ME3100
Modeling and Simulation
Project Report

Tracing the snap through behaviour of a simple truss structure using Arc Length Method

Hima Vamsi(ME17BTECH11009)
Vinay Cheguri(ME17BTECH11017)
Rishi J(ME17BTECH11023)
Praharsh K(ME17BTECH11038)
Aniruthan R(ME17BTECH11007)

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Contents

1. Problem Definition and Background

1.1 Problem Statement

A simple truss problem :

We first consider the simplest possible structure comprised of two truss members with initial length L_0 and cross section A_0 that initially form an angle θ_0 with the horizontal axis as shown in Figure 1.1.

1.2 Background

The truss members are homogeneous and are assumed to be made of an isotropic and linearly elastic material. We also assume that it is impossible for the members to buckle and hence, they can only shrink under compression. Moreover, the fact that trusses can only carry axial forces and can only deform by shrinking or extending, implies that there is no need to discretize this problem. The truss members are connected with a hinge about which they are allowed to rotate, and their lower ends are fixed. A force P is applied at the hinge point subjecting the truss members into compression. With little examination, it is clear that

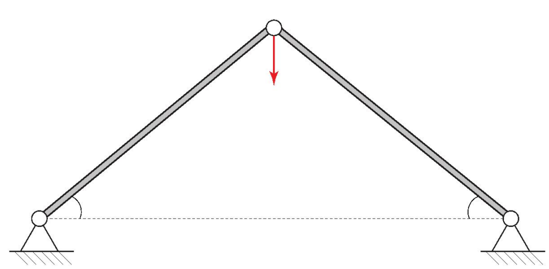


Figure 1.1: A simple structure consisting of 2 linearly elastic truss members that form an initial angle θ_0 with the horizontal plane. The structure is loaded with a force P that subjects the truss members into compression

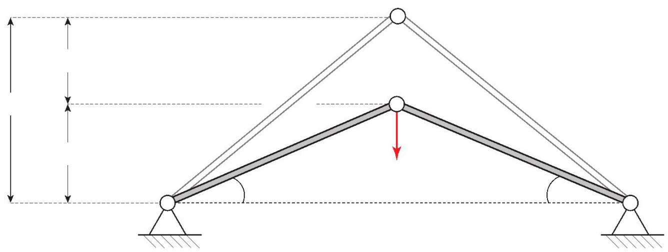


Figure 1.2: A schematic representation of a possible deformed state for the structure consisting of two truss members subjected into compression

the only degree of freedom is the vertical displacement u of the hinge point and our primary goal in this problem is to determine the relationship between P and u.

We will not make any assumptions regarding the magnitude of deformations. The displacement of the hinge u can be arbitrarily large as long as the truss members shrink enough for the displacements to be compatible. A possible deformed configuration or the truss is shown in Figure 1.2. We start from the equilibrium equation, which is expressed in terms of force balance between the externally applied force P and the internally developed forces F_L, keeping in mind that the equation must be written with respect to the deformed state.

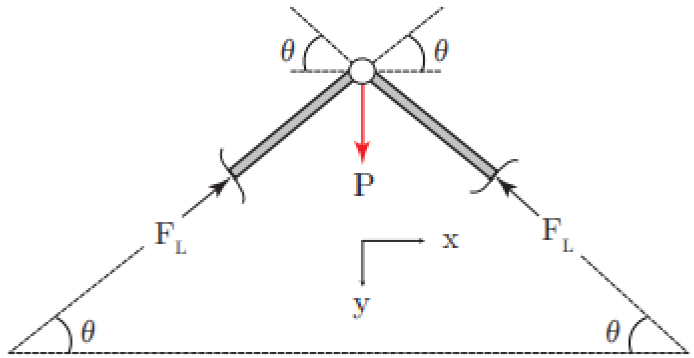


Figure 1.3: Force balance; the compressive/tensile forces developed internally in the trusses must be in equilibrium with the externally applied force P

1.3 Solving Strategy

According to Fig.1.3, We can write:

$$\sum F = 0 \quad \Rightarrow \quad P = 2F_L sin(\theta)$$

The **constitutive** equation is

$$\sigma = E\epsilon \quad \Rightarrow \quad \frac{F_L}{A_0} = E\frac{L_0 - L}{L_0} \quad \Rightarrow \quad F_L = \frac{EA_0}{L_0}(L_0 - L) = k(L_0 - L)$$

where k is a measure of each member's stiffness. Now, we only have to determine the kinematics equations that would relate the hinge's displacement u with the initial and deformed lengths of the truss members. We expect the kinematics equation to be nonlinear as a result of the preliminary assumption that the displacement u can be arbitrarily large. Based on Figure 1.4, we take the Pythagorean theorem for (ABC) and:

$$L^2 = \left(\Delta'\right)^2 + (L_0\cos(\theta_0))^2 = (L_0\sin(\theta_0) - u)^2 + (L_0\cos(\theta_0))^2 = L_0^2 - 2L_0\,u\,sin(\theta_0) + u^2 \Rightarrow$$

$$\boxed{\frac{L}{L_0} = \sqrt{1 - 2\frac{u}{L_0}\sin(\theta_0) + \left(\frac{u}{L_0}\right)^2}}$$

By using this in the force balance equation, we get :

$$P = 2k(L - L_0)\sin(\theta) = 2k(L - L_0)\left(\sin(\theta_0) - \frac{u}{L_0}\right) \Rightarrow$$

$$\frac{P}{2kL_0} = \left(\frac{L}{L_0} - 1\right)\left(\sin(\theta_0) - \frac{u}{L_0}\right) = \left(\frac{1}{\sqrt{1 - 2\frac{u}{L_0}\sin(\theta_0) + \left(\frac{u}{L_0}\right)^2}} - 1\right)\left(\sin(\theta_0) - \frac{u}{L_0}\right) \Rightarrow$$

$$\boxed{\frac{P}{2kL_0} = \left(\frac{1}{\sqrt{1 - 2\frac{u}{L_0}\sin(\theta_0) + \left(\frac{u}{L_0}\right)^2}} - 1\right)\left(\sin(\theta_0) - \frac{u}{L_0}\right)}$$

We now define the normalized load λ and displacement a as follows:

$$\lambda = \frac{P}{2kL_0} \qquad \alpha = \frac{u}{L_0}$$

Now, the expression can be written as:

$$\boxed{\lambda(a) = \left(\frac{1}{\sqrt{1 - 2a\sin(\theta_0) + a^2}} - 1\right)(\sin(\theta_0) - a)}$$

Now if we plot this expression we would get a normalized force–displacement curve λ-α that characterizes the structures behavior and a representative plot is shown in Figure 1.5 below.

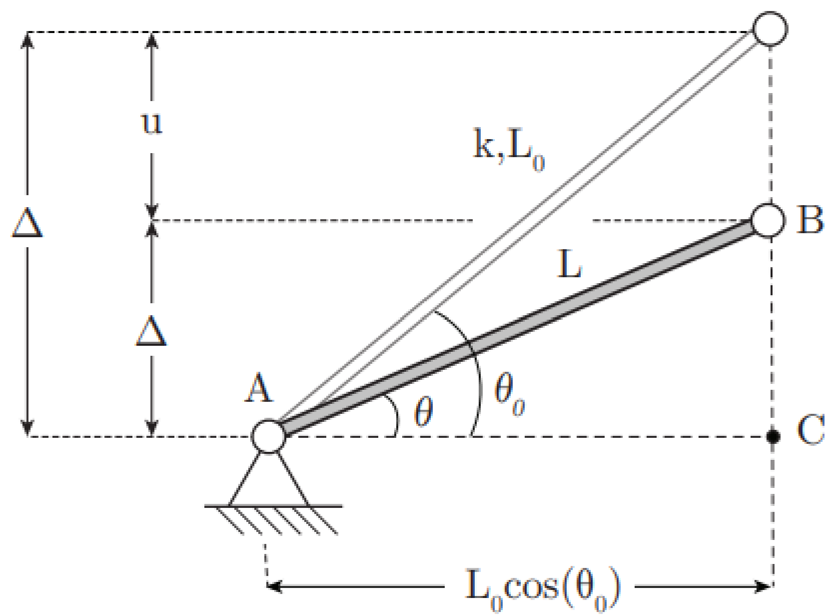


Figure 1.4: Deformed and undeformed states in the case of finite deformations

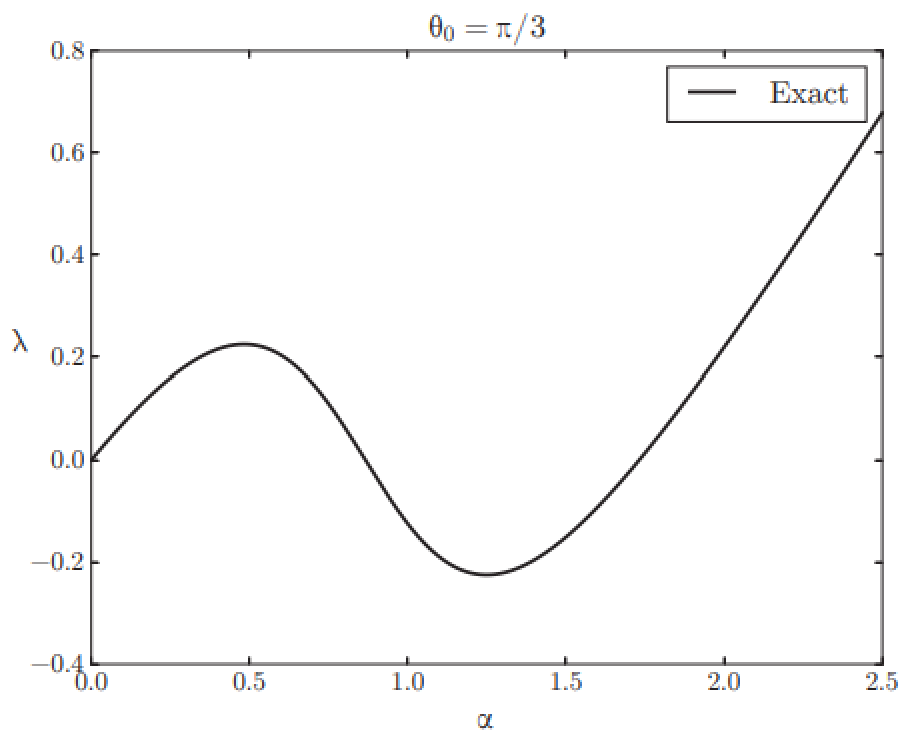
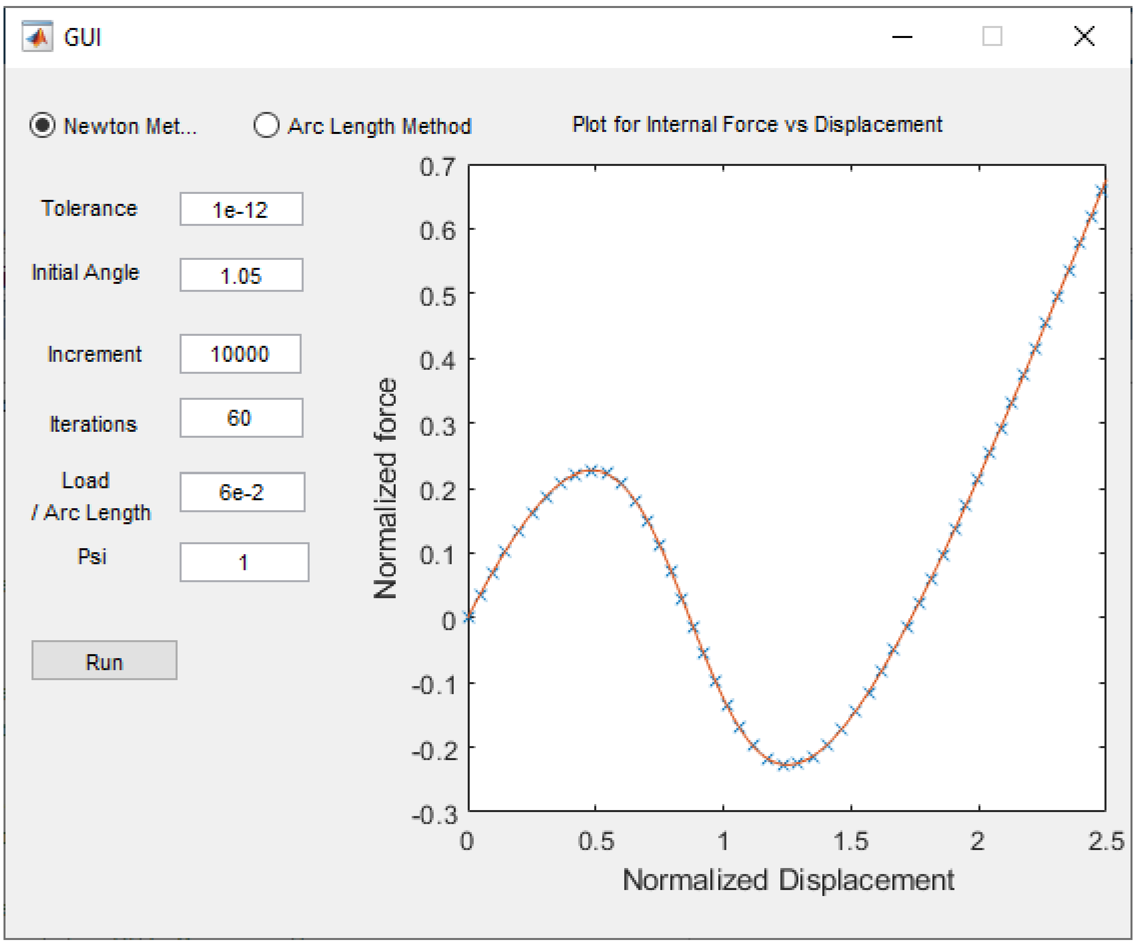


Figure 1.5: A plot of the normalized force displacement curve for the simple truss problem

2. Graphical User Interface

2.1 GUI Layout



2.2 GUI properties

The GUI provides two different methods to apply as radio button options for the given problem (Both of them can be selected at once too) , namely:

1. **Newton Method**
2. **Arc Length Method**

We can edit various parameters to simulate the given problem and its variants.

- **Tolerance :** It dictates the convergence of the solution. once the obtained solution is at the limits of the mentioned tolerance, the solution is said to be converged.
- **Initial Angle :** It is the initial angle as mentioned in the truss problem (θ)
- **Iterations :** It's the maximum number of iterations allowed to the code to reach convergence.
- **Increment :** It's the maximum increments allowed of the load acting on the truss.(for Arc-Length Method)
- **Load Increment/Arc length :** Load Increment is a increment in the load at every step for Newton's Method and Arc length is an essential parameter related to load increment in Arc Length Method.
- ψ **:** It is a parameter essential for Arc Length Method

3. Solving using Newton's Method

3.1 Algorithm

1. Start with $u_0 = 0$ and $\lambda_0 = 0$.

2. Increment λ as $\lambda' = \lambda_0 + \Delta\lambda$ with the predefined incrementation parameter $\Delta\lambda$.

3. Update u as $u' = u_0 + \Delta u$, where $\Delta u = [K_T]_{u_0}^{-1} \cdot (\Delta\lambda q)$.

4. Calculate the displacement correction δu as, $\delta u = -[K_T]_{u'}^{-1} \cdot \hat{R}(u')$.

5. Update $u_0 = u' + \delta u$ and $\lambda_0 = \lambda'$.

6. If $\hat{R}(u'') <$ tolerace, break.

7. else, goto step 2 and repeat.

3.2 Parameters influencing convergence

The convergence depends on the provided tolerance and the predefined load incrementation parameter λ.

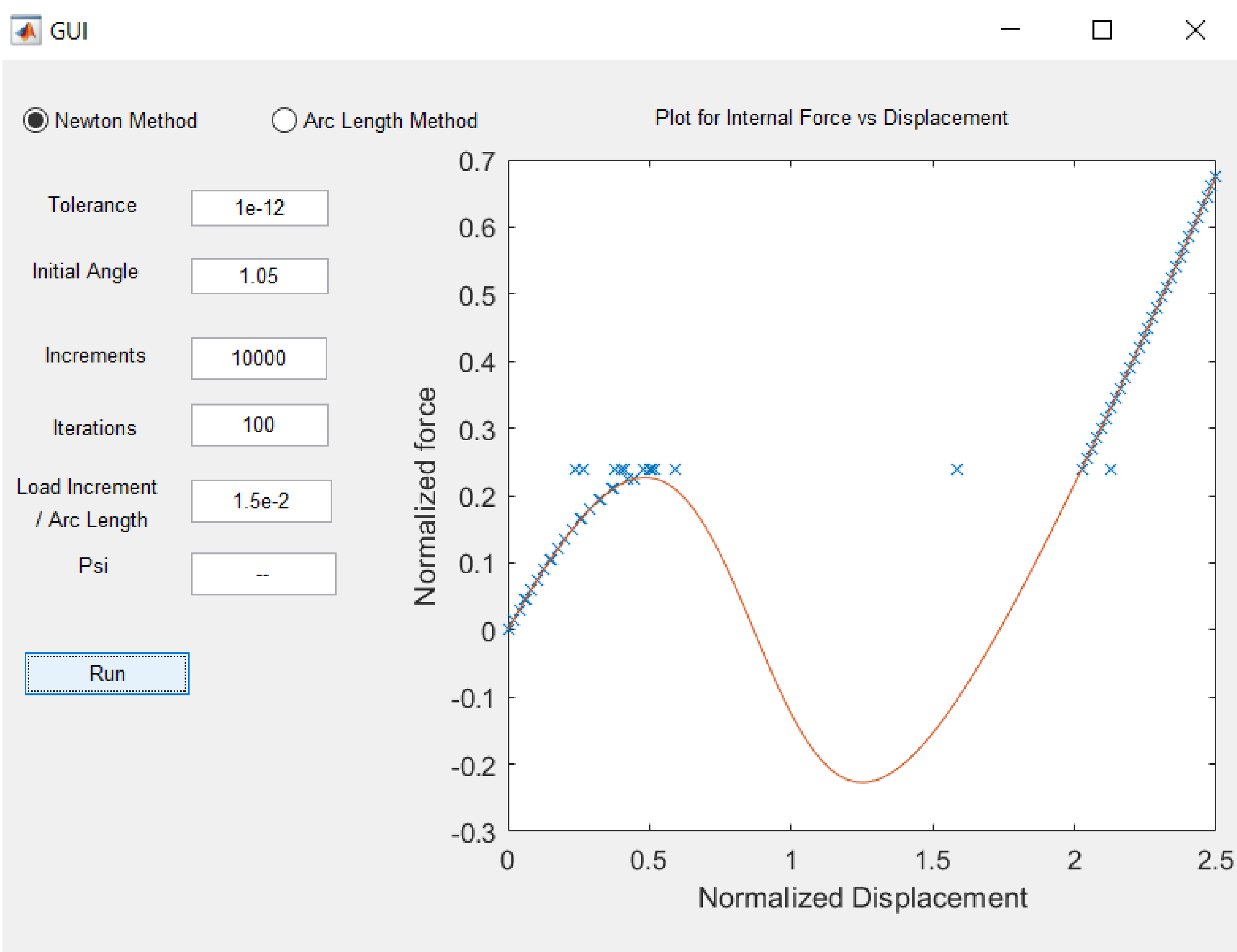
3.3 MATLAB Code

```
function [a_t,al_t]=newton(dl,th0,incr,max_iter,tol)
    n=1;
    iq=zeros(n,1);
    iq(1)=1;

    a=zeros(n,1);
    f=zeros(n,1);
    df=zeros(n,n);
    dfinv=zeros(n,n);

    dls=zeros(2,1);
    dao=zeros(n,1);
    al=0.0;

    a_t=0;
    al_t=0;

    count=1;

    for i=1:incr
        if a(1)>=2.5
            break;
        end
        da=zeros(n,1);

        a=a+da;
```

```
        al=al+dl;

        f=fcn((a),th0,(al));
        fcheck=sqrt(f'*f);

        if fcheck<tol
            iloop=0;
        else
            iters=0;
            while fcheck>tol
                iters=iters+1;
                [df,dfinv]=dfcn(a,th0,al);
                da=-1*(dfinv'*f);

                a=a+da;

                count=count+1;
                a_t(count)=a;
                al_t(count)=al;
                f=fcn(a,th0,al);
                fcheck=sqrt(f'*f);

                if iters>max_iter
%                   disp(['Convergence cannot achieved within',max_iter,'iterations'])
%                   disp('Program stops')
                    return
                end
            end
        end
    end
% disp('The program completed successfully')
end
```

```
function bb=b(x,y)
      bb=1.+x.^2.0 -2.0.*x.*sin(y);
end

function f=fcn(x,y,z)
      bb=b(x(1),y);
            f(1)=(1./sqrt(bb)-1.0)*(sin(y)-x(1))-z;
end

function [df,dfinv]=dfcn(x,y,z)
      bb=b(x(1),y);
      df=zeros(1,1);
            df(1,1)=1-(1.-sin(y)^2.0)/(bb^1.5);
            dfinv=1/df;
end
```

3.4 Plots and Results through GUI



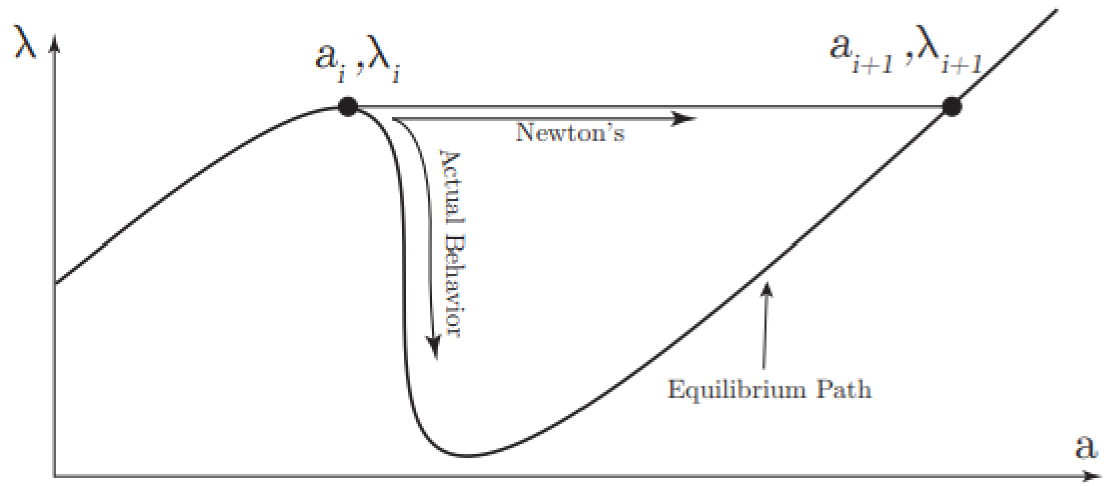
This is result when the Newton method code is run with the inputs:

- Maximum increments = 10000
- Maximum iterations = 100
- tolerance = 10^{-12}
- Load increment = 1.5 $\times 10^{-2}$

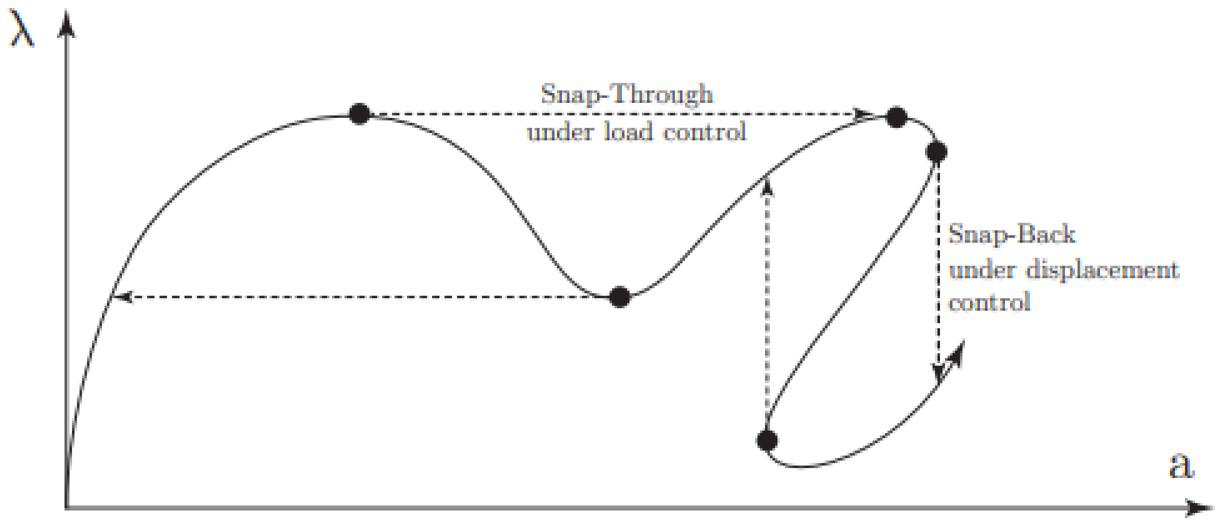
Using newton method, we are unable to trace the complete load-displacement curve. It snaps through beyond the critical point.

3.5 Demerits of Newton's Method

- Newton's method is associated with a major drawback. The method fails to accurately follow the 'equilibrium' path once the tangent stiffness reaches zero. That happens due to the formulation of Newton's method, and in particular that it restricts the parameter λ to change monotonically every increment.



- In the Snap-through case, Newton's method fails in load–control. Now in many cases, one way to circumvent problems like these is to use displacement control, where you can continuously increase the displacements u and still remain on the equilibrium curve. In general however, apart from Snap–Through behaviors under load control, a problem may exhibit Snap–Back behaviors under displacement control or even both.



- As a general rule, if the problem under consideration requires information after its critical/failure points then Newton's method is not a good choice.

- Buckling analysis and non-linear materials that exhibit work softening are just two example problems that cannot be solved using Newton's method.

- The Newton's method cannot accurately predict the solution after a limit point is reached.

4. Solving using Arc Length Method

4.1 Algorithm

At every step of the method apart from the beginning of each increment we can outline the steps to be followed as:

(a) . Every converged increment store the converged displacement and load corrections as $(\Delta u_n, \Delta\lambda_n)$.

(b) Calculate the sign of the product $(\Delta u_{n+1} + \delta u^i)^T.\Delta u_n + \psi^2 \Delta\lambda_n(\Delta\lambda_{n+1} + \delta\lambda)(q^T.q)$

(c) We choose the that leads to the largest DOT product and thus is closer to the previous correction

(d) In the special case where the two solutions give the same dot products, then choose either one

This method for choosing the correct solution is able to help the solution evolve forwards in most cases.
The Arc-Length method initiation for every increment as well as the iterative loops until convergence is achieved are outlined in the pseudo code that follows:

1. Initiate Increment
2. Set $\Delta u = 0$ and $\Delta\lambda = 0$
3. $\delta\bar{u} = 0$ and $\delta u_t = ([K_T]_u^{-1}.q)$
4. Solve arc length equation for $\delta\lambda_1$ and $\delta\lambda_2$
5. Choose the correct solution
6. Update u, λ as $u' =$ u + δu and $\lambda' = \lambda + \delta\lambda$
7. Check for convergence $||R(u', \lambda')|| < tol$
8. If convergence criteria are met then GOTO Step 10
9. Initiate Iteration
   (a) Set $\Delta u = \delta u$ and $\Delta\lambda = \delta\lambda$
   (b) Calculate $\delta\bar{u}$ and δu_t
   (c) Solve arc length equation for $\delta\lambda_1$ and $\delta\lambda_2$
   (d) Choose the correct solution
   (e) Update u, λ as $u' =$ u + δu + Δu and $\lambda' = \lambda + \delta\lambda + \Delta\lambda$
   (f) Check for convergence $||R(u', \lambda')|| < tol$
   (g) If convergence criteria are met then GOTO Step 10
   (h) GOTO Step 9
10. Proceed to next Increment

Finally, we still need to address a final issue that arises in step 5 of the previous algorithm. If we set $\Delta u = 0$ and $\Delta\lambda = 0$ and we do not have any information regarding the last converged increment (i.e. beginning of the analysis) it is impossible to determine the correct solution using the DOT rule since both DOT products will be equal to zero. A way around this issue in such cases is to determine the correct solution based on the sign of the determinant. In particular,

- Calculate the determinant of the Jacobian, namely $[K_T]$, and also it's sign
- Solve arc length equation for $\delta\lambda_1$ and $\delta\lambda_2$
- Choose the $\delta\lambda_i$ whose sign is the same as the determinants

4.2 Parameters influencing convergence

The convergence depends on the provided tolerance, the ψ, and the arc-length.

4.3 MATLAB Code

```
function [a_t,al_t] = arc_length(psi,dll,th0,incr,max_iter
    ,tol)

    n=1;
    iq=zeros(n,1);
    iq(1)=1;

    a=zeros(n,1);
    f=zeros(n,1);
    df=zeros(n,n);
    dfinv=zeros(n,n);

    dls=zeros(2,1);
    dao=zeros(n,1);
    al=0.0;

    a_t=0;
    al_t=0;

    count=1;

    for i=1:incr
        if a(1)>2.5
            break;
        end
        da=zeros(n,1);
        dab=da;
```

```
        dat=da;
        dda1=da;
        dda2=da;
        dda=da;
        dalpha=da;
        dl=0.0;

        [df,dfinv]=dfcn(a+da,th0,al+dl);
        dat=dfinv'*iq;

        [ddl1,ddl2]=arc(psi,dll,da,dab,dat,dl,iq);

        dda1=dab+ddl1*dat;
        dda2=dab+ddl2*dat;

%       det=det(df);
        det=df;

        if det*ddl1>0
            dda=dda1;
            ddl=ddl1;
        else
            dda=dda2;
            ddl=ddl2;
        end

        dalfa=da+dda;
        dlamda=dl+ddl;

        f=fcn((a+dalfa),th0,(al+dlamda));

        fcheck=sqrt(f'*f);

```

```
        if fcheck<tol
            a_t(count)=a;
            al_t(count)=al;
            a=a+dalfa;
            al=al+dlamda;

            count=count+1;
            a_t(count)=a;
            al_t(count)=al;

            dao=dalfa;
            dlo=dlamda;
            iloop=0;
        else
            iters=0;
            while fcheck>tol
                iters=iters+1;
                da=dalfa;
                dl=dlamda;

                f=fcn((a+da),th0,(al+dl));
                [df,dfinv]=dfcn((a+da),th0,(al+dl));

                dab=-dfinv'*f;
                dat=dfinv'*iq;

                [ddl1,ddl2]=arc(psi,dll,da,dab,dat,dl,iq);


                dda1=dab+ddl1*dat;
                dda2=dab+ddl2*dat;

                %                           det=det(df);
```

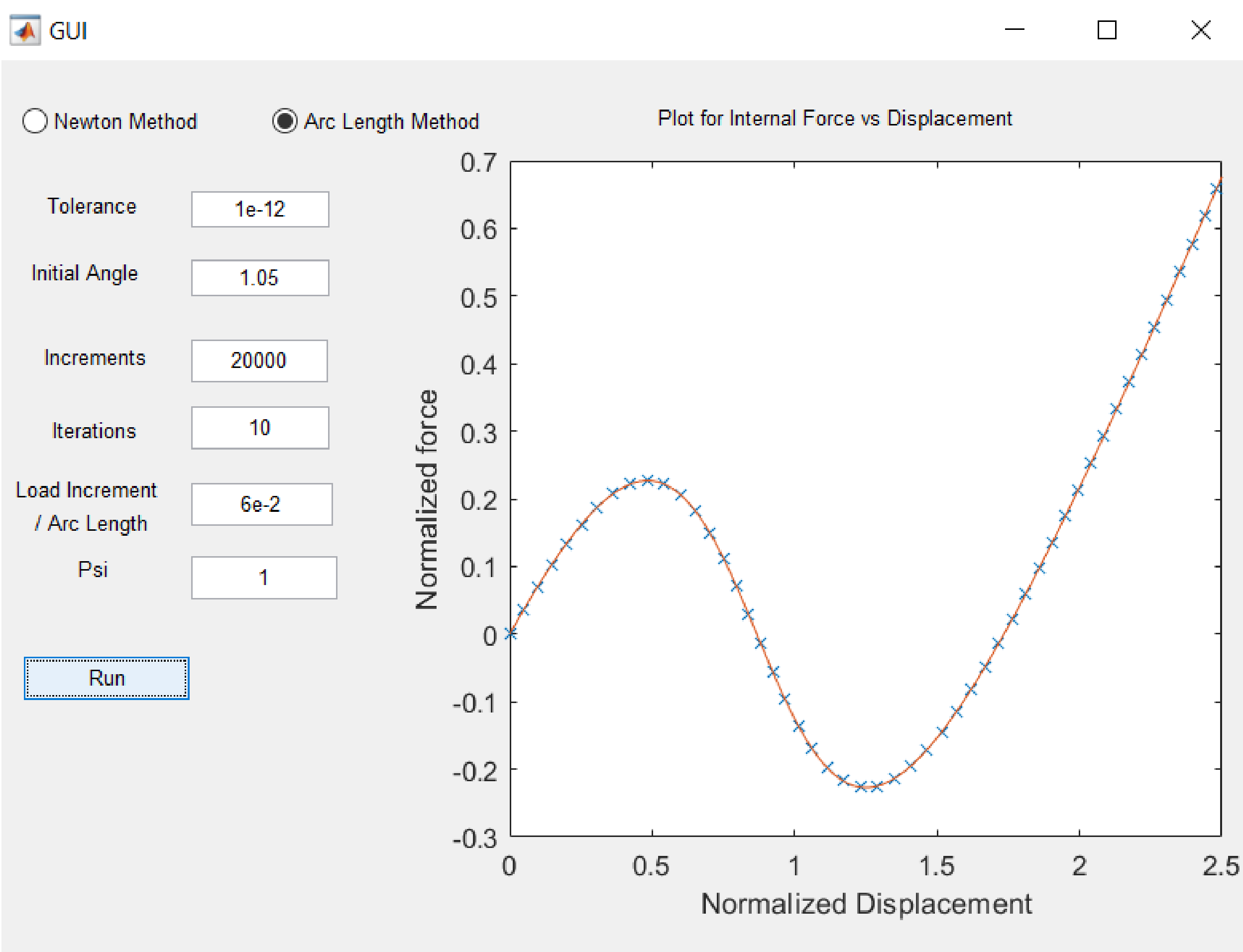
```
det=df;

daomag=(dao'*dao);

if daomag==0.
    if (dl+ddl1)*(det)>0
        dda=dda1;
        ddl=ddl1;
    else
        dda=dda2;
        ddl=ddl2;
    end

else
    aux1=(da+dda1)'*dao;
    aux2=(da+dda2)'*dao;

    aux3=dlamda*(dl+ddl1)*(iq'*iq);
    aux4=dlamda*(dl+ddl2)*(iq'*iq);

    dot1=aux1+(psi^2)*aux3;
    dot2=aux2+(psi^2)*aux4;

    if dot1>dot2
        dda=dda1;
        dd1=ddl1;
    else
        dda=dda2;
        ddl=ddl2;
    end

end
if ddl1==ddl2
```

```
                    dda=dda1;
                    ddl=ddl1;
                end

                dalfa=da+dda;
                dlamda=dl+ddl;

                f=fcn((a+dalfa),th0,(al+dlamda));
                fcheck=norm(f);

                if iters>max_iter
                    iters=max_iter+1;
                    break;
                end

            end

            if iters>max_iter
%                   disp(['Convergence cannot achieved
    within',max_iter,'iterations'])
%                   disp('Program stops')
                hold off
                return
            else
                a_t(count)=a;
                al_t(count)=al;
                a=a+dalfa;
                al=al+dlamda;

                count=count+1;
                a_t(count)=a;
                al_t(count)=al;
                dao=dalfa;
```

```
                    dlo=dlamda;
                end
%                   plot(a,al,'x');
            end
        end
end
%% Helper Functions

function bb=b(x,y)
    bb=1.+x.^2.0-2.0.*x.*sin(y);
end

function f=fcn(x,y,z)
    bb=b(x(1),y);
        f(1)=(1./sqrt(bb)-1.0)*(sin(y)-x(1))-z;
end

function [df,dfinv]=dfcn(x,y,z)
    bb=b(x(1),y);
    df=zeros(1,1);
        df(1,1)=1-(1.-sin(y)^2.0)/(bb^1.5);
        dfinv=1/df;
end

function [ddl1,ddl2]=arc(psi,dll,da,dab,dat,dl,iq)

    c1=dat'*dat+(psi^2)*(iq'*iq);
    c2=2.0*(((da+dab)'*dat)+dl*(psi^2)*(iq'*iq));
    c3=(da+dab)'*(da+dab)+(dl^2)*(psi^2)*(iq'*iq)-(dll^2);

    if c2^2-4*c1*c3>0
        dls=roots([c1,c2,c3]);
        ddl1=dls(1);
```

```
            ddl2=dls(2);
        else
            ddl1=-c2/(2*c1);
            ddl2=-c2/(2*c1);
%               disp('Poosible isssue in arc length')
        end
end
```

4.4 Plots and Results through GUI



This is result when the Arc Length method code is run with the inputs:

- Maximum increments = 20000
- Maximum iterations = 10

- tolerance = 10^{-12}
- Arc-length = 6 $\times 10^{-2}$
- $\psi = 1$

Using the Arc Length method, we are able trace the complete load-displacement curve.

5. Conclusions

Therefore, Arc-Length Method is more suitable than Newton Method to solve non-linear problems like Snap-through and Snap-Back problems.

6. References

- Nonlinear Analysis of Structures: The Arc Length Method: Formulation, Implementation and Applications. [https://scholar.harvard.edu/files/vasios/files/ArcLength.pdf]