Review – Enforcing constraints with Lagrange Multipliers

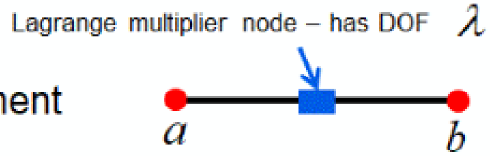
Suppose we wish to enforce a general nonlinear constraint relating displacements component *m* at nodes *a* and *b* $f(u_m^a, u_m^b)$

Augment PVW with a Lagrange Multiplier

$$\int_{V_0} \sigma_{ij} \frac{\partial \eta_i}{\partial x_j} dV + \delta\lambda f(u_m^a, u_m^b) + \lambda \frac{\partial f}{\partial u_m^a} \eta_m^a + \lambda \frac{\partial f}{\partial u_m^b} \eta_m^b = \int_{\partial_2 V_0} t_i^* \eta_i dA0$$

Satisfying this for all $\eta_i, \delta\lambda$ will satisfy equilibrium in weak form and the constraint exactly

FE Implementation: introduce a 'Lagrange Multiplier' element



Element DOF $\mathbf{d} = \begin{bmatrix} u_1^a & u_2^a & u_1^b & u_2^b & \lambda \end{bmatrix}$

Element residual and stiffness:

$$\mathbf{r}^{el} = -\begin{bmatrix} \lambda \partial f / \partial u_1^a \\ \lambda \partial f / \partial u_2^a \\ \lambda \partial f / \partial u_1^b \\ \lambda \partial f / \partial u_2^b \\ f(u_1^a, u_2^a) \end{bmatrix} \qquad \mathbf{k}^{el} = \begin{bmatrix} \lambda \partial^2 f / \partial u_1^{a2} & \lambda \partial^2 f / \partial u_1^a \partial u_2^a & \lambda \partial^2 f / \partial u_1^a \partial u_1^b & \lambda \partial^2 f / \partial u_1^a \partial u_2^b & \partial f / \partial u_1^a \\ & \lambda \partial^2 f / \partial u_2^{a2} & \lambda \partial^2 f / \partial u_2^a \partial u_1^b & \lambda \partial^2 f / \partial u_2^a \partial u_2^b & \partial f / \partial u_2^a \\ & & \lambda \partial^2 f / \partial u_1^{b2} & \lambda \partial^2 f / \partial u_1^b \partial u_2^b & \partial f / \partial u_1^b \\ & Sym & & \lambda \partial^2 f / \partial u_2^{b2} & \partial f / \partial u_2^b \\ & & & & 0 \end{bmatrix}$$

Constraints in EN234FEA

```fortran
subroutine user_constraint(constraint, constraint_flag, n_nodes, node_property_list, dof_list,&          ! Input variables
     n_properties, constraint_properties,nodal_coords,length_coord_array, &
     dof_increment, dof_total, length_dof_array, lagrange_multiplier,  &      ! Input variables
     constraint_stiffness,constraint_residual)      ! Output variables
  use Types
  use ParamIO
  use Mesh_Data, only : node
  implicit none

  integer, intent( in )         :: constraint                                ! Constraint number
  integer, intent( in )         :: constraint_flag                           ! Flag identifying constraint type
  integer, intent( in )         :: n_nodes                                   ! # nodes in the constraint
  integer, intent( in )         :: n_properties                              ! # properties for the element
  integer, intent( in )         :: length_coord_array                        ! Total # coords for element
  integer, intent( in )         :: length_dof_array                          ! Total # Dof for the element
  integer, intent( in )         :: dof_list(n_nodes)                         ! List of DOFs being constrained

  type (node), intent( in )     :: node_property_list(n_nodes)             ! Data structure describing storage for nodal variables - see below
!  type node
!      sequence
!      integer :: flag                          ! Integer identifier
!      integer :: coord_index                   ! Index of first coordinate in coordinate array
!      integer :: n_coords                      ! Total no. coordinates for the node
!      integer :: dof_index                     ! Index of first DOF in dof array
!      integer :: n_dof                         ! Total no. of DOF for node
!   end type node
!   Access these using node_property_list(k)%n_coords eg to find the number of coords for the kth node on the element

  real( prec ), intent( in )    :: nodal_coords(length_coord_array)          ! Coordinates, stored as x1,x2,(x3) for each node in constraint in turn
  real( prec ), intent( in )    :: dof_increment(length_dof_array)           ! DOF increment, stored as du1,du2,du3,du4... for each node in turn
  real( prec ), intent( in )    :: dof_total(length_dof_array)               ! accumulated DOF, same storage as for increment
  real( prec ), intent( in )    :: lagrange_multiplier                       ! Current value of lagrange multiplier

  real( prec ), intent( in )    :: constraint_properties(n_properties)       ! Constraint properties, stored in order listed in input file

  real( prec ), intent( out )   :: constraint_stiffness(length_dof_array,length_dof_array)   ! Constraint stiffness (ROW,COLUMN)
  real( prec ), intent( out )   :: constraint_residual(length_dof_array)                     ! Constraint residual force (ROW)
```

- *Example: for simple equality*

$$u_i^a - u_i^b = 0$$

$$r^{el} = -\begin{bmatrix} \lambda \\ -\lambda \\ u_i^a - u_i^b \end{bmatrix} \qquad [k] = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$

```fortran
  ! Subroutine to set up stiffness for multi--point constraint.  The constraint has one parameter, specifying a (small) compliance.
  constraint_residual = 0.d0
  constraint_stiffness = 0.d0
!
  ndof = node_property_list(1)%n_dof
  dof = dof_list(1)
  constraint_residual(1) = -lagrange_multiplier
  constraint_residual(2) = lagrange_multiplier
  constraint_residual(3) = dof_total(ndof+dof)+dof_increment(ndof+dof)-dof_total(dof)-dof_increment(dof) &
                           -lagrange_multiplier*constraint_properties(1)
  constraint_stiffness(1, 3) = 1.D0
  constraint_stiffness(3, 1) = 1.D0
  constraint_stiffness(2, 3) = -1.D0
  constraint_stiffness(3, 2) = -1.D0
  constraint_stiffness(3,3) = constraint_properties(1)
```

ABAQUS

Degree of freedom version of user subroutine MPC

This version of user subroutine MPC allows for one individual degree of freedom to be constrained and, thus, eliminated at a time. The constraint can be quite general and nonlinear of the form:

$$f(u^1, u^2, u^3, \ldots, u^N, \text{ geometry, temperature, field variables}) = 0.$$

User subroutine interface

```
      SUBROUTINE MPC(UE,A,JDOF,MDOF,N,JTYPE,X,U,UINIT,MAXDOF,
     * LMPC,KSTEP,KINC,TIME,NT,NF,TEMP,FIELD,LTRAN,TRAN)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION A(N),JDOF(N),X(6,N),U(MAXDOF,N),UINIT(MAXDOF,N),
     * TIME(2),TEMP(NT,N),FIELD(NF,NT,N),LTRAN(N),TRAN(3,3,N)


      user coding to define UE, A, JDOF, and, optionally, LMPC


      RETURN
      END
```

Nodal version of user subroutine MPC

The nodal version of user subroutine MPC allows for multiple degrees of freedom of a node to be eliminated simultaneously. The set of constraints can be quite general and nonlinear, of the form

$$f_i(u^1, u^2, u^3, \ldots, u^N, \text{ geometry, temperature, field variables}) = 0 \qquad i = 1, 2, \cdots, \texttt{NDEP}.$$

User subroutine interface

```
      SUBROUTINE MPC(UE,A,JDOF,MDOF,N,JTYPE,X,U,UINIT,MAXDOF,
     * LMPC,KSTEP,KINC,TIME,NT,NF,TEMP,FIELD,LTRAN,TRAN)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION UE(MDOF),A(MDOF,MDOF,N),JDOF(MDOF,N),X(6,N),
     * U(MAXDOF,N),UINIT(MAXDOF,N),TIME(2),TEMP(NT,N),
     * FIELD(NF,NT,N),LTRAN(N),TRAN(3,3,N)


      user coding to define JDOF, UE, A and, optionally, LMPC


      RETURN
      END
```

Augmented Lagrangian

- Goal: Remove zero on diagonal of equation for the constraint

Fix $f(u_i^a, u_i^b) - \epsilon\lambda = 0$

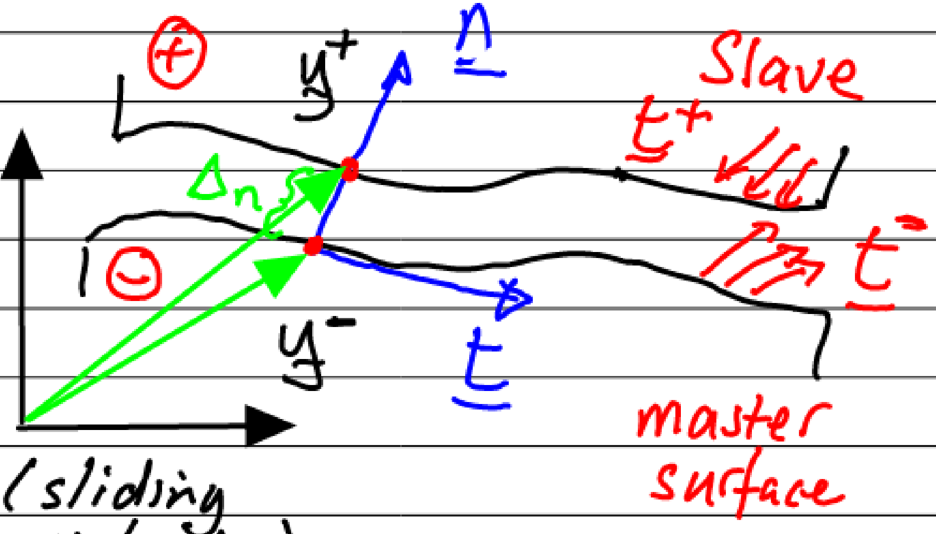
(new constraint equation - ϵ is a small #)

New equations

$$r^{el} = -\begin{bmatrix} \text{same as before} \\ f - \epsilon\lambda \end{bmatrix} \qquad [k^{el}] = \begin{bmatrix} \text{Same as before} & \\ & \epsilon \end{bmatrix}$$

Contact: Very common engineering problem
Tricky to implement in FEA

Contact Mechanics



Geometry: $\underline{t} = \dfrac{\partial y^-}{\partial s}$ $\quad \underline{n} = \underline{e}_3 \times \underline{t}$

$$\Delta_n \underline{n} = y^+ - y^-$$

$$\underline{v}_s = \frac{d\underline{y}^+}{dt} - \frac{d\underline{y}^-}{dt} - \Delta_n \frac{d\underline{n}}{dt} \quad \text{(sliding velocity)}$$

Forces: $T_n = \underline{t}^- \cdot \underline{n}$ $\qquad T_\alpha = \underline{t}^- \cdot \underline{t}^\alpha$

Constitutive equations must relate $(\Delta_n, \underline{v}_s)$ to (T_n, T_α)

Example constitutive laws

- "Hard" frictionless contact

$$\Delta_n \geq 0 \qquad T_n \leq 0 \qquad T_\alpha = 0$$

- Coulomb friction with "hard" contact

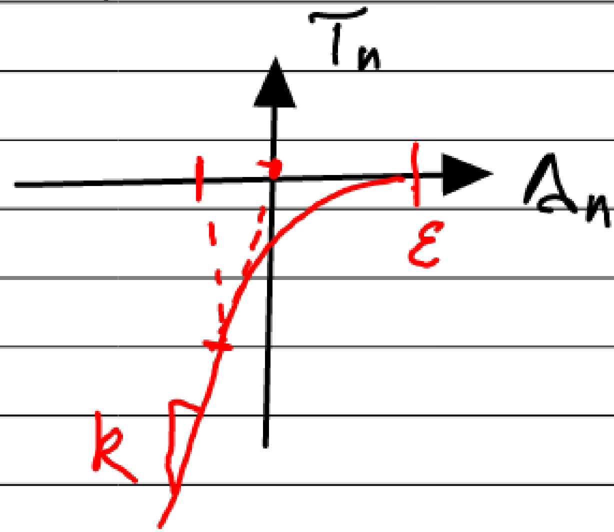
$$\Delta_n \geq 0 \qquad T_n \leq 0$$

$$\left.\begin{array}{l} |\underline{V}_s| = 0 \\ \sqrt{T_\alpha T_\alpha} < \mu |T_n| \end{array}\right\} \text{No slip}$$

$$T_\alpha = -\mu |T_n| \frac{V_s}{|\underline{V}_s|} \qquad \text{slip}$$

Coulomb friction is known to be ill posed

- more sophisticated models will fix this



• "Soft" Contact models

Instead of unilateral constraint
we enforce $\Delta_n \geq 0$ $T_n \leq 0$
with penalty method

ABAQUS uses exponential-linear relation
for $T_n(\Delta_n)$

- Goal is to incorporate contact equations in FEA
- Static & Dynamic analysis uses different method

  Statics: Lagrange multiplier

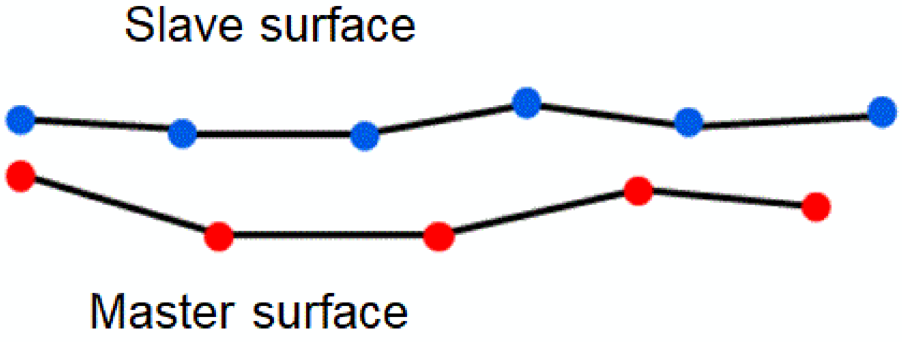
  Dynamics: Predictor - Corrector

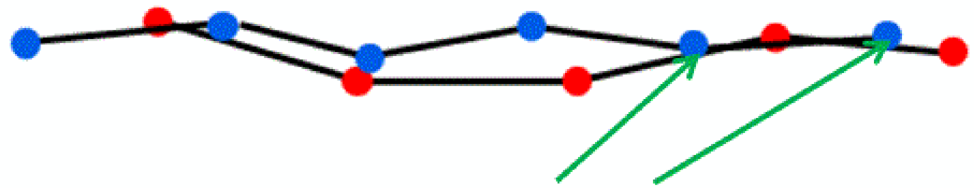
- Focus here on implementing static contact

  Approximations: Hard Contact
  2D
  Neglect friction

  Finite strains & finite sliding

Master - Slave contact algorithm





Constraint will require slave nodes to lie on surface segments that connect master nodes

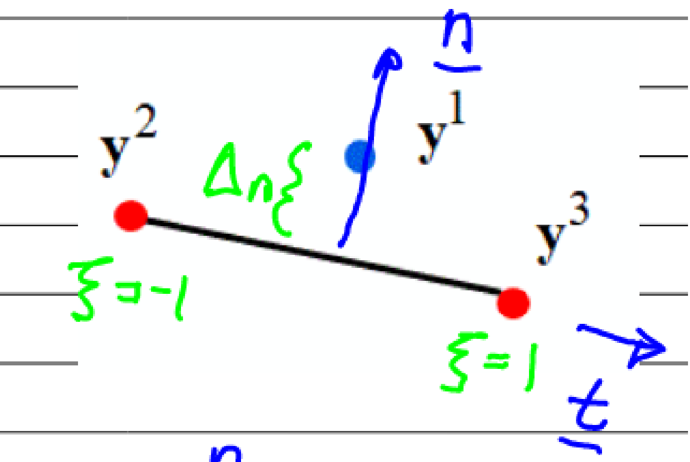
Interpolate geometry of master surface.

Each "slave" interacts with 2 or more "masters"
- Defines connectivity of contact element

Interpolating geometry of master surface

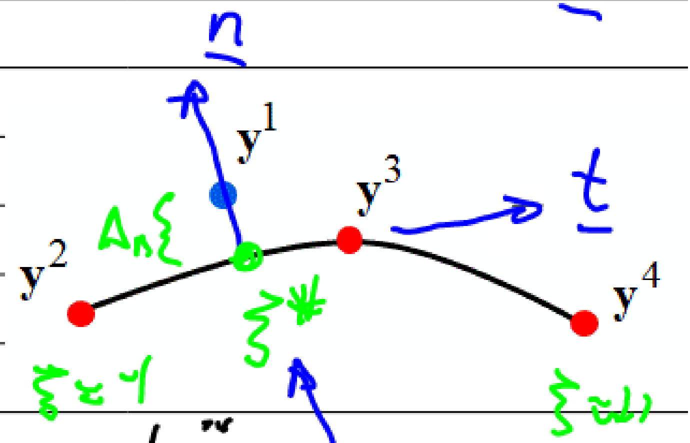
Linear segment

$$y = (1-\xi)/2 \; y^2 + (1+\xi)/2 \; y^3$$



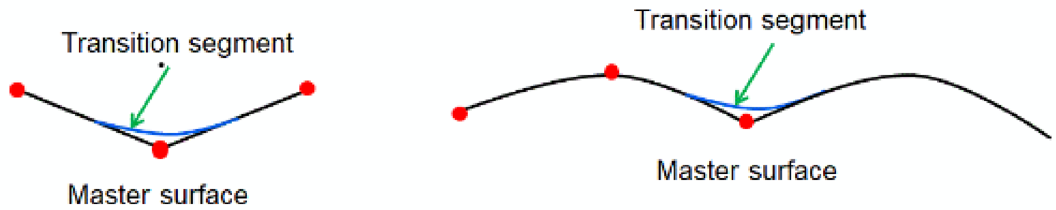
Quadratic

$$y = \xi(1-\xi)/2 \; y^2 + (1-\xi)(1+\xi) \; y^3 + \xi(1+\xi) \; y^4$$



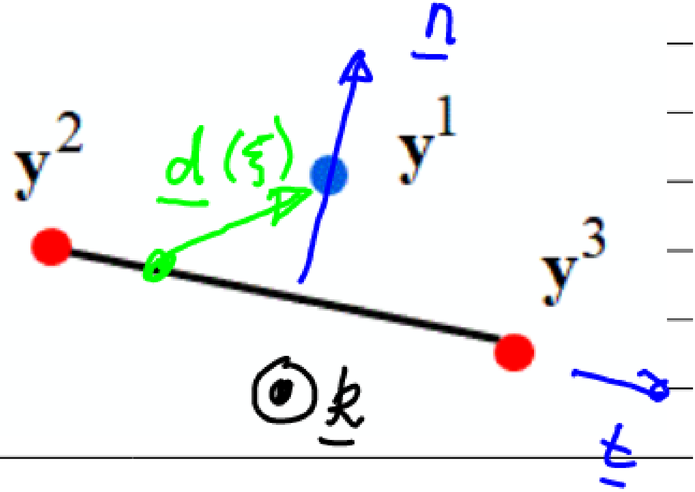
ABAQUS also introduces "transition segments"

Special value of ξ @ contact



Book-keeping: we will need to find ξ^*, $\underline{n}$, $\underline{t}$ Δ_n

Introduce interpolation functions for interacting nodes



$\hat{N}^1 = -1$

$\hat{N}^2 = (\xi-1)/2$ $\quad\quad$ $\hat{N}^3 = (\xi+1)/2$

(similar idea for quadratic & transition segments)

Then $\underline{d} = \underline{y}^1 - \underline{y}(\xi) = -\hat{N}^a \underline{y}^a$ $\quad$ sum on $\underline{a}$

Also $\underline{t} = \dfrac{d\underline{y}}{d\xi} \Big/ \left| \dfrac{\partial \underline{y}}{\partial \xi} \right| = -\dfrac{1}{q} \dfrac{\partial N^a}{\partial \xi} \underline{y}^a$

$\underline{n} = \underline{k} \times \underline{t}$ $\quad\quad$ $q = \left| \dfrac{\partial N^a}{\partial \xi} \underline{y}^a \right|$

To calculate ξ^* note that at ξ^*

$$(\underline{y}' - \underline{y}) \cdot \underline{t} = 0$$

Hence $$\boxed{-N^a(\xi^*)\, \underline{y}^a \cdot \frac{\partial N^b(\xi^*)}{\partial \xi} \underline{y}^b = 0}$$

For linear segments this is a linear eqn for ξ^*

For quadratic segment solve cubic with Newton - Raphson

Finally note $\Delta_n \underline{n} = \underline{y}' - \underline{y}(\xi^*) = -N^a(\xi^*)\, \underline{y}^a$

Hence $$\boxed{\Delta_n = -N^a(\xi^*)\, \underline{y}^a \cdot \underline{n}}$$