segments

Is a struct containing the information about rigid boundary. It consists out of three fields. Field *.points* is a 2x2xN matrix of coordinates structured in the following way.

$$\begin{bmatrix} x0 & y0 \\ x1 & y1 \end{bmatrix} \qquad (1)$$

Coordinates x0 and y0 represent the first point A, while coordinates x1 and y1 represent the second point B of the segment. During the calculation struct Segments is further expanded by the field *.normals* which is a Nx2 matrix. Where N is the number of segments defined in the field *.number*.

When defining the rigid boundary user only has to define the field *.points*. Orientation of each segment has to be respected by the user so that the normal vectors are orientated towards the body of interest. Other two fields are automatically constructed in the solver. Individual segments are independent from each other and do not have to be connected. It is recommended for individual segments to be longer than the size of mesh elements.

propContact

Is a struct containing information about the possible contact nodes defined by the user in GiD. It consists out of three fields. Field *.nodeIDs* is an array that contains global numbering of the nodes. Size of that vector is stored in field *.numberOfNodes.* Both fields are constructed during the parsing from GiD input file. Field *.gap* contains perpendicular distances from each segment to each contact node and it is constructed during calculation of contact. It is a MxN matrix where M is the number of contact nodes and N is the number of segments. Each column is linked to one segment.

User must define the possible contact nodes in GiD. In case of large applied forces, specifying the whole body as a boundary is usually a safe bet since large displacements and deformations in the first step of solver iteration might cause the contact nodes to move far away from the rigid boundary and in turn violating the linear gap function.

removeFullyConstrainedNodes

Is a function that reduces the number of defined contact nodes. It removes fully constrained nodes from the set of possible contact nodes (*propContact.nodeIDs*). Nodes that have both DOFs (x,y) fully constrained do not need to be tested for possible contact with rigid boundary since they will not move. This function is not necessary for the solver, but it can speed up the code.

buildSegmentsData

Is a function that expands a data structure Segments by the field *.normals*. Normal vectors are perpendicular to director and normalized to have the unit length.

buildConstraintMatrix

Is a function that builds the constraint matrix C to be appended to stiffness matrix K. The constraint matrix is built with dimensions MxN, where M is the number of DOFs and N is the number of segments times the number of contact nodes. The matrix is filled with the normal vectors of the segments applied on the right couple of displacement.

buildFullDisplacement

Is a function that rebuilds full expanded displacement vector from the reduced displacement vector. We loop over all degrees of freedom. If we are in location of DOF that was previously deleted

(constrained DOF) we have to add 0, otherwise we simply copy the values from the reduced displacement vector.

lagrange

Is a struct containing information about Lagrange multipliers. It consists out of two fields. Field *.active_nodes* contains the global numbering of the mesh nodes where Lagrange multipliers apply. So only the active contact nodes. While field *.multipliers* stores the values of Lagrange multipliers of that nodes. Struct *lagrange* enables us to visualize the contact nodes.

computeGapFunction

Is a function that expands a data structure *propContact* by the field .gap. Gap function in our case is nothing more than a perpendicular distance from each node to each segment. Gap is calculated in the following way. Point P represents a node in undeformed configuration, while point R is the position of the node after applied displacement u. Gap function is calculated for undeformed configuration u = 0, therefore R = P (2). Point R* is the projection of the point R on the segment AB and can be represented by a linear combination of points A and B (3). As vectors AB and R*R are perpendicular, their dot product must be equal to zero (4). If we substitute (3) into (4) we get a closed form for parametric distance α along the segment AB (5). Now we can easily compute point R*. Gap is then the dot product between normal and vector R*R (6). In case the node is penetrating the distance is negative, otherwise it is positive.

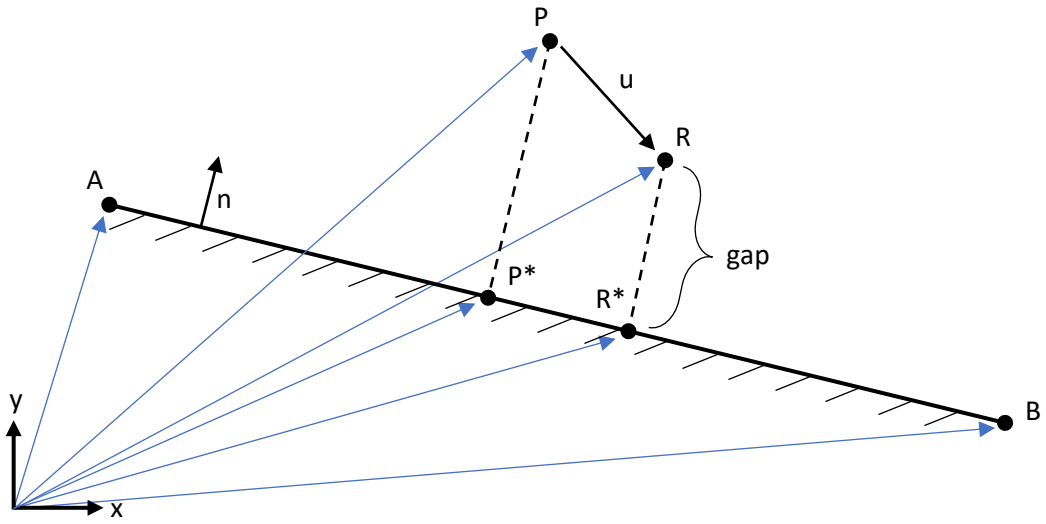

Figure 1: Calculating gap function

$$\mathrm{R} = \mathrm{P} + \mathrm{u} \qquad (2)$$

$$\mathrm{R}^* = (1 - \alpha)\mathrm{A} + \alpha\mathrm{B} \qquad (3)$$

$$(\mathrm{B} - \mathrm{A}) \cdot (\mathrm{R} - \mathrm{R}^*) = 0 \qquad (4)$$

$$\alpha = \frac{(\mathrm{A}-\mathrm{B}) \cdot (\mathrm{R}-\mathrm{A})}{(\mathrm{A}-\mathrm{B}) \cdot (\mathrm{B}-\mathrm{A})} \qquad (5)$$

$$\mathrm{gap} = \mathrm{n} \cdot (\mathrm{R} - \mathrm{R}^*) \qquad (6)$$

detectInactiveNodes

Is a function that detects inactive degrees of freedom in each step of iteration. Inactive DOFs are the ones that are non-penetrating and have non-compressive Lagrange multipliers. To check if a node is non-penetrating, we have to evaluate its gap function to each segment after applied displacement u ≠ 0. Node R must not lie within the area marked on the Figure 2. Calculation is the same as in *computeGapFunction*, but now two extra conditions (7) and (8) must be satisfied.

$$\text{gap} > 0 \qquad (7)$$

$$0 < \alpha \leq 1 \qquad (8)$$

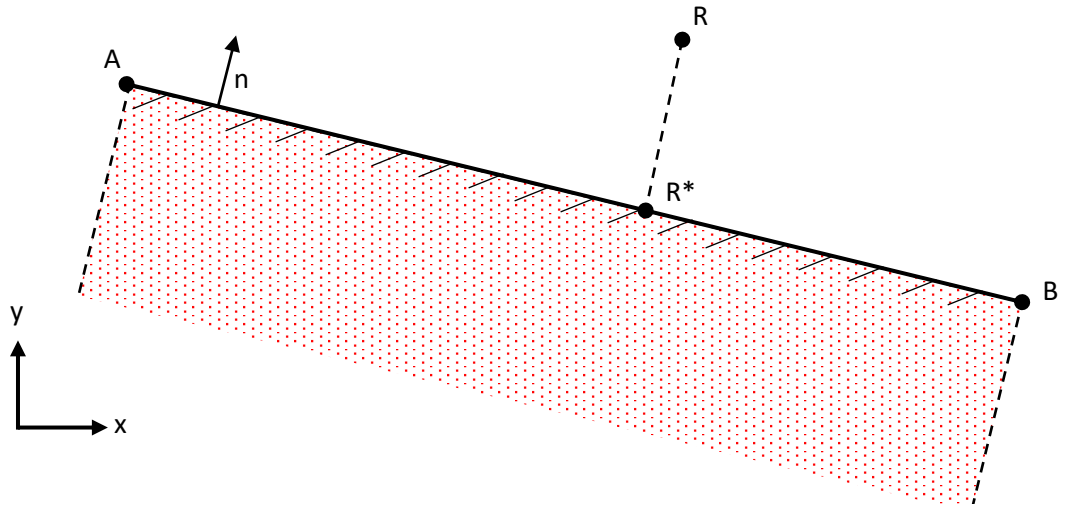

Figure 2: Non-active node R

In addition to that, node must have non-compressive Lagrange multiplier which is checked by (9). If any of aforementioned conditions hold, the node is active.

$$\lambda \leq 0 \qquad (9)$$

solveSignoriniLagrange_1

Is the main function of our solver. It returns the displacement field and the Lagrange multipliers corresponding to a plain stress/strain analysis for the given mesh and geometry together with its Dirichlet and Neumann boundary conditions and the contact constraints for multiple rigids walls by applying the Lagrange multiplier method.

1. Remove fully constrained nodes and compute the initial gap function
2. Compute the master stiffness matrix of the structure
3. Create the expanded system of equations
4. Solve expanded system iteratively

   while

   4.1. Determine the inactive nodes from expanded displacement vector
   4.2. Reduce the system of equations according to the inactive nodes
   4.3. Solve the reduced system of equations
   4.4. Assemble the expanded displacement (Lagrange multiplier) vector
   4.5. Evaluate convergence conditions

   end
5. Get the values for the displacement and the Lagrange multipliers

Expanded system of equations

It is created in the third step of the solver, before entering the while loop.

- *K_exp* is an expanded global stiffness matrix. It consists of global stiffness matrix *K* and constraint matrix *C.*
- *U_exp* is an expanded displacement vector. It contains displacements *u* for each DOF and Lagrange multipliers λ for each node to each segment.
- *F_exp i*s and expanded force vector. It contains applied forces for each DOF and gap functions for each node to each segment.
- NUM = number of segments * number of contact nodes

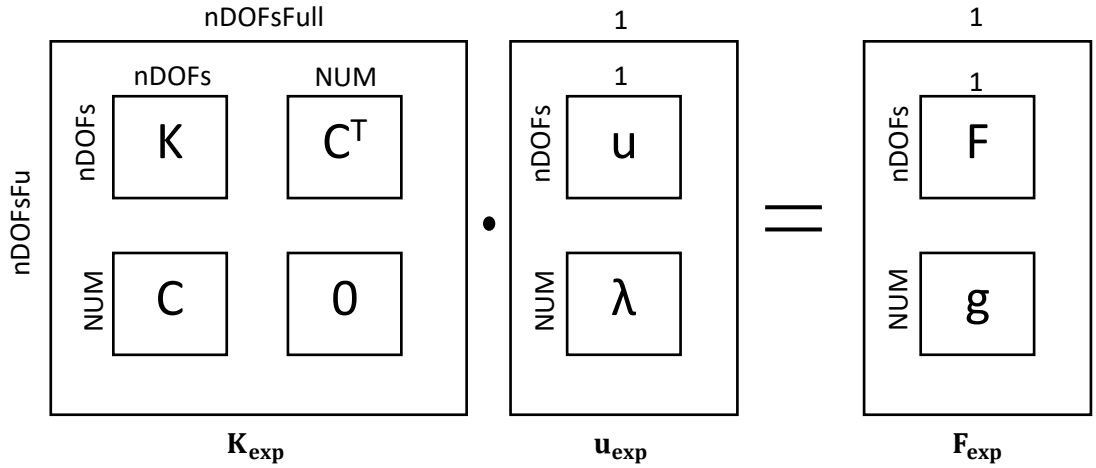

Figure 3: Expanded system of equations

Reduced system of equations

After finding the inactive nodes in each iteration we can merge them together with homDBC and remove corresponding entries in expanded system. We get the reduced system of equations that we can solve to obtain displacements.

$$\mathrm{u_{red}} = \mathrm{K_{red}}^{-1} F_{red} \qquad (10)$$

Convergence conditions

There are two main convergence conditions that need to be meet before we exit the loop. Set of inactive nodes must stay the same as in previous iteration and all Lagrange multipliers have to be valid. In addition, we specify the maximum number of iterations in case our solution does not converge.