SOLUTION PROCEDURE FOR NON-LINEAR
FINITE ELEMENT EQUATIONS

PROJECT REPORT
ECI 284
WINTER 2003

SUBMITTED BY:
RITU JAIN
M.S.
(Structural Engg.)
University Of California, Davis
This project is done to complete the requirement of course ECI284. The goals of the projects were as follows:

- To study various continuation (path-following) methods.
- Get a good understanding of Hyper-Spherical Arc-Length Method.
- Implement a class called HSConstraint in OpenSees.

This report consists of two parts.

In the first part, I have tried to explain what I have learnt and what I have done towards achieving the above goals. It explains briefly about path-following methods in the beginning. Later it explains various aspects of an Arc-Length Method. It gives a brief overview of problems associated with the Arc-Length method.

In the second part, I have explained the algorithm, which I used to implement Arc-Length Method in OpenSees. I give the design of class and the hierarchy of classes to which it is linked in OpenSees. I also summarize the changes I made in OpenSees to include the HSConstraint class.
INDEX

• Introduction to Continuation Methods
• Continuation Methods Vs Traditional methods
• Arc-Length Method
  o Introduction
  o Parameterization
  o Predictor
  o Corrector
  o Step-Length Control
  o Convergence/Divergence Criteria
  o Different intersection Situation
• Algorithm
• Implementation in OpenSees
  o Class Hierarchy in OpenSees
  o Design of class ‘HSConstraint’ in C++
    o Modification in OpenSees classes to include ‘HSConstraint’
• Bibliography
CONTINUATION METHODS

The development and application of numerical procedures for following solution paths in nonlinear static structural problems have received significant attention in last two decades. Apparently an ideal solution method should be able to trace the entire static load-displacement path of a structure, which may include both softening and hardening behavior, together with the presence of load and displacement limit points and possibly bifurcations as well. A considerable number of path-following methods have been proposed and they all can be described within the framework of continuation methods.

Basically, continuation methods add a constraint equation to the original non-linear governing equation of the problem, and then solve the extended system of equations by incremental-iterative procedures such as Newton-Raphson, modified Newton Raphson, or quasi-Newton techniques, to obtain a solution point along the path. In a step-by-step manner, together with changing the value of a parameter contained in the constraint equation, called the path parameter, the solution path can then be traced in terms of a set of points.

Beginning with a known solution $x_0$, the continuation method is to compute further solutions

$$x_1, x_2, x_3, \ldots, x_k, x_{k+1}, \ldots$$

of the extended system of equations for specified values of path parameter in a step-by-step manner until one reaches a target point.

In general iteration methods are required to compute a particular point. These methods normally require suitable starting values in order that the iteration procedure converges to correct solution points since most iteration methods are only locally convergent. A continuation method has four aspects:
The possible parameterizations of a solution method are load control, displacement control, external work control etc. But using either of these alone poses several problems in tracing the solution path. In continuation method an adaptive parameterization is used, which is of more generalized form. And it can be changed as required during the process of path-tracing. In the predictor phase information that belongs to the point previously computed is used to compute a suitable starting value for the corrector phase. And in the corrector phase some numerical procedure is used to find out the solution of the extended system with the initial guess supplied by the predictor. The control of step size along the path is a crucial issue in the design of continuation methods. Once the parameterization, predictor and corrector strategies have been selected, it is hoped that such a step size control procedure can be achieved so that the desired solution path can be obtained at a minimum computational expense.
Continuation methods Vs Traditional non-linear solution methods

The difference between a continuation method and a traditional non-linear solution method can be described in terms of Newton-Raphson method Vs Arc-Length Method.

In Newton-Raphson approach, the load is subdivided into a series of load increments. The load increments can be applied over several load steps. Before each solution, the Newton-Raphson method evaluates the out-of-balance load vector, which is the difference between the restoring forces (the loads corresponding to the element stresses) and the applied loads. The program then performs a linear solution, using the out-of-balance loads, and checks for convergence. If convergence criteria are not satisfied, the out-of-balance load vector is re-evaluated, the stiffness matrix is updated, and a new solution is obtained. This iterative procedure continues until the problem converges.

In some nonlinear static analyses, if the Newton-Raphson method is used alone, the tangent stiffness matrix may become singular (or non-unique), causing severe convergence difficulties. Such occurrences include nonlinear buckling analyses in which the structure either collapses completely or "snaps through" to another stable configuration.
For such situations the arc-length method is useful to avoid bifurcation points and track unloading. The arc-length method causes the Newton-Raphson equilibrium iterations to converge along an arc, as can be seen in figure 1, thereby often preventing divergence, even when the slope of the load vs. deflection curve becomes zero or negative. The constraint equation is forced to be satisfied at each iteration.

Unlike traditional non-linear solution methods, continuation methods can handle post yielding non-linear behavior of materials which show strain-softening after yield point. They cannot be trapped in an infinite loop at the unstable region of the solution curve (since the stiffness matrix is updated based on both load increment and displacement increment).
ARC-LENGTH METHOD

An arc-length method is a continuation method used to solve non-linear finite element equations. The equation that governs the non-linear static structural problems can be expressed in the form:

\[ r(u, \lambda) = f_{int}(u) - \lambda f_{ext} = 0 \]  \hspace{1cm} (1)

Where, \( f_{int}(u) \) are the internal forces which are functions of displacement \( u \), the vector \( f_{ext} \) is a fixed external loading vector and the scalar \( \lambda \) is a load level parameter that multiplies \( f_{ext} \). This equation describes the case of proportional loading in which the loading pattern is kept fixed. These governing equations are called residual force equations.

**Parameterization:**

Generally, the solution path is parameterized using a general scalar equation or an auxiliary surface. The aim is to find its intersection with equation (1). In Arc-Length method this equation is called constraining equation and is given by

\[ a = (\Delta s)^2 - (\Delta l)^2 = \left( \frac{\psi_u}{u_{ref}^2} \Delta u^T S \Delta u + \Delta \lambda^2 \psi_f^2 \right) - (\Delta l)^2 \]  \hspace{1cm} (2)

Where \( \Delta l \) is the radius of desired intersection, and represents an approximation to the incremental arc length (\( \Delta s \)). Scaling matrix \( S \) is usually a diagonal nonnegative matrix that scales the state vector \( \Delta u \) and \( u_{ref} \) is a reference value with the dimension of \( \sqrt{(\Delta u^T S \Delta u)} \). The vector \( \Delta u \) and \( \Delta \lambda \) are incremental and not iterative values, and are starting from the last converged equilibrium state. The differential form of arc-length \( \Delta s \) is given by

\[ ds = \sqrt{\frac{\psi_u^2}{u_{ref}^2} d u^T S d u + d \lambda^2 \psi_f^2} \]  \hspace{1cm} (3)

The entire Arc-Length is given by

\[ s = \int ds \]  \hspace{1cm} (4)

The main essence of arc-length method is that the load parameter \( \lambda \) becomes a variable. With load parameter \( \lambda \) variable we are dealing with n+1
unknown. In order to solve this problem we have $n$ equilibrium equations (1) and the one constraint equation (2).

![Figure 2: Spherical Arc-Length Method](image)

**Predictor:**

In this phase information that belongs to the point previously computed is used to compute a suitable starting value for the corrector phase using following formulas:

$$
\Delta u_p = K_t^{-1} \Delta q_e = \Delta \lambda_p K_t^{-1} f_{ext} = \Delta \lambda_p \delta u_t
$$

(5)

where $K_t$ is the tangent stiffness matrix at the beginning of increment. Substitution equation (5) into the constraining equation (2) one obtains:
The solution for $\Delta \lambda_p$ is readily found:

$$\Delta \lambda_p = \pm \frac{\Delta l}{\sqrt{\psi_{u_f}^2 |\delta u_i^T S \delta u_i| + \psi_f^2}}$$  \hspace{1cm} (7)

where $\Delta l > 0$ is the given increment length. The absolute value of $|\delta u_i^T S \delta u_i|$ is needed if the stiffness matrix is chosen as a scaling matrix, i.e. $S = K_t$, since, after passing the limit point, the stiffness matrix is non-positive so $\delta u_i^T S \delta u_i \leq 0$.

Success of a path-following method depends crucially on the choice of the appropriate sign of the iterative load factor. Predictor Criteria to select the correct sign are as follows:

- Based on the sign of current stiffness determinant: Good for snap-back situation but fails in the case of bifurcation
- Based on incremental predictor work: Insensitive to bifurcation but fails in the situation of snap-back

A Solution could be to switch between these criteria according to the path situation... The question of choosing the right sign $+$ or $-$ in (7) is still a vigorous research topic.

**Corrector:**

In the correction phase, some numerical procedure is used to find out the solution of the extended system. Arc-length solves the augmented system of equations by applying Newton-Raphson to equations (1) and (2)
\[
\begin{align*}
\mathbf{r}^{new}(\mathbf{u}, \lambda) &= \mathbf{r}^{old}(\mathbf{u}, \lambda) + \frac{\partial \mathbf{r}(\mathbf{u}, \lambda)}{\partial \mathbf{u}} \delta \mathbf{u} + \frac{\partial \mathbf{r}(\mathbf{u}, \lambda)}{\partial \lambda} \delta \lambda = \\
&= \mathbf{r}^{old}(\mathbf{u}, \lambda) + \mathbf{K}_t \delta \mathbf{u} - \mathbf{f}_{ext} \delta \lambda = \\
&= 0
\end{align*}
\]

\[
\mathbf{a}^{new} = \mathbf{a}^{old} + 2 \frac{\psi^2}{u^2_{ref}} \Delta \mathbf{u}^T \mathbf{S} \delta \mathbf{u} + 2 \Delta \lambda \delta \mathbf{u} \psi^2_j = 0
\] (8)

Where, \( \mathbf{K}_t = \frac{\partial \mathbf{r}(\mathbf{u}, \lambda)}{\partial \mathbf{u}} \) is the tangent stiffness matrix. The aim is to have \( \mathbf{r}^{new}(\mathbf{u}, \lambda) = 0 \) and \( \mathbf{a}^{new} = 0 \) (or within convergence limits) so the previous system can be written as

\[
\begin{bmatrix}
\mathbf{K}_t & -\mathbf{f}_{ext} \\
2 \frac{\psi^2}{u^2_{ref}} \Delta \mathbf{u}^T \mathbf{S} & 2 \Delta \lambda \psi^2_j
\end{bmatrix}
\begin{bmatrix}
\delta \mathbf{u} \\
\delta \lambda
\end{bmatrix}
= -
\begin{bmatrix}
\mathbf{r}^{old} \\
\mathbf{a}^{old}
\end{bmatrix}
\] (9)

One can solve previous system of two equations for \( \delta \mathbf{u} \) and \( \delta \lambda \) as follows:

\[
\begin{bmatrix}
\delta \mathbf{u} \\
\delta \lambda
\end{bmatrix}
= -\mathbf{K}^{-1}
\begin{bmatrix}
\mathbf{r}^{old} \\
\mathbf{a}^{old}
\end{bmatrix}
\] (10)

Where \( \mathbf{K} \) is the augmented stiffness matrix given by

\[
\mathbf{K} = 
\begin{bmatrix}
\mathbf{K}_t & -\mathbf{f}_{ext} \\
2 \frac{\psi^2}{u^2_{ref}} \Delta \mathbf{u}^T \mathbf{S} & 2 \Delta \lambda \psi^2_j
\end{bmatrix}
\] (11)

The iterative displacement \( \delta \mathbf{u} \) is split into two parts, with the Newton change at the new unknown load level:

\[
\lambda^{new} = \lambda^{old} + \delta \lambda
\] (12)

becomes:

\[
\begin{align*}
\delta \mathbf{u} &= -\mathbf{K}_t^{-1} \mathbf{r}(\mathbf{u}^{new}, \lambda) = -\mathbf{K}_t^{-1} (\mathbf{f}_{init}(\mathbf{u}^{old}) - \lambda^{new} \mathbf{f}_{ext}) \\
&= -\mathbf{K}_t^{-1} (\mathbf{f}_{init}(\mathbf{u}^{old}) - (\lambda^{old} + \delta \lambda) \mathbf{f}_{ext}) = -\mathbf{K}_t^{-1} ((\mathbf{f}_{init}(\mathbf{u}^{old}) - \lambda^{old} \mathbf{f}_{ext}) - \delta \lambda \mathbf{f}_{ext}) \\
&= -\mathbf{K}_t^{-1} (\mathbf{r}(\mathbf{u}^{old}, \lambda^{old}) - \delta \lambda \mathbf{f}_{ext}) = -\mathbf{K}_t^{-1} \mathbf{r}^{old} + \delta \lambda \mathbf{K}_t^{-1} \mathbf{f}_{ext} = \delta \mathbf{u} + \delta \lambda \delta \mathbf{u}_t
\end{align*}
\] (13)

where \( \delta \mathbf{u}_t = \mathbf{K}_t^{-1} \mathbf{f}_{ext} \) is the displacement vector corresponding to fixed load vector \( \mathbf{f}_{ext} \) and \( \delta \mathbf{u} \) is an iterative change that would stem from the standard load-controlled Newton-Raphson at a fixed load level \( \lambda^{old} \). With the solution for \( \delta \mathbf{u} \) from (13), the incremental displacements are:
\[ \Delta u^{new} = \Delta u^{old} + \delta = \Delta u^{old} + \delta \tilde{u} + \delta \lambda \delta u_i \]  \hspace{1cm} (14)

where \( \delta \lambda \) is the only unknown. The constraint equation (2) can be used here and by rewriting it as:

\[ \left( \frac{\psi_u^2}{u_{ref}^2} (\Delta u^{new})^T S (\Delta u^{new}) + (\Delta \lambda^{new})^2 \psi_f^2 \right) = (\Delta l)^2 \]  \hspace{1cm} (15)

then by substitution \( \Delta u^{new} \) and \( \Delta \lambda^{new} \) from (14) and (12) one ends up with following quadratic scalar equation:

\[ \left( \frac{\psi_u^2}{u_{ref}^2} (\Delta u^{old} + \delta \tilde{u} + \delta \lambda \delta u_i)^T S (\Delta u^{old} + \delta \tilde{u} + \delta \lambda \delta u_i) + (\Delta \lambda^{old} + \delta \lambda)^2 \psi_f^2 \right) = (\Delta l)^2 \]  \hspace{1cm} (16)

or, by collecting terms

\[ \left( \frac{\psi_u^2}{u_{ref}^2} \delta u_i^T S \delta u_i + \psi_f^2 \right) \delta \lambda^2 + 2 \left( \frac{\psi_u^2}{u_{ref}^2} \delta u_i^T S (\Delta u^{old} + \delta \tilde{u}) + \Delta \lambda^{old} \psi_f^2 \right) \delta \lambda + \left( \frac{\psi_u^2}{u_{ref}^2} (\Delta u^{old} + \delta \tilde{u})^T S (\Delta u^{old} + \delta \tilde{u}) - \Delta l^2 + (\Delta \lambda^{old})^2 \psi_f^2 \right) = 0 \]

\[ a_1 \delta \lambda^2 + 2a_2 \delta \lambda + a_3 = 0 \]  \hspace{1cm} (17)

The quadratic equation can be solved for \( \delta \lambda \):

\[ \delta \lambda = \delta \lambda_1 = \frac{-a_2 + \sqrt{a_2^2 - a_1 a_3}}{a_1} \; ; \; \delta \lambda = \delta \lambda_2 = \frac{-a_2 - \sqrt{a_2^2 - a_1 a_3}}{a_1} \]  \hspace{1cm} (18)

or, if \( a_1 = 0 \), then:

\[ \delta \lambda = \frac{-a_3}{2a_2} \]  \hspace{1cm} (19)

Then complete change is defined from equation 14:

\[ \Delta u^{new} = \Delta u^{old} + \delta \tilde{u} + \delta \lambda \delta u_i \]  \hspace{1cm} (20)
An ambiguity is introduced in the solution for $\delta \lambda$ in (18). The tangent at the regular point on the equilibrium path has two possible directions, which generally intersect the constraint hyper-surface at two points. So, the solution for $\delta \lambda$ can be characterized as:

- Real roots of opposite sign: Solution process converges normally
- Real roots of equal sign opposite to that of $\Delta \lambda^{\text{old}}$: When going over a flat limit point
- Real roots of equal sign same as that of $\Delta \lambda^{\text{old}}$: encounter a turning point or erratic iteration behavior
- Complex roots: signal a sharp turning point, a bifurcation point, erratic or divergent iterates

For the first two cases correct sign of $\delta \lambda$ in the corrector step can be found out by applying following schemes:

- Positive External work criterion:
  External work expenditure over the predictor step $> 0$
  \[ \Delta W = f_{ext}^T \Delta u = f_{ext}^T K^{-1}_{\lambda} f_{ext} \delta \lambda > 0 \] (21)
  Fails at bifurcation points and turning points because
  \[ f_{ext}^T K^{-1}_{\lambda} f_{ext} = 0 \] (22)
at these points.

- Angle Criterion:
  Calculate $\Delta u^{\text{new}}$ for both $\delta \lambda_1$ and $\delta \lambda_2$ and whichever lies closest to the old incremental step direction $\Delta u^{\text{old}}$ is selected.

  \[ \Delta u_1^{\text{new}} = \Delta u^{\text{old}} + \delta \bar{u} + \delta \lambda_1 \delta u_t \] (23)
  \[ \Delta u_2^{\text{new}} = \Delta u^{\text{old}} + \delta \bar{u} + \delta \lambda_2 \delta u_t \] (24)

  Cosine of the angle should be maximum and is given by following formula:

  \[ \cos \phi = \frac{(\Delta u^{\text{old}})^T (\Delta u^{\text{new}})}{\|\Delta u^{\text{old}}\| \|\Delta u^{\text{new}}\|} = \frac{(\Delta u^{\text{old}})^T (\Delta u^{\text{old}} + \delta \bar{u} + \delta \lambda \delta u_t)}{\|\Delta u^{\text{old}}\| \|\Delta u^{\text{old}} + \delta \bar{u} + \delta \lambda \delta u_t\|} \] (25)
Step-Length Control:

A number of strategies have been proposed for controlling the step length size. In the Arc Length method Automatic increment is used. The new incremental length is found by applying

\[
\Delta l^{\text{new}} = \Delta l^{\text{old}} \left( \frac{I_{\text{desired}}}{I_{\text{old}}} \right)^n
\]  

(26)

\( \Delta l^{\text{old}} \): old incremental factor for which \( I_{\text{old}} \) iterations were required.

\( I_{\text{desired}} \): input, desired number of iterations

\( n \) is set to \( \frac{1}{2} \).

Convergence Criteria:

The arc-length method uses the following convergence criteria:

Displacement Convergence Criteria:

The change in the last correction \( \delta u \) of the state vector \( u \) measured in an appropriate norm, should not exceed a given tolerance \( \varepsilon_u \). Example: in Euclidean norm the termination criterion is given as:

\[
\| \delta u \|_{\text{scaled}} = \sqrt{\delta u^T S \delta u} \leq \varepsilon_u
\]  

(27)

Residual Convergence Criteria

Euclidean norm of residual is compared with some predefined tolerance:

\[
\| r(u, \lambda) \|_{\text{scaled}} = \sqrt{r^T S r} \leq \varepsilon_r
\]  

(28)

Energy Based Convergence Criteria

Above two convergence criteria are combined to give energy based convergence criteria:

\[
\| (\delta u)^T (r) \| = \sqrt{(\delta u)^T S (r)} \leq \varepsilon_u \varepsilon_r
\]  

(29)
Divergence Diagnosis:

The Newton-Raphson method is not guaranteed to converge. Some sort of divergence detection scheme is therefore necessary to stop the erroneous iteration cycle. Divergence can be diagnosed if either of the following inequalities occurs:

\[
\frac{\|\delta u\|_{scaled}}{\|u\|_{scaled}} \geq g_u \tag{30}
\]

\[
\frac{\|r\|_{scaled}}{\|r^{predictor}\|_{scaled}} \geq g_r \tag{31}
\]

Where, \(g_u\) and \(g_r\) are dangerous growth factors.

Different Intersection Situations:
Figure 3: Three different intersection situations

The constraint surface may intersect with the solution path in various ways. Figures 3a-c show three situations which may be more likely to occur in practice.

Figure (3a) exhibits a weak non-linear structural behavior and two solutions $x_{k-1}$ and $x_{k+1}$ exists. In general, ‘trackback’ could happen only if a poor initial guess is constructed in predictor stage.

Situations shown in figures 3b and 3c are more complex, both corresponding to the cases where the structure exhibits strong nonlinear features. In the former case, except for the two solutions, $x_{k-1}$ and the desired $x_{k+1}$ as expected, another two solutions, denoted by $x_{1 k+1}^1$ and $x_{2 k+1}^2$, respectively may exist. If the correct predictor criterion is utilized, $x_{k-1}$ may not be obtained, but $x_{1 k+1}^1$ and $x_{2 k+1}^2$ may be found by the corrector unexpectedly. Furthermore, if $x_{2 k+1}^2$ is attained, the numerical procedure can continue to follow the curve in the right direction, but the solution information between point $x_k$ and $x_{1 k+1}^1$ will be lost. However, if $x_{1 k+1}^1$ is attained, nothing seems wrong with the solution, but the next solution is likely to be $x_k$, i.e., ‘trackback’ occurring again. In the situation (3c), due to strong non-linearity, it is possible that the corrector may converge to point $x_{k-1}$ instead of the desired point $x_{k+1}$, even with a ‘normal’ initial guess from the Predictor’s point of view.
The above discussion suggest that, except for the poor initial guess, ‘trackback’ happens mainly because the constraint surface has more than one intersection with the solution curve, or because the non-linearity of the problem changes too dramatically between $x_k$ and $x_{k+1}$. As a matter of fact the fundamental reason is the deficiency between the path parameter used and the true arc length $s$, which increases when the non-linear features become strong. The only way to decrease the difference is to adopt a proper step size control strategy so that the change between any two adjacent situations is not very sharp and the tangent prediction can be sufficiently close to the desired solution.
ALGORITHM

Progress of the Arc-Length method is described, in relation with figure (2).

1. Starts from a previously converged solution \((u_0, \lambda_0 f_{ext})\).
2. An incremental tangential predictor step \(\Delta u_1, \Delta \lambda_1\) is obtained in prediction phase.
3. The next point \((u_1, \lambda_1 f_{ext})\) is obtained by adding the increments \(\Delta u_1\) and \(\Delta \lambda_1\) to the displacement \(u_0\) and load level \(\lambda_0\). This point will be the starting for the correction phase.
4. Iteration starts and equations given in corrector phase is used to calculate \(\delta \lambda_1\) and \(\delta u_1\) with \(\Delta u^{old} = \Delta u_1\) and \(\Delta \lambda^{old} = \Delta \lambda_1\)
5. New values of \(\Delta u\) and \(\Delta \lambda\) are obtained as follows
   \[\Delta u_2 = \Delta u_1 + \delta u_1\quad \text{and}\quad \Delta \lambda_2 = \Delta \lambda_1 + \delta \lambda_1\]
6. When added to the displacement \(u_0\) and load level \(\lambda_0\), at the end of the previous increment this process would lead to the next point \((u_2, \lambda_2 f_{ext})\)
7. Next iteration again uses the equations in corrector phase to calculate \(\delta \lambda_2\) and \(\delta u_2\) with \(\Delta u^{old} = \Delta u_2\) and \(\Delta \lambda^{old} = \Delta \lambda_2\)
8. New values of \(\Delta u\) and \(\Delta \lambda\) are obtained as follows
   \[\Delta u_3 = \Delta u_2 + \delta u_2\quad \text{and}\quad \Delta \lambda_3 = \Delta \lambda_2 + \delta \lambda_2\]
9. When added to the displacement \(u_0\) and load level \(\lambda_0\), at the end of the previous increment this process would lead to the next point \((u_3, \lambda_3 f_{ext})\), which is the intersection of constraint surface with the solution path and hence the desired solution point.
10. Next step starts with point \((u_3, \lambda_3 f_{ext})\) and new value of step length is calculated using (26). Steps from 1-10 repeated until the entire path is traced.
IMPLEMENTATION OF HYPER-SPHERICAL ARC-LENGTH METHOD IN OPENSEES

Class Hierarchy:

Following tree shows the hierarchy of analysis classes under integrator in OpenSees and shows where Hyper-Spherical Arc Length (HSConstraint) class fits in.
Design of class ‘HSConstraint’ in C++:

Definition of Class HSConstraint is given in a header file called ‘HSConstraint.h’. The header file also contains the declaration of different parameters used in Arc-Length method, for e.g. $\psi_u$, $\psi_f$, $u_{ref}$, scaling matrix $S$, arc length etc. It also declares some vectors and matrices required for the implementation of Arc-Length Method. The constructor of class takes four arguments of type ‘double’: arclength, $\psi_u$, $\psi_f$ and $u_{ref}$. All arguments except arclength have default values set in the constructor. Constructor will be modified later to take value of Scaling matrix also. The destructor of the class deletes all the vectors and matrices objects created.

The class is implemented in a C++ file called ‘HSConstraint.cpp’. The class has the following methods:

1. **int HSConstraint::NewStep(void):** This method implements predictor in the Arc-Length method. Returns ‘0’ if everything go right. In case of error, it returns ‘-1’.

2. **int HSConstraint::update(const Vector &dU):** This method implements Corrector in the Arc-Length method. It takes a vector of displacements as its argument. It is needed because after each iteration the SOE is going to change. Returns ‘0’ if everything go right. In case of error, it returns ‘-1’.

3. **int HSConstraint::domainChanged(void):** This method is called when the arc-length method is first called to set the domain. After that it is called when a solution point is obtained and the process goes to the next step. Returns ‘0’ if everything go right. In case of error, it returns ‘-1’.

4. **int HSConstraint::sendSelf(int cTag, Channel &theChannel):** This method returns ‘-1’ if it is unable to send the data across the channel. Else it returns ‘0’.

5. **int HSConstraint::recvSelf(int cTag, Channel &theChannel, FEM_ObjectBroker &theBroker):** This method returns ‘-1’ if it is unable to receive the data from the channel. Else it returns ‘0’.

6. **void HSConstraint::Print(ostream &s, int flag):** This method is used to print the data in an output stream ‘s’.
Modification in OpenSees classes to include class ‘HSConstraint’

Here is a description of the changes I made in OpenSees:

New Files:
HSConstraint.h: OpenSees/SRC/analysis/integrator/HSConstraint.h
HSConstraint.cpp: OpenSees/SRC/analysis/integrator/HSConstraint.cpp

Updated Files:
Makefile: OpenSees/SRC/analysis/integrator/Makefile
    Added ‘HSConstraint.o’ with obj files names.
classTags.h: OpenSees/SRC/classTags.h
    Added a tag, named ‘INTEGRATOR_TAGS_HSConstraint’ numbered ‘14’ for HSConstraint.cpp.
commands.cpp: OpenSees/SRC/tcl/commands.cpp
    Added code to define an OpenSees command for HSConstraint class.
Makefile: OpenSees/SRC/Makefile
BIBLIOGRAPHY

- Lecture Notes by Prof. Boris Jeremic
- Anders Eriksson, Structural instability analyses based on generalized path-following, Computer methods in applied mechanics and engineering, vol. 156, 1998 45-74
- Reijo Kouhia, Martti Mikkola, Numerical Treatment of multiple bifurcation points