CE394M: FEM solvers and errors

Krishna Kumar

University of Texas at Austin

krishnak@utexas.edu

March 6, 2019

Overview

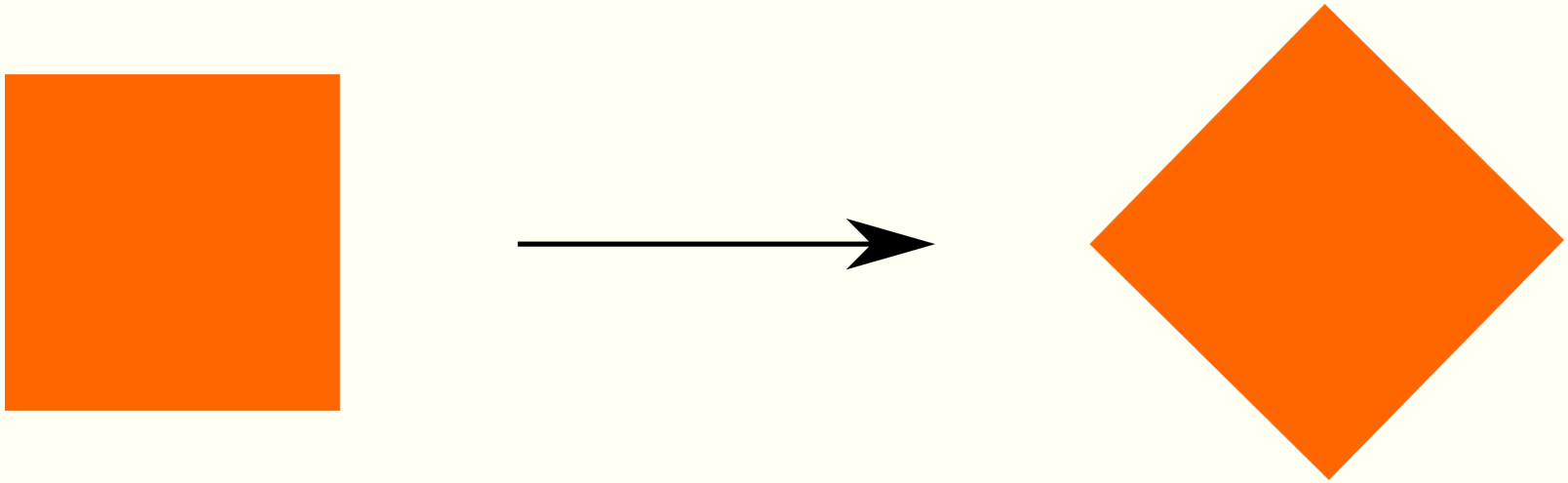
1. Solving non-linear problems
   - Iterative solvers
   - Tangent stiffness
   - Newton Raphson
2. Arc length method
3. Error estimates
   - A priori estimates
   - Posterior error estimation
4. Time-dependent problems
   - Explicit schemes
   - Implicit method

Linear and non-linear problems

- **Linear problems**
  - The response can only be approximated as linear if its deformations/motions are small.
  - In linear analyses, the response to individual load cases can be scaled and added to the results from other linear analyses, which is the principle of superposition.
- **Non-linear problems**
  - Superposition is invalid.
  - The solution is an incremental/iterative process.
  - An iteration is the solution of a system of equations linearised about the current state of the nonlinear physical problem.

Non-linearity in geotechnical engineering

- **Material non-linearity**
  - plasticity
- **Contact**
  - discontinuous source of non-linearity
- **Large deformations and motions of a geotechnical structures**
  - rotation, rigid body motion
  - often ignored, still in the research area.



No strains if linear strain-displacement relations are used.

Linear solvers

To solve a system of linear equations of the form $\mathbf{Ka} = \mathbf{b}$, there are two families of methods of solvers that can be used: direct and iterative.

- Direct solvers solve a system of linear equations in a predefined number of steps.
- Methods are based on Gauss elimination, with the most common method being LU decomposition.
- The time required to solve a linear system increases with the number of computer operations performed.
- For a direct linear solver applied to a dense system of size $n \times n$: CPU time $= Cn^3$.
- If the number of degrees of freedom in a finite element simulation is doubled, the time required to solve the system will increase by a factor of 8!

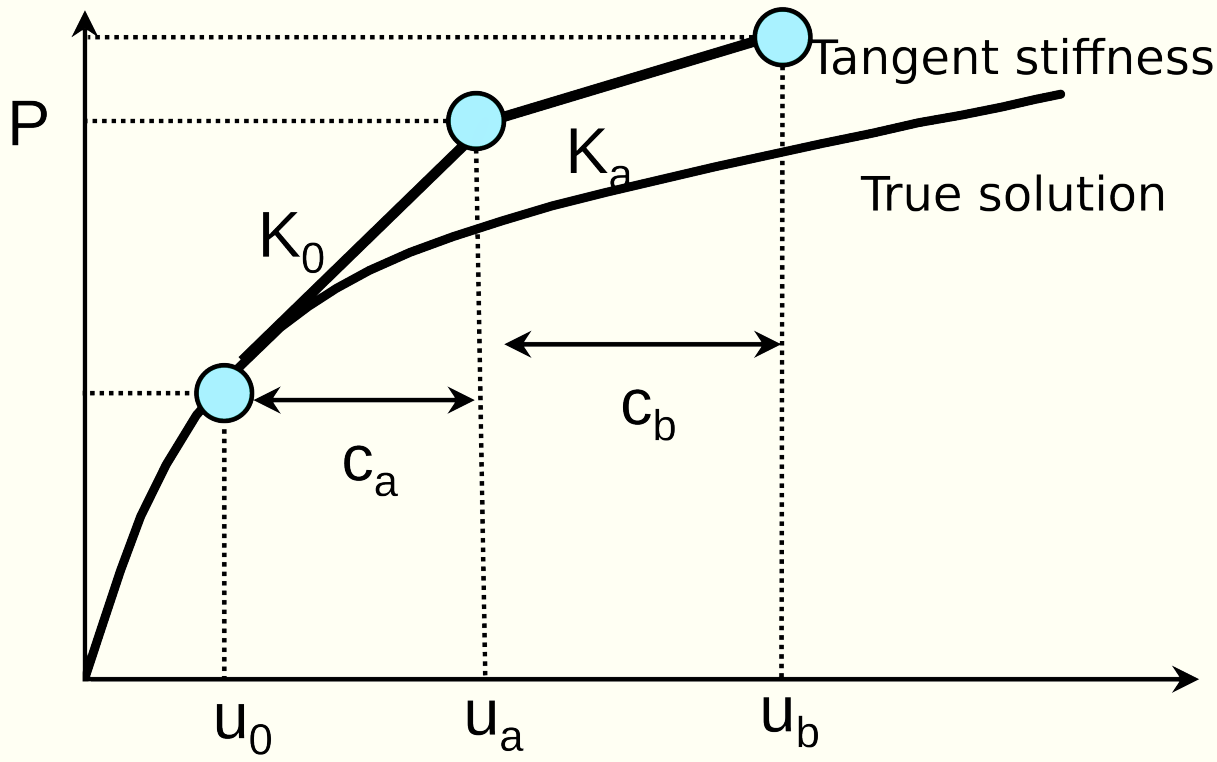
Iterative solvers

The system of equations that we derived from the finite element approximation of the BVP:

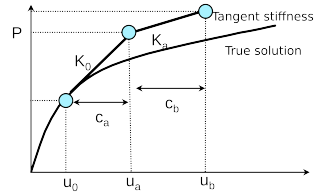
$$F_{\text{int}}(u) - F_{\text{ext}} = 0$$

At this point we remind ourselves that in the case of finite deformations, both F_{int} and F_{ext} are in general very nonlinear terms.
The integration has to be carried over the current volume and surface (of the finite element under consideration) that may in general depend on u in a highly non-linear fashion.

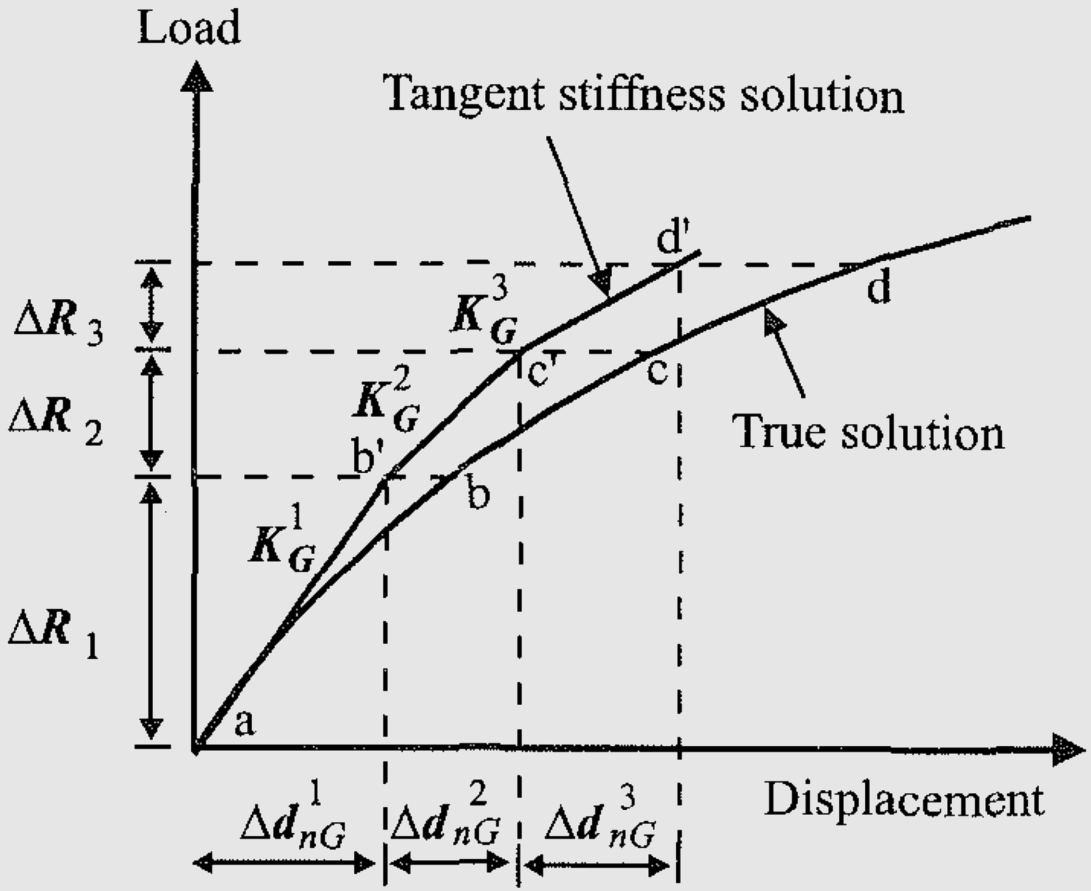
Tangent stiffness method

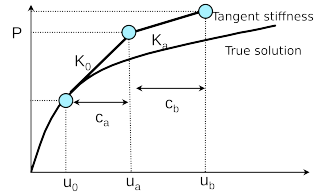


- Many small increments are need to obtain accurate solution
- Need to perform a parametric study to find the optimum incremental
- Defining increments is very important
- Program - SAGE CRISP



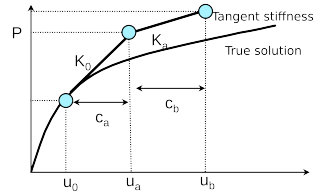
- Many small increments are need to obtain accurate solution
- Need to perform a parametric study to find the optimum incremental
- Defining increments is very important
- Program - SAGE CRISP





- Many small increments are need to obtain accurate solution
- Need to perform a parametric study to find the optimum incremental
- Defining increments is very important
- Program - SAGE CRISP

The analysis starts with the application of ΔR. The incremental global stiffness matrix $[\mathbf{K}_G]^1$ for this increment is evaluated based on the unstressed state of the bar corresponding to point '*a*'. For an elasto-plastic material this might be constructed using the elastic constitutive matrix [**D**]. Equation is then solved to determine the nodal displacements Δd^1. As the material stiffness is assumed to remain constant, the load displacement curve follows the straight line ab'. In reality, the stiffness of the material does not remain constant during this loading increment and the true solution is represented by the curved path '*ab*'. There is therefore an error in the predicted displacement equal to the distance '$b'b$', however in the tangent stiffness Load approach this error is neglected. The second increment of load, ΔR^2, is then applied, with the incremental global stiffness matrix $[\mathbf{K}_G]^2$ evaluated using the stresses and strains appropriate to the end of increment 1, i.e. point 'b.'

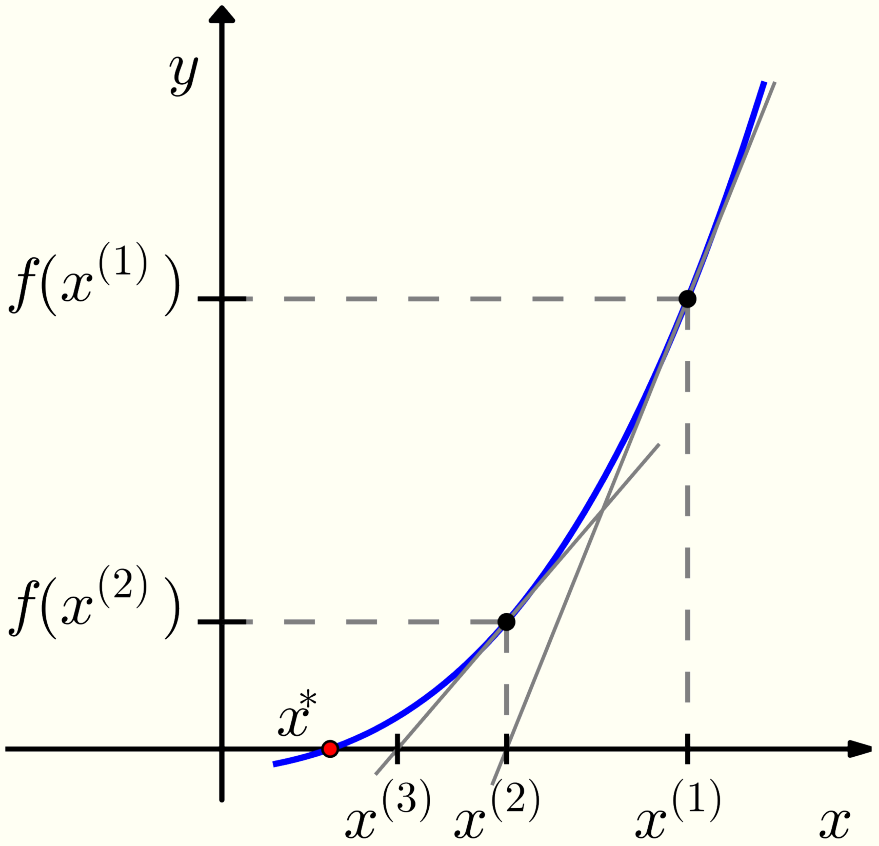


- Many small increments are need to obtain accurate solution
- Need to perform a parametric study to find the optimum incremental
- Defining increments is very important
- Program - SAGE CRISP

In the incremental solution we divide the load into increments $\Delta P = \lambda P$, where λ is also known as a load factor and apply a repeated solution of $\Delta u = K^{-1} \Delta P$.

Basically we divide the load into substeps, and treat each as linear - but that is usually not accurate enough and inefficient.

Newton Raphson method



$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Newton Raphson method

The incremental loading is expressed as follows. The external load vector F_{ext} is gradually increased from 0 in order to reach a desired value $\mathbf{F}^*$. Assuming that $\mathbf{F}^*$ itself remains constant during the analysis in terms of its 'direction' and only its magnitude is changing, we can write $F_{\text{ext}} = q$ known just to simplify our expression for the system of equations. Then we can control how the external load vector increases or decreases by introducing a scalar quantity λ and express the system as follows:

$$R(u) = F_{\text{int}} - F_{\text{ext}} \rightarrow R(u) = F_{\text{int}} - \lambda F_{\text{ext}} = 0$$

Thus by increasing or decreasing λ we can control our load. We are interested in u and λ. At every increment, we change slightly the value of λ and try to determine u satisfying $R(u) = 0$.

$$\Delta u = [K_T]_{u0}^{-1} \cdot (\Delta \lambda q)$$

Newton Raphson method

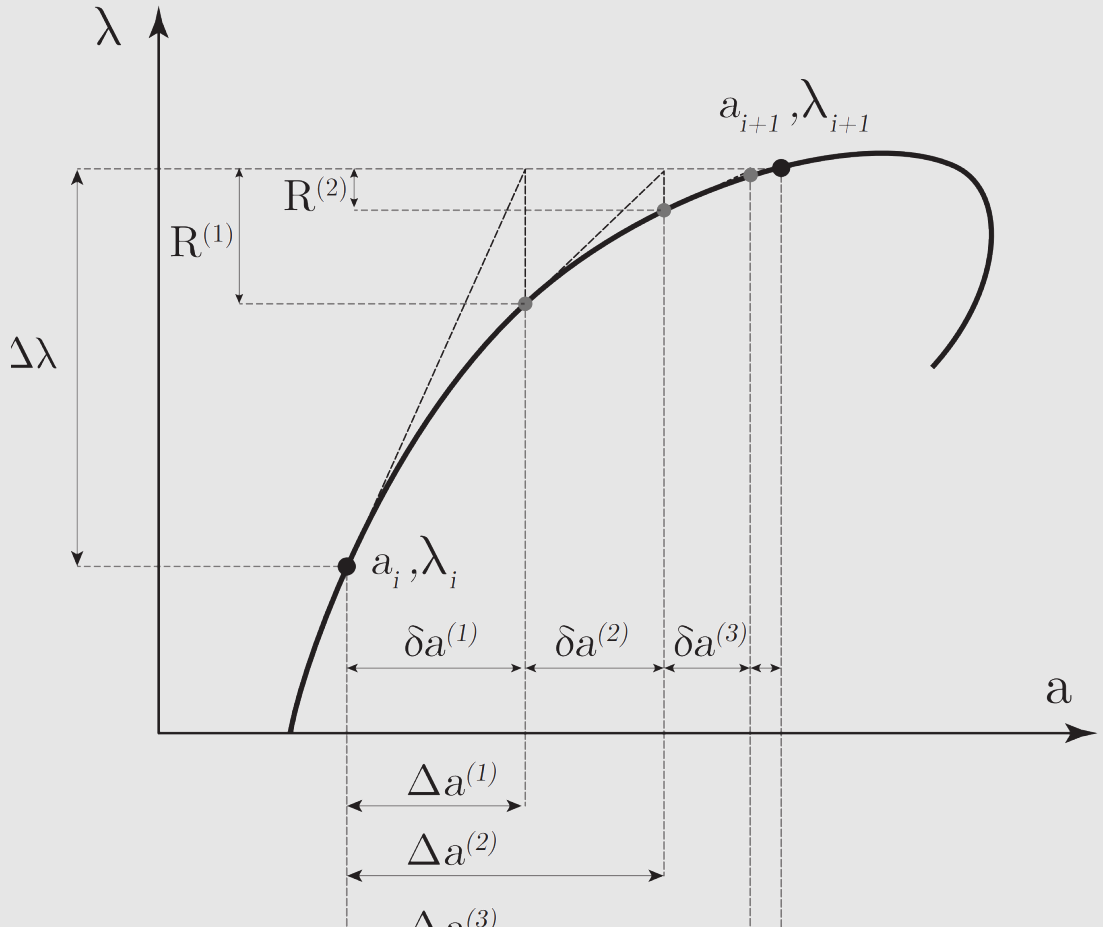
The incremental loading is expressed as follows. The external load vector F_{ext} is gradually increased from 0 in order to reach a desired value $\mathbf{F}^*$. Assuming that $\mathbf{F}^*$ itself remains constant during the analysis in terms of its 'direction' and only its magnitude is changing, we can write $F_{\text{ext}} = q$ known just to simplify our expression for the system of equations. Then we can control how the external load vector increases or decreases by introducing a scalar quantity λ and express the system as follows:

$$R(u) = F_{\text{int}} - F_{\text{ext}} \rightarrow R(u) = F_{\text{int}} - \lambda F_{\text{ext}} = 0$$

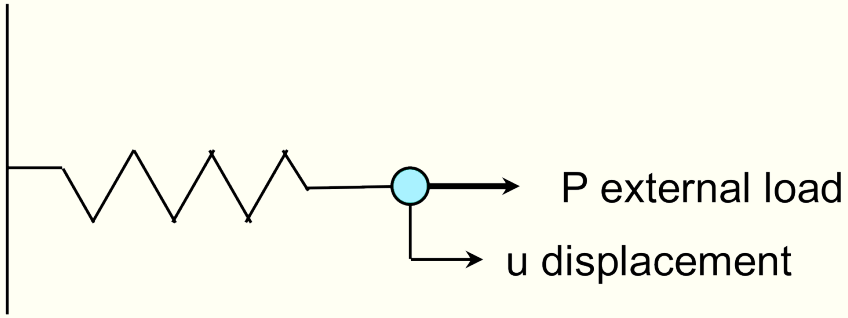
Thus by increasing or decreasing λ we can control our load. We are interested in u and λ. At every increment, we change slightly the value of λ and try to determine u satisfying $R(u) = 0$.

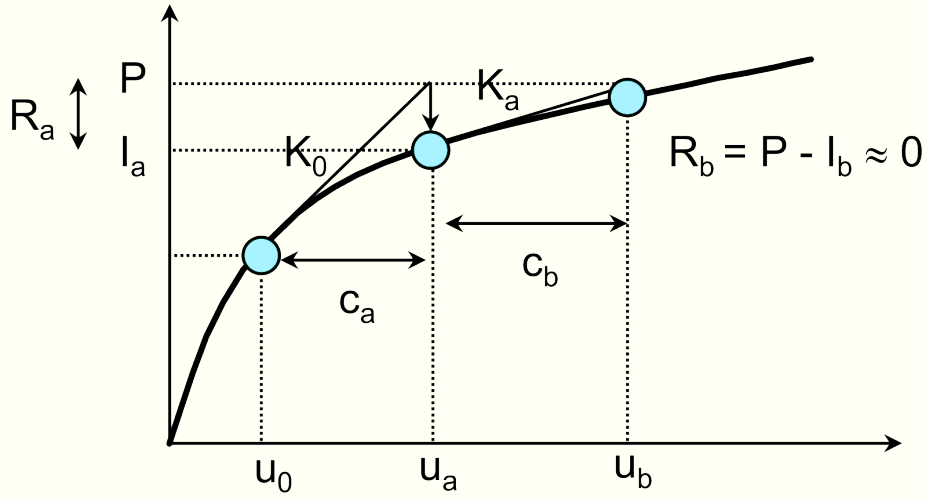
$$\Delta u = [K_T]_{u0}^{-1} \cdot (\Delta \lambda q)$$

where $\left[K^T\right] = [\partial F(u)/\partial u]$ is the "Jacobian" matrix of the system of equations and is commonly referred to as the Stiffness Matrix.

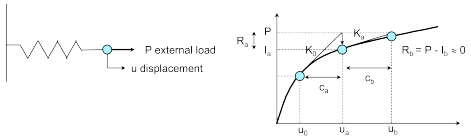


Newton Raphson method





- Using the initial stiffness K_0, apply an increment of load δP, calculate an approximate solution c_a caused by this increment.
- The stiffness K_a is updated using the new position, and the internal force in the spring I_a is calculated.
- If the difference R_a between the total load applied to the spring, P, and I_a is smaller than the tolerance, $u_a = u_0 + c_a$ is the converged solution.



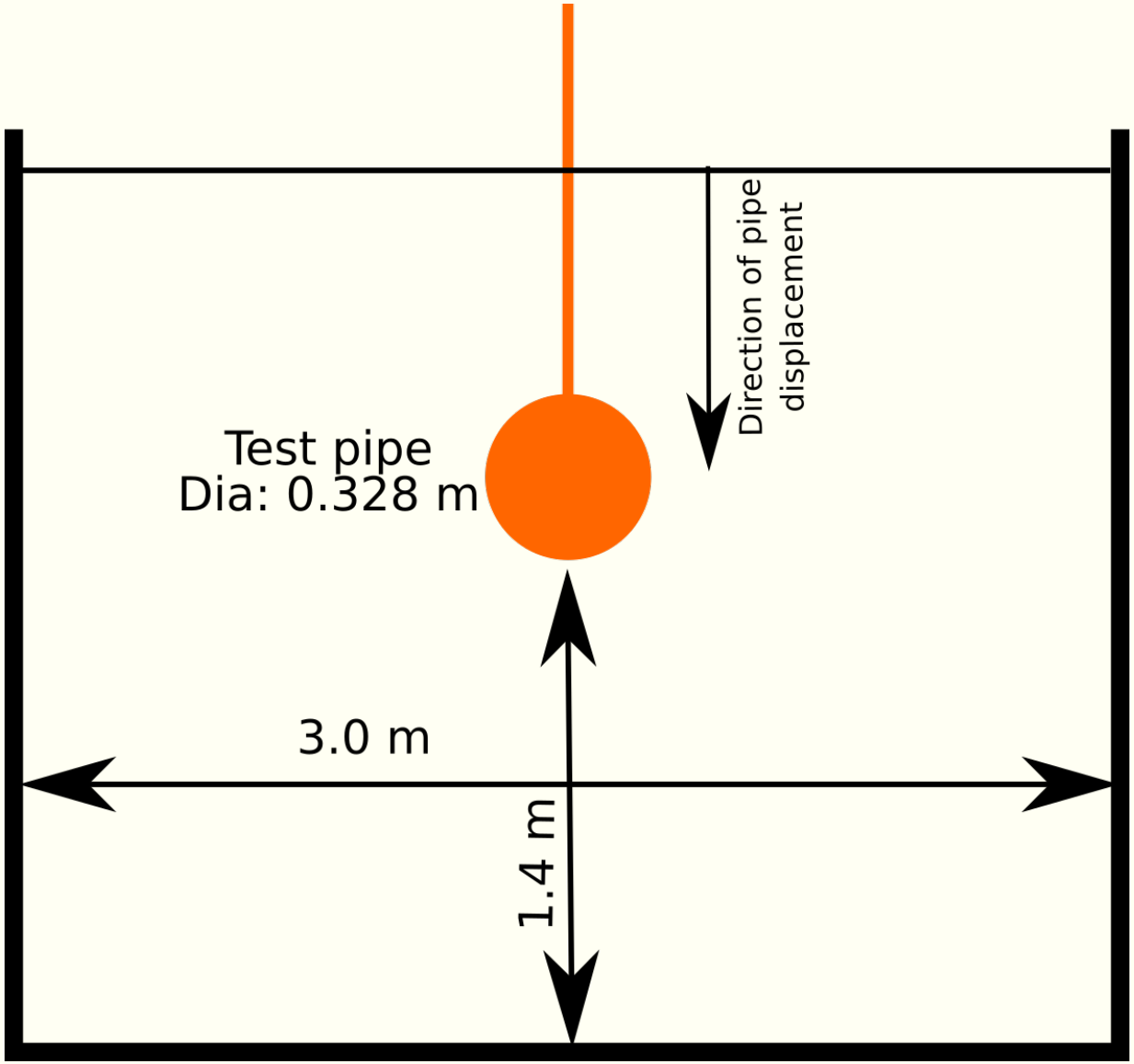
- Using the initial stiffness K_0, apply an increment of load δP, calculate an approximate solution c_a caused by this increment.
- The stiffness K_a is updated using the new position, and the internal force in the spring I_a is calculated.
- If the difference R_a between the total load applied to the spring, P, and I_a is smaller than the tolerance, $u_a = u_0 + c_a$ is the converged solution.

- If R_a is not small, a new displacement correction c_b is calculated by solving $c_b = R_a/K_a$
- The new displacement u_b is updated, and the internal force I_b in the updated configuration is calculated.
- The new force residual R_b is obtained. If $R_b < \textit{tolerance}$, the solution is converged. If not, continue the iteration.

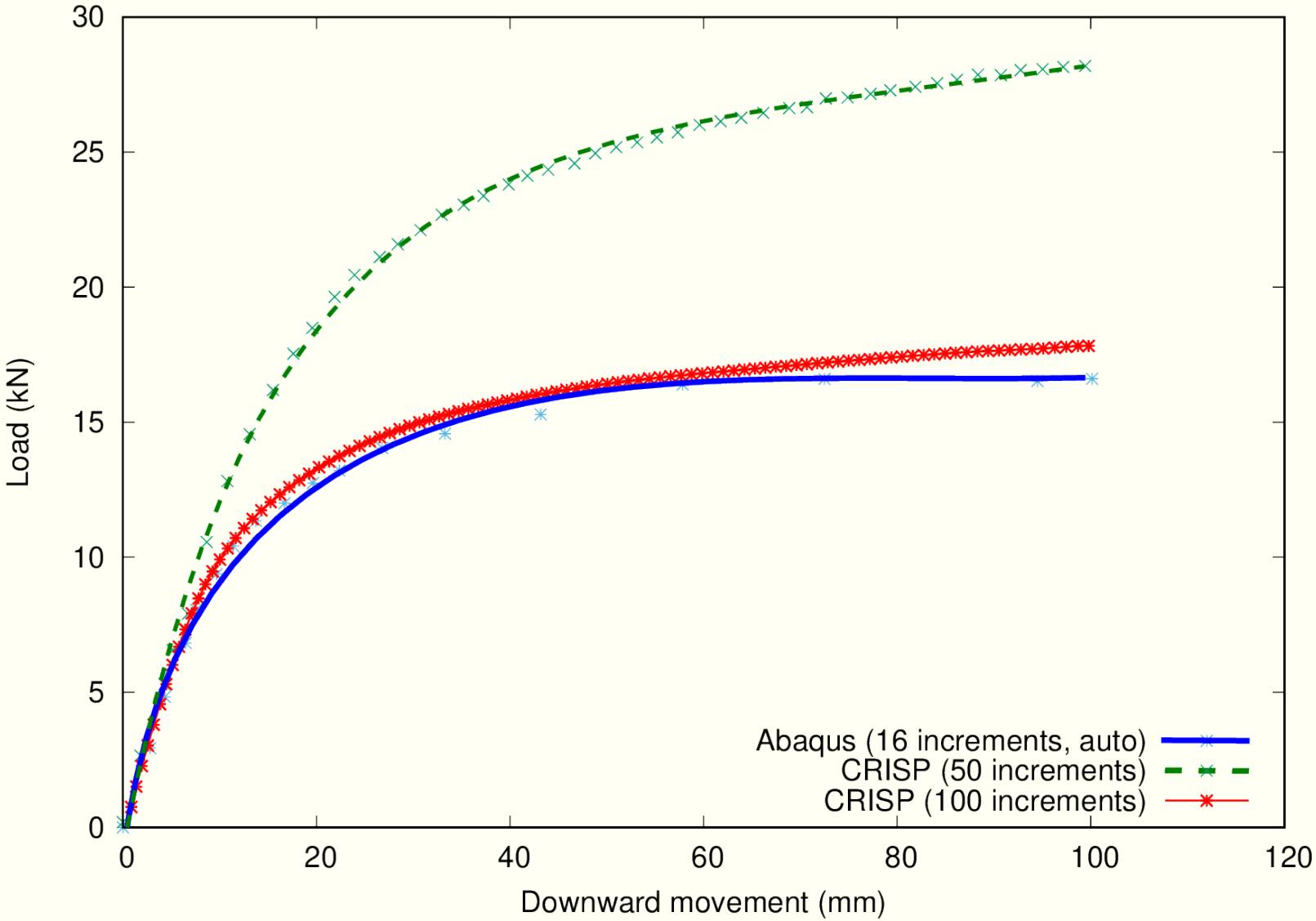
Newton Raphson method: Tolerance and Convergence

- Program - ABAQUS
- Newton-Raphson is the most standard method to solve nonlinear problems in FE.
- A large error in the initial estimate can contribute to non-convergence of the algorithm.
- **Tolerance**
  - must be small enough to ensure that the approximate solution is close to the exact mathematical solution.
  - must be large enough so that reasonable number of iterations are performed.
- **Quadratic convergence**
  - If the tangent stiffness is calculated correctly, R should reduce quadratically from one iteration to the next.

Tangent-Stiffness vs Newton Raphson



Tangent-Stiffness vs Newton Raphson



Newton Raphson iterative procedure: FE example

In Newton's method:

$$R \equiv [K]u - F = 0$$

where **R** is the residual vector, we expand **R** in Taylor's series:

$$\mathbf{R}(\mathbf{u^{r+1}}) = \mathbf{R}(\mathbf{u^r}) + \left(\frac{\partial \mathbf{R}}{\partial \mathbf{u}}\right)^{(r)} \cdot \delta\mathbf{u} + \ldots$$

Omitting higher order terms (2):

$$\left(\frac{\partial \mathbf{R}}{\partial \mathbf{u}}\right)^{(r)} \cdot \delta\mathbf{u} = -\mathbf{R}(\mathbf{u^r})$$

$$[\mathbf{T}(\mathbf{u})^{\mathbf{r}}]\,\delta\mathbf{u} = -\mathbf{R}(\mathbf{u^r})$$

where $[T]$ is called the *tangent matrix*

$$[\mathbf{T}(\mathbf{u})^{\mathbf{r}}] \equiv \left(\frac{\partial \mathbf{R}}{\partial \mathbf{u}}\right)^{(r)}$$

Newton Raphson iterative procedure: FE example

The terms of the tangent stiffness matrix are given by

$$\mathbf{T} := \mathbf{K} + \frac{\partial \mathbf{K}}{\partial \mathbf{u}}\,\mathbf{u} \qquad \rightarrow \qquad T_{ij} := K_{ij} + \sum_{m=1}^{n} \frac{\partial K_{im}}{\partial u_j} u_m \ .$$

For our 2×2 reduced stiffness matrix, we have

$$\begin{aligned}
T_{22} &= K_{22} + \frac{\partial K_{22}}{\partial u_2} u_2 + \frac{\partial K_{23}}{\partial u_2} u_3 \\
T_{23} &= K_{23} + \frac{\partial K_{22}}{\partial u_3} u_2 + \frac{\partial K_{23}}{\partial u_3} u_3 \\
T_{32} &= K_{32} + \frac{\partial K_{32}}{\partial u_2} u_2 + \frac{\partial K_{33}}{\partial u_2} u_3 \\
T_{33} &= K_{33} + \frac{\partial K_{32}}{\partial u_3} u_2 + \frac{\partial K_{33}}{\partial u_3} u_3 \ .
\end{aligned}$$

Newton Raphson iterative procedure: FE example

The component definition of the tangent matrix at the element level is:

$$[\mathbf{T}(\mathbf{u})^{\mathbf{r}}] = K + \frac{\partial K}{\partial u} \rightarrow \quad T_{ij} = K_{ij} + \sum_{m=1}^{n} \frac{\partial K_{im}}{\partial u_j} u_m$$

The residual vector after r iteration is:

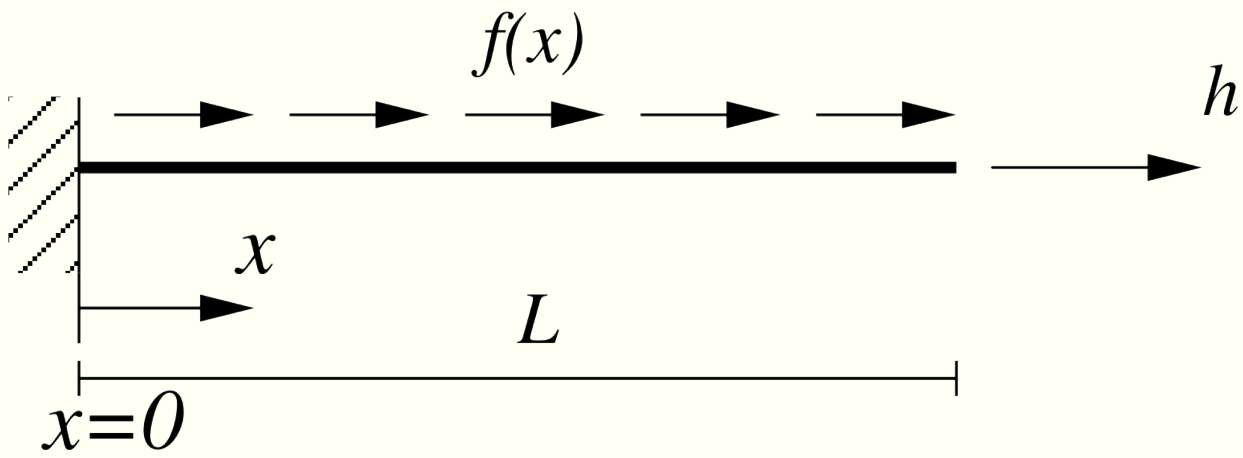
$$R(u^{(r)}) = [K(u^r)]u^r - F$$

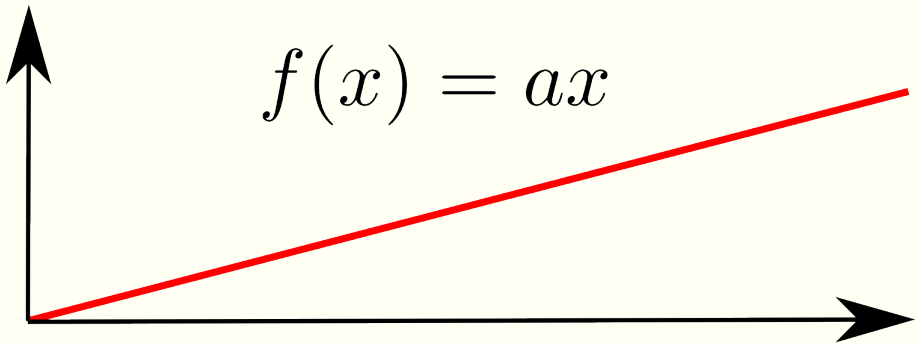
The solution at $r+1$:

$$u^{(r+1)} = u^r + \delta u$$

$$\delta \mathbf{u} = [\mathbf{T}(\mathbf{u})^{\mathbf{r}}]^{-1} * \mathbf{R}(\mathbf{u}^{\mathbf{r}})$$

Newton Raphson iterative procedure: FE example





The constitutive equation for this problem: $\sigma = E_0(1+\varepsilon)\varepsilon$

Newton Raphson iterative procedure: FE example

For the axial loading of a nonlinear bar:

$$-\frac{dN}{dx} = f$$

$$N = g(\frac{du}{dx})$$

$$N = AE\left(1 + \frac{du}{dx}\right)\frac{du}{dx}$$

The weakform:

$$-\frac{d}{dx}\left(AE\left(1 + \frac{du}{dx}\right)\frac{du}{dx}\right) = f$$

Newton Raphson iterative procedure: FE example

Multiplying the weak form by a weight function:

$$-\int_0^L w\frac{d}{dx}\left(AE\left(1+\frac{du}{dx}\right)\frac{du}{dx}\right)dx=\int_0^L wfdx$$

Applying boundary condition (Dirichlet condition of $u(0)=0$) and doing integration by parts:

$$-\int_0^L AE\left(1+\frac{du}{dx}\right)\frac{du}{dx}\frac{dw}{dx}dx=\int_0^L wfdx$$

Newton Raphson iterative procedure: FE example

The element stiffness matrix has the form

$$\mathbf{K}^e = \frac{AE_0}{h^2}(h + u_1 - u_2)\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The tangent stiffness matrix is given by

$$\mathbf{T}^e = \mathbf{K}^e + \frac{\partial \mathbf{K}^e}{\partial \mathbf{u}}\,\mathbf{u}\,.$$

After going through the algebra, we get

$$\mathbf{T}^e = \frac{AE_0}{h^2}(h + 2u_1 - 2u_2)\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

For the two element mesh, the assembled global tangent stiffness matrix is

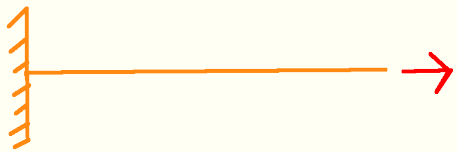
$$\mathbf{T} = \frac{AE_0}{h^2}\begin{bmatrix} h + 2(u_1 - u_2) & -h - 2(u_1 - u_2) & 0 \\ & 2h + 2(u_1 - u_3) & -h - 2(u_2 - u_3) \\ \text{Symm.} & & h + 2(u_2 - u_3) \end{bmatrix}$$

Newton Raphson iterative procedure: FE example

Let $A = 0.01$ m^2, $L = 2$ m, $E_0 = 100$ MPa, $R = 100$ kN. Since there are two elements, $h = 1$ m.

For the first Newton iteration, let the initial guess be

$$\mathbf{u}_0 = \begin{bmatrix} u_2^0 \\ u_3^0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



Then, the first Newton iteration gives

$$\mathbf{K}_0 = 10^6 \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}, \qquad \mathbf{T}_0 = 10^6 \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}, \qquad \mathbf{u}_1 = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}.$$

The second Newton iteration gives

$$\mathbf{K}_1 = 10^6 \begin{bmatrix} 1.8 & -0.9 \\ -0.9 & 0.9 \end{bmatrix}, \qquad \mathbf{T}_1 = 10^6 \begin{bmatrix} 1.6 & -0.8 \\ -0.8 & 0.8 \end{bmatrix}, \qquad \mathbf{u}_2 = \begin{bmatrix} 0.1125 \\ 0.2250 \end{bmatrix}.$$

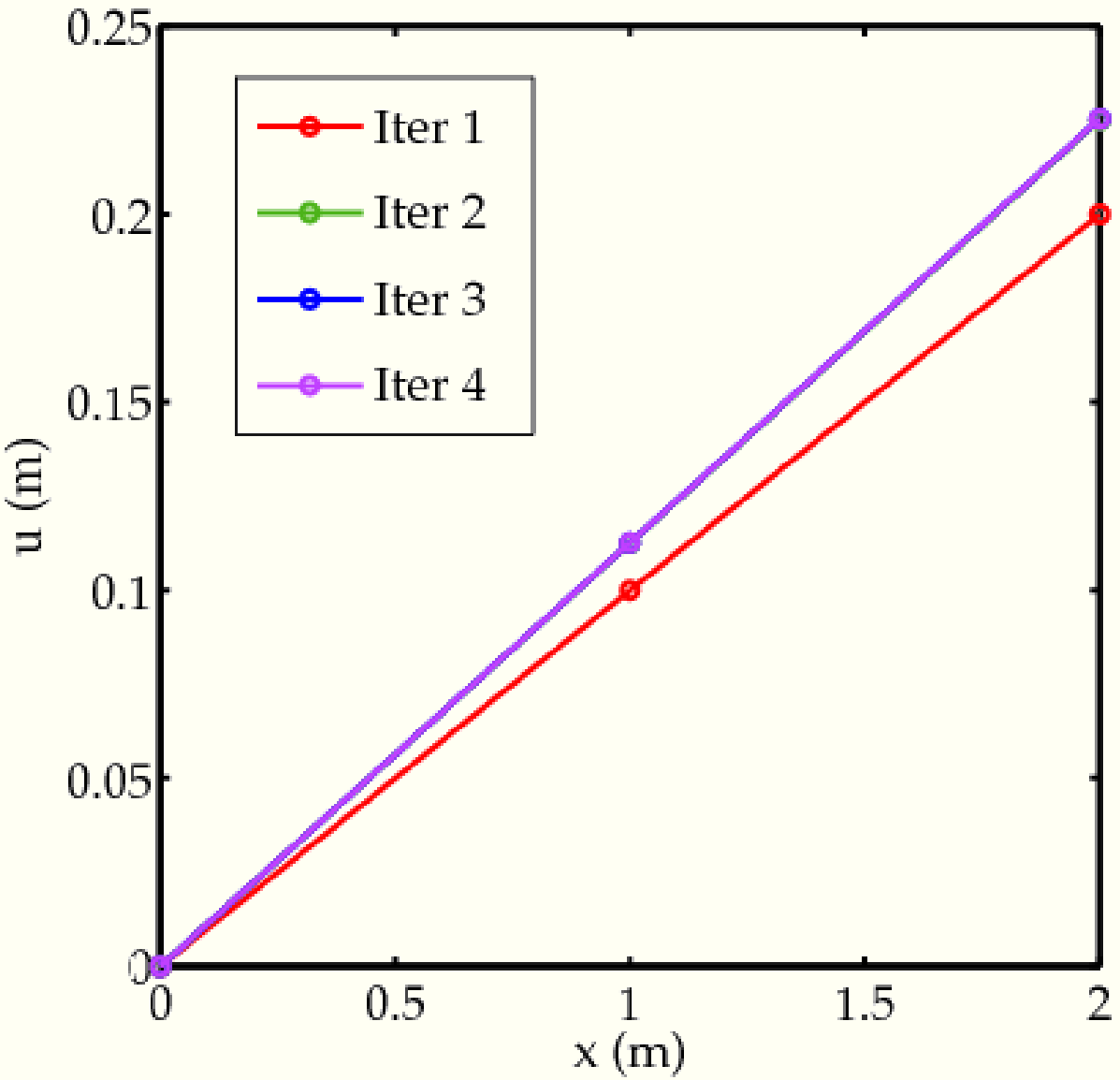
The third Newton iteration gives

$$\mathbf{K}_2 = 10^6 \begin{bmatrix} 1.775 & -0.8875 \\ -0.8875 & 0.8875 \end{bmatrix}, \qquad \mathbf{T}_2 = 10^6 \begin{bmatrix} 1.55 & -0.775 \\ -0.775 & 0.775 \end{bmatrix}, \qquad \mathbf{u}_3 = \begin{bmatrix} 0.1127 \\ 0.2254 \end{bmatrix}.$$

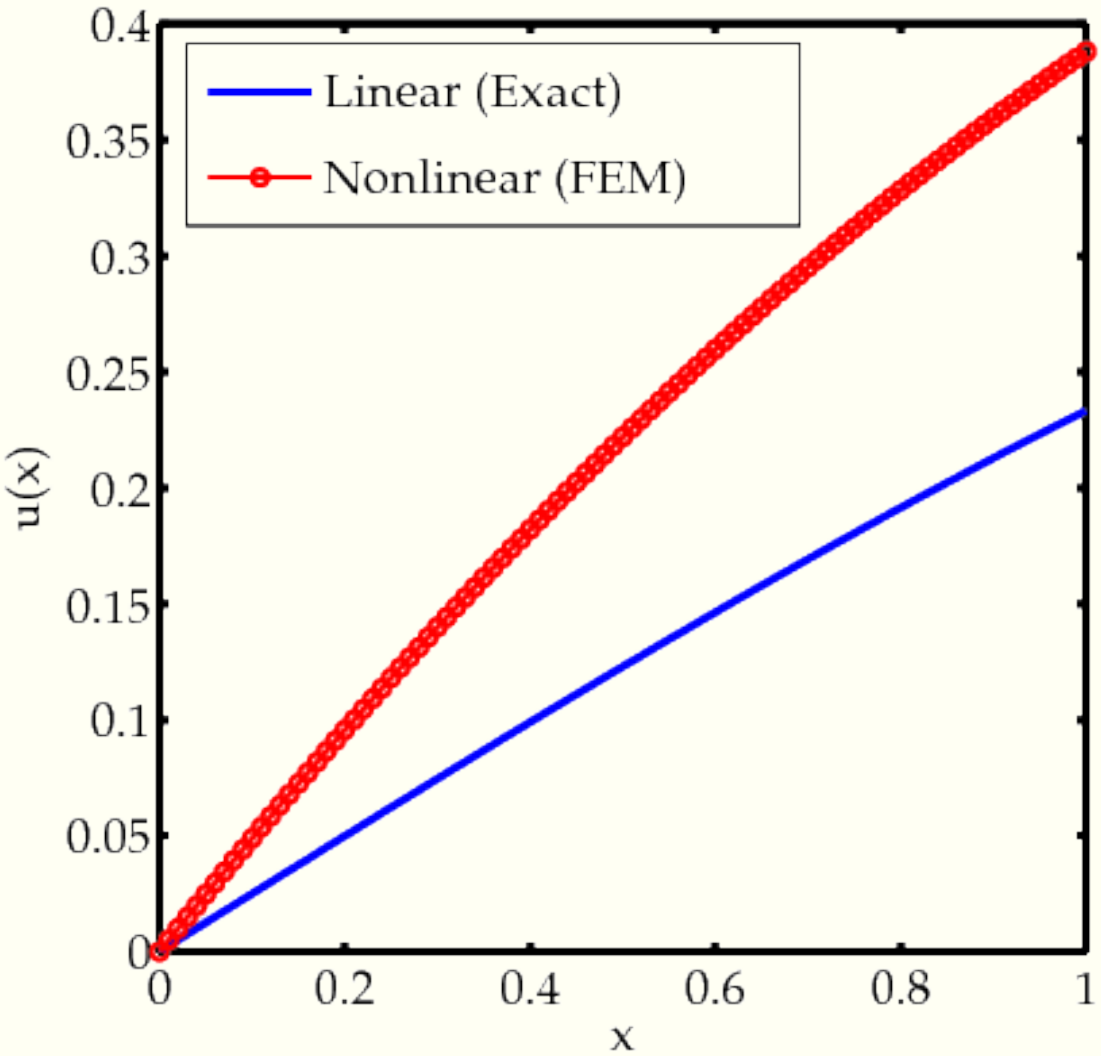
The fourth Newton iteration gives

$$\mathbf{K}_3 = 10^6 \begin{bmatrix} 1.7746 & -0.8873 \\ -0.8873 & 0.8873 \end{bmatrix}, \qquad \mathbf{T}_3 = 10^6 \begin{bmatrix} 1.5492 & -0.7746 \\ -0.7746 & 0.7746 \end{bmatrix}, \qquad \mathbf{u}_4 = \begin{bmatrix} 0.1127 \\ 0.2254 \end{bmatrix}.$$

Newton Raphson iterative procedure: FE example

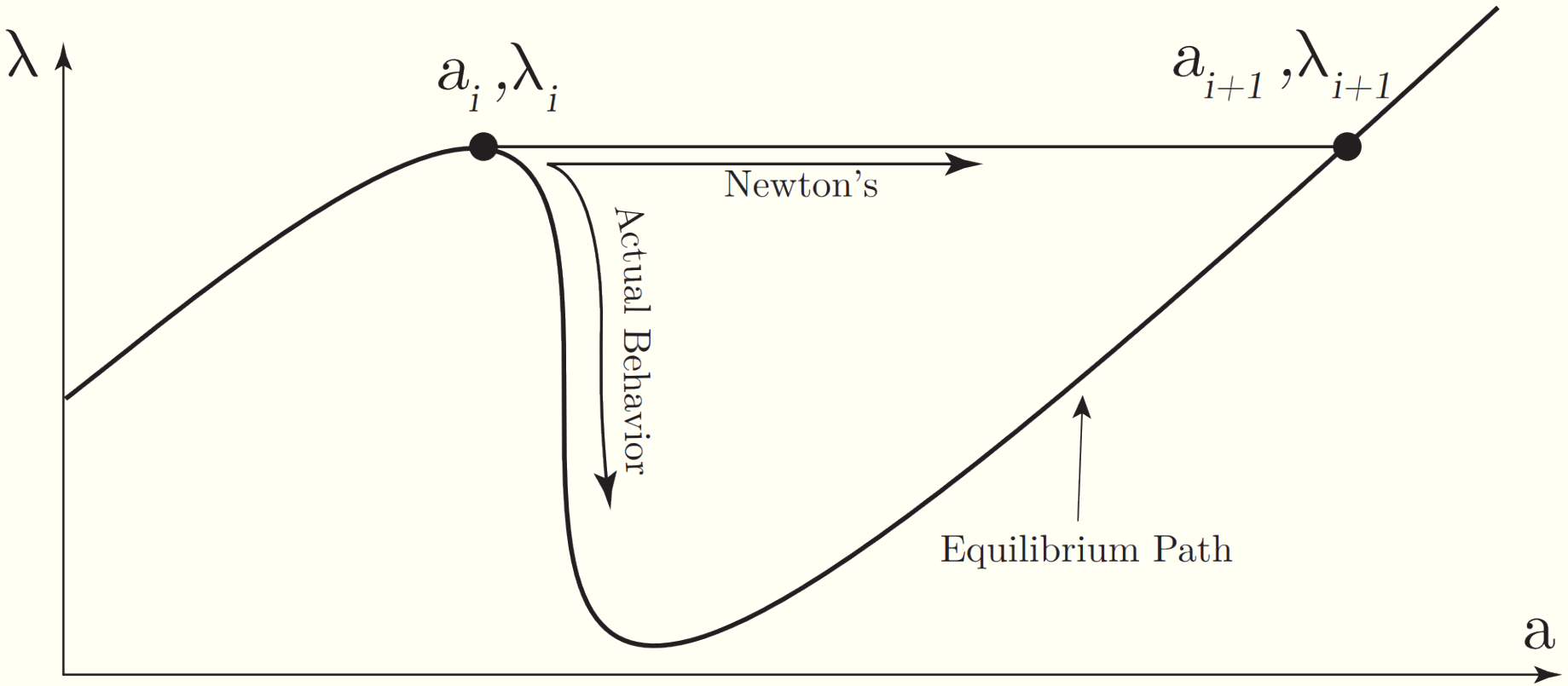


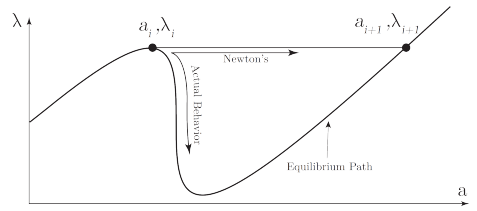
Newton Raphson iterative procedure: FE example



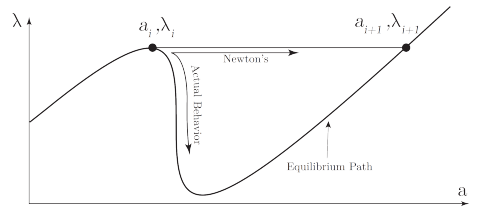
FEM displacement solution for nonlinear bar under distributed axial load. We have used 100 linear elements in these calculations.

Newton Raphson: Drawbacks

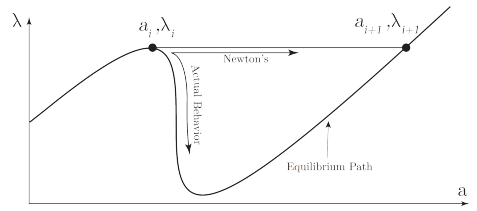




Newton's method fails to accurately follow the 'equilibrium' path once the tangent stiffness reaches zero. That happens due to the formulation of Newton's method, and in particular that it restricts the parameter λ to change monotonically every increment 3 . The definition of a limit point then (saddle points excluded), suggests that in order to remain on the equilibrium path you need to change your loading pattern depending on whether the limit point is a local maximum or maximum in the $u-\lambda$ space.

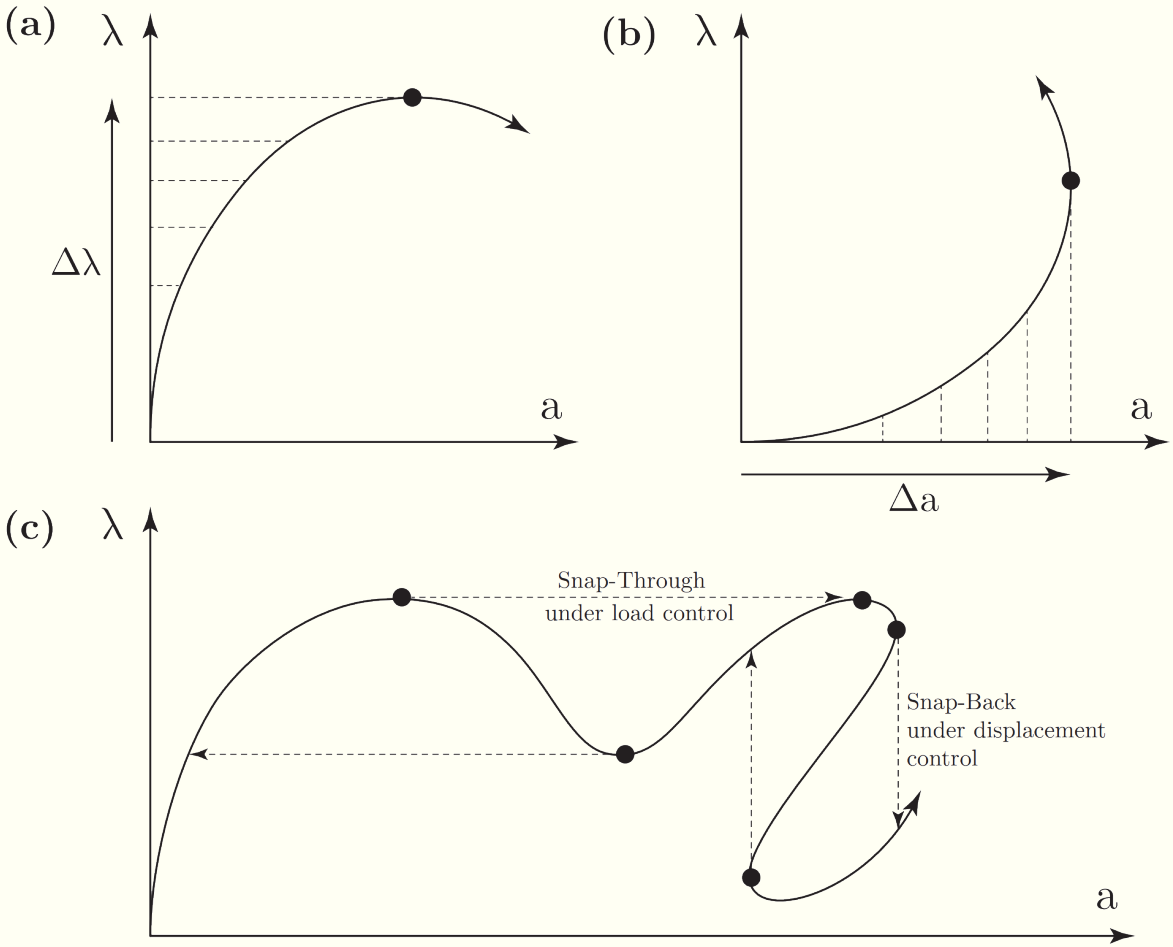


In terms of mechanical systems then, this method is able to solve any non-linear system of equations very efficiently but only up to the critical point (if any). In the case shown in figure, Newton's method fails in load–control. Now in many cases, one way to circumvent problems like these is to use 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. The main problem is, that in most actual applications, the structural response, and therefore the equilibrium path, for the structure under consideration is unknown, and therefore one does not know what type of behavior to expect.



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. Furthermore, very often, strong nonlinearities that arise in finite deformation problems may eventually lead to such behaviors and thus it is necessary to introduce a numerical technique to solve such problem with strongly nonlinear behaviors.

Newton Raphson: Drawbacks



(a) A system that is unstable under load control (Snap–Through instability), (b) A system that is unstable under displacement control (Snap–Back Instability), (c) A system that is unstable under both displacement and load control

Arc length method

System of the non-linear (in general) equations to solve:

$$F_{\text{int}} - F_{\text{ext}} = 0 \rightarrow F_{\text{int}} - \lambda q = 0$$

Suppose (u_0, λ_0) satisfy the system of equations and thus belongs to the 'equilibrium path'. Arc Length postulates a simultaneous variation in both the displacements Δu and the load vector coefficient $\Delta\lambda$.

$$R(u', \lambda') = F_{\text{int}}(u_0 + \Delta u) - (\lambda_0 + \Delta\lambda)q = 0$$

If the above is satisfied for $(u_0 + \Delta u, \lambda_0 + \Delta\lambda)$ then this point also belongs to the 'equilibrium path' and we can successfully update the solution. However, immediate satisfaction is not possible, so we need to provide the necessary corrections $(\delta u, \delta\lambda)$, aiming that the new point $(u_0 + \Delta u + \delta u, \lambda_0 + \Delta\lambda + \delta\lambda)$ will satisfy.

$$R(u'', \lambda'') = F_{\text{int}}(u_0 + \Delta u + \delta u) - (\lambda_0 + \Delta\lambda + \delta\lambda)q = 0$$

Arc length method

System of the non-linear (in general) equations to solve:

$$F_{\text{int}} - F_{\text{ext}} = 0 \rightarrow F_{\text{int}} - \lambda q = 0$$

Suppose (u_0, λ_0) satisfy the system of equations and thus belongs to the 'equilibrium path'. Arc Length postulates a simultaneous variation in both the displacements Δu and the load vector coefficient $\Delta\lambda$.

$$R(u', \lambda') = F_{\text{int}}(u_0 + \Delta u) - (\lambda_0 + \Delta\lambda)q = 0$$

If the above is satisfied for $(u_0 + \Delta u, \lambda_0 + \Delta\lambda)$ then this point also belongs to the 'equilibrium path' and we can successfully update the solution. However, immediate satisfaction is not possible, so we need to provide the necessary corrections $(\delta u, \delta\lambda)$, aiming that the new point $(u_0 + \Delta u + \delta u, \lambda_0 + \Delta\lambda + \delta\lambda)$ will satisfy.

$$R(u'', \lambda'') = F_{\text{int}}(u_0 + \Delta u + \delta u) - (\lambda_0 + \Delta\lambda + \delta\lambda)q = 0$$

Arc length is a very efficient method in solving non-linear systems of equations when the problem under consideration exhibits one or more critical points. In terms of a simple mechanical loading-unloading problem, a critical point could be interpreted as the point at which the loaded body cannot support an increase of the external forces and an instability occurs.

Unlike the Newton-Method, the Arc Length method postulates a simultaneous variation in both the displacements Δu and the load vector coefficient $\Delta\lambda$. The main difference is that both Δu and $\Delta\lambda$ are unknowns in contrast to Newton's method where $\Delta\lambda$ was given and we had to iteratively solve for Δu.

Arc length method

System of the non-linear (in general) equations to solve:

$$F_{\text{int}} - F_{\text{ext}} = 0 \rightarrow F_{\text{int}} - \lambda q = 0$$

Suppose (u_0, λ_0) satisfy the system of equations and thus belongs to the 'equilibrium path'. Arc Length postulates a simultaneous variation in both the displacements Δu and the load vector coefficient $\Delta\lambda$.

$$R(u', \lambda') = F_{\text{int}}(u_0 + \Delta u) - (\lambda_0 + \Delta\lambda)q = 0$$

If the above is satisfied for $(u_0 + \Delta u, \lambda_0 + \Delta\lambda)$ then this point also belongs to the 'equilibrium path' and we can successfully update the solution. However, immediate satisfaction is not possible, so we need to provide the necessary corrections $(\delta u, \delta\lambda)$, aiming that the new point $(u_0 + \Delta u + \delta u, \lambda_0 + \Delta\lambda + \delta\lambda)$ will satisfy.

$$R(u'', \lambda'') = F_{\text{int}}(u_0 + \Delta u + \delta u) - (\lambda_0 + \Delta\lambda + \delta\lambda)q = 0$$

The system of equations takes the form:

$$\left[K^T\right]_{u_0+\Delta u} \cdot \delta u - \delta\lambda q = -\left[F_{\text{int}}(u_0 + \Delta u) - (\lambda_0 + \Delta\lambda)q\right] = -R(u', \lambda')$$

δu and $\delta lambda$ are the unknowns for whom we need to solve. If the u vector however, has dimensions $N \times 1$ then we have a total of N equations that we need to solve for $N + 1$ unknowns (N unknowns δu and 1 unknown $\delta\lambda$). Equations then are not sufficient to determine $\delta u, \delta\lambda$. The supplementary equation that completes the system is called the Arc Length Equation.

Arc length method

The Arc length equation:

$$(\Delta \mathbf{u} + \delta \mathbf{u})^T \cdot (\Delta \mathbf{u} + \delta \mathbf{u}) + \psi^2 (\Delta \lambda + \delta \lambda)^2 (\mathbf{q}^T \cdot \mathbf{q}) = \Delta l^2$$

where ψ and Δl are user defined parameters. In a sense Δl defines how far to search for the next equilibrium point and it is analogous (but not directly equivalent) to the load increment $\Delta \lambda$ we used in Newton's method.

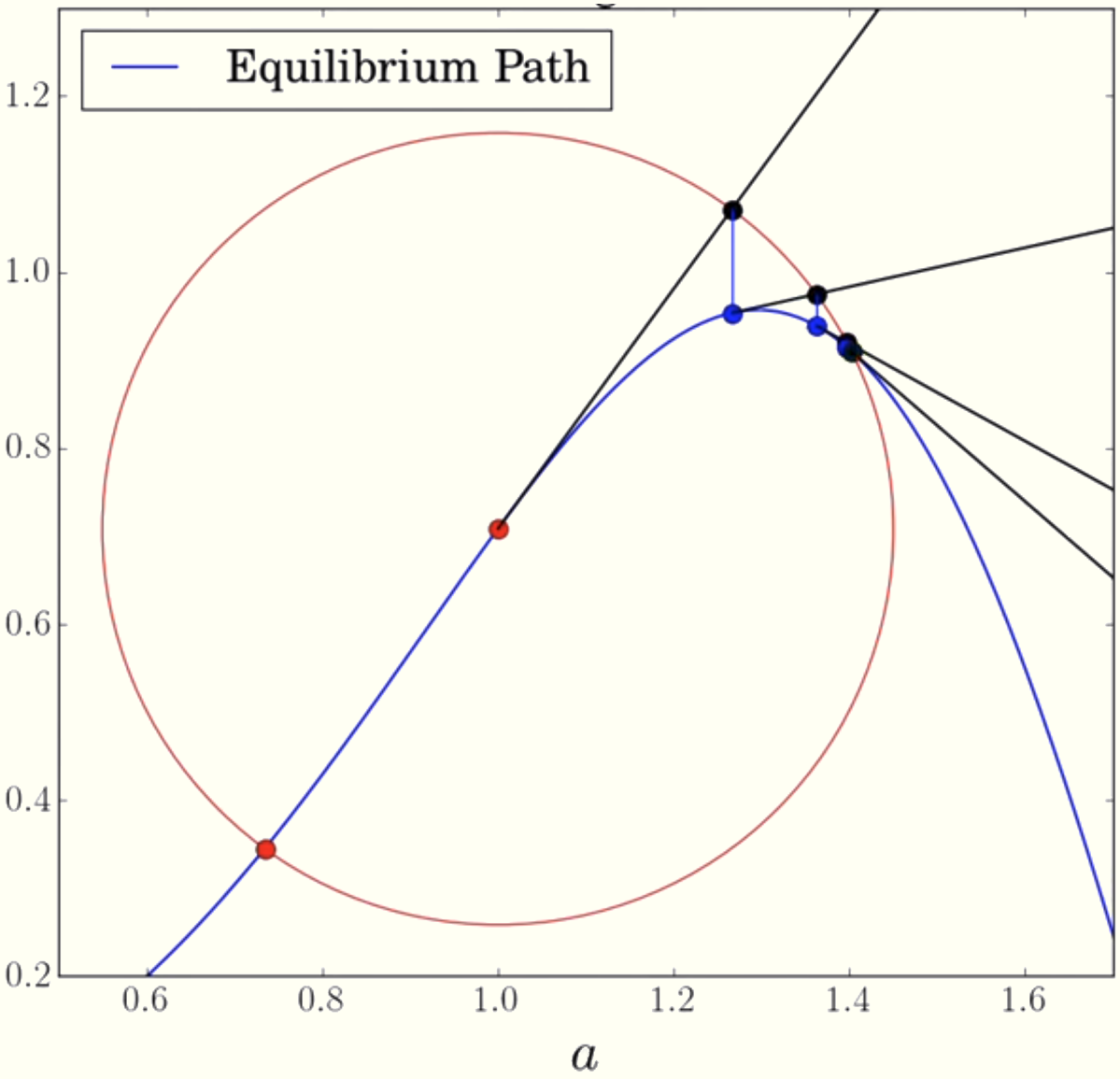
The Arc length equation:

$$(\Delta\mathbf{u}+\delta\mathbf{u})^T\cdot(\Delta\mathbf{u}+\delta\mathbf{u})+\psi^2(\Delta\lambda+\delta\lambda)^2(\mathbf{q}^T\cdot\mathbf{q})=\Delta l^2$$

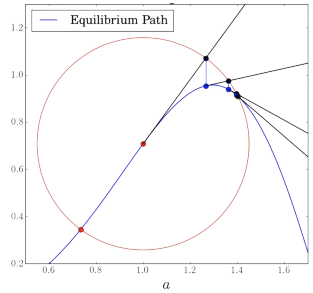
where ψ and Δl are user defined parameters. In a sense Δl defines how far to search for the next equilibrium point and it is analogous (but not directly equivalent) to the load increment $\Delta\lambda$ we used in Newton's method.

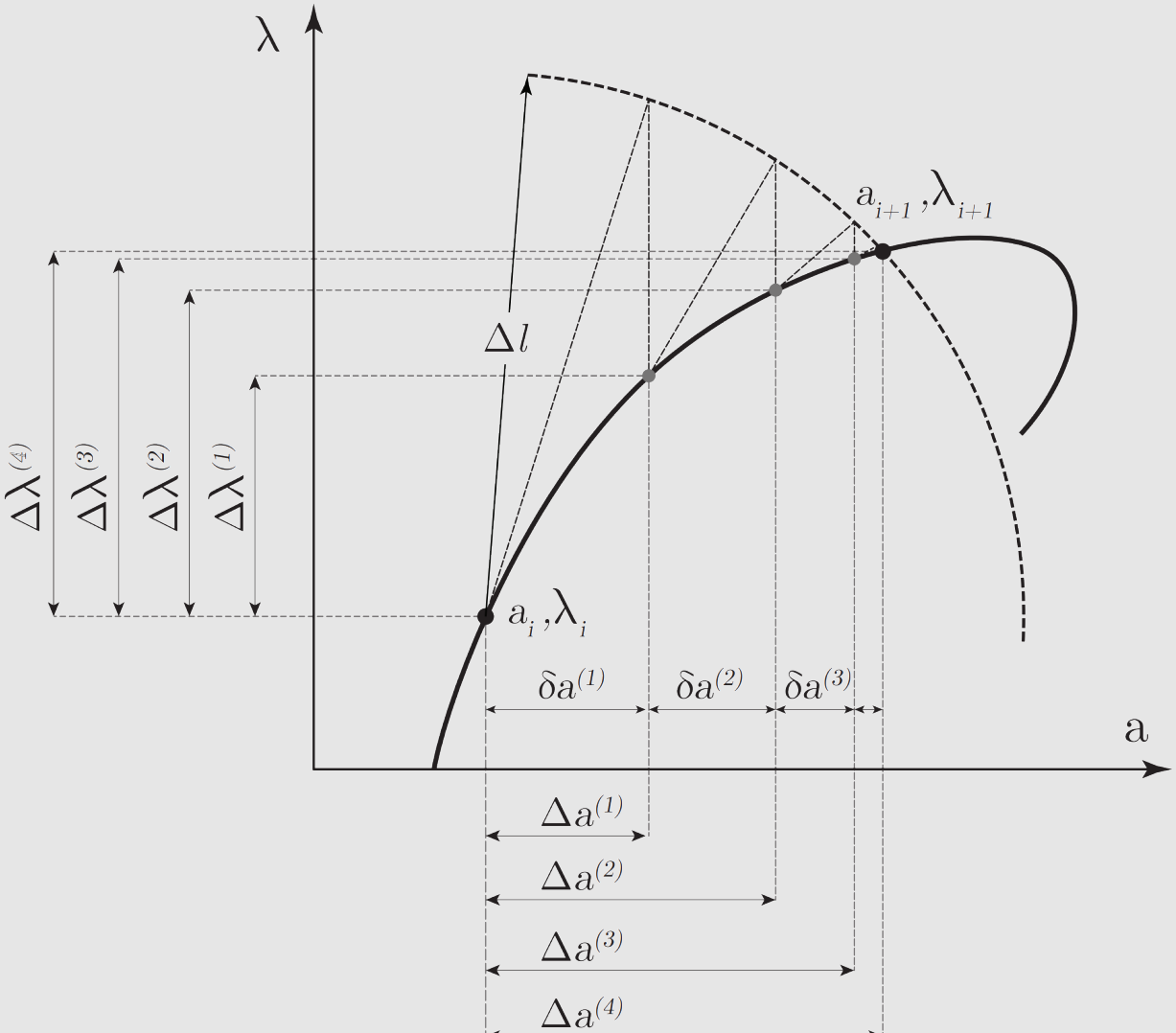
The system of equations is solved for $\delta u, \delta\lambda$ and updates the previous corrections $\Delta u, \Delta\lambda$ to be $\Delta u' = \Delta u + \delta u$ and $\Delta\lambda' = \Delta\lambda + \delta\lambda$ respectively. The method continues to provide such incremental corrections $\delta u, \delta\lambda$ until convergence is achieved. When $\psi = 1$, the method is also called the **Spherical Arc-Length Method** because the points $\Delta u', \Delta\lambda'$ belong to a circle with radius Δl. In its most general form for arbitrary ψ can be geometrically interpreted as a hyper-ellipse in the multidimensional displacement-load space $(u - \lambda)$.

Arc length method

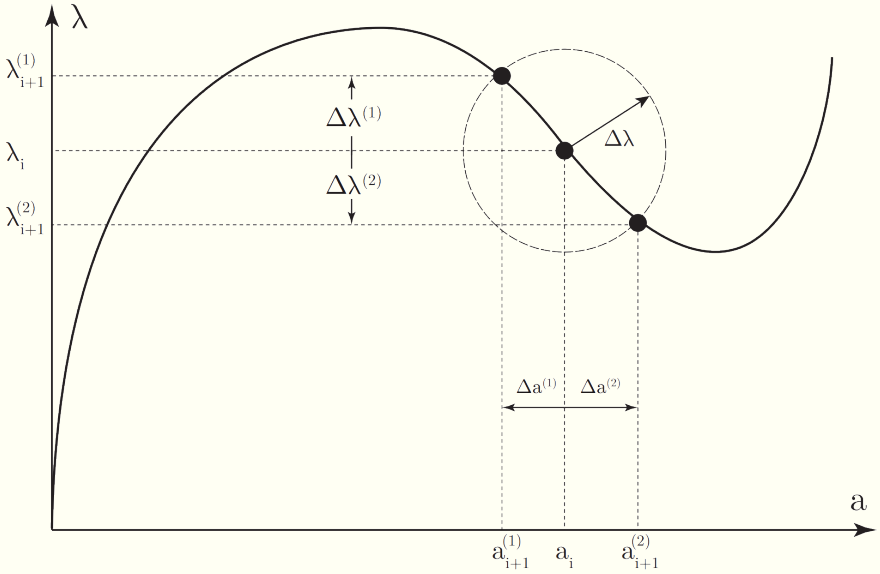


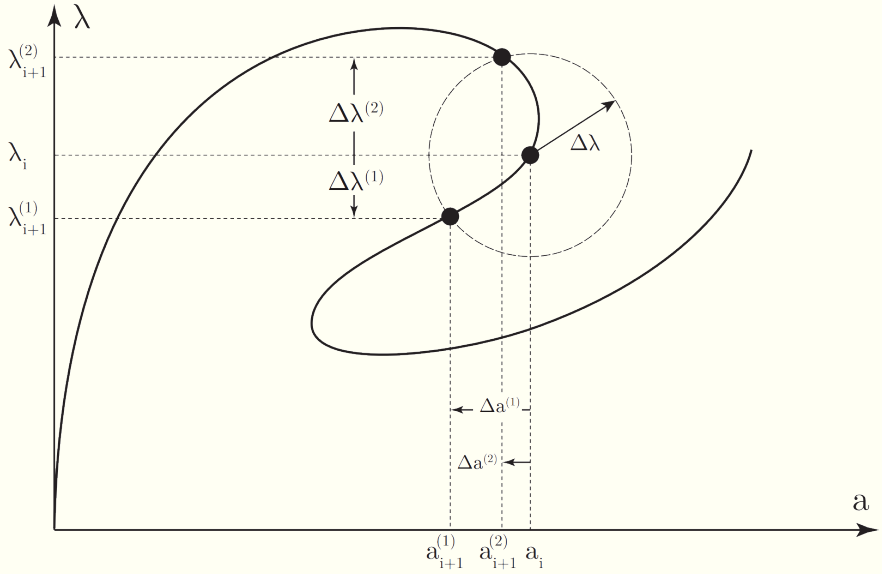
Arc length method

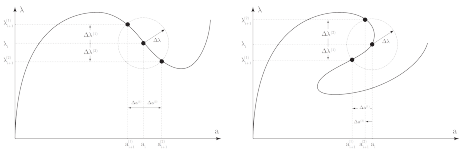




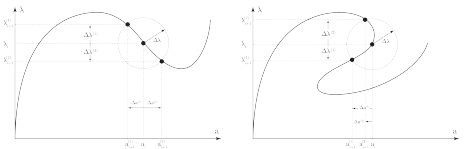
Arc length method: Drawbacks







Crisfield's [1983] implementation of Arc-length however, leads to one of the method's most important drawbacks. The quadratic equation would in general lead to two distinct solutions for $\delta\lambda$ which will in turn lead to two distinct solutions for δu. Thus, every iteration, the solver determined two sets of solutions, namely $(\delta u_1, \delta\lambda_1)$ and $(\delta u_1, \delta\lambda_1)$. This is no surprise since a circle (or a hyperellipse) would always intersect a curve in two points if its center is located on the curve.



The issue that arises then, is to develop a robust algorithm that would be able to accurately determine the correct set of $(\delta u, \delta \lambda)$ to update the solution. In general, we would like to choose the set, so that the solution 'evolves forwards'. This term 'forward evolution' is commonly used in the context of this method since choosing the wrong set would make the solution move back towards a previously converged point, and not in the desired (forward) direction. It is interesting to note, that an effective solution to this problem that works for all applications is yet to be found and as a result, many times programs like ABAQUS fail to converge to the correct solution or fail to 'evolve forwards'.

A priori estimates

- A priori error estimation is the analysis of the errors in a finite element simulation before actually performing an analysis.
- It is an abstract concept since it cannot quantify the error in a computed simulation, but analysis can tell something about whether the finite element solution will converge to the exact solution as the mesh is refined, and how quickly it will converge if the element type is changed or the mesh is refined.

Functional norms

Before considering the error, we need to decide how it should be measured. The question is how to measure a function? The answer to this is to use function norms. A common norm is known as the L^2 norm. For a function on a domain Ω, the L^2 norm is defined as:

$$\|u\|_0 = \left(\int_\Omega u^2 d\Omega \right)^{1/2}$$

This norm provides a measure of 'how big' a function is. A measure of the difference between two functions u and v is provided by:

$$\|u - v\|_0 = \left(\int_\Omega (u - v)^2 d\Omega \right)^{1/2}$$

It is only possible that $\|u - v\|_0 = 0$ if $u = v$.

Functional norms

The L^2 norm measures the 'magnitude' of a function, but does not reflect any details of the derivatives. Other norms are the H^1 norm:

$$\|u\|_0 = \left(\int_\Omega u^2 + \nabla u \cdot \nabla u d\Omega \right)^{1/2}$$

We will use function norms to measure the difference between the exact solution and the finite element solution. For example, we can define a particular measure of the error e as:

$$\|u - u_h\|_0 = \left(\int_\Omega (u - u_h)^2 d\Omega \right)^{1/2}$$

where u is the exact solution and u_h is the finite element solution.

How good is the FE solution?

Inevitably, the finite element solution is not exact. But how do we know that it bears any relation to the exact solution? If the mesh is refined, will the error reduce? We may say intuitively yes, but this is not very satisfactory. This is where a priori analysis helps. A priory error estimate:

$$\|u - u_h\|_s \leq Ch^\beta \|u\|_{k+1}$$

- s is the norm of interest,
- k is the polynomial order of the shape functions (complete polynomials),
- $\beta = min(k + 1 - s, 2(k + 1 - m))$,
- m is the highest order derivative appearing in the weak form
- C is an unknown constant that does not depend on h or the exact solution u.

What is of key interest is the exponent β.

Error in the temperature/displacement field

For a steady heat conduction or elasticity problem, for the error in the temperature/displacement field:

- $s = 0$
- $m = 1$

Therefore, for $k \geq 1$,

$$\|u - u_h\|_s \leq Ch^{k+1} \|u\|_{k+1}$$

The solution converges with order $O(h^{k+1})$. Using quadratic shape functions, the error converges with order $O(h^3)$. So, if the element size is halved, the error will be reduced by a factor of eight.

Error in the strain/flux field

For a steady heat conduction or elasticity problem, for the error in the strain/flux field we have:

- $s = 1$
- $m = 1$

Therefore, for $k \geq 1$,

$$\|u - u_h\|_s \leq Ch^k \|u\|_{k+1}$$

For a given element, the order of convergence in the first derivative is one order less than for the unknown field itself. Hence, the stresses in a finite element simulation are usually '*less*' accurate than the displacements.

Posterior error estimation and adaptivity

A *posterior error* estimation involves quantification of the error after a simulation has been preformed.
If we have an a posterior error estimator for a particular problem, a simulation can been performed, the error estimated in various part of the domain and the mesh and finite element type can be adjusted in order to reduce the error to a prescribed value. This is known as *adaptivity*.
The two obvious strategies are known as:

- *h-adaptivity*: this involves modify the mesh.
- *p-adaptivity*: this involves changing the finite element type.

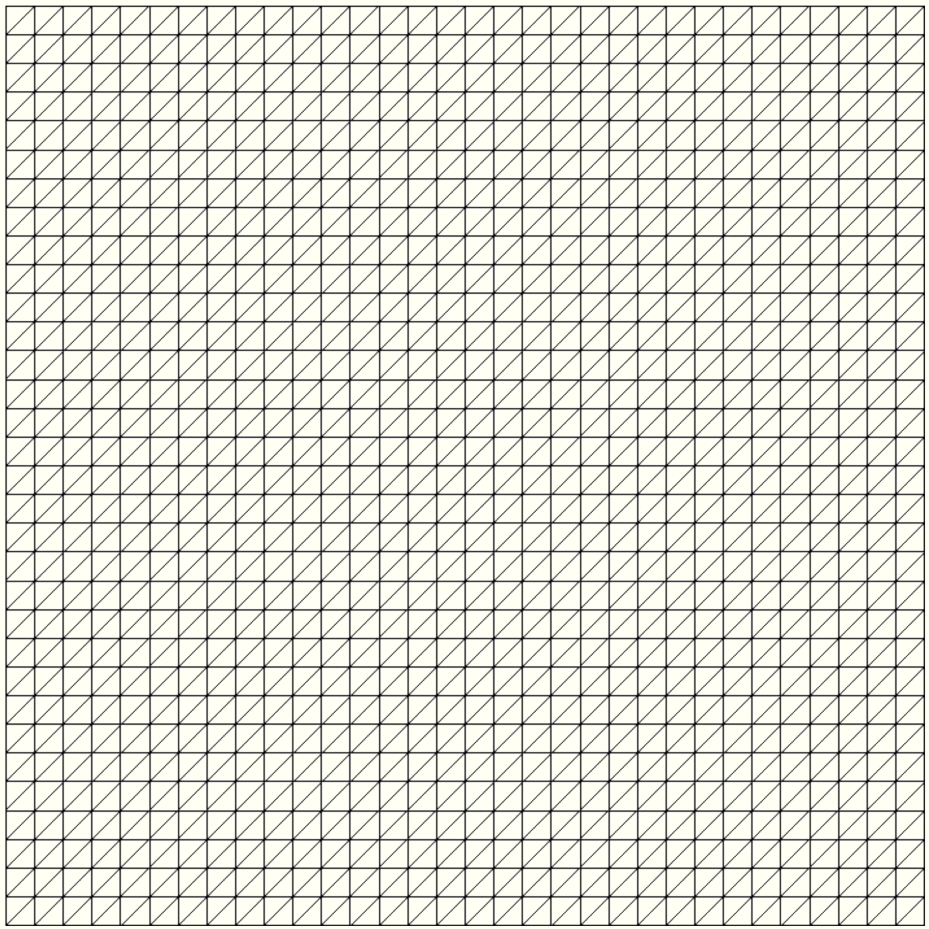
A *posterior error* estimation involves quantification of the error after a simulation has been preformed.
If we have an a posterior error estimator for a particular problem, a simulation can been performed, the error estimated in various part of the domain and the mesh and finite element type can be adjusted in order to reduce the error to a prescribed value. This is known as *adaptivity*.
The two obvious strategies are known as:

- *h-adaptivity*: this involves modify the mesh.
- *p-adaptivity*: this involves changing the finite element type.

A *posterior error* estimation involves quantification of the error after a simulation has been preformed. With an estimate of the error, a judgement can be made as to the reliability of the result. It is then also possible to adapt the finite element mesh to reduce the error. The goal of adaptivity is minimise the error for a given computational effort. It is a rich area of activity in applied and computational mathematics.

Errors, compute-time, and memory

An elasticity problem is solved on a square domain using a uniform structured mesh of linear triangular elements with n_1 vertices in each direction (n_1 is very large).



Errors, compute-time, and memory

1. Based on a priori estimates, what reduction in the error could you expect? If the total number of elements is doubled and the element order is raised to quadratic,
2. Provide an estimate of the increase in the required computational time if using LU decomposition.
3. How much more computer memory is required to store the stiffness matrix? (Give the factor increase.)

Decrease in error

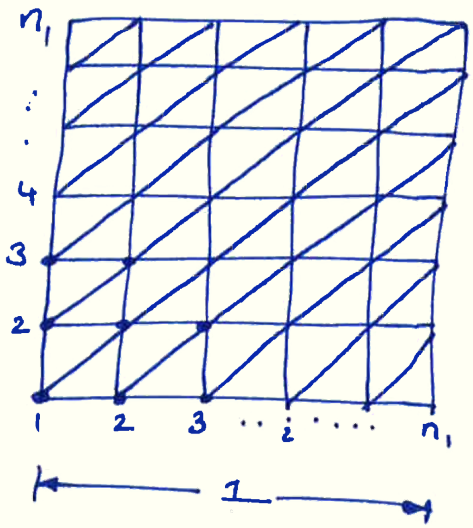
of vertices in each direction = n_1

For a large mesh:

# of vertices (nodes) = n_1^2

# of elements $\approx 2 n_1^2$ (large mesh)

Element Size $h_1 = \frac{1}{n_1}$



If # of elements is doubled

$$2\left(2 n_1^2\right) = 2 n_2^2$$

n_2 - New # of nodes.

$$n_2 = \sqrt{2}\, n_1$$

New element Size $h_2 = \frac{1}{n_2} = \frac{1}{\sqrt{2} n_1}$

Linear elements exhibit $O(h^2)$ convergence in the displacement norm. $(1/\sqrt{2})^2 = 1/2$

$\therefore$ Error will be halved.

Computational time

b.

Total # Nodes = # vertices + # face nodes

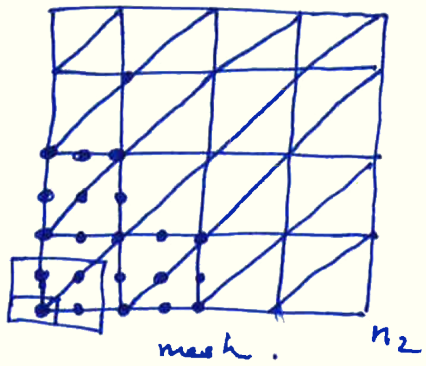
vertices $= n_2^2$

for each vertex nodes $\Rightarrow$ 3 face nodes

face nodes $= 3n_2^2$

Nodes $= n_2^2 + 3n_2^2$

$\Rightarrow 4n_2^2 = 4(\sqrt{2}n_1)^2 = 8n_1^2$



For LU decomposition, time scale $O(n^3)$

$\therefore$ Computational time increases by a factor $(8^3) = 512$

Non-zero elements in the sparse stiffness matrix

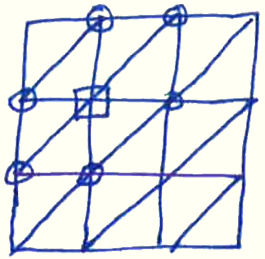
of Non zero terms in stiffness matrix is equal to

$$\sum_{i=1}^{\#\text{Nodes}} C_i \quad , \; C_i = \#\text{of nodes, node '}i\text{' is connected to}$$

For linear elements

$$C_i = 7$$

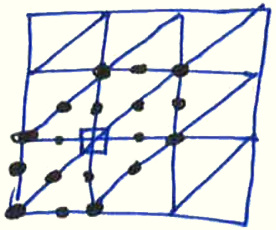
$$\therefore \text{Non zero entries} = 7n_1^2$$



For Quadratic elements:

Vertex nodes: $C_i = 19$

Face/edge nodes $C_i = 9$



Vertex

Memory increase

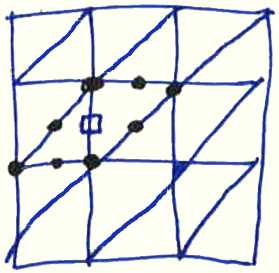
Non zero entries

$\Rightarrow 19 * \underbrace{n_2^2}_{\#\text{vertex}} + 9 \times \underbrace{(3 * n_2^2)}_{\#\text{facenode}}$

$\Rightarrow 46\, n_2^2 = 46\, (\sqrt{2} n_1)^2$

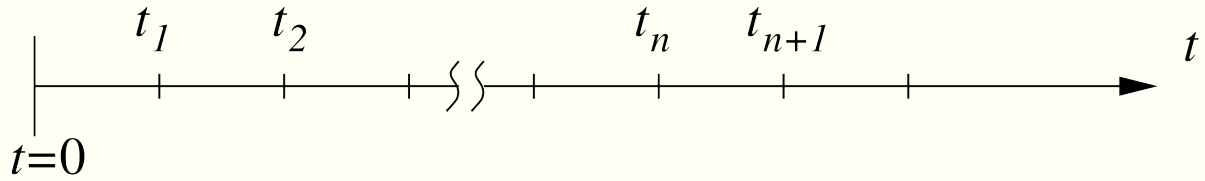
$= 92\, n_1^2$

$\therefore \frac{92}{7}$ more memory.



Time-dependent problems

We wish to develop schemes to deal with semi-discrete finite element equations in a step-by-step fashion. Starting at time $t = 0$, we want to solve a problem up to time $t = t_{final}$. The time interval $(0, t_{final})$ will be 'chopped' into increments Δt:



Time increment: $\Delta t = t_{n+1} - t_n$
What is needed is a strategy to advance the solution from t_n to t_{n+1}.
There are many different techniques for this. We will consider some simple examples and the most commonly used methods.

Forward Euler

Given an equation of the form:

$$\dot{y} = f(y, t)$$

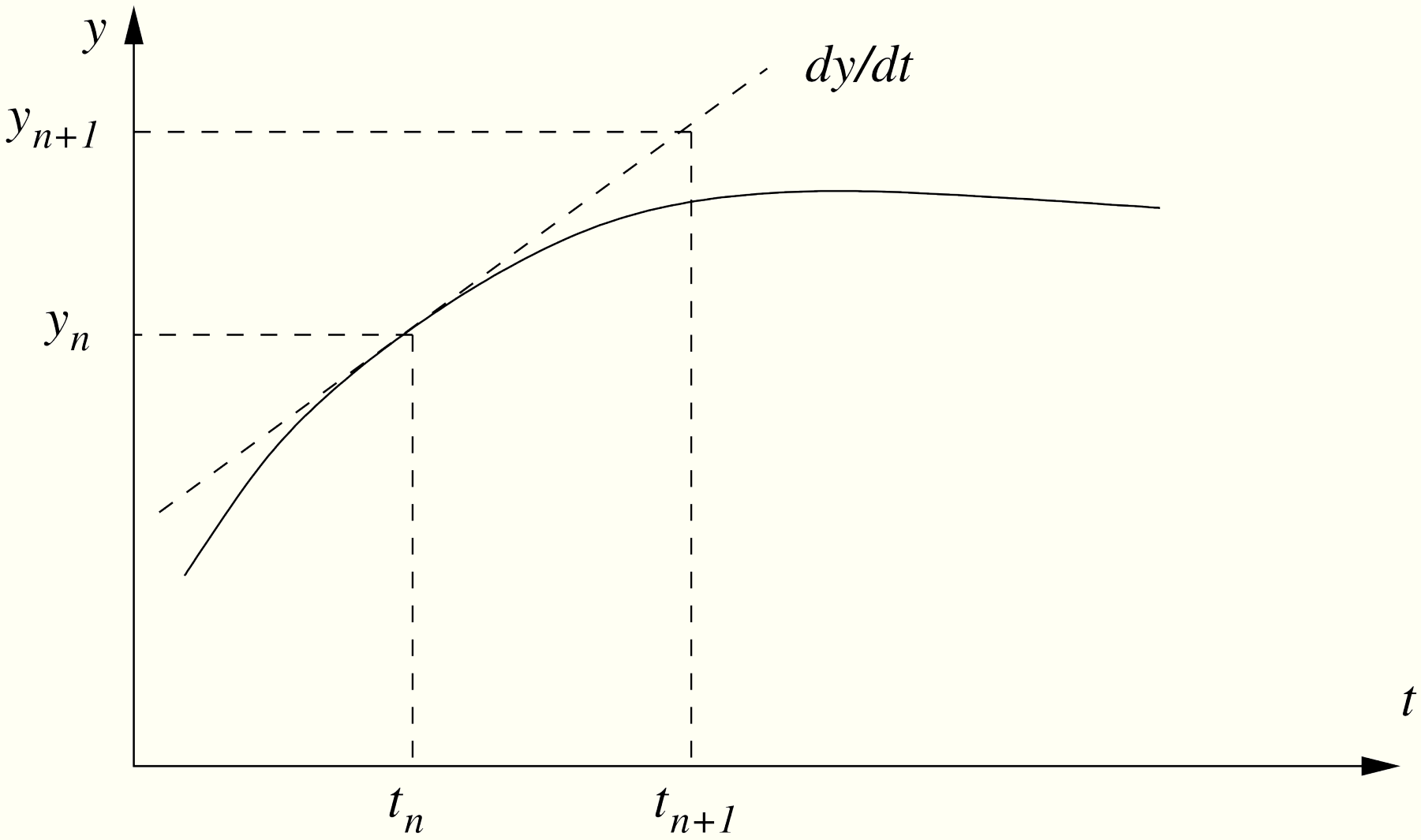
for which we know everything at time $t_n (y_n \mathrm{and} f(y_n, t_n))$, we want to advance to time t_{n+1} and find y_{n+1}. The simplest approach is the forward Euler method. It involves:

$$y_{n+1} = y_n + \Delta t f(y_n, t_n)$$

Re-arranging,

$$\frac{y_{n+1} - y_n}{\Delta t} = f(y_n, t_n)$$

Forward Euler



Forward Euler

We can apply the forward Euler scheme to the nodal degrees of freedom a by replacing y with $\mathbf{a}$ and f with $\dot{\mathbf{a}}$,

$$\frac{\mathbf{a}_{n+1} - \mathbf{a}_n}{\Delta t} = \dot{\mathbf{a}}_n$$

We would like to eliminate the time derivative term $\dot{\mathbf{a}}$ from the semi-discrete heat equation, $\mathbf{M}\dot{\mathbf{a}} + \mathbf{K}\mathbf{a} = \mathbf{b}$, so that we are left with only $\mathbf{a}$. In the forward Euler expression, we have $\dot{\mathbf{a}}_n$, therefore we consider the heat equation at time step t_n,

$$\mathbf{M}\dot{\mathbf{a}}_n + \mathbf{K}\mathbf{a}_n = \mathbf{b}_n$$

Inserting the forward Euler expression for $\dot{\mathbf{a}}$, we have

$$\mathbf{M}\frac{\mathbf{a}_{n+1} - \mathbf{a}_n}{\Delta t} + \mathbf{K}\mathbf{a}_n = \mathbf{b}_n$$

Taking all the known terms (those at t_n) to the right-hand side,

$$\mathbf{M}\mathbf{a}_{n+1} = \mathbf{M}\mathbf{a}_n + \Delta t(\mathbf{b}_n - \mathbf{K}\mathbf{a}_n)$$

Forward Euler

We have a system on linear equation that we can solve to find $\mathbf{a}_{n+1}$. Now the benefit of a lumped mass matrix can be seen. If $\mathbf{M}$ is lumped, there is no need to solve a system of equations and finding $\mathbf{a}_{n+1}$. is trivial.
System of equations for a lumped mass matrix:

$$\begin{bmatrix} M_{11} & 0 & 0 \\ 0 & M_{22} & 0 \\ 0 & 0 & M_{33} \end{bmatrix} \begin{bmatrix} a_{1,n+1} \\ a_{2,n+1} \\ a_{3,n+1} \end{bmatrix} = \begin{bmatrix} \hat{b}_{1,n} \\ \hat{b}_{2,n} \\ \hat{b}_{3,n} \end{bmatrix}$$

Solution:

$$a_{i,n+1} = \hat{b}_{i,n}/M_{ii}$$

This is the key to the efficient simulation of some large scale time-dependent finite element problems. We can start at $t = 0$ and advance in time. The process is simply repeated until the desired final time is reached.

The forward Euler scheme is an example of an explicit scheme since f depends only on data at t_n (or possibly earlier steps). The forward Euler scheme is very simple, but unfortunately not very useful for the heat equation. Firstly, it is not particularly accurate (only $O(\Delta t)$), although this is not the biggest issue. The problem is stability. If Δt is too large, the solution a will grow rapidly and uncontrollably. The forward Euler method is notoriously unstable.

Forward Euler

We have a system on linear equation that we can solve to find $\mathbf{a}_{n+1}$. Now the benefit of a lumped mass matrix can be seen. If $\mathbf{M}$ is lumped, there is no need to solve a system of equations and finding $\mathbf{a}_{n+1}$. is trivial.
System of equations for a lumped mass matrix:

$$\begin{bmatrix} M_{11} & 0 & 0 \\ 0 & M_{22} & 0 \\ 0 & 0 & M_{33} \end{bmatrix} \begin{bmatrix} a_{1,n+1} \\ a_{2,n+1} \\ a_{3,n+1} \end{bmatrix} = \begin{bmatrix} \hat{b}_{1,n} \\ \hat{b}_{2,n} \\ \hat{b}_{3,n} \end{bmatrix}$$

Solution:

$$a_{i,n+1} = \hat{b}_{i,n}/M_{ii}$$

This is the key to the efficient simulation of some large scale time-dependent finite element problems. We can start at $t = 0$ and advance in time. The process is simply repeated until the desired final time is reached.

Forward Euler

The time step at which a method becomes unstable is known at the *critical time step*. Conditionally stable methods are stable for: $\Delta t \leq \Delta t_{\text{crit}}$

- The critical time step often scales with eigenvalues of the finite element system, which in turn depend on the element type and size.
- As the element size is reduced, the critical time step reduces.
- The heat equation is a so-called stiff equation. This means that the restriction on the time step for explicit schemes is particularly severe. It scales according to $\Delta t \propto h^2$

where h is a measure of the size of an element. This is a very strong restriction, and means that explicit schemes are not usually practical for the heat equation.

Second-order Central difference method

$$\dot{\mathbf{a}}_n = \frac{\mathbf{a}_{n+1} - \mathbf{a}_{n-1}}{2\Delta t}$$
$$\ddot{\mathbf{a}}_n = \frac{\mathbf{a}_{n-1} - 2\mathbf{a}_n + \mathbf{a}_{n+1}}{\Delta t^2}$$

This algorithm is explicit – it is possible to express $\mathbf{a}_{n+1}$ in terms of quantities at n and earlier times. This scheme is $O(\Delta t^2)$ accurate and it is conditionally stable. It is stable for:

$$\Delta t \leq \frac{2}{\omega_{h,max}}$$

where $\omega_{h,max}$ is the highest natural frequency. Therefore:

$$\Delta t \leq \frac{h}{c}$$

where c is the dilatation wave speed ($c = \sqrt{(E/\rho)}$) and h is the length of an element, which is the Courant, Friedrichs and Lewy (CFL) stability condition required for a wave to travel through an element.

Backward Euler

$$y_{n+1} = y_n + \Delta t f(y_{n+1}, t_{n+1})$$

Note now that f is evaluated at y_{n+1} and not at y_n. However, we do not know y_{n+1}; it is what we want to find.

$$\mathbf{a}_{n+1} = \mathbf{a}_n + \Delta t \dot{\mathbf{a}}_{n+1}$$

$$\frac{\mathbf{a}_{n+1} - \mathbf{a}_n}{\Delta t} = \dot{\mathbf{a}}_n$$

Since the time derivative term is evaluated at t_{n+1}, we pose the semi-discrete heat equation at t_{n+1},

$$\mathbf{M}\dot{\mathbf{a}}_{n+1} + \mathbf{K}\mathbf{a}_{n+1} = \mathbf{b}_{n+1}$$

Inserting the backward Euler expression for $\dot{\mathbf{a}}$, we have

$$\mathbf{M}\frac{\mathbf{a}_{n+1} - \mathbf{a}_n}{\Delta t} + \mathbf{K}\mathbf{a}_{n+1} = \mathbf{b}_{n+1}$$

Backward Euler

Collecting the unknown terms on the left-hand side and the known terms on the right- hand side:

$$(\mathbf{M} + \Delta t \mathbf{K})\, \mathbf{a}_{n+1} = \Delta t \mathbf{b}_{n+1} + \mathbf{M}\mathbf{a}_n$$

- Now, even if $\mathbf{M}$ is a diagonal matrix, we still need to solve a system of equations to find $\mathbf{a}_{n+1}$ because $\mathbf{K}$ is not diagonal.
- A system must be solved because the backward Euler method is an *implicit method*.
- In stepping from t_n to t_{n+1}, it relies on data at t_{n+1} which is what characterises implicit methods.
- The consequence is that a system of equations must be solved.

Collecting the unknown terms on the left-hand side and the known terms on the right- hand side:

$$(\mathbf{M} + \Delta t \mathbf{K})\,\mathbf{a}_{n+1} = \Delta t \mathbf{b}_{n+1} + \mathbf{M}\mathbf{a}_n$$

- Now, even if $\mathbf{M}$ is a diagonal matrix, we still need to solve a system of equations to find $\mathbf{a}_{n+1}$ because $\mathbf{K}$ is not diagonal.
- A system must be solved because the backward Euler method is an *implicit method*.
- In stepping from t_n to t_{n+1}, it relies on data at t_{n+1} which is what characterises implicit methods.
- The consequence is that a system of equations must be solved.

The backward Euler method requires far more computational effort at each time step compared to the forward Euler method, yet it has the same order of accuracy ($O(\Delta t)$). However it is unconditionally stable. That is, there is no critical time step and the solution will remain bounded for any Δt. While each step is more expensive relative to an explicit method, it is possible that far fewer total steps will be required.

Summary of time-stepping schemes

- Explicit schemes are characterised by conditional stability.
- Explicit schemes can sometimes be very efficient as there is no need to solve systems of equations.
- Implicit schemes can be unconditionally stable (although not all implicit schemes are unconditionally stable).
- Implicit schemes require the solution of a system of equations, which can be expensive.
- Whether a scheme is explicit or implicit does not infer anything regarding accuracy.