

Non-linear analysis - Basis



Code_Aster, Salome-Meca course material

GNU FDL licence (<http://www.gnu.org/copyleft/fdl.html>)

Outline

- ▶ Description of non-linear problems
- ▶ Theoretical elements for solving non-linear problems
- ▶ Solving non-linear problems with Newton
- ▶ Using *Code_Aster*

What is a non-linear problem in mechanics ?

Non-linear problems

▶ Non-linearities come from :

- Kinematic (movement and strains) : large displacement, large rotation, large strains
- Material : non-linear response, history-dependent response
- Contact/friction

▶ The three non-linearities should be coupled

▶ Numerical simulation require expert analysis

Non-linear material

▶ Material non-linearity

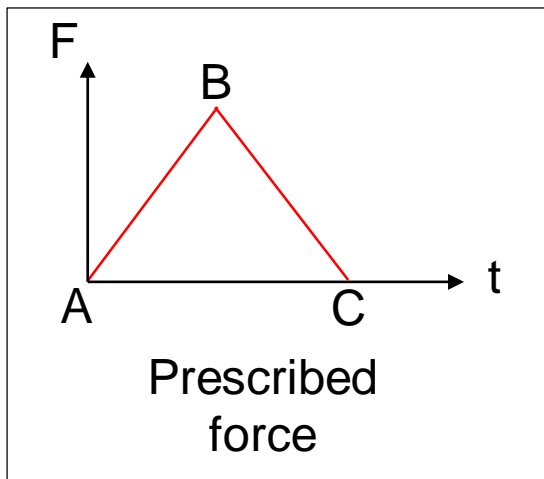
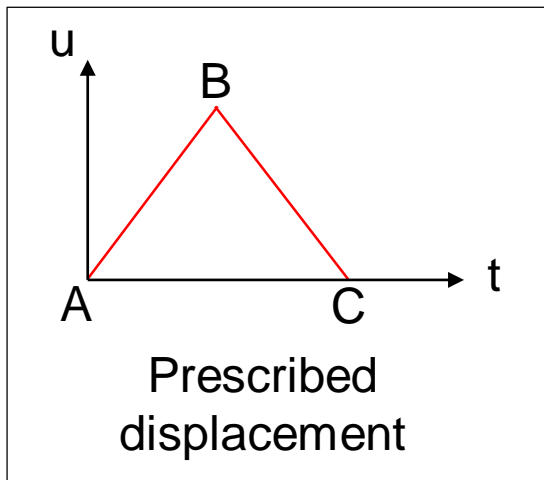
- Experimental identification : identify macroscopical response force/displacement



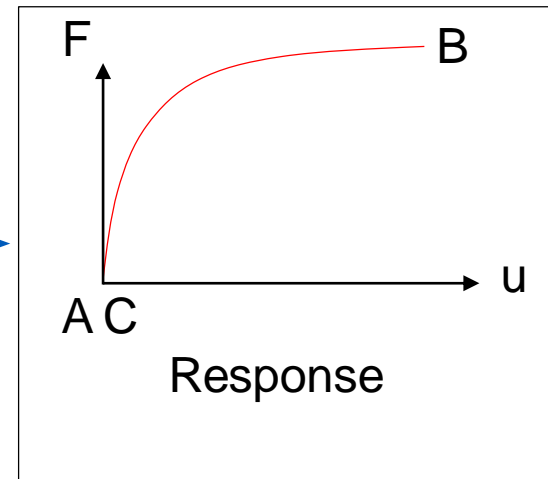
- Most materials are non-linear and/or history dependent

Non-linear material

- ▶ Displacement/force function is not linear **but not** history dependent



$$F = \varphi(u) \quad \text{with} \quad \varphi(u) \neq a \cdot u + b$$

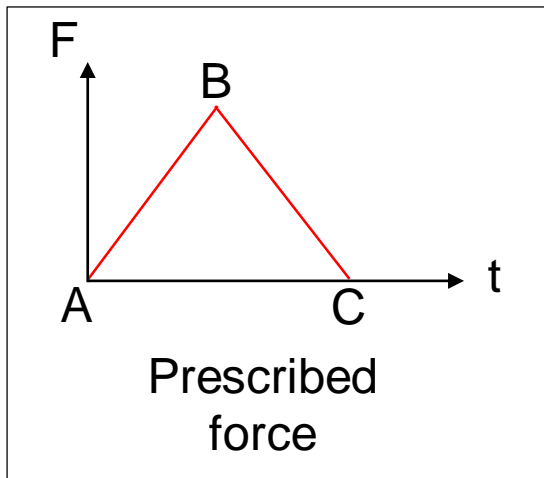
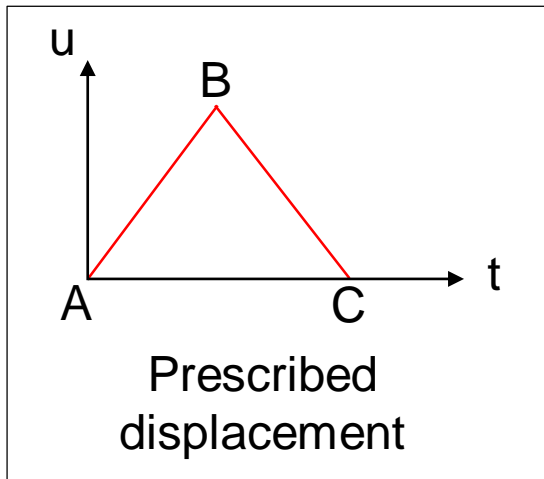


$$u(C) = u(A)$$

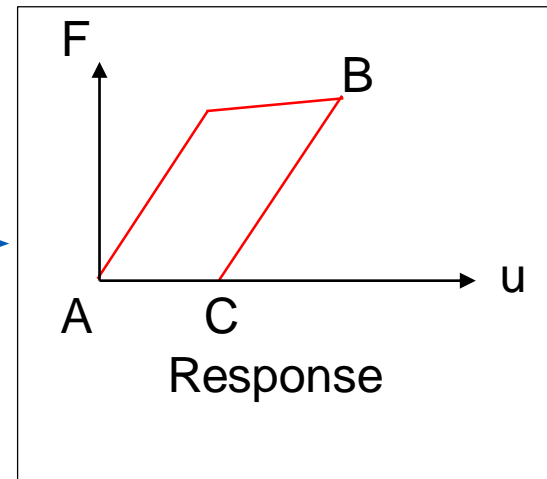
$$F(C) = F(A)$$

Non-linear material

- ▶ Displacement/force function is not linear **and** history dependent



$$F = \varphi(u) \quad \text{with} \quad \varphi(u) \neq a \cdot u + b$$

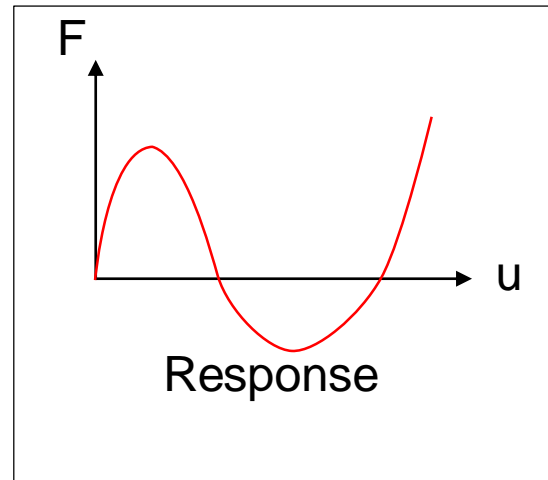
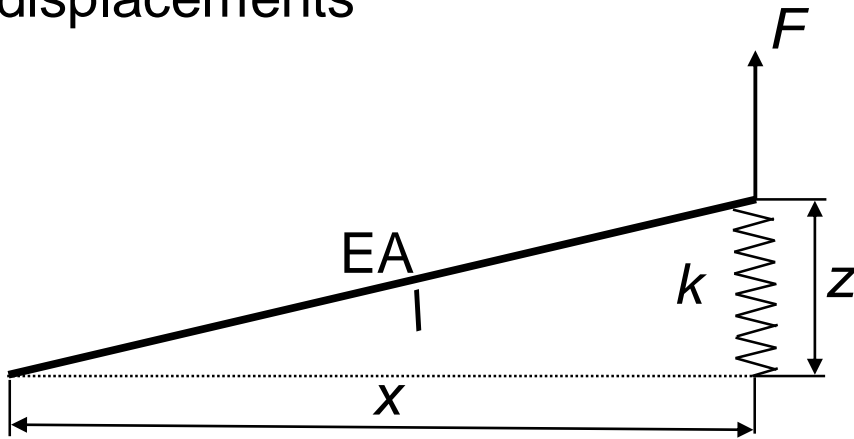


$$u(C) \neq u(A)$$

Residual displacement
Next loading : initial displacement $\neq 0$

Non-linear kinematic

- ▶ Truss/beams/shells elements with large rotation and/or large displacements



E : Young modulus

A : section

L : length

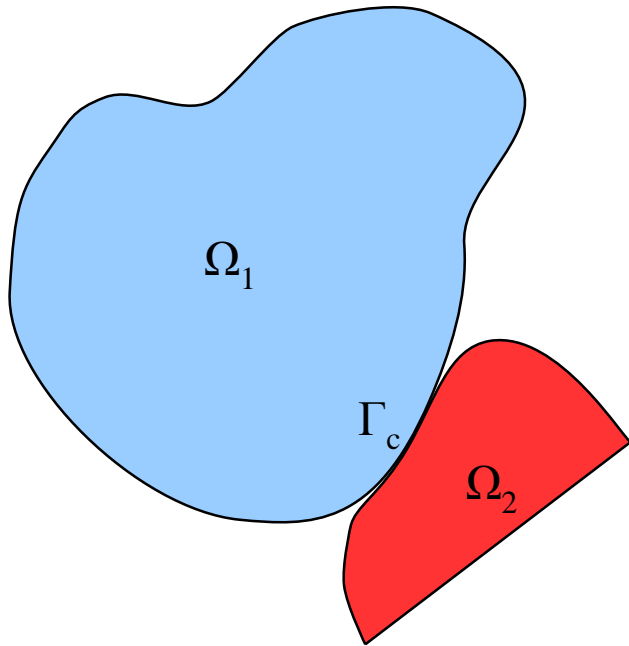
F : load

u : displacement

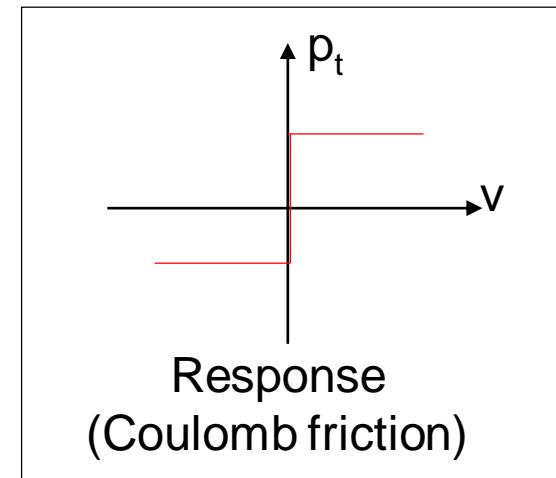
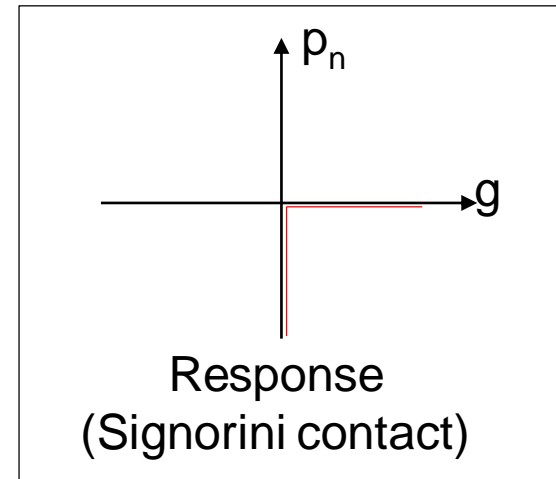
k : rigidity of the spring

Non-linear contact/friction

- ▶ Contact and friction: a very difficult non-linear problem



p_n : normal pressure
 p_t : tangent pressure
 g : distance between solids (gap)
 v : tangential speed between solids

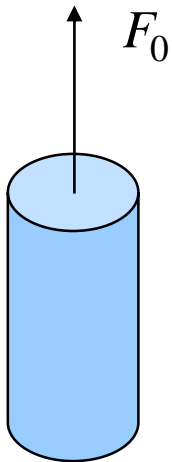


Some theoretical elements

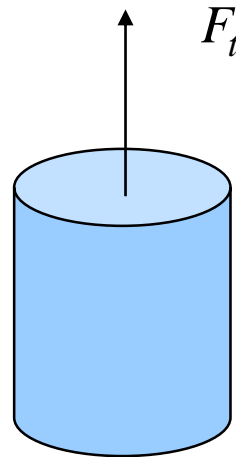
Equations – Continuous form

► Measure of stress : Cauchy (true) stress

$$\sigma = \lim_{\Delta S \rightarrow 0} \frac{\Delta F}{\Delta S}$$



$$\sigma_0 = \lim_{\Delta S_0 \rightarrow 0} \frac{\Delta F_0}{\Delta S_0}$$



$$\sigma_t = \lim_{\Delta S_t \rightarrow 0} \frac{\Delta F_t}{\Delta S_t}$$

Cauchy stress : measure on **deformed** configuration

► Measure of strain :

- for finite strain, it's an arbitrary choice, depending on behavior law

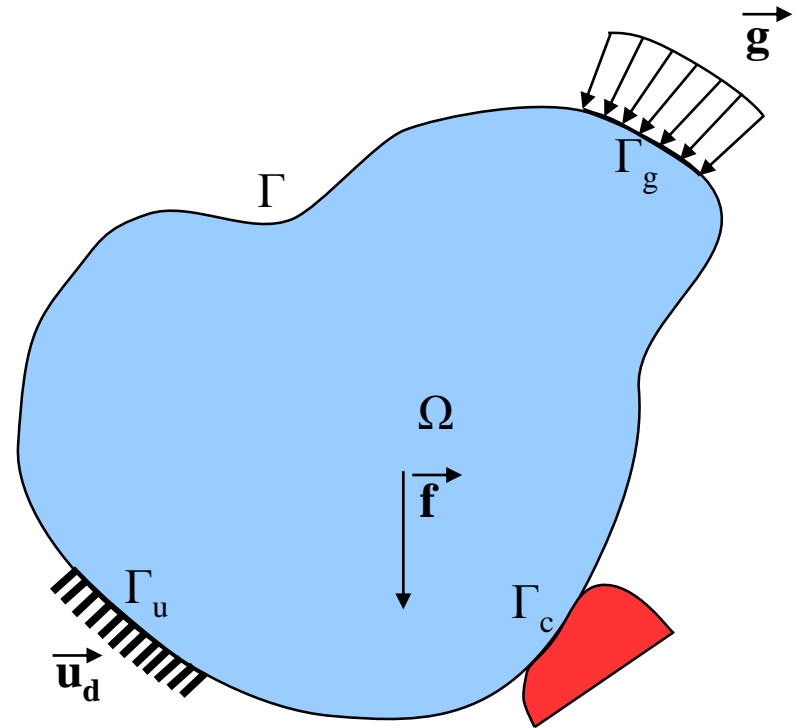
- For small strain
$$\varepsilon(u) = \frac{1}{2} (\nabla u + \nabla^t u)$$

Equations – Continuous form

► Defining solid Ω which is in equilibrium with external forces

- Γ is external boundary
- Γ_u to apply prescribed displacement \mathbf{u}_d
- Γ_g to apply prescribed force \mathbf{g}
- Γ_c to define contact/friction
- \mathbf{f} is volumic force

$$\begin{cases} \operatorname{div} \sigma + f = 0 & \text{in } \Omega \\ \sigma \cdot n = g & \text{on } \Gamma_g \\ u = u_d & \text{on } \Gamma_u \\ L_{cf}(u) & \text{on } \Gamma_c \end{cases}$$



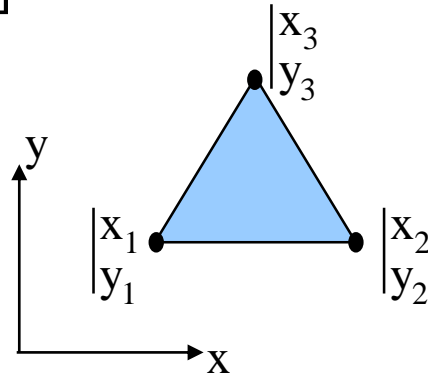
Equations – Finite element approximation

► Main shapes for elements :

- segment, triangle, quadrangle, hexaedron, tetraedron, pyramid, prism
- linear or quadratic (curved or straight edges)

► Geometry discretization :

- Coordinates in element $\{x\}$
- Discretized coordinates at nodes $\{x_i\}$
- Shape functions $[N_g]$

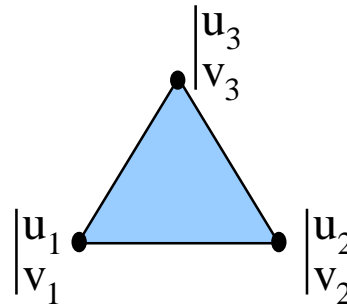


$$\{x\} = [N_g] \{x_i\}$$

Equations – Finite element approximation

► Discretization of the unknowns:

- Unknowns depending on physic : displacement for mechanics temperature for thermic, ...
- Displacements in element $\{u\}$
- Discretized displacement at nodes $\{u_i\}$
- Shape functions $[N_u]$



$$\{u\} = [N_u] \cdot \{u_i\}$$

We suppose isoparametric elements $[N] = [N_g] = [N_u]$

Equations – Finite element approximation

► Weak form of equilibrium (weighted residuals) :

- Find $u \in E_u$ with $\forall \tilde{u} \in E_{\tilde{u}}$:

$$\int_{\Omega} \sigma(u) : \varepsilon(\tilde{u}) . d\Omega = \int_{\Omega} f(u) : \tilde{u} . d\Omega + \int_{\Gamma_g} g(u) : \tilde{u} . d\Gamma$$

- Virtual displacements \tilde{u} and virtual strains $\varepsilon(\tilde{u}) = \frac{1}{2}(\nabla \tilde{u} + \nabla^t \tilde{u})$
- With E_u space of kinematically admissible displacements :

$$E_u = \{u \text{ with } u(X) = u_d \quad \forall X \in \Gamma_u\}$$

- With $E_{\tilde{u}}$ space of virtual displacements :

$$E_{\tilde{u}} = \{\tilde{u} \text{ with } \tilde{u}(X) = 0 \quad \forall X \in \Gamma_u\}$$

► The weak form is the mechanical concept of **virtual power**

- Galerkin form : $\{u\} = [N].\{u_i\}$ and $\{\tilde{u}\} = [N].\{\tilde{u}_i\}$

Equations – Finite element approximation

- ▶ Non-linear problem : stress depending on displacement $\sigma(u)$
 - From displacement u to strain ε : **kinematic** non-linearity
 - From strain ε to stress σ : **material** non-linearity

- ▶ Non-linear material

$$\sigma(u) = \sigma(\varepsilon(u))$$

- ▶ Non-linear material depending on material's history (plasticity,...) :

$$\sigma(u) = \sigma(\alpha, \varepsilon(u))$$

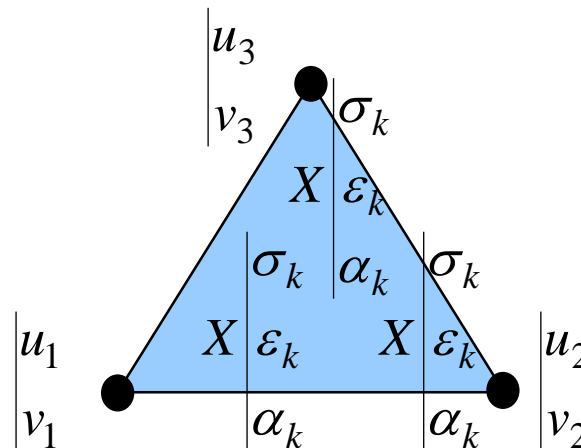
Internal variables α

Equations – Finite element approximation

- ▶ Numerical quadrature : to compute continuous som, using a Gauss approximation

$$\int_{\Omega} \sigma \cdot d\Omega \rightarrow \sum_{k=1}^{\text{nb points}} \sigma_k \cdot \omega_k$$

- ▶ Stress, strain and internal variables : discretized at **numerical points** X (quantities post_fixed by `_ELGA` in *Code_Aster*)
- ▶ Displacement : discretized at **nodes** ● (quantities post_fixed by `_NOEU` in *Code_Aster*)



Equations – Finite element approximation

► Computation of internal forces

$$\int_{\Omega} \sigma(u) : \varepsilon(\tilde{u}) . d\Omega \rightarrow \{L_{\text{int}}(u)\} = [Q(u)].\{\sigma(u)\}$$

► Non-linear internal forces : depending on u

- Non-linear kinematic $[Q(u)]$
- Non-linear material $\{\sigma(u)\}$

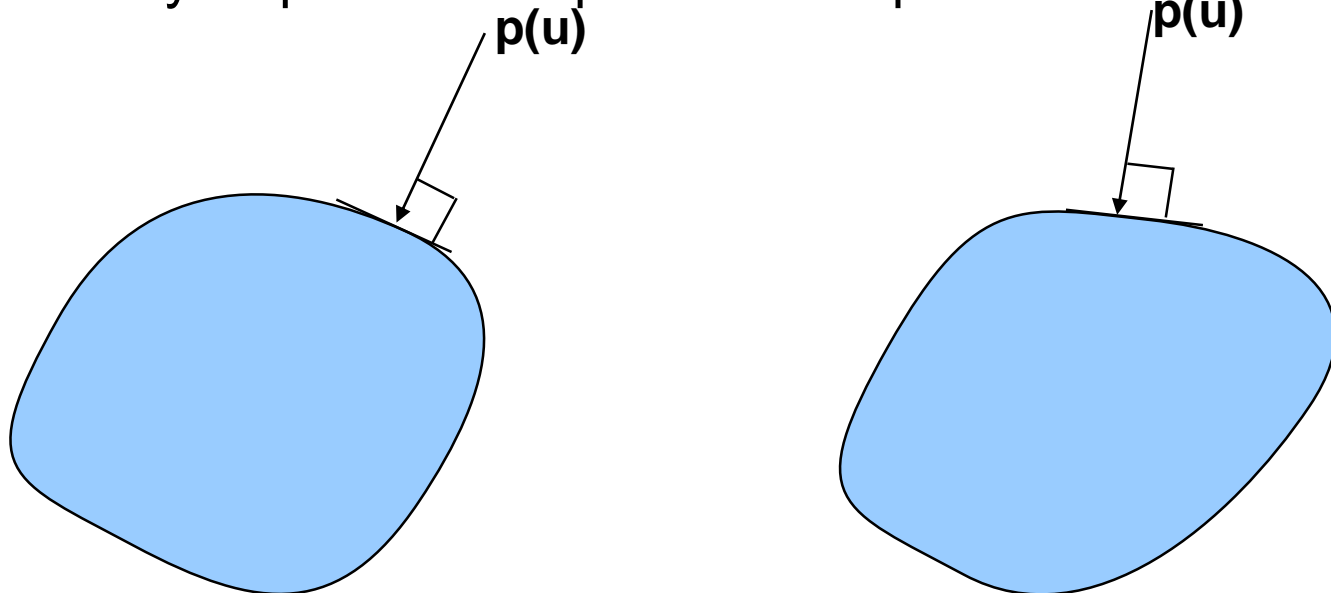
Equations – Finite element approximation

► Computation of external forces

$$\int_{\Omega} f(u) : \tilde{u} . d\Omega + \int_{\Gamma_g} g(u) : \tilde{u} . d\Gamma \rightarrow \{L_{ext}(u)\}$$

► Non-linear external forces : depending on u

- Non-linear when following forces : pressure always normal to boundary → pressure depends on displacement



Equations – Finite element approximation

► Final equilibrium equation : discretized form

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

Non-linear equation, depending on u

General algorithm for non-linear problem

Solving non-linear problem

► To solve non-linear problem : an incremental algorithm

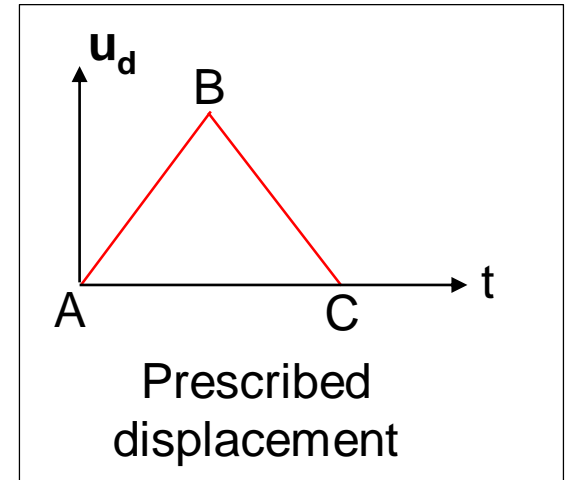
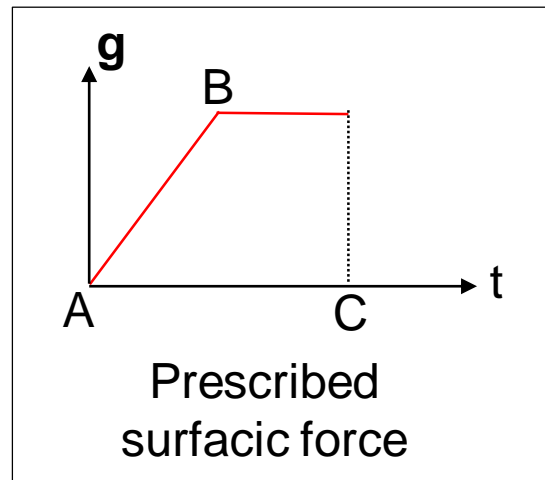
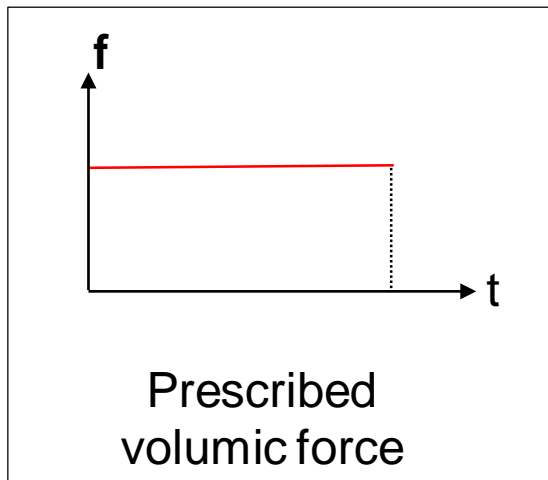
- Problem is parametrized by the parameter t
- t is **not real time** (quasi-static problem)

► Why parametrization ?

- Real boundary conditions should be applied by non-constant values
- Non-linear material produced non-linear equation, precision should depend on incremental step size
- A non-linear problem is easier to solve when cut by small step size

Solving non-linear problem

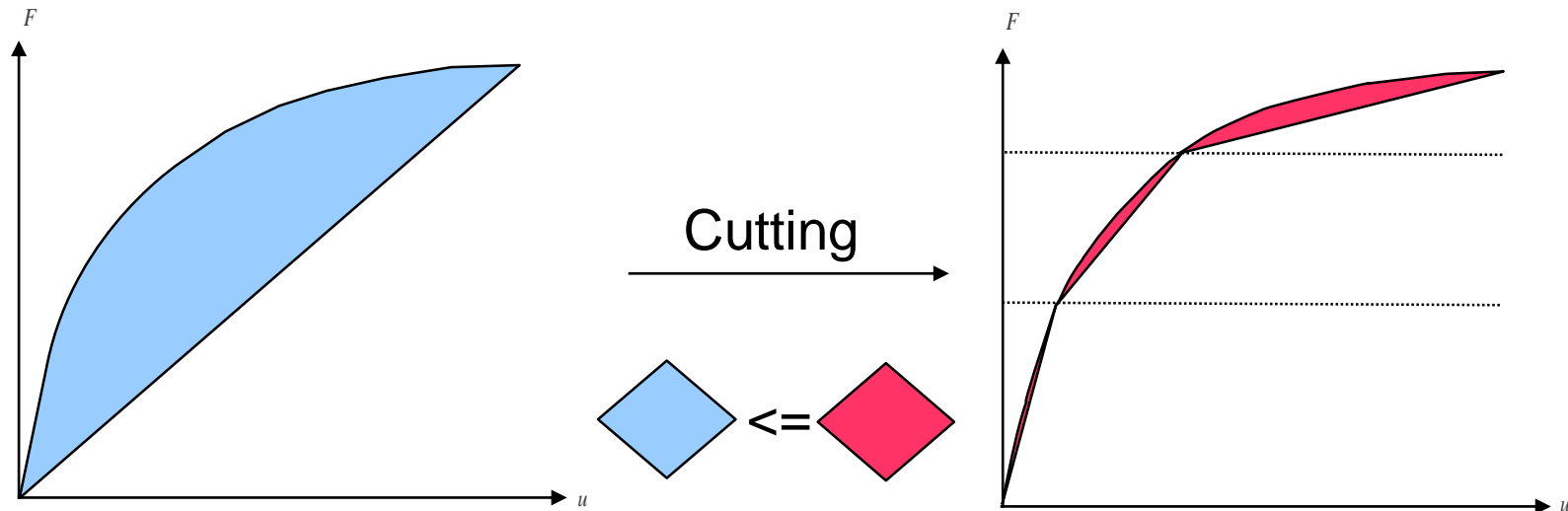
- ▶ Parametrization because boundary conditions should be applied by non-constant values



Some examples of prescribed boundary conditions

Solving non-linear problem

- ▶ Parametrization to cut a non-linear problem to decrease degree of non-linearity



High degree of non-linearity on complete problem

Low degree of non-linearity at each step

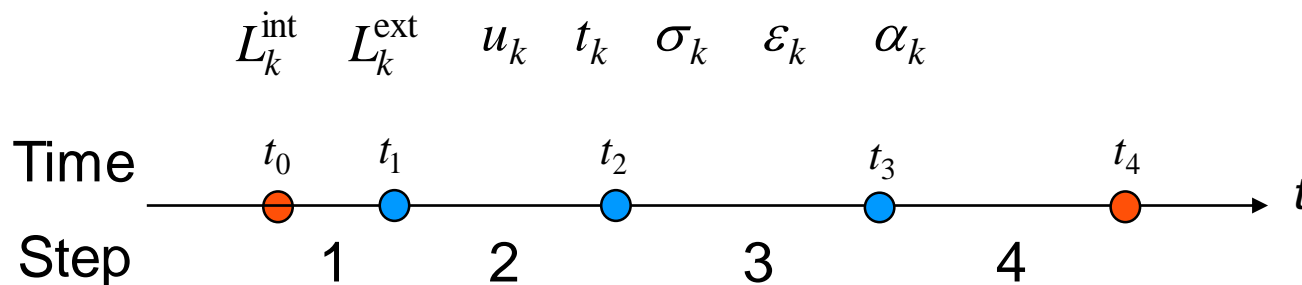
Solving non-linear problem

► The parameter t in the equations :

- Internal forces : dependence with respect to t is **implicit**, it results from the integration of the constitutive relation in time
- External forces : dependence with respect to t is **explicit** (applied BC) or **implicit** (following forces)

$$\{L_{\text{int}}(u(t))\} - \{L_{\text{ext}}(t, u(t))\} = \{0\}$$

► « Time » discretization, all quantities are parametrized by step k :



Incremental quantities : $u_k = u_{k-1} + \Delta u$

Solving non-linear problem

▶ Solving a non-linear equation $F(x) = 0$: the Newton's method

- Construction of the series $(x^n)_{n \in \mathbb{N}}$
- Taylor series expansion of the first order

$$0 = F(x^n) \approx F(x^{n-1}) + F'(x^{n-1}) \cdot (x^n - x^{n-1})$$

- Next value of the term of the series

$$x^n = x^{n-1} - \frac{1}{F'(x^{n-1})} F(x^{n-1})$$

▶ Properties :

- Quadratic convergence near the solution
- Computation of $F'(x^{n-1})$ should be very expensive
- $F'(x^{n-1}) \neq 0$

Solving non-linear problem

- ▶ Newton's method for equilibrium equation, at step k :

$$\begin{aligned} \{L_{\text{int}}(u_k(t_k))\} - \{L_{\text{ext}}(t_k, u_k(t_k))\} &= \{0\} \\ \{L_{\text{int}}(u(t))\} - \{L_{\text{ext}}(t, u(t))\} &= \{0\} \end{aligned}$$



Simplify : we are at step k

- ▶ Internal forces :

$$L^{\text{int},n} \approx L^{\text{int},n-1} + \left(\frac{\partial L^{\text{int}}}{\partial u} \right)_{u^{n-1}} \cdot \delta u^n \quad \text{with} \quad \delta u^n = (u^n - u^{n-1})$$

- ▶ External forces :

$$L^{\text{ext},n} \approx L^{\text{ext},n-1} + \left(\frac{\partial L^{\text{ext}}}{\partial u} \right)_{u^{n-1}} \cdot \delta u^n \quad \text{with} \quad \delta u^n = (u^n - u^{n-1})$$

Solving non-linear problem

- ▶ Newton's method for equilibrium equation, at step k :

$$L^{\text{int},n-1} - L^{\text{ext},n-1} + \left(\frac{\partial L^{\text{ext}}}{\partial u} - \frac{\partial L^{\text{int}}}{\partial u} \right)_{u^{n-1}} \cdot \delta u^n = 0$$

$$K^{n-1} \cdot \delta u^n = R^{n-1}$$

- ▶ K^{n-1} is tangent matrix

$$K^{n-1} = \left(\frac{\partial L^{\text{int}}}{\partial u} \right)_{u^{n-1}} - \left(\frac{\partial L^{\text{ext}}}{\partial u} \right)_{u^{n-1}}$$

- ▶ R^{n-1} is equilibrium residual

$$R^{n-1} = \left(L^{\text{ext},n-1} - L^{\text{int},n-1} \right)$$

Solving non-linear problem

▶ Global algorithm :

- 1) Compute internal and external forces
- 2) Compute tangent matrix
- 3) Solve linear system
- 4) Update displacements
- 5) Evaluate convergence

Solving non-linear problem

- ▶ Compute internal and external forces :
 - Exact computation is **necessary** (software requirement)
 - Non-linear behavior is evaluated at each Gauss point
 - Complex behavior laws computation should be expensive
 - Precision of computation of some behavior laws should depend on increment of displacement :
 - Visco-plasticity
 - Non-radial loading
 - **PETIT_REAC** strain measure

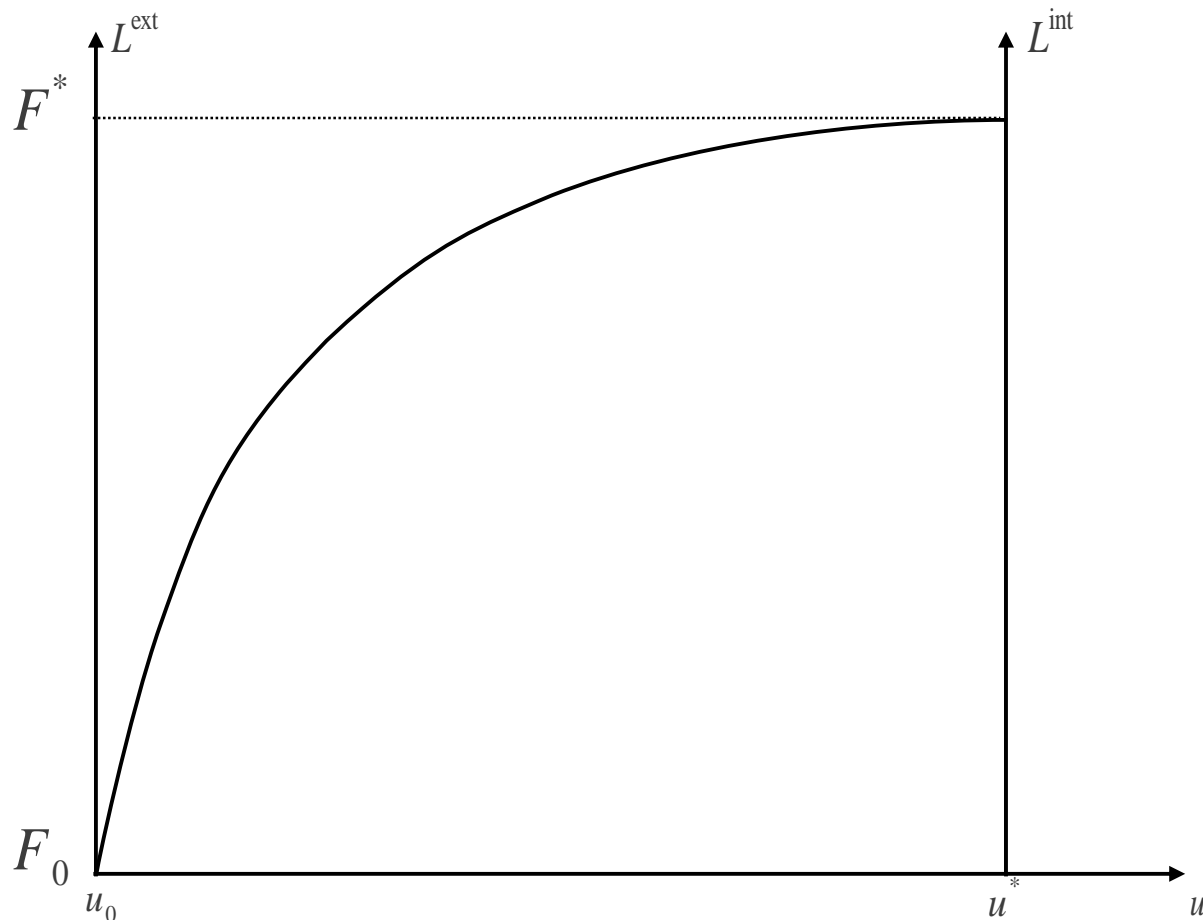
Solving non-linear problem

► Compute tangent matrix :

- Exact computation is NOT necessary
- Tangent matrix is evaluated at each Gauss point
- Tangent matrix computation should be expensive

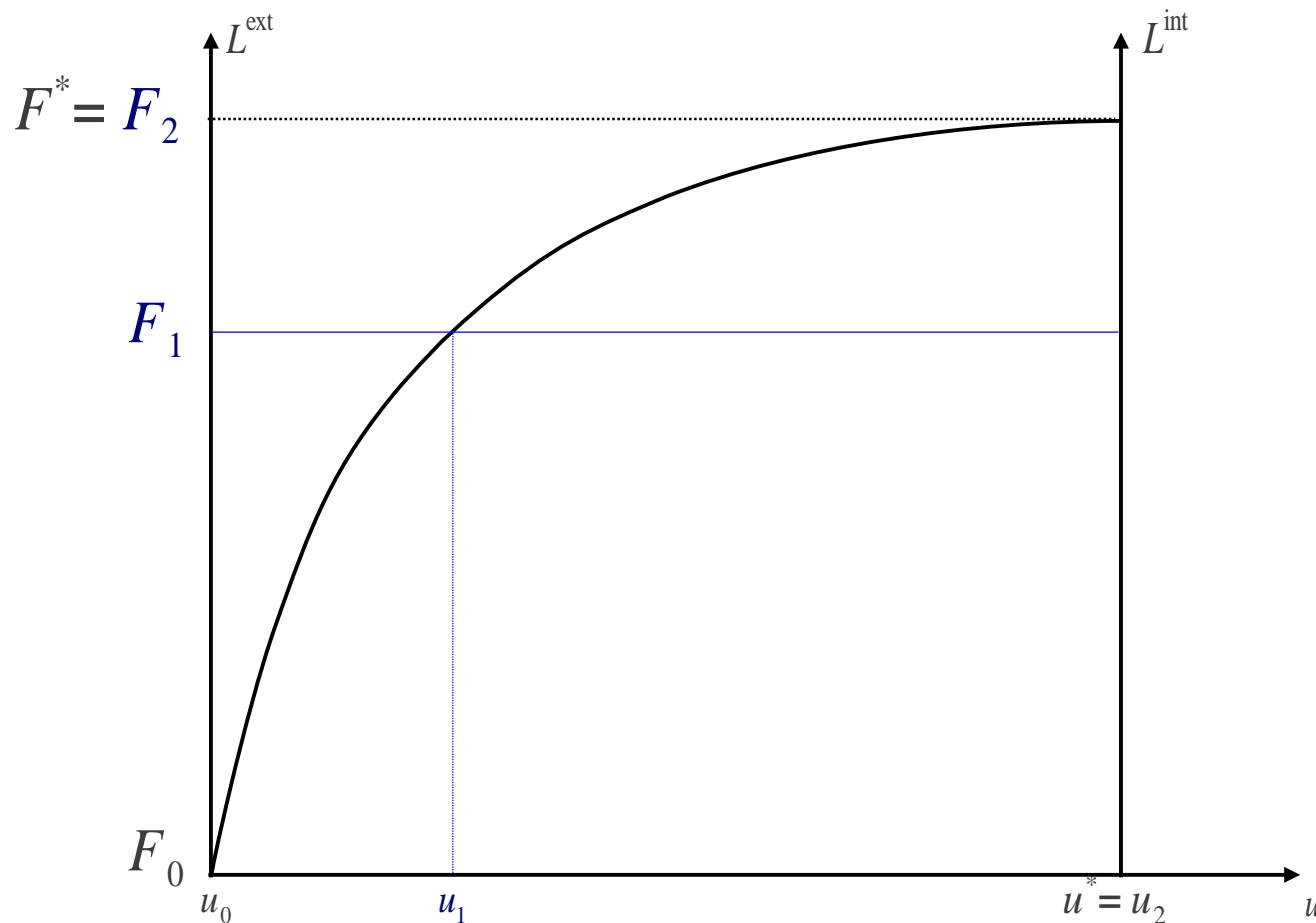
Solving non-linear problem – Newton by graphic

- ▶ Find the solution (u^*, F^*) where F^* is known and u^* is unknown



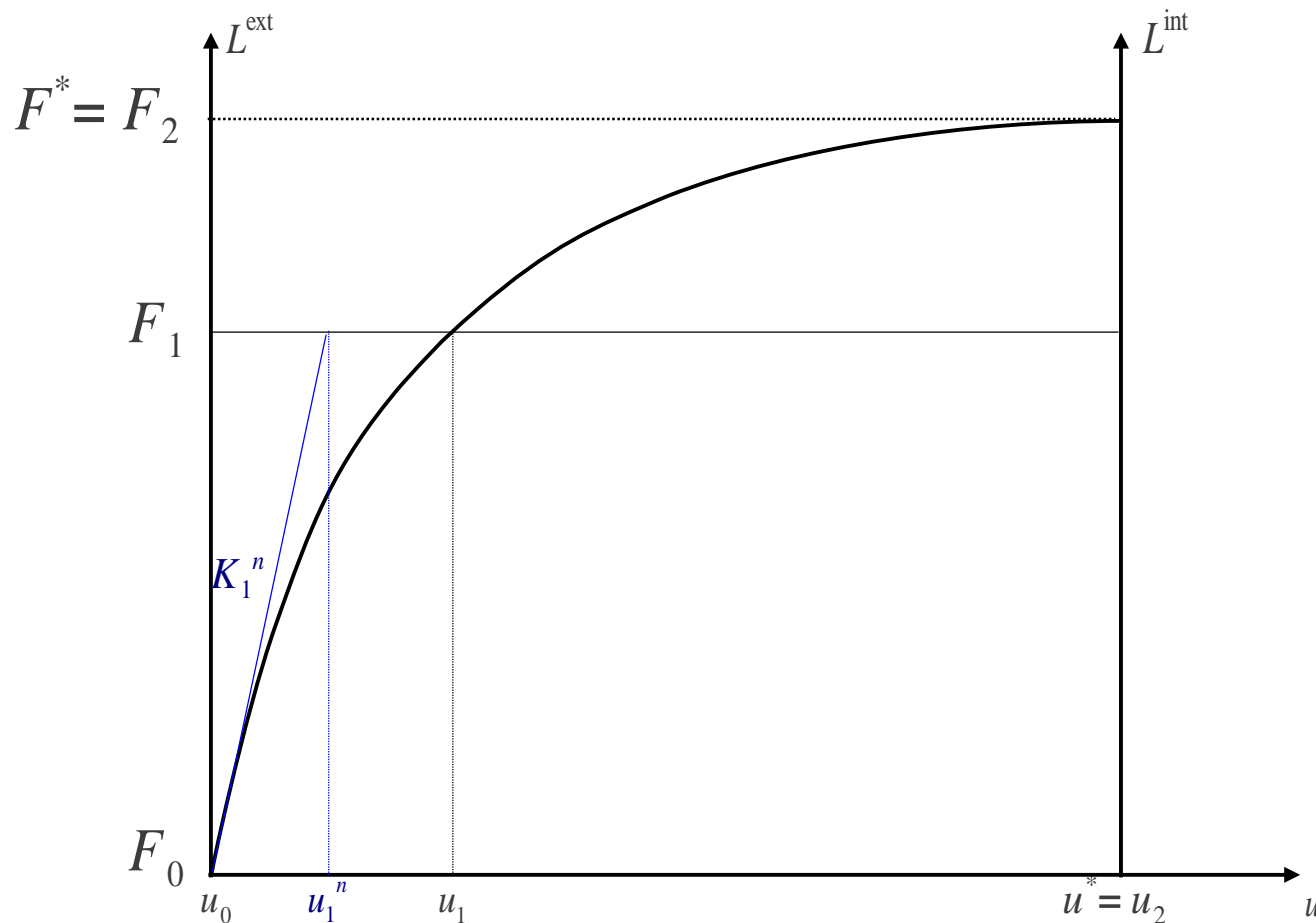
Solving non-linear problem – Newton by graphic

- ▶ Dividing F^* in two increments (reducing degree of non-linearity) :
find (u_1, F_1) and (u_2, F_2)



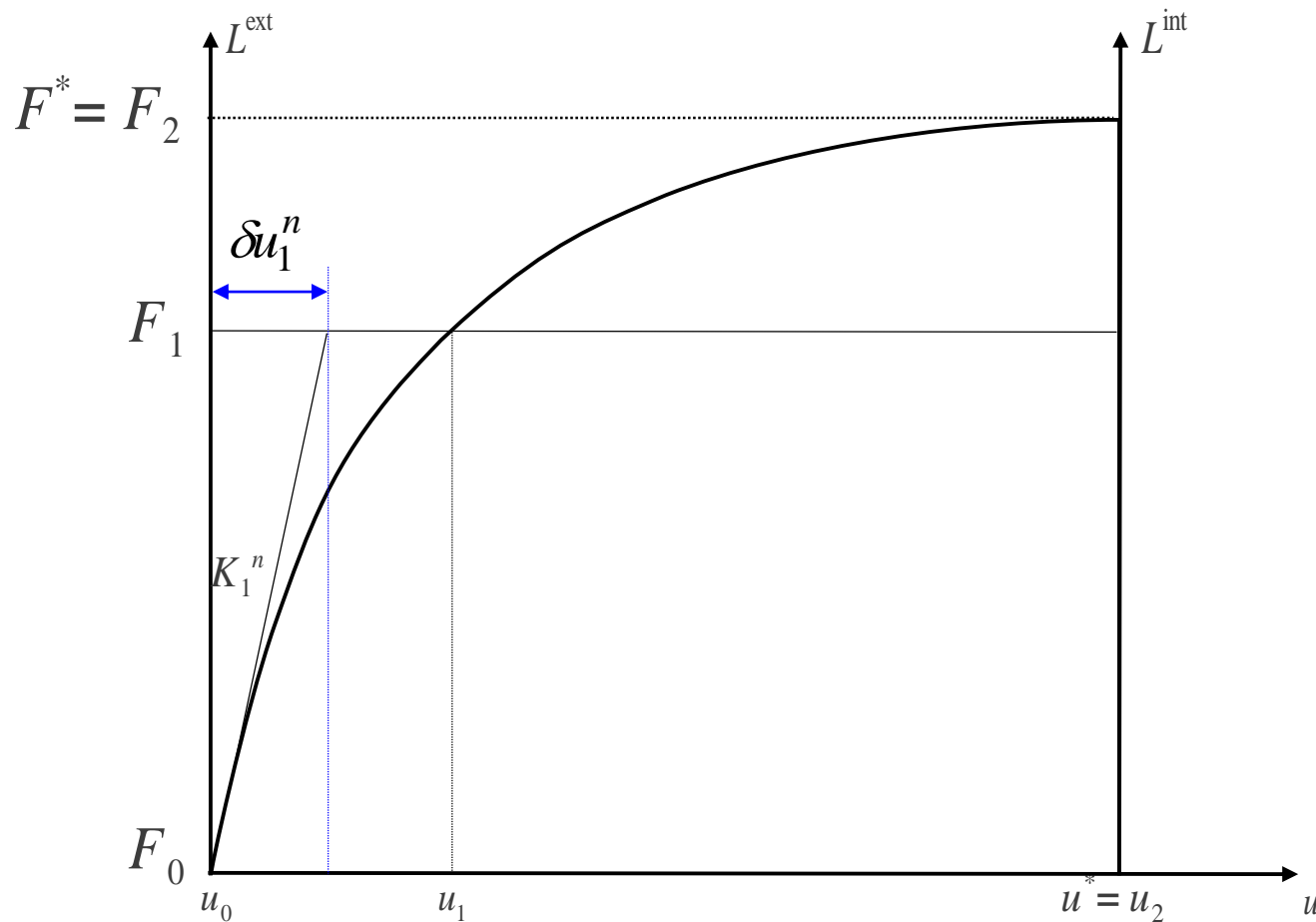
Solving non-linear problem – Newton by graphic

- ▶ Compute tangent matrix K_1^n for iteration Newton n and step 1



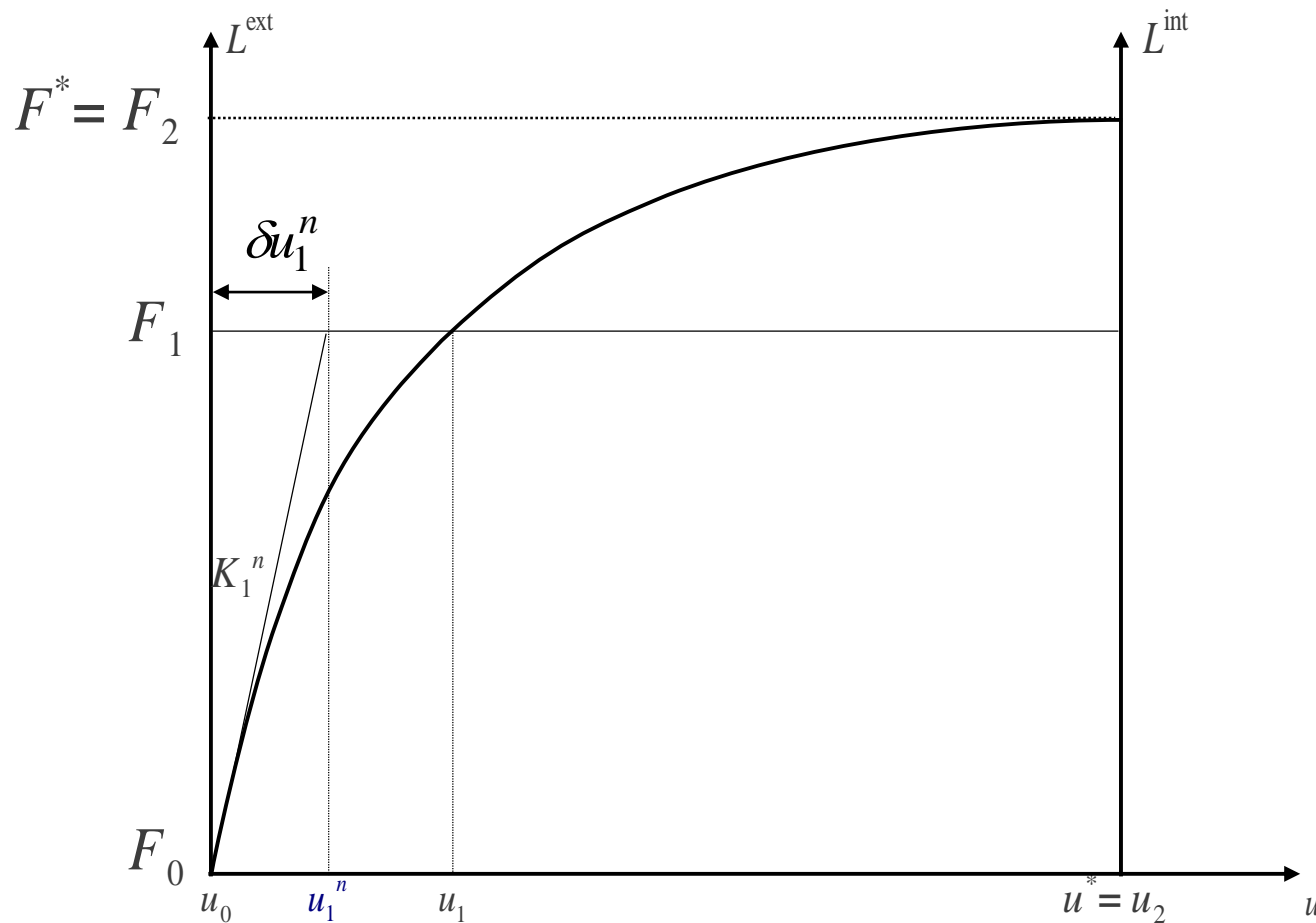
Solving non-linear problem – Newton by graphic

► Solving $K_1^n \cdot \delta u_1^n = F_1 - F_0 \rightarrow \delta u_1^n$



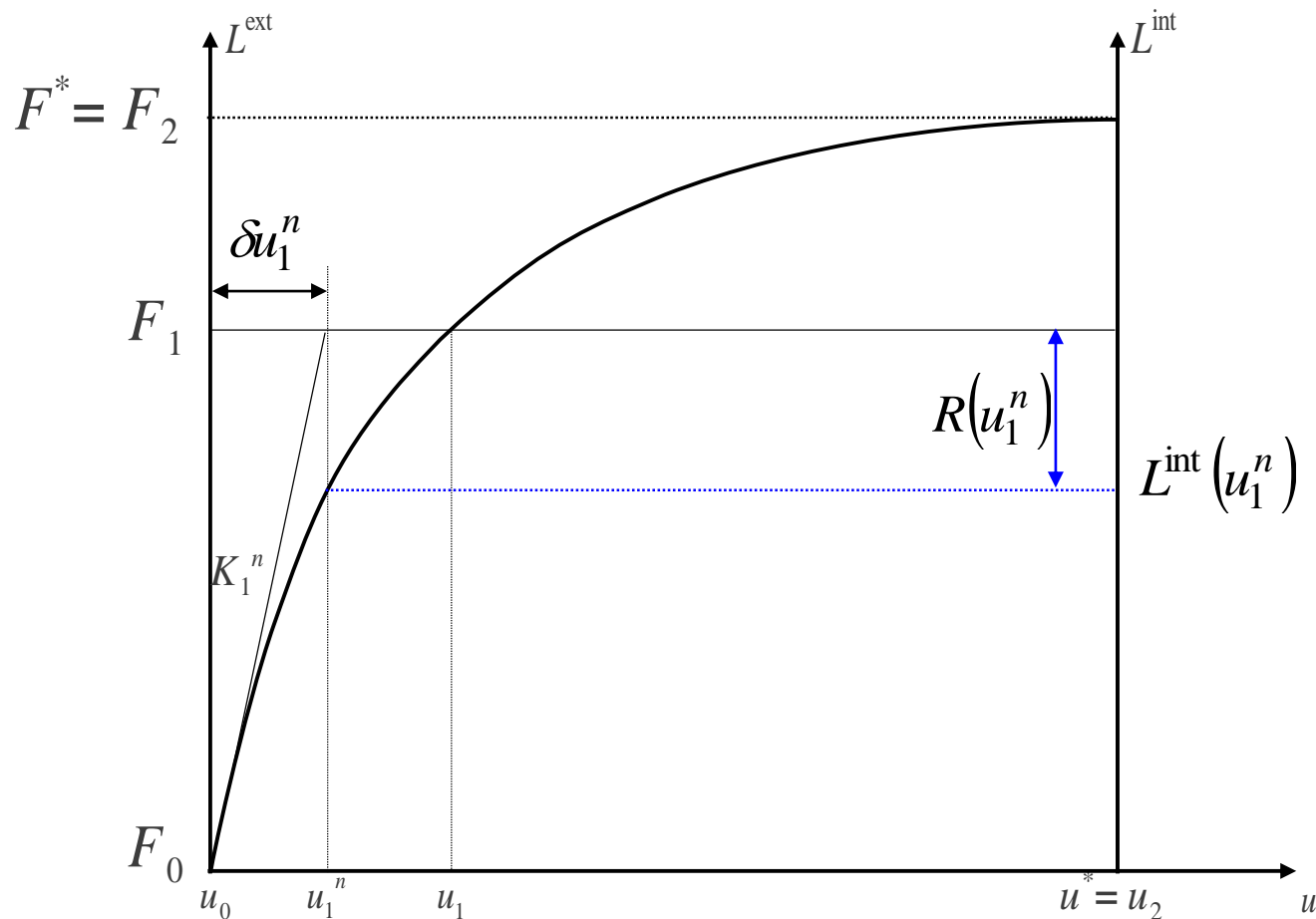
Solving non-linear problem – Newton by graphic

- Update displacements $u_1^n = u_1^{n-1} + \delta u_1^n$

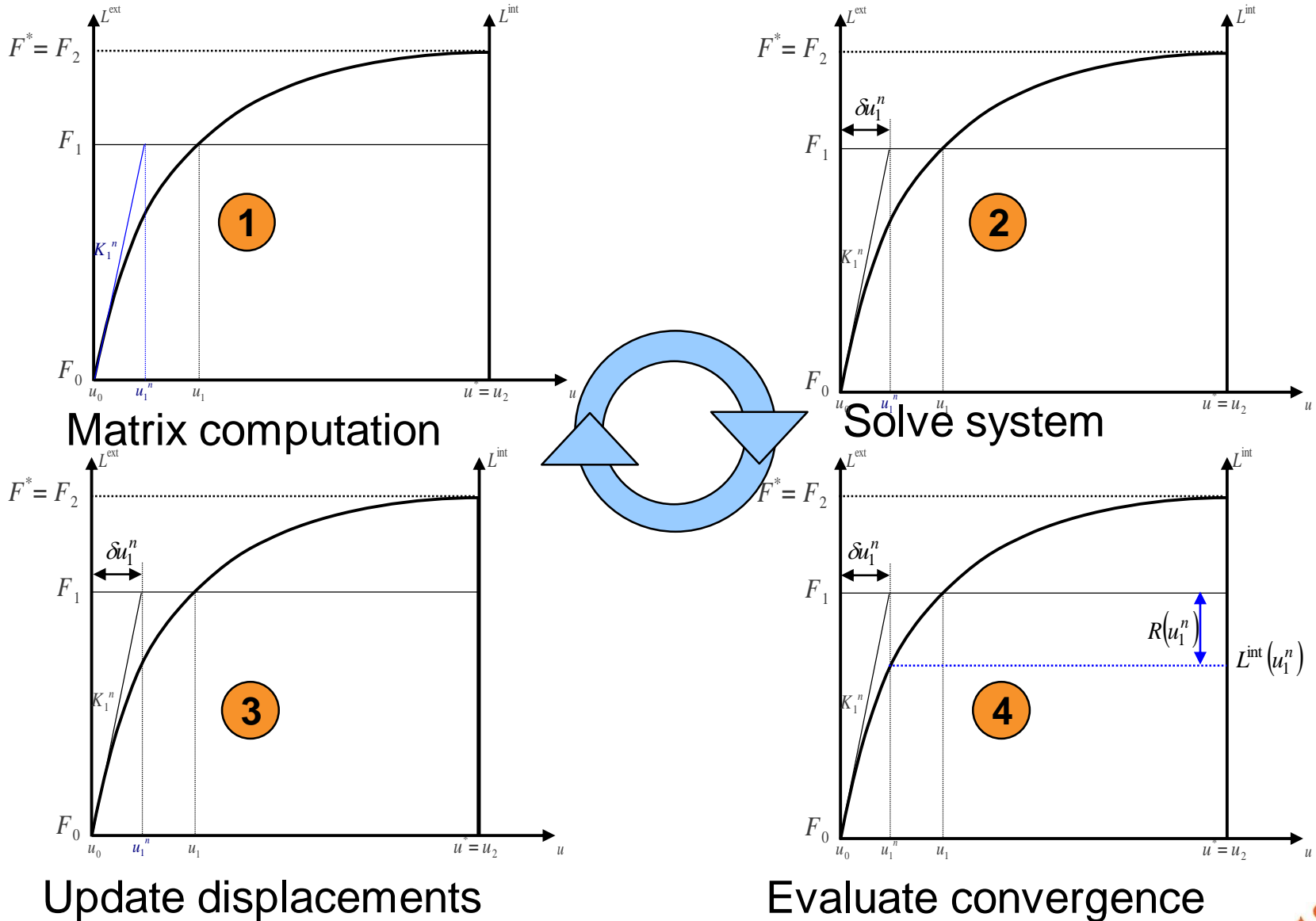


Solving non-linear problem – Newton by graphic

- ▶ Compute $L^{\text{int}}(u_1^n)$ and $R = F_1 - L^{\text{int}}(u_1^n)$



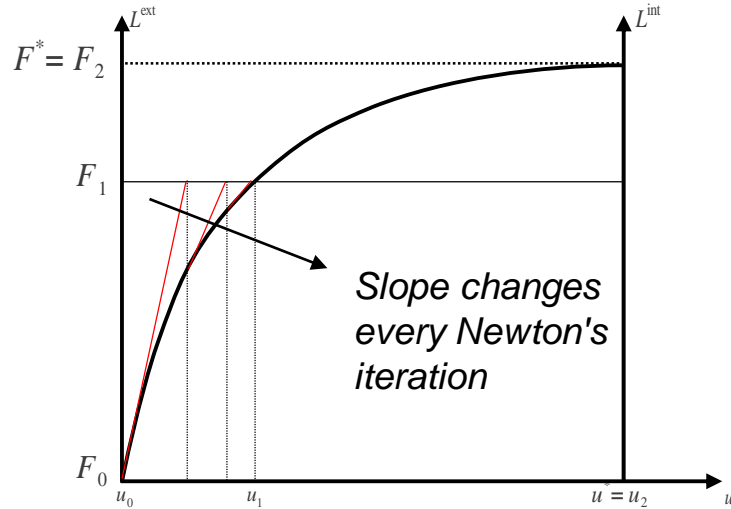
Solving non-linear problem – Newton by graphic



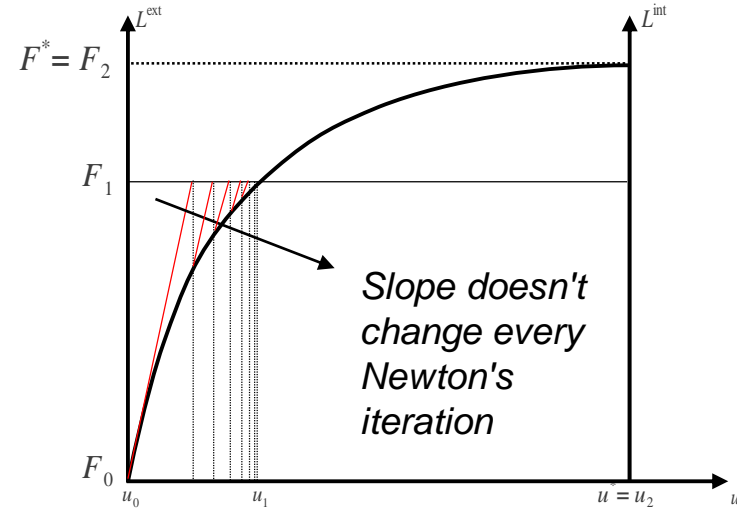
Solving non-linear problem

1 Matrix computation :

- True Newton method : K_i^n is evaluate every step and every Newton iteration
- Quasi-Newton method : K_i^n is evaluate every i Newton's iteration and every n time's step



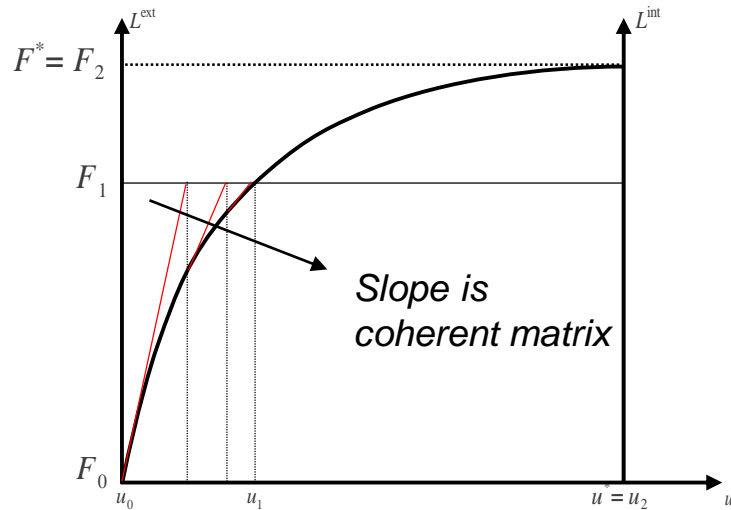
True Newton method



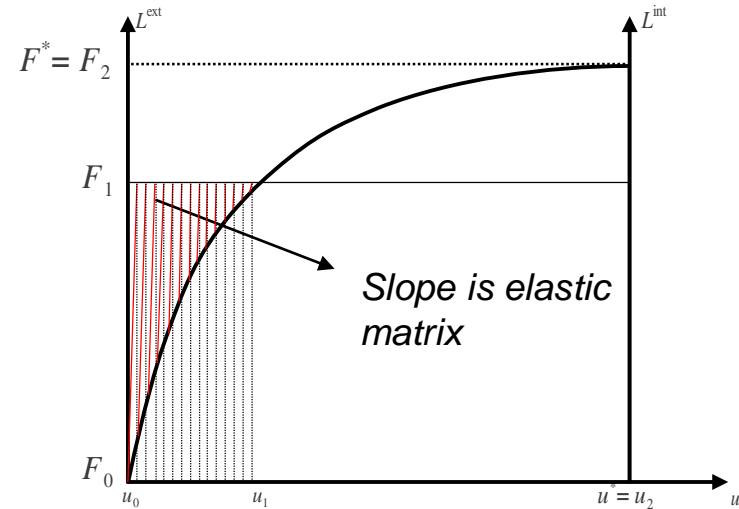
Quasi Newton method

Solving non-linear problem

- 1 Matrix computation :
 - Quasi-Newton method : compute approximate matrix (elastic matrix)



True Newton method



Quasi Newton method

Solving non-linear problem

1 Matrix computation : Why quasi-Newton methods ?

- Compute exact matrix every iteration : matrix must be factorized → very expensive
- Make more iterations but each iteration is quicker
- Elastic matrix achieve convergence for all standard generalized materials with a lot of iterations (thousands). Elastic matrix is computed and factorized ONE time : very cheap
- Elastic matrix is the better choice for unloading loading's cases

Solving non-linear problem

- ② Solve system : using direct solver or iterative solver

Solving non-linear problem

3 Update displacement - Improving by line-search - **Method**

- Solving $K^{n-1}.\delta u^n = R^{n-1}$ is equivalent to **minimizing** functional
- Solving $K^{n-1}.\delta u^n = R^{n-1}$ give the **direction** δu^n of the solution
- Solving functional minimisation give the **line-search coefficient** ρ
- Update displacements

$$u^n = u^{n-1} + \rho.\delta u^n$$

Solving non-linear problem

3 Update displacement - Improving by line-search - **Functional**

- Functional (scalar) J is :

$$u \rightarrow J(u) = \int_{\Omega} \Phi(\varepsilon(u)).d\Omega - \int_{\Omega} f.u.d\Omega - \int_{\Gamma} t.u.d\Gamma$$

- With $\Phi(\varepsilon(u))$ is the density of free energy, for hyperelastic material :

$$\varepsilon = \frac{\partial \Phi}{\partial \sigma}$$

- This functional is **convex**, minimizing convex function \rightarrow its gradient **vanishes** :

$$\nabla J(u).\tilde{u} = 0 \quad \forall \tilde{u} \in E_{\tilde{u}}$$

- Gradient vanishes \Leftrightarrow Virtual power

$$\int_{\Omega} \sigma(u) : \varepsilon(\tilde{u}).d\Omega = \int_{\Omega} f(u) : \tilde{u}.d\Omega + \int_{\Gamma_g} g(u) : \tilde{u}.d\Gamma \quad \forall \tilde{u} \in E_{\tilde{u}}$$

- Find equilibrium : **minimize J scalar functional**

Solving non-linear problem

3 Update displacement - Improving by line-search - **Minimize functional**

- Derivative of the functional

$$\left\{L_{\text{int}}(u^{n-1} + \rho\delta u^n)\right\} - \left\{L_{\text{ext}}(u^{n-1} + \rho\delta u^n)\right\} = \{0\}$$

- Scalar functional

$$g(\rho) = \left\langle \delta u^n \right\rangle \cdot \left\{L_{\text{int}}(u^{n-1} + \rho\delta u^n)\right\} - \left\{L_{\text{ext}}(u^{n-1} + \rho\delta u^n)\right\} = \{0\}$$

- Using classical method for minimizing scalar convex function (dichotomy for instance)

Solving non-linear problem

- 3 Update displacement - Improving by line-search - **Notes**
- Only few iterations : it's not necessary to find *exact* zero of g
 - Only compute internal and external forces (no matrix)
 - Algorithms for minimizing scalar function are very efficient and very simple

Solving non-linear problem

4 Evaluate convergence

- Absolute (**RESI_GLOB_MAXI**)

$$\|L^{ext} - L^{int, n-1}\|_{\infty} \leq \zeta$$

- Relative (**RESI_GLOB_RELA**)

$$\frac{\|L^{ext} - L^{int, n-1}\|_{\infty}}{\|L^{ext}\|_{\infty}} \leq \zeta$$

- By reference : giving stress reference (**RESI_REFE_RELA**)

$$\|L^{ext} - L^{int, n-1}\|_k \leq \gamma \cdot L_k^{ref}$$

Using non-linear in Code_Aster

Non-linear in *Code_Aster*

- ▶ Material is steel with Von Mises plasticity with isotropic hardening, traction curve from file

```
FSIGM = LIRE_FONCTION (UNITE=21, PARA='EPSI')  
  
STEEL = DEFI_MATERIAU (ELAS=_F (YOUNG=210.E9, NU=0.3),  
                      TRACTION=_F (SIGM=FSIGM))
```

- ▶ Steel on all mesh

```
CHMA = AFFE_MATERIAU (MAILLAGE=MESH,  
                     AFFE=_F (TOUT='OUI',  
                              MATER='STEEL'))
```

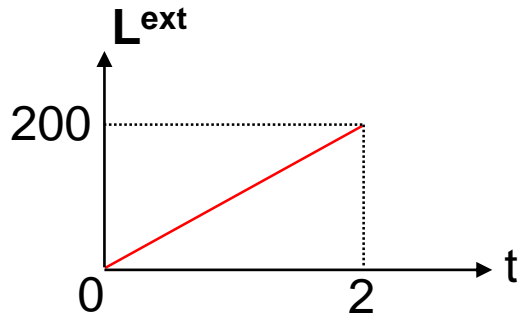
Non-linear in *Code_Aster*

- ▶ Plasticity with isotropic hardening, small strains

```
RESUN = STAT_NON_LINE (...  
    CHAM_MATER = CHMA,  
    COMPORTEMENT = _F(  
        TOUT = 'OUI',  
        RELATION = 'VMIS_ISOT_TRAC',  
        DEFORMATION = 'PETIT'),  
    ...)
```

Non-linear in *Code_Aster*

- ▶ Loading : parametrized by time – Using function in **FONC_MULT**



$$\{L^{ext}(t, u(t))\} = g(t) \cdot \{L^{ext}(u)\}$$

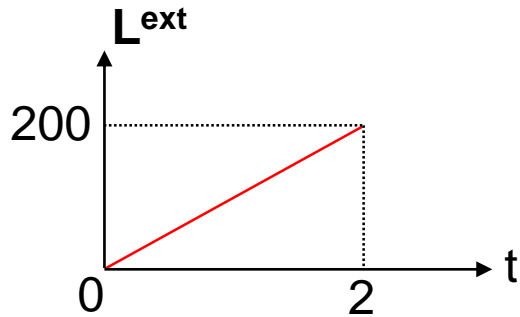
```
RAMPE = DEFI_FONCTION(PARA='INST', VALE=(0,0,2,2))
```

```
LOAD = AFFE_CHAR_MECA(MODELE=MOD,  
                      PRES_REP=_F(PRES=100.,  
                                  GROUP_MA='TOTO'))
```

```
RESUN = STAT_NON_LINE(...  
                      EXCIT=_F(CHARGE =LOAD,  
                               FONC_MULT=RAMPE)  
                      ...)
```

Non-linear in Code_Aster

- ▶ Loading : parametrized by time – Using function directly in BC



$$\{L^{ext}(t, u(t))\} = g(t) \cdot \{L^{ext}(u)\}$$

```
RAMPE = DEFI_FONCTION(PARA='INST', VALE=(0, 0, 2, 200))
```

```
LOAD = AFFE_CHAR_MECA_F(MODELE=MOD,  
                        PRES_REP=_F(PRES=RAMPE,  
                                    GROUP_MA='TOTO'))
```

```
RESUN = STAT_NON_LINE(...  
                      EXCIT=_F(CHARGE =LOAD)  
                      ...)
```

Non-linear in *Code_Aster*

- ▶ Computing is parametrized by time

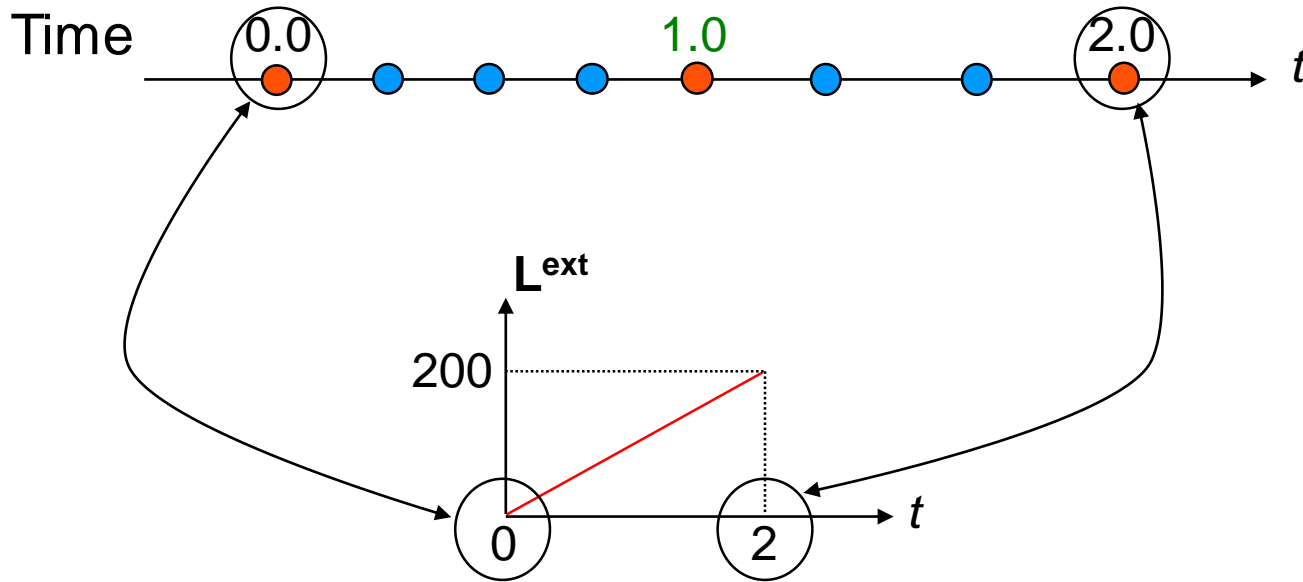


```
L_INST=DEFI_LIST_REEL( DEBUT=0.0,  
                       INTERVALLE=(  
                           _F(JUSQU_A=1.0,NOMBRE=4, ),  
                           _F(JUSQU_A=2.0,NOMBRE=3, ),  
                       ), )
```

```
RESUN = STAT_NON_LINE(...  
                       INCREMENT=_F(LIST_INST=L_INST)  
                       ...)
```

Non-linear in *Code_Aster*

- ▶ Loading and time definition must be **consistent**



- ▶ Computation on **part** of step

```
RESUN = STAT_NON_LINE (...  
    INCREMENT=_F (LIST_INST=L_INST,  
    INST_FIN =1.0)  
    ...)
```

Non-linear in Code_Aster – Advanced controls

- ▶ Automatic step cut : no convergence → automatic cutting

```
L_INST=DEFI_LIST_REEL( DEBUT=0.0,  
                      INTERVALLE=(  
                        _F(JUSQU_A=1.0,NOMBRE=4,)  
                        _F(JUSQU_A=2.0,NOMBRE=3,)  
                      ),)
```

```
DEFLIST = DEFI_LIST_INST(  
          DEFI_LIST = _F(  
              LIST_INST = L_INST,)  
          ECHEC      = _F(  
              EVENEMENT = 'ERREUR',  
              ACTION    = 'DECOUPE',))
```

```
RESUN = STAT_NON_LINE(...  
                    INCREMENT=_F(LIST_INST=DEFLIST)  
                    ...)
```

Non-linear in *Code_Aster* – Advanced controls

► Automatic step adaptation :

```
L_INST=DEFI_LIST_REEL( DEBUT=0.0,  
    INTERVALLE=( _F(JUSQU_A=2.0,NOMBRE=1,) ,  
    ) , )
```

```
DEFLIST = DEFI_LIST_INST( DEFI_LIST = _F(  
    METHODE      = 'AUTO`'  
    LIST_INST    = L_INST ,  
    PAS_MINI     = 1.E-6) , )
```

```
RESUN = STAT_NON_LINE (...  
    INCREMENT=_F(LIST_INST=DEFLIST)  
    ...)
```


Non-linear in *Code_Aster* – Advanced controls

- ▶ Quasi-Newton method : recompute matrix every 2 Newton's iteration

```
RESUN = STAT_NON_LINE (...  
                NEWTON=_F (REAC_ITER=2 , ) , )
```

- ▶ Quasi-Newton method : recompute matrix every 3 time's step

```
RESUN = STAT_NON_LINE (...  
                NEWTON=_F (REAC_INCR=3 , ) , )
```

- ▶ Quasi-Newton method : using elastic matrix

```
RESUN = STAT_NON_LINE (...  
                NEWTON=_F (MATRICE='ELASTIQUE' , ) , )
```

- ▶ Default values in *Code_Aster*

```
RESUN = STAT_NON_LINE (...  
                NEWTON=_F ( REAC_INCR=1 ,  
                          REAC_ITER=0 ,  
                          MATRICE='TANGENTE' , ) , )
```



Non-linear in *Code_Aster* – Advanced controls

- ▶ Convergence criterion : careful when change values !



```
RESUN = STAT_NON_LINE (...  
                CONVERGENCE=_F(  
                RESI_GLOB_MAXI=1.E-6,  
                RESI_GLOB_RELA=1.E-6,))
```

- ▶ Changing criterion convergence may produces **WRONG** results
- ▶ Maximum number of Newton's iteration



```
RESUN = STAT_NON_LINE (  
                CONVERGENCE=_F(ITER_GLOB_MAXI=20,))
```

- ▶ Elastic matrix uses thousands of Newton's iterations
- ▶ Default values in *Code_Aster*

```
RESUN = STAT_NON_LINE (...  
                CONVERGENCE=_F(  
                ITER_GLOB_MAXI=10,  
                RESI_GLOB_RELA=1.E-6,))
```

End of presentation

Is something missing or unclear in this document?
Or feeling happy to have read such a clear tutorial?

Please, we welcome any feedbacks about Code_Aster training materials.
Do not hesitate to share with us your comments on the Code_Aster forum
[dedicated thread](#).