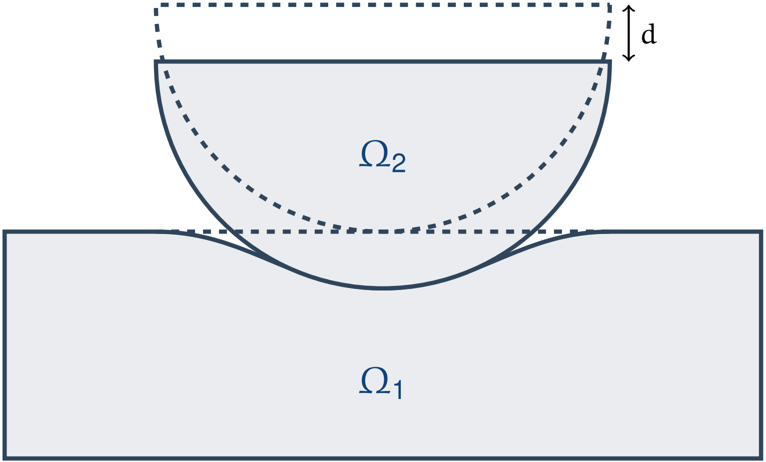
INTERNODES in contact mechanics

February 5, 2023

Bruno Ploumhans & Fabio Matti

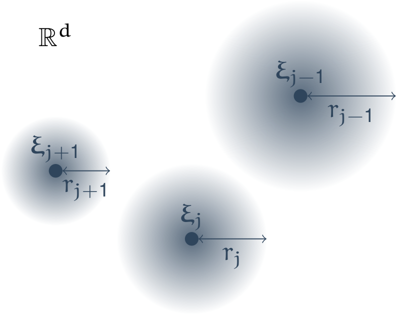
- Recap of INTERNODES method
- Test cases and results
- Implementation discussion


d
Ω_2
Ω_1

Radial basis interpolant of $g : \mathbb{R}^d \to \mathbb{R}$:

$$\Pi(\mathbf{x}) = \sum_{m=1}^{M} g(\xi_m)\phi(\|\mathbf{x} - \xi_m\|, r_m) \quad (1)$$

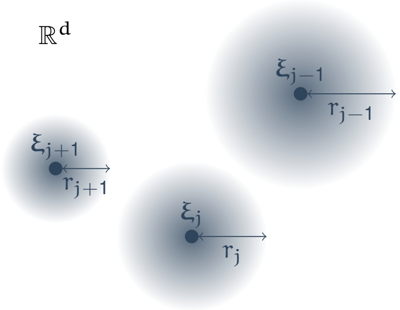
- Radial basis function ϕ
- Interpolation nodes $\xi_1, \ldots, \xi_M$
- Radius parameters $r_1, \ldots, r_M$



Radial basis interpolant of $g : \mathbb{R}^d \to \mathbb{R}$:

$$\Pi(\mathbf{x}) = \sum_{m=1}^{M} g(\xi_m)\phi(\|\mathbf{x} - \xi_m\|, r_m) \quad (1)$$

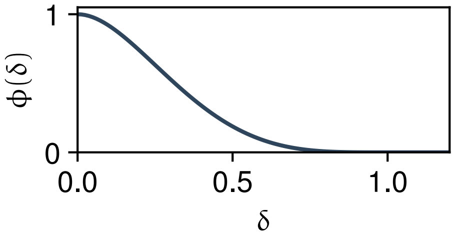
- Radial basis function ϕ
- Interpolation nodes $\xi_1, \ldots, \xi_M$
- Radius parameters $r_1, \ldots, r_M$



Wendland C^2 radial basis function

$$\phi(\delta) = (1 - \delta)_+^4 (1 + 4\delta)$$

with $\delta = \|\mathbf{x} - \xi_m\|/r$ are well suited



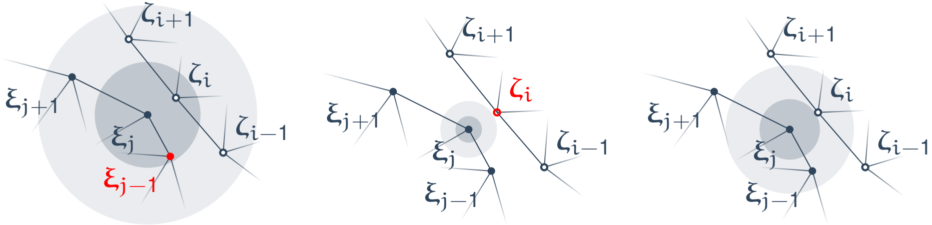
Conditions on radius parameters

There exist $c \in (0, 1)$ and $C \in (c, 1)$ such that

$$\forall i : \#\{j \neq i : \|\xi_i - \xi_j\| < r_i\} < 1/\phi(c) \qquad \text{(Condition 1)}$$

$$\forall i \neq j : \|\xi_i - \xi_j\| \geqslant c r_j \qquad \text{(Condition 2)}$$

$$\forall i, \exists j : \|\zeta_i - \xi_j\| \leqslant C r_j \qquad \text{(Condition 3)}$$



Denoting $\mathbf{g}_\zeta = (g(\zeta_1), \ldots, g(\zeta_N))^\mathsf{T}$ and $\mathbf{g}_\xi = (g(\xi_1), \ldots, g(\xi_M))^\mathsf{T}$ we can write

$$\mathbf{g}_\zeta = \underbrace{\mathbf{D}_{NN}^{-1}\boldsymbol{\Phi}_{NM}\boldsymbol{\Phi}_{MM}^{-1}}_{\mathbf{R}_{NM}}\mathbf{g}_\xi$$

with rescaling $\mathbf{D}_{NN}^{-1}$ to obtain exact interpolation of constant functions and the radial basis matrices

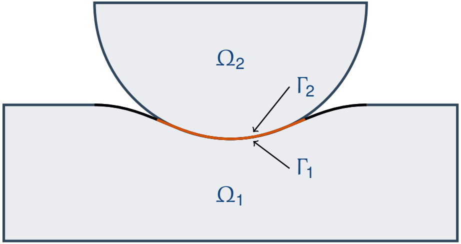
$$(\boldsymbol{\Phi}_{MM})_{ij} = \phi(\|\boldsymbol{\xi}_i - \boldsymbol{\xi}_j\|, r_j) \quad i, j \in \{1, \ldots, M\}$$

$$(\boldsymbol{\Phi}_{NM})_{ij} = \phi(\|\zeta_i - \boldsymbol{\xi}_j\|, r_j) \quad i \in \{1, \ldots, N\},\ j \in \{1, \ldots, M\}$$

The INTERNODES system of equations

Two bodies Ω_1 and Ω_2 with interfaces Γ_1 and Γ_2.

$$\underbrace{\left[\begin{array}{cc|cc|c} \mathbf{K}_{\Omega_1\Omega_1} & \mathbf{K}_{\Omega_1\Gamma_1} & & & \\ \mathbf{K}_{\Gamma_1\Omega_1} & \mathbf{K}_{\Gamma_1\Gamma_1} & & & -\mathbf{M}_{\Gamma_1} \\ \hline & & \mathbf{K}_{\Omega_2\Omega_2} & \mathbf{K}_{\Omega_2\Gamma_2} & \\ & & \mathbf{K}_{\Gamma_2\Omega_2} & \mathbf{K}_{\Gamma_2\Gamma_2} & -\mathbf{M}_{\Gamma_2}\mathbf{R}_{\Gamma_2\Gamma_1} \\ \hline & & & -\mathbf{R}_{\Gamma_1\Gamma_2} & \end{array}\right]}_{=\mathbf{A}} \left[\begin{array}{c} \mathbf{u}_{\Omega_1} \\ \mathbf{u}_{\Gamma_1} \\ \hline \mathbf{u}_{\Omega_2} \\ \mathbf{u}_{\Gamma_2} \\ \hline \boldsymbol{\lambda} \end{array}\right] = \underbrace{\left[\begin{array}{c} \mathbf{f}_{\Omega_1} \\ \mathbf{f}_{\Gamma_1} \\ \hline \mathbf{f}_{\Omega_2} \\ \mathbf{f}_{\Gamma_2} \\ \hline \mathbf{d} \end{array}\right]}_{=\mathbf{b}} \qquad (2)$$



Valid solution only if inequality constraints hold:

1. No interface nodes are in tension:

$$\boldsymbol{\lambda} \cdot \mathbf{n} \leqslant 0 \tag{3}$$

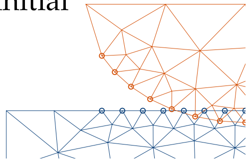
2. No interpenetrating interface nodes:

$$\mathbf{d}' \cdot \mathbf{n} \geqslant 0 \tag{4}$$

Procedure for solving contact problems

Start from an initial configuration and iterate:

initial



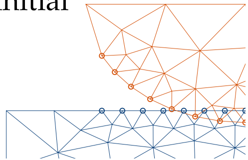
Algorithm for contact mechanics

Procedure for solving contact problems

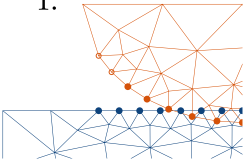
Start from an initial configuration and iterate:

1. Find interface nodes and radius parameters r_m

initial



1.

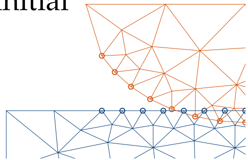


Procedure for solving contact problems

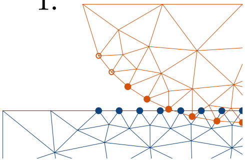
Start from an initial configuration and iterate:

1. Find interface nodes and radius parameters r_m
2. Assemble $\mathbf{A}$ and $\mathbf{b}$, and solve the linear system $\mathbf{Ax} = \mathbf{b}$

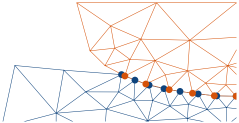
initial



1.



2.



Algorithm for contact mechanics

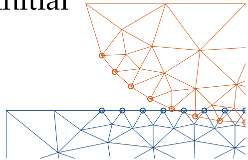
Procedure for solving contact problems

Start from an initial configuration and iterate:

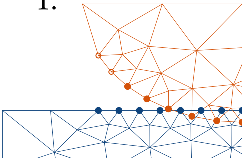
1. Find interface nodes and radius parameters r_m
2. Assemble $\mathbf{A}$ and $\mathbf{b}$, and solve the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$

If the inequality constraints are verified $\implies$ **RETURN**

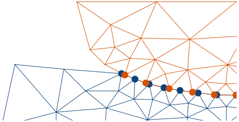
initial



1.



2.



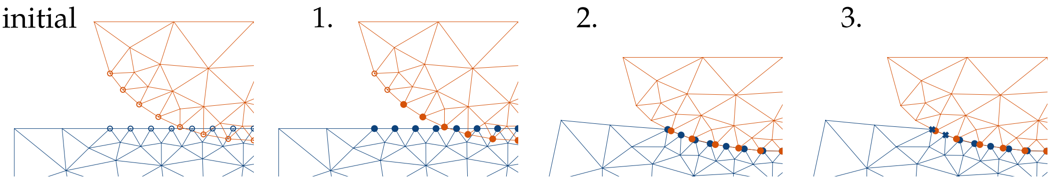
Procedure for solving contact problems

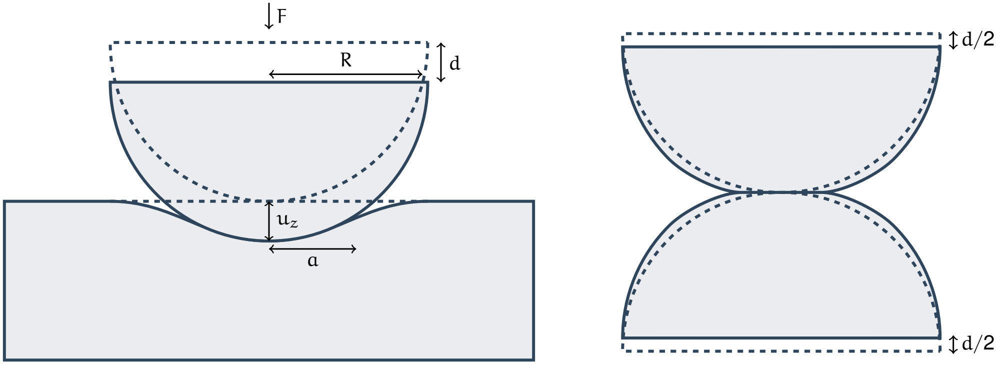
Start from an initial configuration and iterate:

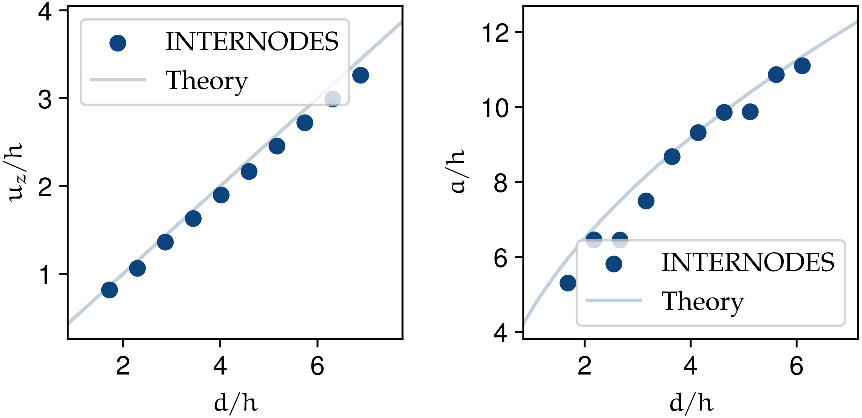
1. Find interface nodes and radius parameters r_m
2. Assemble $\mathbf{A}$ and $\mathbf{b}$, and solve the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$

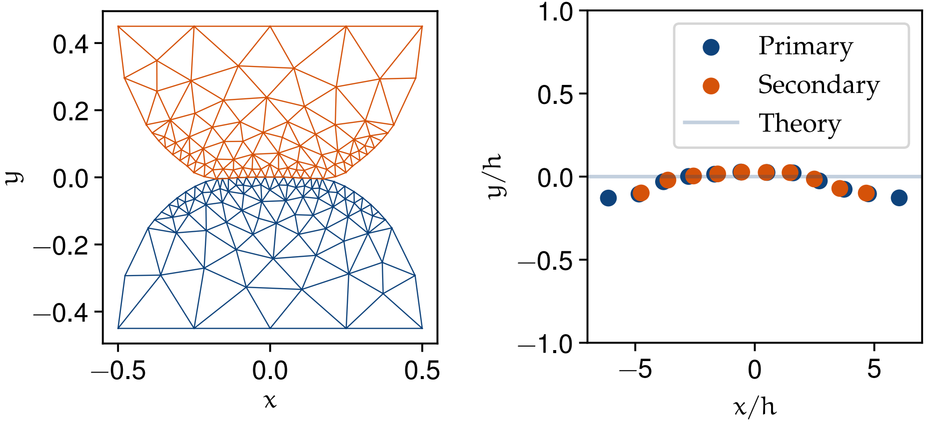
If the inequality constraints are verified $\implies$ **RETURN**

3. Update the set of active nodes




F
R
d
u_z
a
d/2
d/2


INTERNODES
Theory
u_z/h
d/h
1
2
3
4
2
4
6
a/h
d/h
4
6
8
10
12
2
4
6
INTERNODES
Theory


0.4
0.2
0.0
−0.2
−0.4
y
−0.5
0.0
0.5
x
1.0
0.5
0.0
−0.5
−1.0
y/h
−5
0
5
x/h
Primary
Secondary
Theory

Starting point

- Existing Akantu contact mechanics implementation, using the "penalty" method, centered around `ContactMechanicsModel` and its `ContactDetector`.
- Original MATLAB reference implementation of INTERNODES by a previous student (Yannis).
- Python reference implementation of INTERNODES by another previous student (Moritz).
- Partial C++ implementation of INTERNODES by Moritz.

The goal is to provide a usable INTERNODES implementation in Akantu, with attention to a few points:

- Robust implementation and extensive tests, to validate the method.
- Common architecture for penalty and INTERNODES contact, to make future developments easier and keep the code neatly organized.
- Similar Python interface for penalty and INTERNODES contact, to make switching between the methods easy.

Because three implementations were not enough, we decided to write a fourth one...

We refactored Moritz's Python prototype and used it as a testing ground.

- Carefully documented every step, with references to the relevant papers.
- Convergence check prototyping and first working implementation.
- Test case prototyping and first working implementation.

All of these improvements were ported to C++.

C++: the convergence check

- Brief reminder of what the convergence check is.
- Missing in Moritz's C++ implementation, had to be ported from the extended Python prototype.
- Normal computations need special attention to work both in 2D and 3D.
- The set of active nodes changes at each iteration.
  - Number of λ variables changes at each iteration, but we cannot resize the DOFs (degrees of freedom) in Akantu.
  - Allocate one λ variable for each candidate node.
  - Add identity submatrix for the inactive variables to preserve numerical stability.

For radius parameter search, we need to list the nodes in a (relatively small) circle. How can we do this efficiently?

- Option 1: Iterate over each node, check if it's in the circle.

For radius parameter search, we need to list the nodes in a (relatively small) circle. How can we do this efficiently?

- Option 1: Iterate over each node, check if it's in the circle.
- Option 2: We can build a spatial grid instead, and only check neighboring cells!

  - Also used by the penalty method, so grid building is shared between the methods via `AbstractContactDetector`.

- Penalty: separate contact mechanics implementation and solid mechanics implementation, bridged by a `CouplerSolidContact`.
  - Dispatches to contained `ContactMechanicsModel`.
  - Dispatches to contained `SolidMechanicsModel`.

- Penalty: separate contact mechanics implementation and solid mechanics implementation, bridged by a `CouplerSolidContact`.
  - Dispatches to contained `ContactMechanicsModel`.
  - Dispatches to contained `SolidMechanicsModel`.
- INTERNODES: the `ContactMechanicsInternodesModel` contains a solid mechanics implementation.
  - Contains the INTERNODES contact mechanics code.
  - Dispatches to contained `SolidMechanicsModel`.

Architecture concerns

- Penalty: separate contact mechanics implementation and solid mechanics implementation, bridged by a `CouplerSolidContact`.
  - Dispatches to contained `ContactMechanicsModel`.
  - Dispatches to contained `SolidMechanicsModel`.
- INTERNODES: the `ContactMechanicsInternodesModel` contains a solid mechanics implementation.
  - Contains the INTERNODES contact mechanics code.
  - Dispatches to contained `SolidMechanicsModel`.
- Refactor INTERNODES to use a coupler too?

- Penalty: separate contact mechanics implementation and solid mechanics implementation, bridged by a `CouplerSolidContact`.
  - Dispatches to contained `ContactMechanicsModel`.
  - Dispatches to contained `SolidMechanicsModel`.
- INTERNODES: the `ContactMechanicsInternodesModel` contains a solid mechanics implementation.
  - Contains the INTERNODES contact mechanics code.
  - Dispatches to contained `SolidMechanicsModel`.
- Refactor INTERNODES to use a coupler too?
  - Problem: Sharing the coupler code is hard due to algorithmic differences. Lots of templates, template specialization and inheritance leading to very complicated code.

Architecture concerns

- Penalty: separate contact mechanics implementation and solid mechanics implementation, bridged by a `CouplerSolidContact`.
  - Dispatches to contained `ContactMechanicsModel`.
  - Dispatches to contained `SolidMechanicsModel`.
- INTERNODES: the `ContactMechanicsInternodesModel` contains a solid mechanics implementation.
  - Contains the INTERNODES contact mechanics code.
  - Dispatches to contained `SolidMechanicsModel`.
- Refactor INTERNODES to use a coupler too?
  - Problem: Sharing the coupler code is hard due to algorithmic differences. Lots of templates, template specialization and inheritance leading to very complicated code.
  - Problem: The standalone contact models are not usable, yet they extend `Model`.

Architecture concerns: what we really want

The goal is to easily swap from one contact mechanics implementation to the other using the Python interface. This can be done without refactoring to a coupler.

```
  mesh = aka.Mesh(spatial_dimension)
  mesh.read(mesh_file)
- model = aka.CouplerSolidContact(mesh)
+ model = aka.ContactMechanicsInternodesModel(mesh)
  model.applyBC(...)
  # and so on...
```

Current status: works. However, Python uses dynamic typing hence the common interface is not well defined.

We can introduce a well-defined C++ common interface implemented by both `CouplerSolidContact` and `ContactMechanicsInternodesModel`.

```
class SolidContactModel : public Model {
public:
  virtual ~SolidContactModel() = 0;

  // Polymorphic accessors
  virtual Model & getContactMechanicsModel() = 0;
  virtual AbstractContactDetector & getContactDetector() = 0;
  virtual SolidMechanicsModel & getSolidMechanicsModel() = 0;

  // Helper methods, for example:
  template <typename FunctorType>
  inline void applyBC(const FunctorType & func) {
    getSolidMechanicsModel().applyBC(func);
  }
  // ...
};
```

Contact mechanics renames

- For a long time, Akantu only had a single implementation of contact mechanics, which is why its classes use generic names such as `ContactMechanicsModel` or `ContactDetector`.
- Now that there is a second contact mechanics implementation, we propose renaming the penalty method classes to more appropriate names.
  - `ContactMechanicsModel` → `ContactMechanicsPenaltyModel`.
  - `ContactDetector` → `ContactDetectorPenalty`.
  - `CouplerSolidContact` → `CouplerSolidContactPenalty`.
- We intend to carry out these impactful refactors with Nico's approval once INTERNODES is merged.

It’s possible to step through Akantu C++ code, print variables, etc... even when running it through a Python script. A very handy trick for debugging !

```
$ gdb python
```

It’s possible to step through Akantu C++ code, print variables, etc... even when running it through a Python script. A very handy trick for debugging !

```
$  gdb python
Reading symbols from python...
Reading symbols from /usr/lib/debug/.build-id/53/d7bffd6d10967f934c73627bd679a8ae9f62db.debug...
(gdb)  b findContactNodes
```

It's possible to step through Akantu C++ code, print variables, etc... even when running it through a Python script. A very handy trick for debugging !

```
$ gdb python
Reading symbols from python...
Reading symbols from /usr/lib/debug/.build-id/53/d7bffd6d10967f934c73627bd679a8ae9f62db.debug...
(gdb) b findContactNodes
Function "findContactNodes" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
```

It's possible to step through Akantu C++ code, print variables, etc... even when running it through a Python script. A very handy trick for debugging !

```
$ gdb python
Reading symbols from python...
Reading symbols from /usr/lib/debug/.build-id/53/d7bffd6d10967f934c73627bd679a8ae9f62db.debug...
(gdb) b findContactNodes
Function "findContactNodes" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (findContactNodes) pending.
(gdb) r test_contact2d_circle.py
```

It's possible to step through Akantu C++ code, print variables, etc... even when running it through a Python script. A very handy trick for debugging !

```
$ gdb python
Reading symbols from python...
Reading symbols from /usr/lib/debug/.build-id/53/d7bffd6d10967f934c73627bd679a8ae9f62db.debug...
(gdb) b findContactNodes
Function "findContactNodes" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (findContactNodes) pending.
(gdb) r test_contact2d_circle.py
...
Thread 1 "python" hit Breakpoint 1, 0x00007fffbd83cdf0 in akantu::ContactDetectorInternodes
::findContactNodes(akantu::NodeGroup&, akantu::NodeGroup&)@plt () from /home/bp/akantu/build/src/libakantu.so
(gdb) next
```

It's possible to step through Akantu C++ code, print variables, etc... even when running it through a Python script. A very handy trick for debugging !

```
$  gdb python
Reading symbols from python...
Reading symbols from /usr/lib/debug/.build-id/53/d7bffd6d10967f934c73627bd679a8ae9f62db.debug...
(gdb) b findContactNodes
Function "findContactNodes" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (findContactNodes) pending.
(gdb) r test_contact2d_circle.py
...
Thread 1 "python" hit Breakpoint 1, 0x00007fffbd83cdf0 in akantu::ContactDetectorInternodes
::findContactNodes(akantu::NodeGroup&, akantu::NodeGroup&)@plt () from /home/bp/akantu/build/src/libakantu.so
(gdb) next

Thread 1 "python" hit Breakpoint 1, akantu::ContactDetectorInternodes
::findContactNodes (this=0x154e5f0, master_node_group=..., slave_node_group=...)
    at /home/bp/akantu/src/model/contact_mechanics_internodes/contact_detector_internodes.cc:105
105     void ContactDetectorInternodes::findContactNodes(NodeGroup & master_node_group,
                                                          NodeGroup & slave_node_group) {
(gdb)
```