speed

Using the same slot coater model, we will continue both in back pressure (as before), and web speed. Although we are now changing two parameters instead of one, the amount of work required to prepare this continuation run is much more than double the work required for the single parameter continuation run. The additional work comes about because we need to account for all of the *Goma* boundary conditions that are dependent on these two parameters.

The modification to *arc.pl* is simple. Comment out the previous lines (the `VACUUM_VAL` single parameter continuation setup), and uncomment the following:

```
# Parameter settings for a slot-coater hunting run.
@params =     ("VACUUM_VAL", "WEBSPEED_VAL");
@start_vals = (      -11444,             50);
@end_vals =   (      -10664,             60);
$eval_shift = -400.0;
$eval_base = -100.0;
$max_steps_per_segment = 20;
```

This setup indicates a global continuation run from the steady state solution with back pressure = -11444 dynes/cm^2 and web speed = 50 *cm*/*s* to the steady state solution with back pressure = -10664 dynes/cm^2 and web speed = 60 *cm*/*s*.

To set up the hunting run in *Goma*, first copy the following files:

From	To
`slot-baseH.geom`	`slot-base.geom`
`slot-cont-baseH.input`	`slot-cont-base.input`
`slot-stab-baseH.input`	`slot-stab-base.input`

We need to set up a hunting run (*Goma* terminology) for multi-parameter continuation. What boundary conditions are affected? For back pressure, we still have only the one boundary condition for the upstream bead free surface. However, changing the web speed requires modification of *five* different boundary condition numerical values! Why is there so much work for changing a single parameter? The answer to this question, and the reason multi-parameter continuation runs tend to require more preparation than single parameter continuation runs, is that we need to account for every numerical value appearing in any boundary condition that varies as the continuation parameter(s) vary.

If the web speed increases, we must account for the change in the two boundary conditions that represent the web. In the unedited input deck, the web boundary conditions for the u-velocity are:

```
BC = VELO_SLIP  SS  21  {1.0/beta}  {webspeed}  0.0  0.0
BC = U  NS  9  {webspeed}
```

These two boundary conditions (or, more precisely, the two web speed numerical values) need to have associated hunting conditions. We are assuming a constant film thickness, so there is an additional constraint on the inflow rate. In our model, we specify a final film thickness of $h = 90\ \mu\ m$. Our initial web speed of $u_{web} = 50$ *cm*/*s* gives a volumetric flow rate of $q = h\ u_{web} = 0.45$ *cm*^{2}/*s*. The slot (inlet) width is $s = 279\ \mu\ m$. Thus, the average inflow speed must be $u_{in} = q/s = 16.13$ *cm*/*s*. We assume a parabolic profile at the inflow. The `BC` cards in the unedited input deck that create this boundary condition are:

```
BC = GD_LINEAR SS 3 R_MOMENTUM1 0 VELOCITY2 0 0.0  1.0
BC = GD_PARAB SS 3 R_MOMENTUM1 0 MESH_POSITION1 0 { -6.0 * inflow_speed
 * x6_new * x5_new/S2/S2 }  { 6.0 * inflow_speed * (x6_new + x5_new)/S2/S2 }
 { -6.0 * inflow_speed/S2/S2 }
BC = VELO_TANGENT SS 3 0 0.0 0.0  0.0
```

The *aprepro* variable `inflow_speed` is computed in the associated geometry file (and is equal to 16.13 cm/s). We need to specify a hunting condition for each of these three numerical values in the `BC = GD_PARAB` card.

There is a lot of freedom in exactly how, and where, one would implement the *aprepro* calculations. The author prefers to put as much of the "ugly" calculations in the geometry file, and refer to them

via *aprepro* variables from *Goma*'s input file. The details follow.

We choose the keyword for the back pressure as before, `VACUUM_VAL`. Our keyword for webspeed will be `WEBSPEED_VAL`. Recall in the geometry file that the *aprepro* variable `S2_new` is the slot (inlet) width, and `x5_new` and `x6_new` are the left and right *x*-coordinates of the inlet. The relevant lines in `slot-base.geom` are:

```
# Vacuum backpressure     [=] cgs            {vacuum = VACUUM_VAL }
# Beginning Flow rate (q) [=] cm^2/s         {flow_rate_a = WEBSPEED_VAL * film
_thickness}
# Ending Flow rate (q)    [=] cm^2/s         {flow_rate_b = WEBSPEED_VAL_END *
film_thickness}
# ------------------------------------------------------------------------
#                Useful Parameters For Hunting Values
# ------------------------------------------------------------------------
# starting inflow speed = { avg_inflow_a = flow_rate_a / S2_new }
# ending inflow speed = { avg_inflow_b = flow_rate_b / S2_new }
# starting C1 = { inflow_C1a = -6.0 * avg_inflow_a * x6_new * x5_new/S2_new/S2_
new }
# ending C1 = { inflow_C1b = -6.0 * avg_inflow_b * x6_new * x5_new/S2_new/S2_ne
w }
# delta C1 = { delta_C1 = abs(inflow_C1b - inflow_C1a) }
# starting C2 = { inflow_C2a = 6.0 * avg_inflow_a * (x6_new + x5_new)/S2_new/S2
_new }
# ending C2 = { inflow_C2b = 6.0 * avg_inflow_b * (x6_new + x5_new)/S2_new/S2_n
ew }
# delta C2 = { delta_C2 = abs(inflow_C2b - inflow_C2a) }
# starting C3 = { inflow_C3a = -6.0 * avg_inflow_a/S2_new/S2_new }
# ending C3 = { inflow_C3b = -6.0 * avg_inflow_b/S2_new/S2_new }
# delta C3 = { delta_C3 = abs(inflow_C3b - inflow_C3a) }
# delta_webspeed = { delta_webspeed = abs((WEBSPEED_VAL_END) - (WEBSPEED_VAL))
}
# delta_vacuum = { delta_vacuum = abs((VACUUM_VAL_END) - (VACUUM_VAL)) }
```

The payoff for all of these extra computations is a much cleaner `slot-cont-base.input` file. Here are the relevant lines:

```
{Include(slot.geom)}
FEM file                                = slot.exoII
Output EXODUS II file                   = slot-cont-out.exoII
Initial Guess                           = read_exoII_file slot-cont-in.exoII
Continuation                            = hfirst
Maximum number of path steps            = MAX_STEPS
------------------------------------------------------------------------------
Hunting Specifications
------------------------------------------------------------------------------
Number of hunting conditions = -1
HC = BC  11  0  0  { inflow_C1a }  { inflow_C1b }  { delta_C1 / 20.0 }  { delta
_C1 / 100.0}  { delta_C1 / 10.0 }
HC = BC  11  1  0  { inflow_C2a }  { inflow_C2b }  { delta_C2 / 20.0 }  { delta
_C2 / 100.0}  { delta_C2 / 10.0 }
HC = BC  11  2  0  { inflow_C3a }  { inflow_C3b }  { delta_C3 / 20.0 }  { delta
```

```
_C3 / 100.0}  { delta_C3 / 10.0 }
HC = BC 13 1 0 WEBSPEED_VAL WEBSPEED_VAL_END { delta_webspeed / 20.0 } { delta_
webspeed / 100.0 } { delta_webspeed / 10.0 }
HC = BC 14 0 0 WEBSPEED_VAL WEBSPEED_VAL_END { delta_webspeed / 20.0 } { delta_
webspeed / 100.0 } { delta_webspeed / 10.0 }
HC = BC 24 1 0 VACUUM_VAL VACUUM_VAL_END { delta_vacuum / 20.0 } { delta_vacuum
 / 100.0 } { delta_vacuum / 10.0 }
END OF HC
BC = GD_PARAB   SS  3  R_MOMENTUM1  0  MESH_POSITION1  0  { inflow_C1a }  { inf
low_C2a }  { inflow_C3a }
BC = VELO_SLIPSS  21  {1.0/beta}  WEBSPEED_VAL  0.0  0.0
BC = U  NS  9  WEBSPEED_VAL
BC = CAPILLARY  SS  23  {surface_tension}  VACUUM_VAL  0.0
```

The file specifications at the beginning are the same as in the single parameter continuation run. To break down the `HC` conditions, consider this one:

```
HC = BC  11  1  0  { inflow_C2a }  { inflow_C2b }  { delta_C2 / 20.0 }  { delta
_C2 / 100.0}  { delta_C2 / 10.0 }
```

The meaning of each of these values is:

`BC`	A boundary condition floating point value will change.
`11`	Boundary condition with ID 11. This corresponds to the `BC = GD_PARAB` card. You can determine the BC ID by running *Goma* with the *-bc_list* command line option.
`1`	Vary the second floating point parameter (0-based counting). This is the C2 coefficient.
`0`	Allow variable path lengths in the continuation. A 1 would indicate a fixed path step.
`{ inflow_C2a }`	Reference the *aprepro* variable defined in the geometry file `slot-base.geom` representing the starting value of C2 (for this local continuation run).
`{ inflow_C2b }`	Reference the *aprepro* variable defined in the geometry `slot-base.geom` representing the ending value of C2.
`{ delta_C2 / 20.0 }`	Set our starting path step size to 1/20*th*, or 5%, of the total path length.
`{ delta_C2 / 100.0 }`	Set our minimum path step size to 1% of the total path length.
`{ delta_C2 / 10.0 }`	Set our maximum path step size to 10% of the total path length.

There are some peculiarities with the `HC` conditions. The three path length values (starting, minimum, and maximum) need to be positive values. The code will determine which direction to go by the beginning and ending values. The `HC` starting step sizes must all be the same relative ratio of the total path steps. This is why the author prefers setting the starting step sizes to the same ratio, as opposed to setting the starting values in a more independent way.

The output files (i.e., `slot-cont-out-N.exoII`) are all indexed by the parameter values for the first parameter listed in the `@params` setting in *arc.pl*. If there is a preferred reference parameter, this should be the first one in the `@params` list.

We omit any example output because it reports information in almost exactly the same way as single parameter continuation.

When Things Don't Go Smoothly...

Inevitably, something will go wrong when running *arc.pl*. Loss of disk space, machine crashes, a manual remesh step, etc., are all examples of events that are not scripting errors, but nonetheless "break" the script. In this section we highlight some of the more popular accidents and how to trick the script into continuing. It would be unacceptable to recompute the entire global continuation run simply because the very last local continuation run was interrupted by a machine crash...

The important players in these fixes are the input files required to start the local continuation run, and the file `start_vals.out`. Recall that at the end of the global continuation run, each of the local continuation runs' starting parameter value is written out to `start_vals.out`. At the beginning of the LSA phase, `start_vals.out` is read to determine which `*.exoII` files to locate. You can always generate your own `start_vals.out` file by noting which parameter values (*N*) appear in the `slot-cont-out-N.exoII` files. `start_vals.out` is simply a 1-line text file of these parameter values, separated by spaces. Following are a couple of accidents, and the workaround.

Interrupt during a local continuation run. Let *N* be the parameter value for the beginning of the local continuation run at which failure occurred. We need to set up a new global continuation run with the starting parameter value equal to *N*.

1 Back up `slot-base.exoII`. This is the geometry file used for the very first step (`FEM file` card).
2 Copy `slot-cont-in-N.exoII` to `slot-base.exoII` (for geometry).
3 Copy `slot-cont-in-N.exoII` to `slot-cont-in.exoII` (for initial guess).
4 Back up `start_vals.out`. You will need these parameter values for the LSA computations later (unless you are comfortable recreating `start_vals.out` yourself).
5 Edit *arc.pl* for a global continuation run from *N* to the original ending value. If a multi-parameter continuation run is performed, compute the starting values for the other parameters, too.
6 Set the `$CONTINUATION_ONLY` *Perl* variable to 1 in *arc.pl*.
7 Rerun the script, *arc.pl slot*. You will now have the same `slot-cont-out-N.exoII` files at its completion that would have been generated had there not been an interruption.
8 Replace `slot-base.exoII` with the backup, and generate a `start_vals.out` with all of the starting values (not just the ones from this last set).
9 Set `$CONTINUATION_ONLY` to 0, and `LSA_ONLY` to 1 in *arc.pl*.

Interrupt during LSA calculations. Let *N* be the parameter value for the interrupted LSA calculation. Note that the script's logic will skip an LSA computation for parameter *M* if `slot-stab-out-`

M `out` exists.

1. Remove the `slot-stab-out-`*N*`.out` file. If it exists (even if empty), then *arc.pl* believes it has already performed LSA for this steady state solution.
2. Set the `$LSA_ONLY` *Perl* variable to 1 in *arc.pl*.
3. Rerun the script, *arc.pl slot*.

General Notes

It is better to use as small a continuation step size as possible. This cuts down on deformation gradient failures, and will catch any rapid transitions in eigenvalues better. Unlike a fixed mesh continuation run, it is better to have a fixed continuation step size.

Currently, there does not appear to be a consensus about how to handle mesh displacement (`DX` and `DY`) boundary conditions. It is the author's experience that allowing all of the mesh displacements to be perturbed for the eigenanalysis causes numerical instability in the eigensolver. This was also true when using another eigensolver (ARPACK). When the mesh displacements were not allowed to be perturbed on the solid boundaries during the eigenanalysis, the numerical instability went away. However, there was no obvious transition from stable to unstable solutions, according to the eigenvalues. The continuation procedure does "ramp up" to the point where the author believes the solution becomes unstable. That is, as the continuation parameter approaches the critical parameter value, the continuation step sizes go to zero, causing failure of the continuation run. This is a current topic of investigation.