KU LEUVEN

ANNA Newtonian Noise Analysis: Recent developments

Georgia Kuci, Stijn François and Geert Degrande

Structural Mechanics Section, Department of Civil Engineering, KU Leuven

EMR Workshop, Maastricht, 26.08.2026

Introduction

Recent developments

- Updated finite element formulation and integration strategy
  - no dedicated mesh refinement near the test mass
  - adaptive integration of the element Newtonian noise matrices
- Extended element implementation and verification
  - solid and surface finite elements
  - spatially varying density
- Shared native C++ core
  - MATLAB interface through MEX
  - Python interface through nanobind

Adaptive integration

Triangular integration domains

- For the current triangular integration domain, identify the closest corner node to the mirror position $\mathbf{x}_0$,

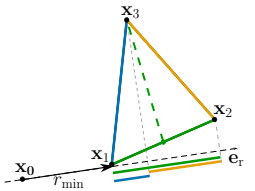
$$r_{\min} = \min_i \|\mathbf{x}_i - \mathbf{x}_0\| \ , \qquad \mathbf{e}_{\mathrm{r,min}} = \frac{\mathbf{x}_{\min} - \mathbf{x}_0}{\|\mathbf{x}_{\min} - \mathbf{x}_0\|} \ . \tag{1}$$

- For each edge (i, j) of the triangle, compute its projected length along the radial direction $\mathbf{e}_{\mathrm{r,min}}$,

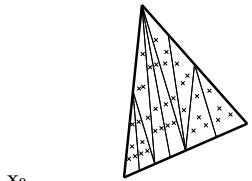
$$l_{ij} = \left|(\mathbf{x}_j - \mathbf{x}_i) \cdot \mathbf{e}_{\mathrm{r,min}}\right| \ . \tag{2}$$

  If $\max(l_{ij}) > \alpha r_{\min}$, split the edge with the largest projected length at its midpoint. The default value is $\alpha = 0.1$.

- Repeat recursively until all subtriangles satisfy the refinement criterion, then apply the standard quadrature rule.



(a)



(b)

KU LEUVEN

Adaptive integration

Quadrilateral integration domains

- For the current quadrilateral integration domain, identify the closest corner node to the mirror position $\mathbf{x}_0$,

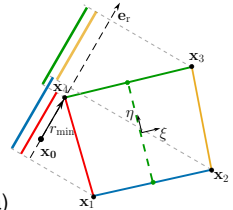
$$r_{\min} = \min_i \|\mathbf{x}_i - \mathbf{x}_0\| \ , \qquad \mathbf{e}_{\mathrm{r,min}} = \frac{\mathbf{x}_{\min} - \mathbf{x}_0}{\|\mathbf{x}_{\min} - \mathbf{x}_0\|} \ . \tag{3}$$

- For each edge (i, j) of the quadrilateral, compute its projected length along the radial direction $\mathbf{e}_{\mathrm{r,min}}$,

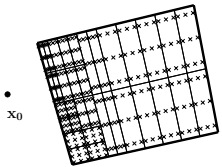
$$l_{ij} = \left|(\mathbf{x}_j - \mathbf{x}_i) \cdot \mathbf{e}_{\mathrm{r,min}}\right| \ . \tag{4}$$

  If $\max(l_{ij}) > \alpha r_{\min}$, split the quadrilateral along the corresponding parametric direction (ξ or η).

- Repeat recursively until all sub-quadrilaterals satisfy the refinement criterion, then apply the standard quadrature rule.



(a)



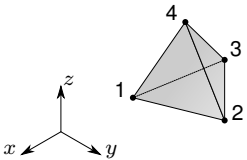
(b)

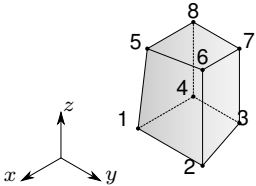
KU LEUVEN

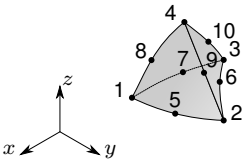
Element library

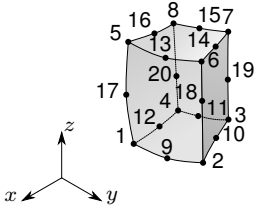
Solid elements

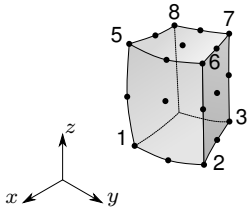
- Solid (volume) elements: `solid4`, `solid8`, `solid10`, `solid20`, `solid27`, `solid125`.

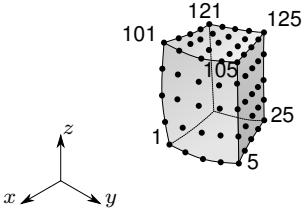










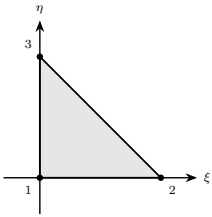


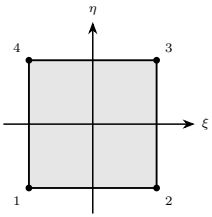
KU LEUVEN

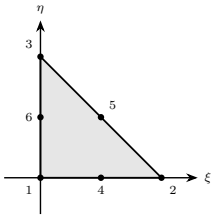
Element library

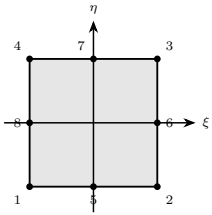
Surface elements

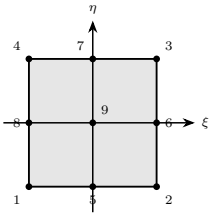
- Surface (boundary) elements: `surf3`, `surf4`, `surf6`, `surf8`, `surf9`, `surf25`.

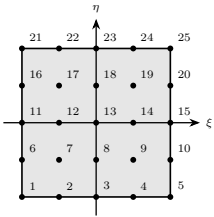












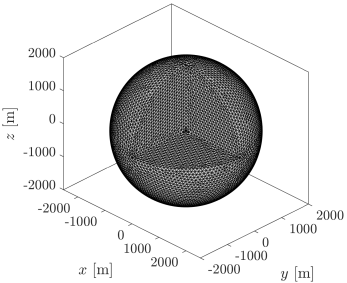
KU LEUVEN

ANNA Workflow

Newtonian noise computation

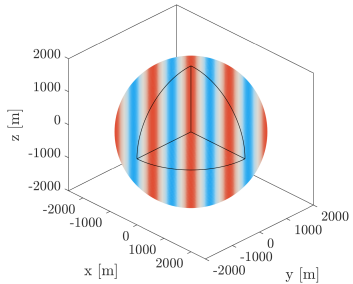
1. FE mesh

- Generated in Gmsh (or any external meshing software).
- Supports solid (volume) and surface (boundary) elements.



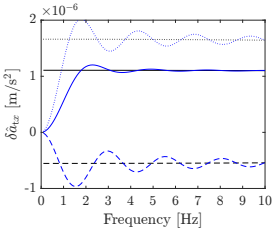
2. Seismic wave field

- Input seismic displacement field $\underline{\hat{\mathbf{u}}}(\omega)$.
- Defined at the FE nodes and stored as a $(3N \times 1)$ vector.



3. NN computation

- Assemble NN matrices (`asmnn`): $\mathbf{A}_\mathrm{t}(\mathbf{x}_0)$, $\mathbf{A}_\mathrm{b}(\mathbf{x}_0)$, $\mathbf{A}_\mathrm{s}(\mathbf{x}_0)$ $(3 \times 3N)$
- Compute NN acceleration: $\delta\hat{\mathbf{a}}_\mathrm{i}(\mathbf{x}_0, \omega) = \mathbf{A}_\mathrm{i}(\mathbf{x}_0)\, \underline{\hat{\mathbf{u}}}(\omega)$



* Analytical reference solution for validation from Harms (2019).

KU LEUVEN

Examples

Overview

- Examples are provided with both the MATLAB and Python interfaces
  - `example1` – Plane harmonic P-wave in a full space.
  - `example2` – Plane harmonic S-wave in a full space.
  - `example3` – Harmonic Rayleigh wave in a halfspace.
  - `example4` – Test case with alternative element types.
  - `example5` – Test case with a height-dependent density field.

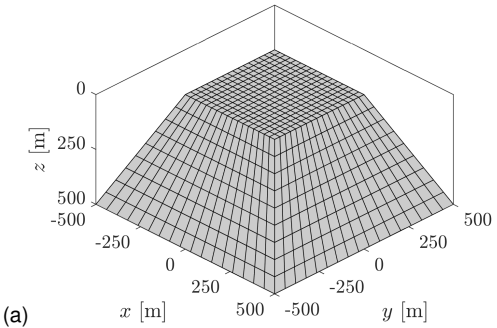
- Typical example structure (`example1`–`example3`):
  - Gmsh geometry (`.geo`) and mesh (`.msh`)
  - main MATLAB/Python script (`.m`/`.py`)
  - analytical reference solution
  - post-processing script

- Additional examples (`example4` and `example5`): separate cases are provided for
  - `solid4` (`surf3`),
  - `solid8` (`surf4`),
  - `solid10` (`surf6`),
  - `solid20` (`surf8`),
  - `solid27` (`surf9`),
  - `solid125` (`surf25`).

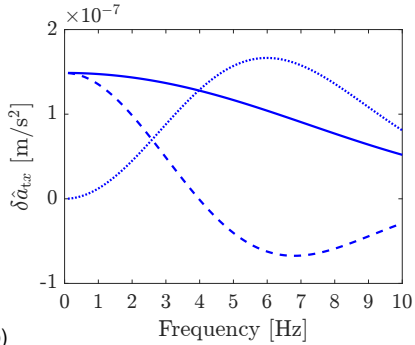
Examples

Alternative element types

- Verification (`Example 4`) extended to all supported solid element types `solid4`, `solid8`, `solid10`, `solid20`, `solid27` and `solid125`.
- Surface contribution $\mathbf{A}_\mathrm{s}$ evaluated either from the faces of the solid elements or independently from a surface (boundary) element mesh.
- Agreement at machine precision between the two evaluations of $\mathbf{A}_\mathrm{s}$.



(a)



(b)

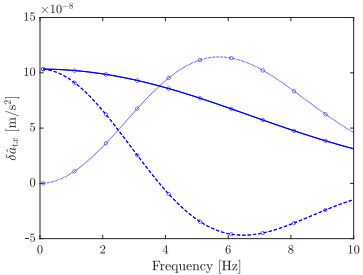
KU LEUVEN

Examples
Spatially varying density

- Spatially varying density (`Example 5`) represented either as
  - a per-node density field `NodalDensity`, interpolated within each element via the element shape functions,
  - an element-wise constant density through `Materials`, using the mean of the element's nodal density values.
- Verification for a linearly varying density field,

$$\rho(z) = \rho_0 \left(0.5 + 0.5\,\frac{z}{h_{\mathrm{dom}}}\right).$$

- Good agreement between both density representations.



ANNA Newtonian Noise Analysis

Structure

- ANNA is organized into three components:
  - `anna-core`: shared native C++ library,
  - `anna-matlab`: MATLAB/MEX interface and MATLAB functions,
  - `anna-python`: Python/nanobind interface and Python functions.
- The performance-critical routines are implemented in `anna-core`:
  - Element shape functions, DOF numbering and quadrature,
  - Newtonian noise element routines and `asmnn` global assembly.
  - Code optimized for performance.
- MATLAB and Python use the same C++ implementation for the computation of the Newtonian noise matrices.
- The C++ core can also be compiled and linked directly with external simulation software (SPECFEM, Salvus), for **in-core Newtonian noise calculation**, or (alternatively) as a postprocessing step.

ANNA Newtonian Noise Analysis

Implementation / compilation / installation

- MATLAB and Python share the same `anna-core` C++ implementation.

`anna-core`

- `include/`, `src/` – shape functions, `nn_*` element routines, `asmnn` assembly
- `tests/` – C++ unit tests (CTest)
- no external dependencies

Build: `cmake -S . -B build`
`cmake --build build`

`anna-matlab`

- `mex/` – MEX wrappers (`*_mex.cpp`)
- `matlab/` – pure MATLAB helper functions
- `examples/`, `manual/`, `tests/`

Build: `annamake`
compiles `anna-core` and every MEX function via CMake into `build/`

Install: add the toolbox folder to the MATLAB path

`anna-python`

- `nanobind/` – nanobind wrappers (`*_nb.cpp`)
- `src/anna/` – Python package
- `examples/`, `manual/`, `tests/`

Build: `python -m build --wheel`
via `scikit-build-core` + `nanobind`: builds `anna-core` and the bindings into a wheel

Install: `pip install anna***.whl`

ANNA Newtonian Noise Analysis

Licensing

- ANNA is distributed under a dual-licensing model:
  - **Open source – GNU GPLv3**
    - Directly available
    - Free for educational and non-commercial use
    - Redistribution/integration allowed under the GPLv3 terms; cannot be integrated, in full or in part, into closed-source (commercial) software
  - **Commercial license**
    - Required for use in closed-source / proprietary software
    - A tailored agreement needs to be drafted case-by-case by KU Leuven Research & Development (LRD)